Pocket

Fabio Brivio

l'Umanista Informatico

XML, HTML, CSS, SQL, Web, Internet, database, programmazione e Google per le Scienze Umane

l'Umanista Informatico

Fabio Brivio

APOGEO

ISBN: 978-88-503-1100-2

~
Seguici su Twitter @apogeonline
Sito web: Apogeonline

L'Umanista Informatico è stato pubblicato per la prima volta in edizione cartacea a febbraio 2009. L'edizione ePUB ha cominciato a circolare in Rete a luglio 2010, nel momento in cui l'editoria digitale italiana assomigliava a una gigantesca start-up nazionale. Pochi mesi dopo è seguita la versione MOBI per i lettori Kindle.

Il contenuto tra le edizioni digitali e quella cartacea è pressoché immutato - salvo la correzioni di alcuni inevitabili refusi. La grande differenza sta nel prezzo e soprattutto nella modalità di distribuzione.

L'edizione cartacea è distribuita e commercializzata nelle librerie al costo di 7,90€. Gli ebook sono rilasciati gratuitamente sotto una licenza Creative Commons che ne autorizza il riutilizzo per fini non commerciali.

Questa scelta è stata motivata da due fattori.

Come autore ho amato scrivere questo libro. Rappresenta un momento importante del mio percorso di formazione, l'archetipo di una nuova figura professionale che continuo a sviluppare e in cui continuo a credere. Permettere la libera circolazione del testo, raggiungere un numero di lettori più ampio, vale molto per me.

Come editor nella primavera del 2010 mi trovavo a preparare le prime pubblicazioni ePUB di Apogeo. La conoscenza della tecnologia era poco più che teorica e un vero test sul campo non era possibile visto che il mercato ancora non esisteva (la vendita cominciò il 18 ottobre). Rilasciare delle pubblicazioni gratuite è stata quindi l'occasione per saggiarne la qualità, raccogliere le reazioni dei lettori e migliorare progressivamente il catalogo e il *workflow* che andavamo configurando.

Con il tempo le motivazioni non sono cambiate e l'ebook continua a essere liberamente scaricabile dal sito dell'editore e da tutti i negozi che lo vogliono offrire ai propri clienti.

I feedback di te lettore che stai guardando queste righe visualizzate su uno schermo sono un modo per ripagare il mio lavoro e quello di Apogeo: una recensione su Anobii, Goodreads, il tuo sito o lo store (Amazon, iBookstore, Kobo, BookRepublic, Simplicissimus...) da cui hai scaricato l'ebook; un tweet a @f4b10b o @apogeonline, o ancora una mail a fabio.brivio@gmail.com.

Inoltre se alla fine della lettura l'ebook ti avrà lasciato qualcosa puoi sempre considerare di acquistarne una copia cartacea ;-)

Buona lettura!

Fabio

Sigarètt senza nomm e büceer senza storia
hann faa i ghirigori nella mia strana memoria
tatüagg invisìbil che me càgnen de nòcc
e una vita tiràda cumè un nastru de scotch...

~

Sigarette senza nome e bicchieri senza storia
hanno fatto i ghirigori nella mia strana memoria
tatuaggi invisibili che mi mordono di notte
e una vita tirata come un nastro di scotch...

La balàda del Genesio - Davide Van de Sfroos

L’informatica e gli umanisti

L’idea di questo libro nasce durante i primi mesi della mia “avventura” in Apogeo. Era l’inverno 2004/05 e mi trovavo impiegato in uno stage che in pochi mesi si sarebbe trasformato nel mio lavoro.

Entravo in redazione con in tasca una laurea in Storia e un master in Informatica e Comunicazione, durante il quale avevo cambiato sensibilmente il mio modo di intendere la Rete e le macchine in genere, provato a giocare un po’ con tecnologie e programmi, e maturato la convinzione di aver qualcosa da dire, ma soprattutto fare, in questo mondo. Ancora però non mi era chiaro come…

Il problema degli umanisti

Come ogni studente laureando o laureato in Scienze Umane sa bene, le nostre lauree (del vecchio o del nuovo ordinamento) sono tanto “ammirate” dal mondo del lavoro, quanto snobbate per possibili assunzioni in favore di discipline più tecniche. Questo perché l’umanista è spesso visto come quello che parla, scrive e magari pensa pure bene, ma a conti fatti è privo di reali abilità pratiche e quindi produttive.

Nel mondo del lavoro servono tecnici, informatici in particolar modo. Alla gente che *sa*, si preferisce la gente che *sa fare*. E l’umanista troppo spesso *sa*, ma non *sa fare*. O almeno così si pensa.

Insomma, l’umanista sembra avere poco da offrire a un’azienda e inoltre con il computer ha poca dimestichezza.

Il problema è noto e alcuni atenei hanno cercato di porvi rimedio inserendo insegnamenti di informatica di livello base o il conseguimento dell’ECDL di livello base, la famigerata patente europea del computer.

L’idea non è in sé sbagliata; in effetti sono ancora troppo pochi gli studenti che si avvicinano al computer solo nel momento in cui è necessario elaborare la tesi. Internet si è discretamente diffusa, ma troppo spesso è anche vista come sinonimo di *file sharing* e posta elettronica. Google, in virtù della sua semplicità e della sua potenza, viene utilizzato in larga misura, ma pochissimi ne sfruttano appieno le reali capacità. Infine le varie soluzioni di comunicazione mobili, cioè integrate nei cellulari di terza generazione, si stanno sì diffondendo rapidamente, ma sono più che altro subite passivamente: nei cellulari la posta elettronica si confonde con SMS e MMS, mentre la Rete è troppo spesso il portale dell’operatore telefonico da cui scaricare immagini, video e suonerie.

Insomma, a voler essere pessimisti per molti, troppi, umanisti il computer è ancora inteso come l’evoluzione della macchina da scrivere, Internet è un’icona sul desktop da cui scaricare musica e film e la rivoluzione mobile una chimera di cui vengono percepiti, goduti (e pagati) solo gli aspetti deteriori o ludici.

In questo scenario catastrofico è chiaro che il conseguimento dell’ECDL può assumere un certo valore, ma esso non va sopravvalutato. A conti fatti, studiando il Syllabus si imparano alcune nozioni inerenti l’architettura hardware e software di un PC (utili quantomeno per capire se una macchina è una Panda o una Porsche), ci si muove con discreta confidenza tra le principali applicazioni del

pacchetto Office di Microsoft e si viene sommariamente introdotti a Internet utilizzando il browser e il client di posta elettronica di casa Microsoft.

In pratica si mette in condizione l'umanista di poter fare qualcosa: così istruito il titolare di una laurea in una disciplina umanistica è almeno pronto per svolgere attività di segretariato. E poiché troppo spesso l'uomo è quello che sa fare, si incanalano molti cervelli "pensanti" su carriere di ripiego, allontanandoli dalla possibilità di poter provare a dare un reale contributo alla crescita di un'azienda.

Certo, questo scenario è esagerato, anzi volutamente esagerato. Tuttavia è necessario per fissare e far risaltare alcuni punti chiave.

- Gli umanisti sono penalizzati nel mondo del lavoro in favore dei "tecnici".
- Un problema oggettivo per il loro inserimento in azienda è la limitata conoscenza informatica.
- L'ECDL da solo non è sufficiente a risolvere questo problema e rischia di limitare le reali capacità di chi ha investito almeno quattro o cinque anni della sua formazione nella studio delle "scienze dell'uomo", per costruirsi un modello critico di interpretazione della realtà, vale a dire la capacità di porsi in maniera oggettiva e creativa di fronte a un problema.
- Quello che si fa fatica a definire è una modalità di insegnamento dell'informatica adatta alle peculiarità degli umanisti. Non insipida ma stimolante. Che vada oltre la singola nozione tecnica. Che ricordi come la separazione tra scienze umane e tecnologia non è poi così marcata. Che spieghi come protocolli e programmi siano basati su linguaggi che, proprio perché tali, hanno una specifica sintassi, adatta però alle possibilità un computer e di conseguenza limitata, se paragonata alle capacità del cervello umano, che per comunicare usa linguaggi assai più complicati e che è in grado di scrivere opere di una complessità impensabile per un computer, opere come la *Divina Commedia* o la *Critica della ragion pura*, opere che gli umanisti studiano e imparano a comprendere nel corso della loro formazione…

Lo scopo di questo libro

Senza negare quanto appena affermato, bisogna ora smorzare un po' i toni.

Quello che voglio passi senza ambiguità è il messaggio che informatica e scienze umanistiche hanno molto in comune. Questo però non significa che chi "conosce di Filosofia, Lettere o Storia" possa di punto in bianco entrare nei processi di sviluppo e di gestione dei sistemi informatici e informativi, sostituendosi a tecnici e sviluppatori.

Vuol dire invece che un umanista ha tutte le capacità per comprendere ed entrare nel merito dei processi di sviluppo e comunicazione che regolano tali sistemi, il che si può tradurre nella capacità di gestire con abilità informazioni di vario genere.

L'informazione non è mai univoca e può essere trattata in modi differenti, a seconda delle necessità. Un cuoco può aver bisogno di una ricetta in particolare o di tutte le ricette di torta di mele esistenti. Lo studente può aver bisogno della bibliografia di un autore o della critica a una certa opera. L'editore deve sapere quanti libri produce in un anno, ma anche i costi del singolo libro e i dati di vendita. L'analista finanziario deve conoscere l'andamento della borsa di New York, ma

anche quello dei titoli del suo portafoglio. E così via.

Informazioni differenti ma tra loro legate, spesso semanticamente affini e quindi relazionabili una con l'altra. Informazioni che possono stare in un singolo computer o essere distribuite in Rete, da cui vanno recuperate e presentate quando necessario. Informazioni che vanno organizzate e gestite, ormai esclusivamente in forma digitale, ma sempre informazioni.

E l'umanista è per la sua formazione la persona meglio preposta a guardare criticamente le informazioni, a metterle in relazione, a ricavare informazione da informazione. Lo scoglio da superare sta tutto nel conoscere gli strumenti e le logiche attraverso cui le informazioni, o dati, vengono gestite con l'informatica. Si tratta di imparare a comprendere un certo tipo di linguaggio, proprio delle macchine, o almeno di certe macchine.

Questo libro si concentra proprio sulle modalità di gestione e codifica delle informazioni nel mondo digitale, con particolare attenzione alle tecnologie che ruotano, a diverso titolo, intorno alla "galassia Internet".

Lo scopo è quello di spiegare alcuni concetti alla base dell'informatica, con un linguaggio un po' meno tecnico ma forse per questo più adatto a chi non ha una formazione specifica; spiegare non tanto come far accadere una certa cosa (o meglio non solo) ma come questa accade, in modo che poi sia possibile operare delle scelte, pianificare dei processi e perché no, costruire o collaborare praticamente alla costruzione di un sistema per la gestione e la comunicazione di informazioni.

Per fare questo è necessario partire da un visione alta, trascurando alcuni aspetti in favore di altri, su cui invece bisogna scendere a "punta di spillo", in modo che alla logica e alla teoria si affianchi la prassi. Senza la pretesa di costruire nulla di speciale, né di essere esaurienti, ma con l'augurio di comprendere come le cose speciali sono costruite.

Non si tratta di Informatica Umanistica

Questo volume non nasce e non vuole porsi nel solco della cosiddetta Informatica Umanistica. Quest'ultima è una disciplina universitaria ormai definita (o in via di definizione).

L'Umanista Informatico è la mia personale risposta alle problematiche sopra ricordate, frutto della mia esperienza sul campo. Questo non vuol dire che quanto scritto qui sia in antitesi con quello che si può imparare frequentando un corso di Informatica Umanistica, anzi, per certi aspetti ne è complementare e per altri in sovrapposizione. Di certo il punto di osservazione è differente.

Questo libro non ha quindi la pretesa di sostituirsi a nessun manuale specifico. Non ne ha, e dato il taglio "tascabile" non può averne, la profondità. Semplicemente vuole suggerire un diverso punto di vista. Vuole raccontare e provare a spiegare, non tanto insegnare. Vuole essere di aiuto a chi non ha avuto modo di studiare all'università l'informatica, a chi non vuole rassegnarsi all'ECDL. Ai curiosi e a quelli che pensano di poter dire la loro.

Non si tratta di un manuale di informatica

In questo libro vengono introdotti diversi concetti tecnici e qua e là vengono presentate alcune operazioni o illustrate nozioni che permettono di lavorare operativamente su documenti di vario tipo.

Tuttavia questo non è e non vuole essere il classico manuale che insegna a svolgere un determinato tipo di operazioni con il computer.

Se siete alla ricerca di manuali specifici su alcune tecnologie questo libro non fa per voi.

Se invece volete cominciare a capire come funzionano e come lavorare con alcune diffuse tecnologie e linguaggi informatici, allora forse avete per le mani una risposta alle vostre domande.

Struttura del testo

Seguendo questo approccio i capitoli che seguono sono stati così organizzati.

- Capitolo 1. Spiega come le macchine gestiscono le informazioni. Per lavorare con profitto nel mondo dell'informatica è necessario imparare a guardare i file da una diversa angolazione, capendo come distinguere i tre livelli su cui si articola l'informazione nel mondo digitale: struttura, comportamento e presentazione.
- Capitolo 2. Racconta una tecnologia, XML, che permette di strutturare le informazioni seguendo criteri logici e semantici utili alla sua gestione in diversi ambiti e spiega in che modo e perché chi ha formazione umanistica può ottenere qui ottimi risultati.
- Capitolo 3. Parla di Internet, del Web e delle sue lingue, ma anche della sua architettura e dei suoi protocolli. Tutta l'esperienza della Rete si basa su alcuni concetti e strutture chiave che è facile padroneggiare una volta che se ne sono comprese logica e sintassi.
- Capitolo 4. Affronta l'organizzazione e il recupero delle informazioni nelle basi di dati, l'anima di ogni sistema informativo, piccolo o grande che sia.
- Capitolo 5. Inizia ai linguaggi di programmazione e alla programmazione informatica in senso stretto, cercando di evidenziarne le parti fondanti di cui viene introdotta la sintassi attraverso il linguaggio PHP.
- Appendice. Guida a un utilizzo più consapevole del motore di ricerca Google introducendo l'uso dei suoi operatori di ricerca e fornendo qualche suggerimento per trovare prima l'informazione ricercata.

Nota per il lettore

Questo libro non è stato pensato come un testo di consultazione rapida, ma per essere letto dall'inizio alla fine.

Il filo conduttore è mostrare come le macchine operano sulle informazioni e non tanto imparare a svolgere un determinato compito. Va da sé che leggendo i vari capitoli imparerete diverse nozioni e comincerete a utilizzare alcune tecnologie che potranno tornarvi utili in progetti di lavoro più o meno grandi.

Certo, niente impedisce di consultare solo il capitolo o la parte che più interessa, ma una lettura dall'inizio alla fine è consigliata e preferibile.

Se avete poca confidenza con il computer, soprattutto nelle prime pagine vi troverete un po' spaesati, storditi da parole e concetti. Non spaventatevi e abbiate il coraggio di proseguire: se ho lavorato bene durante la scrittura, alla fine di ogni capitolo vi rimarrà impresso quello che serve e, forti di queste conoscenze, potrete poi ritornare ad approfondire il resto, in base a curiosità o, come più spesso capita nel mondo del lavoro, a necessità.

Un'ultima raccomandazione. L'informatica parla una lingua i cui termini, spesso oscuri acronimi, non sono sempre di facile comprensione. Ho cercato di spiegarli velocemente via via che entravano in scena, tra parentesi, in incisi o in note. Questa scelta rende forse il testo meno scorrevole, ma più compatto. In linea di massima tutte le informazioni necessarie a comprendere l'argomento trattato

sono così immediatamente consultabili, senza dover ricorrere a glossari o altri apparati paratestuali.

Spero che questo vi aiuti a muovere i primi passi in questo mondo apparentemente difficile ma affascinante e sempre più necessario.

Risorse online

Come già ricordato, questo libro non può e non vuole sostituirsi a nessun manuale tecnico. Per questo se volete approfondire lo studio delle tecnologie qui presentate dovete necessariamente consultare altre fonti.

Molte sono facilmente rintracciabili in Rete, attraverso Google o un altro motore di ricerca. Guide base e avanzate sono gratuitamente disponibili su molti siti.

Chi invece preferisce la carta stampata, troverà in ogni capitolo riferimenti a testi specifici, quasi tutti editi da Apogeo nella collana Pocket (la stessa di questo libro), quindi accessibili sia come formato, sia come prezzo e adatti alle esigenze di chi si avvicina alla tecnologia. Inoltre tutti i titoli della collana Pocket (come del resto tutta la produzione editoriale di Apogeo) sono presenti nei cataloghi di Google Ricerca Libri (http://books.google.it/) che ne offre anteprime gratuite, spesso più che sufficienti ad appagare i più comuni dubbi o curiosità.

Convenzioni utilizzate

Il flusso del testo è spesso spezzato alcuni box di approfondimento connotati in maniera particolare. La lettura non è obbligatoria ai fini della comprensione, ma vivamente consigliata. Al lettore la scelta finale in base a interessi e necessità.

NOTA *Contiene informazioni interessanti, a volte tecniche, relative all'argomento trattato. Talvolta riporta approfondimenti e curiosità.*

TERMINOLOGIA *Spiega termini nuovi o poco familiari.*

RIFERIMENTI *Suggerisce indicazioni circa altri testi o siti Internet in cui approfondire gli argomenti trattati.*

SUGGERIMENTO *Fornisce spunti e consigli per risparmiare tempo ed evitare confusioni, come per esempio il modo più semplice per eseguire una determinata operazione.*

ATTENZIONE *Evidenzia indicazioni da non trascurare per evitare le difficoltà in cui gli utenti, soprattutto alle prime armi, possono imbattersi.*

L'autore

Fabio Brivio laurea in Storia Medievale e master in Informatica e Comunicazione, è in Apogeo dal 2004. Editor per la manualistica informatica, cura la produzione e pubblicazione degli ebook e si interessa alla ricerca e allo sviluppo di soluzioni per l'editoria digitale. Ha coordinato lo sviluppo della intranet utilizzata dagli editori del gruppo Feltrinelli. Per Apogeo ha firmato *ePub per autori, redattori, grafici*, *Trovare su Internet*, *Internet per tutti* e l'ebook *Il mestiere dell'editor*. Quando non ha a che fare con un libro o con qualche frammento di codice che proprio non ne vuole sapere di funzionare, si ricorda di avere un blog, fabiob.eu, oppure lancia sassi nello stagno su Twitter @f4b10b.

Capitolo 1

L’informazione tra forma e formattazione

I computer sono stati inventati per svolgere delle operazioni di calcolo connesse a vari tipi di informazione. Questo è tanto importante, quanto ormai trasparente, al punto che è facile non farci caso.

Le informazioni che vengono gestite da una macchina sono di varia natura, testuali e grafiche, audio e video. Spesso un’informazione è composta: si tratta dei cosiddetti *contenuti multimediali*, dove audio e video si fondono magari arricchiti da contenuti testuali.

TERMINOLOGIA *In questo libro il termine* macchina *è usato come sinonimo di* computer.

Nei computer, a un livello basso di elaborazione, le informazioni vengono trattate come flussi di dati, sequenze di bit su cui sono svolte tutte le operazioni necessarie: apertura, lettura, copia, incolla, cancella, modifica, ripristina, cerca, sostituisci e così via.

Per poter svolgere la giusta operazione sul giusto flusso di dati, elaborando l’informazione nella maniera voluta dall’utente, una macchina deve poter scegliere senza incertezza. Le informazioni immesse in un computer vengono per questo archiviate in *file*, memorizzati, o forse sarebbe più corretto dire allocati, in una *cartella* (*directory* o *folder*) del *filesystem*.

Un filesystem è uno spazio di memoria (normalmente una parte, se non tutto il disco fisso di un computer) organizzato e predisposto alla conservazione dei dati presenti nei file. Nelle interfacce grafiche dei sistemi operativi esso viene appunto rappresentato attraverso la metafora della cartella.

NOTA *I più attenti, sicuramente chi non è digiuno di filosofia, lo avranno forse già notato durante l’utilizzo di un PC. Vale comunque la pena ricordarlo: la struttura che siamo soliti vedere rappresentata quando ci spostiamo tra le cartelle di un computer non ha niente di diverso da tante tassonomie con cui gli umanisti si sono dovuti confrontare nel corso dei loro studi (Figura 1.1). Le categorie care ad Aristotele sono tutt’altro che superate e trovano in informatica ampi spazi di utilizzo. Questo concetto, apparentemente banale, non è da trascurare: quando si pensa ai “talenti” che un umanista possiede, spesso non si considera la confidenza con le strutture di organizzazione della conoscenza che egli è portato a sviluppare quasi spontaneamente durante i suoi studi.*

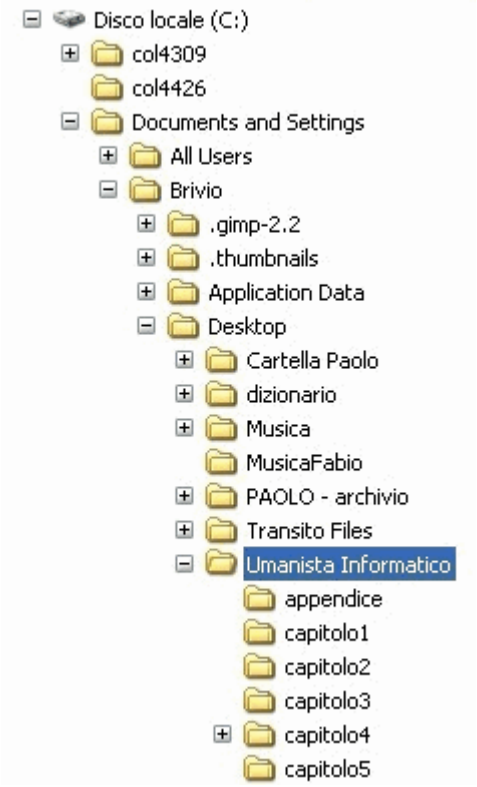


Figura 1.1 Tassonomia di un file system.

TERMINOLOGIA *Tutti i filesystem hanno una cartella "madre" che in gergo viene detta* root *(radice) e che spesso viene indicata con una barra inclinata o slash (/).*

Quindi, ricapitolando, *Cartella/Sottocartella/File*: grazie a questa struttura, a questo modello organizzativo, una macchina riesce a rintracciare i dati senza ambiguità, o meglio riesce a operare sui file in maniera corretta.

Tuttavia trovare il file giusto è solo una parte del lavoro; il resto è leggerlo, rappresentarlo, cioè mostrarlo all'utente che lo ha richiesto, ed eventualmente svolgere delle operazioni (per esempio modificarlo).

Entra qui in scena il concetto (importante) della codifica delle informazioni nei file.

Questione di codifica

"A tipo di informazione, tipo di file": su questa regola si basa gran parte del lavoro di un computer. Ma andiamo con ordine.

A un livello molto semplice, molto basso o vicino alla macchina, il computer non fa altro che eseguire delle operazioni di calcolo sui dati, sui file, da esso gestiti.

Parlare di calcolo quando i file non sono necessariamente numerici, cioè non contengono numeri, potrebbe sembrare un po' strano. In che senso vengono svolte delle operazioni di calcolo su un documento di testo?

Un breve esempio può essere d'aiuto. Utilizzando un comune editor di testo, come Microsoft Word, e volendo modificare l'allineamento di alcuni paragrafi, l'utente non deve far altro che selezionarli e quindi fare clic sull'apposito pulsante (Figura 1.2).

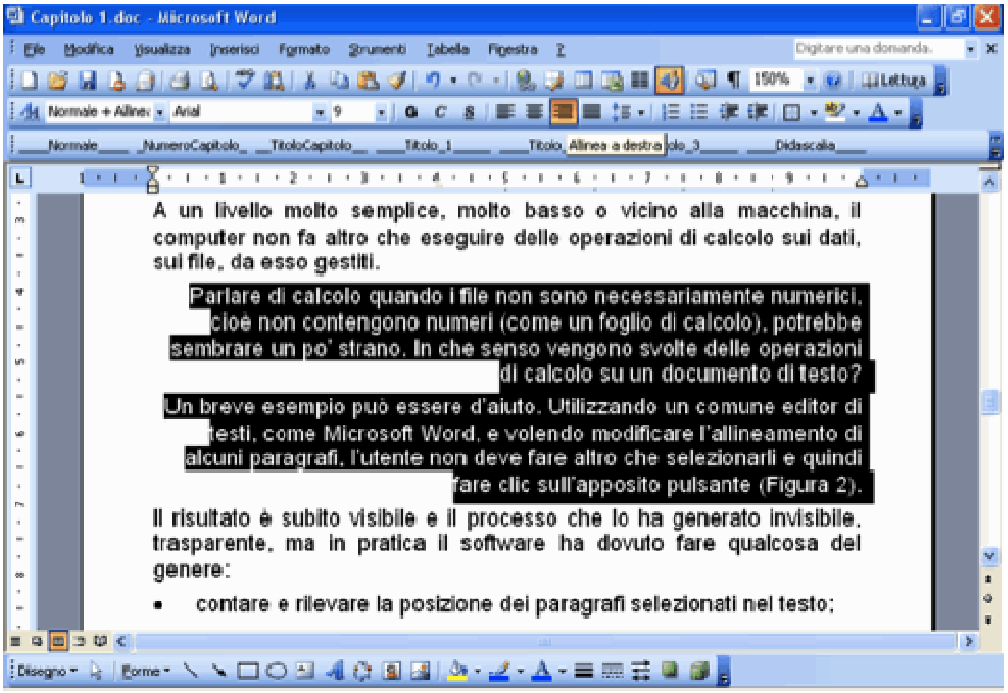


Figura 1.2 Anche per semplici operazioni di editing di un testo, il computer svolge diverse operazioni di calcolo.

Il risultato è subito visibile, mentre il processo che lo ha generato è invisibile, trasparente, ma in pratica il software ha dovuto fare qualcosa del genere:

- contare e rilevare la posizione dei paragrafi selezionati nel testo;
- individuare, recuperare e contarne i contenuti, vale a dire i caratteri e gli spazi nella giusta sequenza;
- leggere la dimensione orizzontale del documento, l'asse X, e ridisporre, ricollocare, nella sequenza rilevata tutti i caratteri a partire, per esempio, dal margine destro della pagina.

Semplificando: conteggio del testo di partenza, archiviazione di queste informazioni, recupero e definizione dei parametri necessari a calcolare e quindi ricollocazione del testo nella maniera voluta, "stampa" del testo a video.

Tutto questo è possibile poiché il testo, o il contenuto che si sceglie di manipolare, è codificato con un *metodo* che ne definisce tutte le specifiche, le qualità. Il metodo si occupa di connotare un dato o un insieme di dati in un certo modo prima di memorizzarlo in un file, quindi un programma dotato della *grammatica*, cioè delle regole proprie del metodo in uso, si occupa di leggerlo, rappresentarlo all'utente e soprattutto di compiere su di esso le operazioni necessarie.

Le operazioni di calcolo sono così possibili a partire dalla codifica in uso, dal metodo. Il software è in grado di interpretarla e da lì partire con le operazioni necessarie.

Questo avviene sempre, per ogni file, per ogni software e per ogni operazione applicata. Anche all'apertura di un file, il programma non fa altro che richiamare tutta la grammatica del metodo in uso, per riproporre all'utente una rappresentazione a video del file in oggetto.

NOTA *Un programma può lavorare su uno o su più metodi: dipende dalla sua complessità e flessibilità.*

Metodi ed estensioni

A questo punto si pone però un problema: come riesce un programma a determinare in quale

modo è stata codificata l'informazione presente in un file? Cosa impedisce a un software di usare un metodo al posto di un altro? Perché un editor di testo come Word lavora correttamente con una certa tipologia di file ma si blocca nel tentativo di aprirne altri?

Come sopra ricordato, un software lavora normalmente con uno o più metodi. Normalmente si tratta di metodi analoghi, cioè adatti a gestire tipologie di dati simili, per esempio testi.

Tuttavia, "metodo analogo" non significa "metodo uguale", quindi, a meno che una certa imprecisione non sia tollerabile (provare ad aprire un file con un metodo solo simile alla codifica applicata può essere causa di perdita di informazioni), è sempre necessario utilizzare solo ed esclusivamente il metodo con cui un file, un documento, è codificato.

Per evitare simili problemi le macchine fanno riferimento all'estensione dei file.

NOTA *Questo è particolarmente vero per i sistemi operativi Windows. I sistemi operativi Mac OS X o Unix, utilizzano invece altre modalità per associare un file a un programma. Queste modalità non vengono qui discusse per motivi di sintesi e perché non necessarie alla comprensione del testo.*

L'*estensione* di un file corrisponde alle lettere che seguono il punto (.) nel nome di un file. Grazie a essa un computer, o meglio il sistema operativo installato su un computer, è in grado di stabilire in quale modo è stato codificato un file e di conseguenza avviare il programma adatto ad aprirlo. Normalmente questo avviene facendo clic sull'icona che rappresenta il file di interesse; inoltre, se su una macchina è presente un software in grado di aprire un dato file, a esso viene di norma associata un'apposita icona.

TERMINOLOGIA *Il* sistema operativo *è il programma che gestisce e controlla tutte le operazioni di base che avvengono in una macchina. I principali sistemi operativi oggi in uso sono Windows di Microsoft e Mac OS X di Apple, a cui si affiancano diverse distribuzioni del sistema GNU/Linux (Ubuntu, Fedora, Knoppix e così via).*

La Figura 1.3 esemplifica bene il comportamento di un sistema operativo Windows di fronte ai file le cui estensioni sono riconosciute: se il programma corrispondente è presente, il file viene connotato da un'apposita icona. Nel caso però del file `vettore.svg`, questa eventualità non è soddisfatta e il computer utilizza quindi un'icona neutra per indicare la non associazione del file a un programma specifico (per aprire questo file sarà quindi prima necessario installare sulla macchina il relativo software).

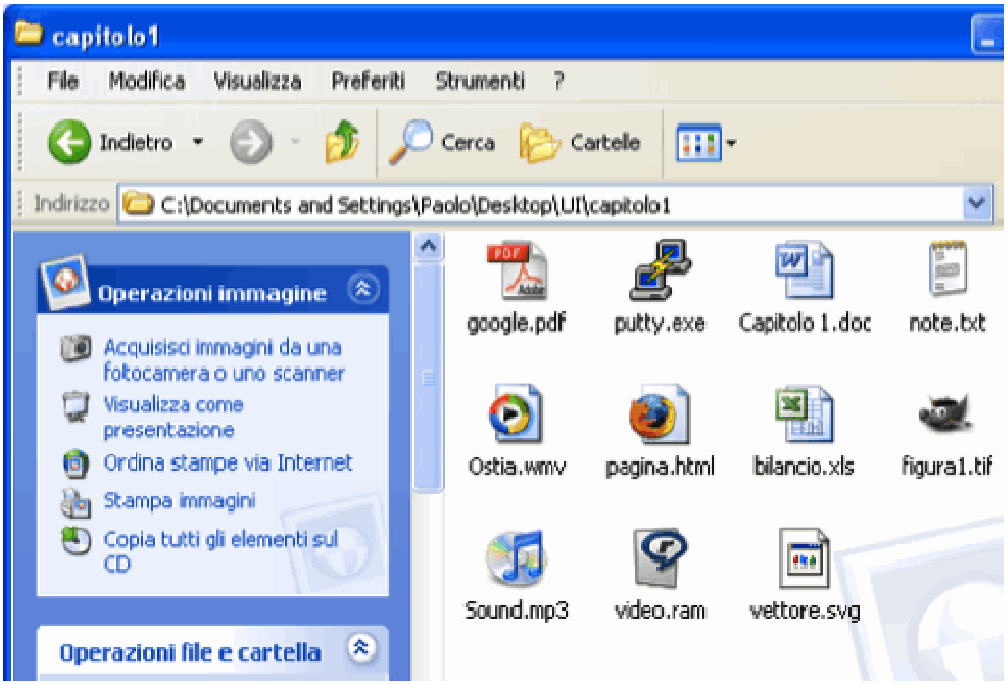


Figura 1.3 Un computer è in grado di associare a ogni estensione un programma specifico, se presente nella sua dotazione.

Provando ad aprire un file con un programma non specifico, cioè non in grado di riconoscere e interpretare il metodo con cui i dati sono stati codificati, il risultato può essere disastroso. La Figura 1.4 mostra proprio questa eventualità: un'immagine (in formato GIF) viene prima aperta con un programma (Gimp) in grado di leggerla e interpretarla correttamente, quindi con Blocco Note (un software per la gestione di semplici contenuti testuali). Nel secondo caso il programma si dimostra incapace di fornire una corretta rappresentazione del contenuto. Il risultato è un output testuale non utilizzabile e interpretabile, se non per i primi tre caratteri ("GIF", su sfondo grigio nella figura), dopodiché il software ha mostrato la sua impossibilità di proseguire nella lettura dei dati codificati con un metodo diverso da quello a lui noto.

Figura 1.4 A ogni programma, il giusto file e… il giusto metodo.

Non subire l'estensione

A questo punto dovrebbe apparire chiara l'importanza dell'estensione, troppo spesso sottovalutata.

Nell'utilizzo quotidiano dei computer, molti problemi di gestione dei file derivano dal fatto che non si presta abbastanza attenzione all'estensione, precludendosi così alcune possibilità: per esempio, un file con una data estensione potrebbe presentare problemi quando aperto con un software ma non con un altro. Un caso comune è quello dei documenti di testo: può capitare di avere difficoltà con Word della suite Office di Microsoft e di risolvere tutto usando Writer, il software per lavorare sui testi della suite OpenOffice, ma anche viceversa.

NOTA *Vale la pena ricordare che, salvo rarissimi casi, per ogni estensione esistono più software adatti. Esemplare è il caso appena ricordato di Office e OpenOffice: entrambi sono suite per l'ufficio complete e potenti in grado di lavorare con file di testo, fogli di calcolo e presentazioni. La compatibilità tra le due suite non è del 100 percento, ma è comunque molto alta: OpenOffice può gestire e anche creare file nei principali formati della suite Office (*`.doc, .xls, .ppt`*).*

Imparare a orientarsi tra le estensioni è quindi importante e la prima cosa è assicurarsi che esse siano sempre visibili. Spesso capita infatti che i sistemi operativi Windows operino di default in una modalità tale da impedire la visualizzazione delle estensioni dei file conosciuti.

ATTENZIONE *Questa modalità, oltre a ostacolare la comprensione del tipo di file, è pericolosa in quanto molti virus contano su di essa per camuffarsi agli occhi degli utenti. È il caso degli allegati che normalmente arrivano per posta elettronica: un file allegato potrebbe avere una doppia estensione: una fittizia (per esempio* `.doc`*) seguita da quella reale (per esempio* `.exe`*>, cioè quella dei file eseguibili). Salvato il file sul PC, il sistema operativo nasconde l'estensione reale e mostra quella fittizia. Se l'utente prova ad aprirlo, il virus si attiva e può cominciare a fare danni.*

SUGGERIMENTO *Per ovviare a questo inconveniente è sufficiente aprire il menu* Strumenti *di una qualsiasi cartella e selezionare* Opzioni cartella*, quindi attivare la scheda* Visualizzazione *e spuntare la casella* Nascondi le

estensioni per i tipi di file conosciuti. *Infine premere* OK *(Figura 1.5).*

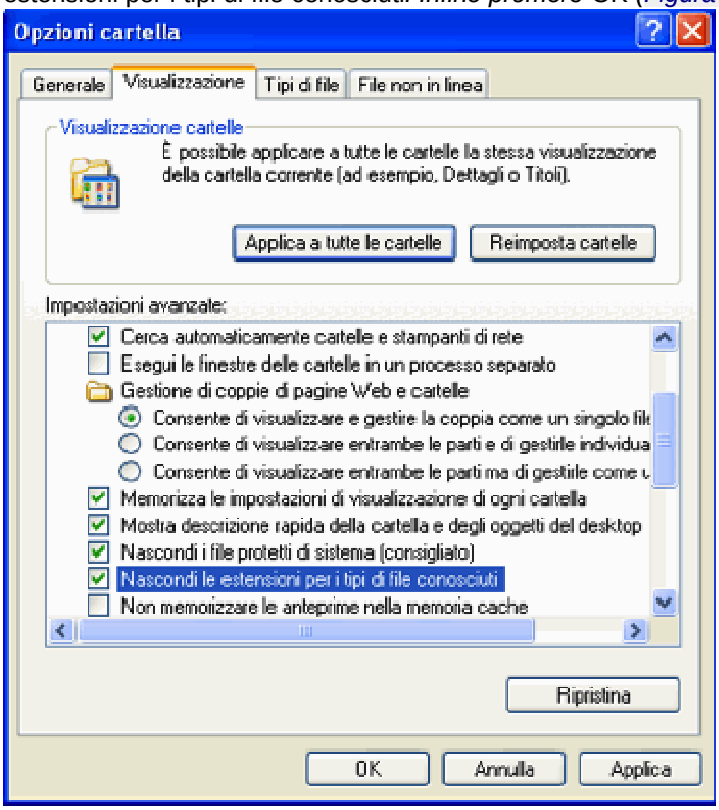


Figura 1.5 Nei sistemi Windows, la casella Nascondi le estensioni per i tipi di file conosciuti non deve essere selezionata.

Inoltre è anche possibile istruire un sistema operativo in modo che davanti a una certa estensione faccia sempre uso di un dato programma.

SUGGERIMENTO *L'operazione è semplice: basta selezionare il file con il tasto destro del mouse, quindi scegliere la voce* Proprietà *e sulla scheda* Generale *fare clic sul pulsante* Cambia*. Nella finestra* Apri con *si può quindi scegliere un programma dall'elenco, oppure ricercarlo sul computer tramite il pulsante* Sfoglia*. Terminata la scelta bisogna premere* OK *(Figura 1.6).*

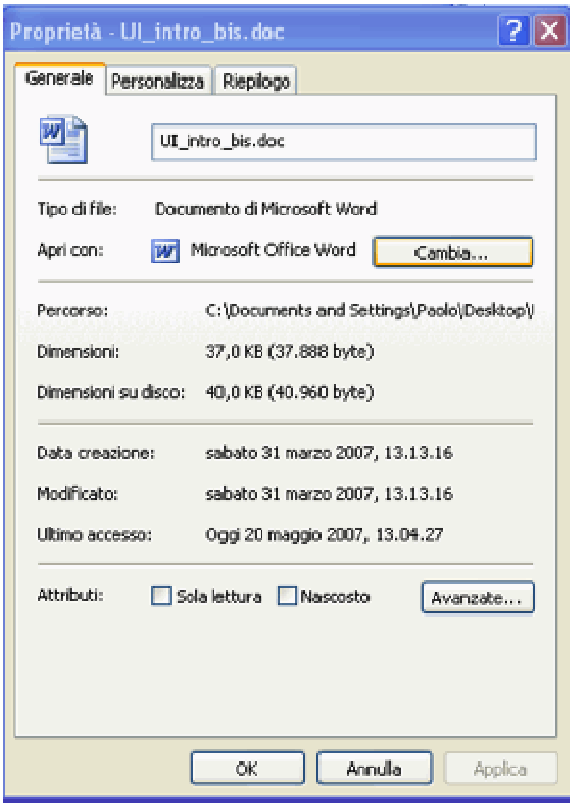


Figura 1.6 Dalla finestra Proprietà è possibile specificare con quale programma aprire un file.

Infine vale la pena ricordare che l'estensione può anche essere modificata manualmente. Tale operazione non è mai consigliabile: se l'estensione è applicata correttamente e il sistema operativo è istruito su quale software usare per trattare i file da essa connotati, non esiste un motivo valido per alterarla.

Tuttavia può capitare che un file arrivi (per esempio come allegato via posta elettronica) su un computer senza estensione. Questa eventualità è frequente quando si lavora su un sistema Windows e si riceve un file precedentemente generato su un sistema operativo diverso, per esempio Mac OS X, dove le modalità di gestione dei file sono differenti. Si tratta di uno dei piccoli inconvenienti in cui è facile imbattersi, legati alla ben più ampia problematica inerente la compatibilità tra sistemi.

In questi casi il problema è facilmente risolvibile, conoscendo il tipo di file ricevuto: basta rinominare il file avendo cura di digitare dopo il nome il punto e l'estensione corretta (Figure 1.7 e 1.8).

ATTENZIONE *Questa operazione in sé molto semplice (basta selezionare il file con il tasto destro del mouse e quindi scegliere la voce* Rinomina*) non va però fatta con leggerezza ma solo quando si conosce e si è sicuri della fonte, cioè l'utente, da cui si riceve il file. Un messaggio di posta elettronica ricevuto da fonte ignota e che invita a rinominare con una certa estensione un file allegato deve sempre essere guardato con sospetto.*

Figura 1.7 Prima di modificare l'estensione di un file Windows ricorda che il file potrebbe non essere poi utilizzabile.

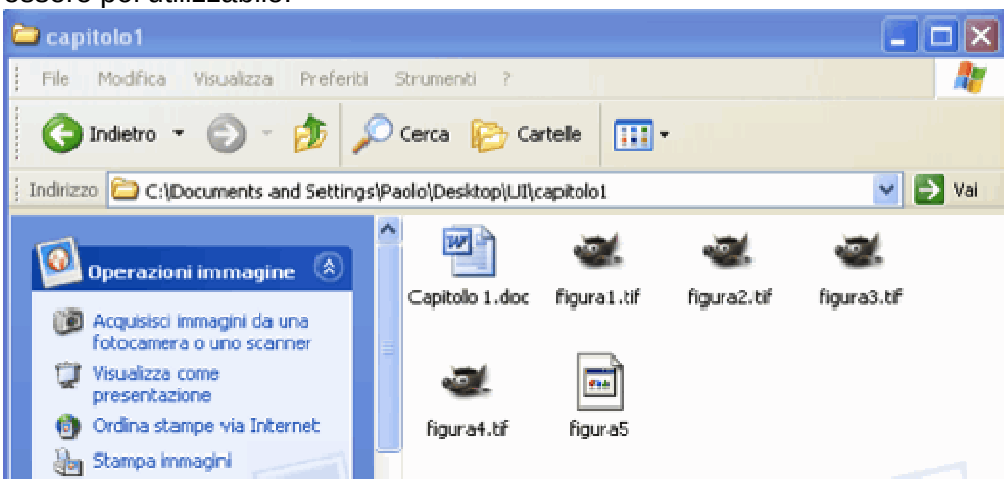


Figura 1.8 Davanti a un file senza estensione (figura5), Windows non sa come comportarsi: al file viene così assegnata un'icona neutra.

Estensioni comuni

Prima di concludere questa parte è utile elencare alcune estensioni, comuni e non, indicando a quale tipologia di file fanno riferimento. Ritorneremo su alcune di queste estensioni, o meglio sulle caratteristiche di alcuni file con tali estensioni, nei prossimi capitoli.

Estensione	File
`.txt`	File di testo non formattato.
`.doc`	File di testo formattato da Microsoft Word.
`.rtf`	File di testo formattato in grado di garantire un'ottima compatibilità tra programmi di videoscrittura diversi (per esempio Microsoft Word e OpenOffice Writer).
`.pdf`	File di testo e immagini in grado di garantire la piena compatibilità, in lettura e in stampa, tra sistemi operativi diversi.
`.xls`	File contenente fogli di calcolo formattato da Microsoft Excel.
`.ppt` `.pps`	File contenenti presentazioni in forma di slide formattato da Microsoft PowerPoint.
`.xml`	File di dati strutturati, privi di qualsiasi caratteristica stilistica, ideali per la distribuzione degli stessi tra piattaforme (sistemi operativi, software, applicazioni) diverse.

.html .xhtml	File di dati strutturati in modo tale da essere distribuiti sul Web ed essere facilmente interpretati e quindi rappresentati dai browser (Internet Explorer, Firefox, Safari, Opera e così via).
.css	File contenente le istruzioni di presentazione di una pagina web.
.php .asp	File contenenti istruzioni scritte in un linguaggio di programmazione utilizzati per creare pagine web dai contenuti dinamici.
.gif .jpg .tif	File contenenti oggetti grafici (immagini e fotografie).

Formati e formattazione: che cosa i software aggiungono alle informazioni?

Nella tabella precedente si è spesso fatto riferimento ai file e alle informazioni in essi contenute con i termini “formattato” e “strutturato”, anche in relazione alla loro “presentazione”. È il momento di approfondire la questione.

Come ricordato, i software che normalmente vengono usati per lavorare sui contenuti (siano essi testuali, grafici, audio o video) utilizzano un metodo, o meglio codificano i dati con un metodo grazie al quale è possibile rappresentarne e modificarne le qualità e le proprietà, sia a livello macro, sia a livello atomico.

In pratica i programmi danno forma all’informazione, la *formattano*.

Chiariamo questo concetto fondamentale.

Quando si usa Word per scrivere un testo, per prima cosa è necessario definire la dimensione della pagina (di default un A4) e i relativi margini. Queste proprietà si applicano a tutto il file in lavorazione, a tutto il documento, a un livello macro.

Nella scrittura si passa poi alla definizione dei titoli, di primo, secondo, terzo livello e così via, specificandone font e relativa grandezza. Analogo lavoro viene fatto per il testo, normalmente organizzato in paragrafi, all’interno dei quali può capitare di dare maggior enfasi ad alcuni termini, normalmente utilizzando corsivo, grassetto o sottolineato. Tutte queste proprietà si applicano al documento in lavorazione a un livello più o meno atomico, più o meno circoscritto: solo un dato termine, solo i titoli di un certo livello e così via.

L’utente è quindi in grado di organizzare con semplicità il documento, mentre il programma si occupa di codificare le scelte relative alla “forma” rendendole disponibili ogni volta che sia necessario.

La stessa cosa avviene per tutte le tipologie di documenti. Per esempio un file di Excel, un foglio di calcolo, fa dell’organizzazione dei dati in tabelle, righe, colonne e celle la sua forza. In pratica permette di formattare le informazioni in una maniera, tabellare appunto, che ben si adatta al loro controllo per fini di calcolo. Inoltre a ogni riga, colonna e cella l’utente può attribuire altre qualità in termini di font, colore, sfondo e così via.

Discorso analogo potrebbe essere fatto per PowerPoint, dove la formattazione delle informazioni avviene sulla base delle slide, e per qualsiasi altro *editor*.

TERMINOLOGIA *Il termine* editor *definisce i software che permettono di lavorare sui dati, qualsiasi tipo di dati, modificandoli, integrandoli o cancellandoli. Editor sono quindi Word, Excel, PowerPoint, ma anche qualsiasi programma in grado di gestire e operare sulle informazioni. Oltre agli editor per i testi, esistono quindi editor*

specifici per la computer grafica, adatti per lavorare su fotografie e immagini, come Photoshop e Gimp, o per lavorare sui video, per esempio Premiere o Final Cut, o sull'audio, come Cubase. Pur nelle differenze di utilizzo e di interfaccia, tutti sono accomunati dal fatto di permettere la definizione precisa delle qualità delle informazioni trattate: un paragrafo o una parola, il contenuto di una cella o di una slide, una sezione di pochi pixel di un'immagine digitale, pochi fotogrammi di un filmato, alcune note di un brano audio. Nella terminologia informatica, editor si contrappone a player*, o* reader*, che indica invece i programmi che non permettono la modifica di un contenuto ma solo la sua rappresentazione. Player sono per esempio Adobe Reader per i PDF, oppure Windows Media Player e iTunes per musica e video. Va da sé che un editor svolge di solito anche le funzioni di reader (si pensi a Microsoft Word).*

Ricapitoliamo quanto fin qui detto. I concetti su cui fissare l'attenzione sono due: il primo è che a un dato file (connotato da una specifica estensione) corrispondono determinate possibilità di formattazione delle informazioni (per esempio il font in un documento di testo, la dimensione di un'immagine, il timbro di un'esecuzione musicale e così via); il secondo è che buona parte del lavoro di un editor consiste nel permettere all'utente di aggiungere all'informazione altra informazione, o più precisamente ***metainformazione,*** vale a dire informazione sull'informazione, inerente cioè la struttura, il comportamento e la presentazione dei dati.

La formattazione di un file, o meglio di un contenuto, si articola infatti su questi tre livelli e alla loro comprensione è dedicata la parte finale di questo capitolo.

TERMINOLOGIA *La formattazione è quindi il carattere distintivo di un file. Per questo motivo è abbastanza frequente riferirsi a un tipo di file con il termine "formato" seguito dall'estensione o dal nome del software utilizzato per gestirlo. Espressioni come "formato doc", "formato gif" o "formato excel" sono quindi del tutto giustificate.*

Struttura, comportamento e presentazione dell'informazione

Prima di proseguire bisogna sottolineare che i tre i livelli della formattazione dell'informazione contenuta in un file non necessariamente sono sempre presenti. Per esempio, un dato può essere trattato solo per quanto riguarda la sua struttura e la sua presentazione, oppure solo per la sua struttura e il suo comportamento. Come vedremo meglio più avanti, un dato difficilmente può però essere trattato solo per quanto riguarda la sua presentazione, cioè come privo di informazioni di struttura: struttura e presentazione sono un binomio fondamentale nella gestione delle informazioni, e la seconda acquista senso solo in relazione alla prima.

In tutto questo, il comportamento è una sorta di "jolly" che si colloca idealmente a un livello intermedio, tra struttura e presentazione. In alcuni file, il dato può avere dei comportamenti che come risultato hanno quello di fornire un nuovo dato, una nuova informazione, che viene opportunamente strutturata e se serve presentata.

NOTA *In questo libro si parla di struttura, comportamento e presentazione dell'informazione prevalentemente in relazione a contenuti testuali, quelli con cui più facilmente un umanista si trova ad avere a che fare.*

Struttura

La struttura di un file può essere definita come l'ordine logico attraverso il quale i dati che lo compongono vengono organizzati.

Questo livello è di fondamentale importanza, in quanto la giusta comprensione di un contenuto è possibile solo attraverso una buona organizzazione dello stesso.

Una pagina di un libro, per esempio, può essere vista come un flusso di testo. Per facilitarne la lettura il flusso viene suddiviso in parti. Ecco quindi che entrano in scena titoli di primo, secondo e terzo livello, paragrafi, citazioni, note, termini con enfasi particolare e così via.

La sfera di competenza della struttura di un contenuto ha a che vedere con la semantica delle parti che in questo modo possono essere messe in relazione. Un titolo di primo livello ha un peso semantico maggiore di un titolo di secondo livello. Un titolo di primo livello precede sempre un titolo di secondo livello che precede a sua volta un titolo di terzo.

Dopo un titolo si trovano uno o più paragrafi, che possono essere intervallati da elenchi, liste, oppure da citazioni, note, suggerimenti e così via.

Per finire, all'interno di un paragrafo alcuni termini possono differenziarsi ulteriormente: è il caso degli acronimi NdT o NdR (*Nota del Traduttore*, *Nota del Redattore*) ma gli esempi potrebbero essere molti.

NOTA *Quella appena ricordata è proprio la struttura del libro che avete tra le mani, struttura che è stata definita attraverso il programma Microsoft Word durante le fasi della scrittura.*

L'importanza della struttura di un contenuto si intuisce meglio quando questa viene a mancare. Per esempio, la Figura 1.9 mostra il testo di questo capitolo (scritto con Microsoft Word) visualizzato con l'editor Blocco Note, che come ricordato non applica alcuna formattazione ai dati trattati. Tutto il testo, cioè tutta l'informazione, è presente, ma in assenza di struttura dei contenuti, la metainformazione, la lettura diventa più difficile: titoli, paragrafi, note, ogni elemento viene interpretato e collocato sullo stesso piano e l'organizzazione semantica dei contenuti viene del tutto ignorata.

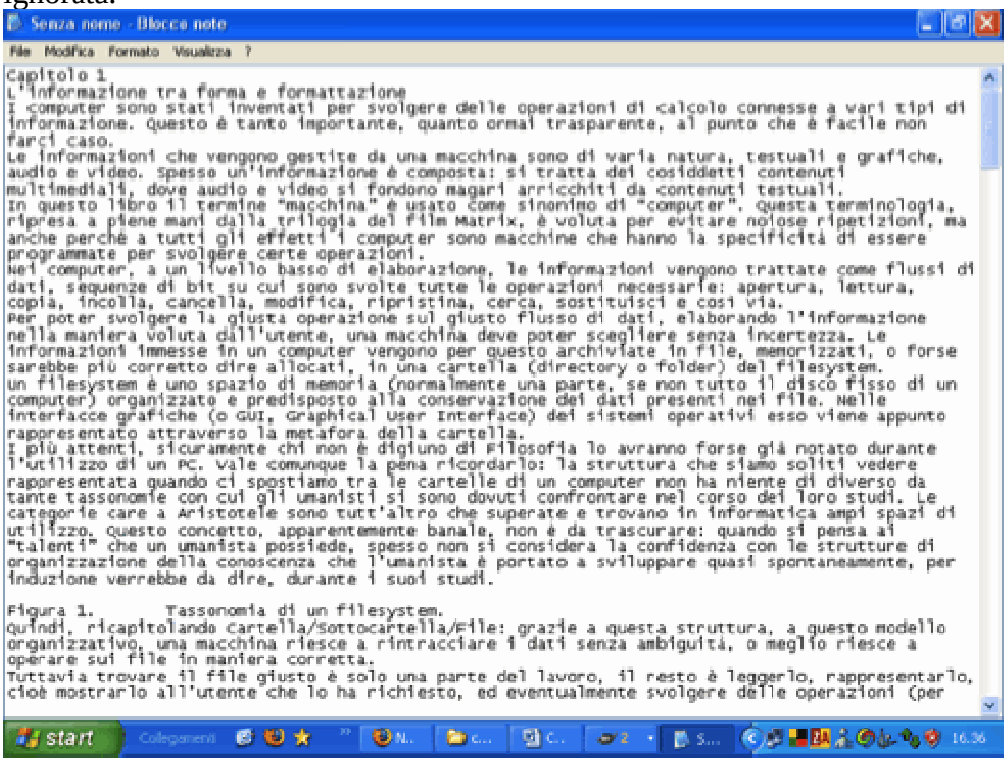


Figura 1.9 Testo non strutturato nell'editor Blocco Note.

La stessa cosa sarebbe successa anche partendo da un file di Excel o di PowerPoint: la struttura tabellare o in slide sarebbe del tutto andata persa.

NOTA *Ci si potrebbe chiedere a questo punto come il concetto di struttura sia applicabile anche a file grafici. In questo caso il contenuto è un'immagine, o una serie di immagini (nei video), strutturata su una griglia di pixel, l'elemento base di ogni immagine bitmap. Questa sequenza di pixel è quindi la struttura stessa dell'immagine, sui cui un editor di fotoritocco, come Photoshop o Gimp, può intervenire.*

Comportamento

Una volta strutturati, i contenuti, o meglio le informazioni, possono essere consultati e quindi utilizzati per creare nuova informazione.

Esempi di questo sono i fogli di calcolo o le pagine web “dinamiche”.

Per capire che cosa si intende per comportamento dell’informazione e che ruolo gioca in questo contesto, è prima però necessario fare un passo indietro.

L’informazione è sempre stata organizzata con lo scopo di fornire una base di conoscenza da cui partire per crearne nuova. La grossa novità introdotta dalle macchine è la possibilità di integrare nei documenti digitali alcune funzionalità che consentono di automatizzare la generazione di un certo tipo di informazione.

I fogli di calcolo sono esemplificativi di questo aspetto. Normalmente sono usati per gestire informazioni di tipo numerico in una struttura tabellare. Su celle, colonne e righe è però possibile programmare delle operazioni di calcolo, che possono fornire risultati in forma numerica o anche grafica. Somme, sottrazioni, moltiplicazioni, divisioni, calcoli semplici e complessi, gestioni di IVA, sconti e così via: tutto questo e molto altro può essere delegato alla macchina, che partendo da un dato o un insieme di dati (i valori contenuti nelle celle) è in grado di controllarne un certo comportamento, fornendo all’utente l’informazione attesa, strutturata come programmato (una nuova cella della tabella o una rappresentazione grafica del dato).

Quello che dobbiamo ancora sottolineare è che il comportamento dell’informazione è dipendente dalla struttura, senza la quale non sarebbe possibile programmare alcun tipo di elaborazione. Basta pensare alle informazioni presenti in una tabella di Excel: svincolate dall’ordinamento delle celle, senza struttura né ordine, effettuare delle operazioni su di loro sarebbe molto improbabile.

Un altro esempio di comportamento dell’informazione lo si trova in molte pagine web, che si possono dividere tra statiche e dinamiche.

SUGGERIMENTO *Sulle pagine web statiche e dinamiche torneremo nel Capitolo 3.*

Una pagina web ***statica***, con tutti i suoi contenuti, i suoi dati e le sue informazioni, esiste in quanto tale e non cambia, mai.

Una pagina web ***dinamica***, non esiste in assoluto, ma cambia in relazione a determinate scelte dell’utente. Il cambiamento può essere circoscritto ai contenuti, ma può anche comprendere la sfera della struttura e della presentazione: anche qui il comportamento dipende da come la pagina è stata programmata, cioè dal tipo di informazioni che deve mostrare e come.

Qualche esempio chiarirà meglio il concetto. Di norma, tutte le volte che ci viene chiesto di autenticarci a un sito, con ID e password, stiamo per accedere a pagine web dinamiche (Figura 1.10). È il caso dei servizi di webmail: le pagine con i messaggi di posta in arrivo, in uscita, eliminati e così via non esistono in assoluto ma vengono generate e quindi inviate solo all’utente il cui login ha avuto esito positivo, cioè che si è autenticato con successo. In definitiva le pagine con i messaggi di posta elettronica sono programmate in modo da comportarsi differentemente in base all’utente. I contenuti, i messaggi, cambiano in relazione all’utente: solo l’utente `brivio@apogeonline.com` può vedere i messaggi che sono stati inviati alla sua casella di posta elettronica (Figura 1.11).

Questo è solo un esempio, ma la casistica è veramente ampia e mai come oggi il Web è popolato

da pagine e da documenti dinamici, che cambiano e si comportano diversamente a seconda dei dati forniti dall'utente e delle operazioni compiute dall'utente sulla pagina visualizzata.

Quando Amazon (`http://www.amazon.com`) ricorda al lettore che chi "è interessato al libro X, è interessato anche al libro Y"; quando Google (`http://www.google.it`) serve una pagina dei risultati in base alle chiavi di ricerca digitate; quando ancora su Google, nella home page personalizzata, detta iGoogle (`http://www.google.it/ig`), vengono spostati o aggiunti i box delle informazioni con il trascinamento del mouse (Figura 1.12); quando si usano servizi web per pagare il bollo dell'auto, l'assicurazione, gestire un conto corrente; o ancora quando si usa un servizio OPAC (*Online Public Access Catalog*), in tutti questi casi si è in presenza di pagine dinamiche il cui comportamento è stato programmato per soddisfare le necessità dell'utente che a esse si rivolge. Pagine in grado di effettuare delle operazioni sulle informazioni che l'utente chiede e inserisce, per fornire nuova informazione, quella desiderata.

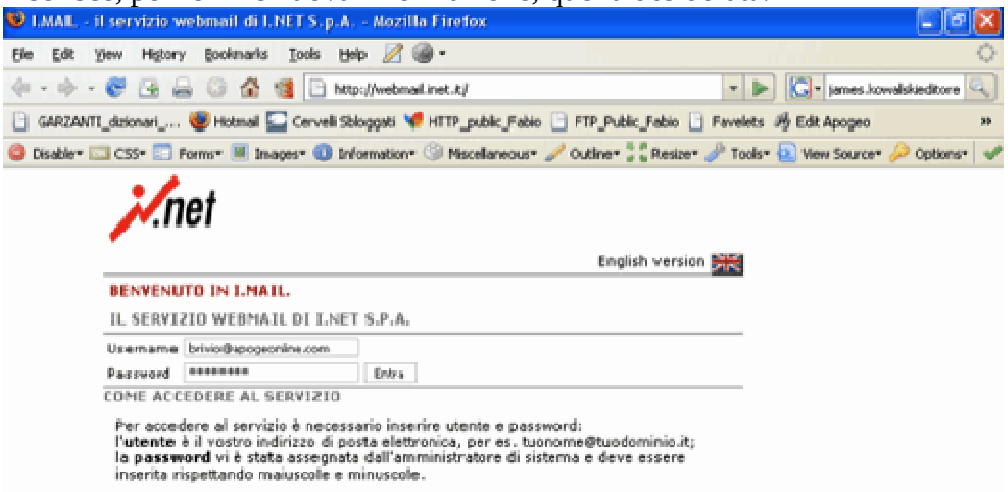


Figura 1.10 Per visualizzare la posta elettronica è necessario prima autenticarsi.

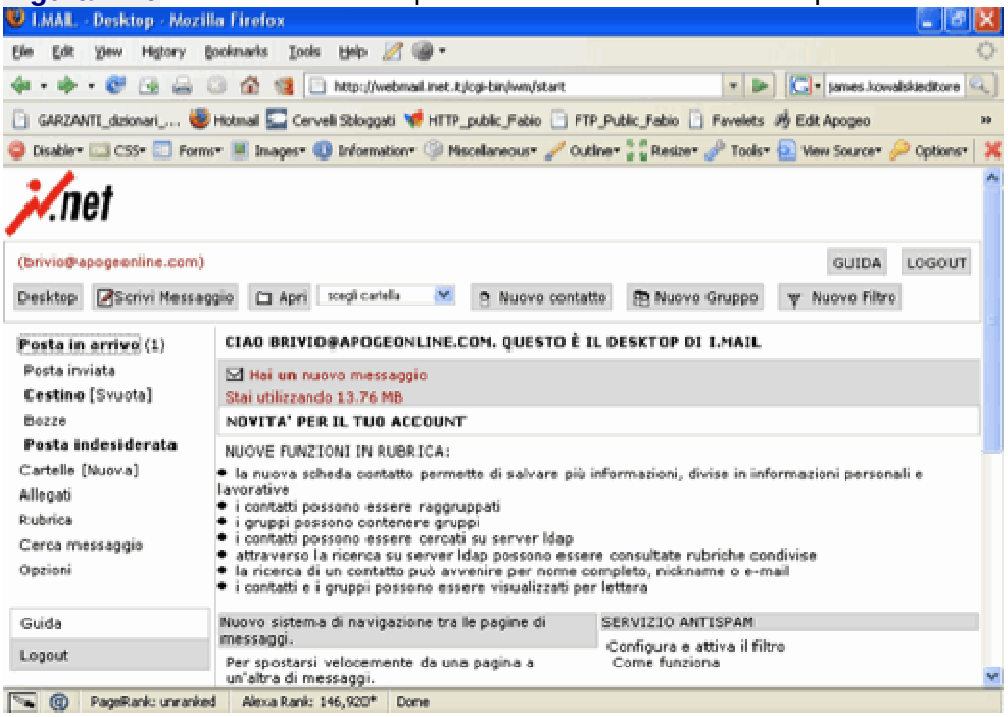


Figura 1.11 Le pagine dei servizi di webmail sono un esempio di pagine dinamiche.

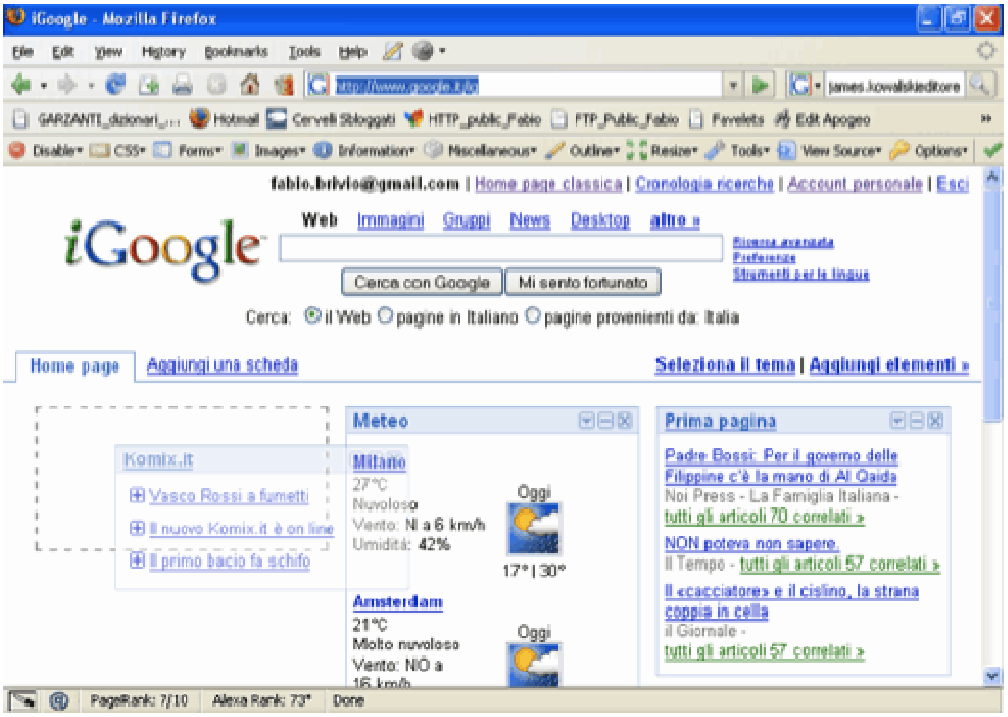


Figura 1.12 iGoogle permette a ogni utente di costruire una pagina su misura, con box informativi dal posizionamento variabile.

Presentazione

Il terzo livello della formattazione dell'informazione riguarda la sua presentazione o, volendo usare una metafora, l'abito con cui si sceglie di mostrare un'informazione.

La presentazione di un'informazione è, istintivamente, il primo elemento che viene percepito dall'utente, e per questo è di particolare importanza.

La presentazione riguarda tutto quello che è "stile": il corpo, il font, il colore di un paragrafo, ma anche i suoi margini e la sua interlinea; la misura della gabbia e l'allineamento di un testo ma anche la tipologia di un elenco (puntato, numerato con cifre arabe o romane); quindi grassetto, corsivo e sottolineato, ma anche l'utilizzo di maiuscolo e minuscolo.

In un documento riguarda la presentazione anche il posizionamento di un elemento, testuale e non (come per esempio un'immagine), e la sua dimensione. Nell'intestazione, al piede, a centro pagina con il testo che gli scorre attorno: tutto questo è presentazione. Ma è presentazione anche il colore e lo spessore della griglia di una tabella (in un documento di testo, ma anche in un foglio di calcolo), la grandezza delle celle e l'eventuale colore di sfondo.

Al fine della comprensione di un'informazione, una buona presentazione, calibrata all'utenza a cui ci si rivolge, conta almeno quanto la struttura dell'informazione stessa.

Per esempio, in un libro per bambini il testo deve essere in un corpo maggiore e l'interlinea più ampia rispetto a un manuale universitario. Il testo in una pagina web dovrebbe sempre avere una giustezza contenuta ed essere allineato a sinistra, per agevolarne la lettura a video, e i link dovrebbero essere immediatamente identificabili, di norma grazie a colori evidenti.

SUGGERIMENTO *Leggere a video è ben diverso che leggere su carta. Righe troppo lunghe e testi giustificati male si prestano al Web (Figure 1.13 e 1.14). Quando un contenuto è dedicato alla Rete, la presentazione delle informazioni assume delle peculiarità che non vanno trascurate e che influiscono sull'usabilità di una pagina web.*

Figura 1.13 La presentazione di questa pagina web non agevola la sua lettura.

Figura 1.14 Lo stesso contenuto presentato su più righe e con a capo più ravvicinati guadagna in leggibilità.

Come ricordato in precedenza, il livello della presentazione lavora necessariamente sopra quello della struttura. Non è possibile applicare alcuna informazione di stile a un testo se prima tale testo non è stato in qualche maniera strutturato. E tanto più la struttura è dettagliata e profonda, tanto più è possibile gestire e sviluppare una presentazione elaborata e precisa.

Per comprendere meglio questo, fondamentale, concetto, basta guardare le pagine di questo libro. I titoli di primo livello sono più grandi di quelli di secondo. Le note hanno rientri e proprietà del

testo (corsivo) che i normali paragrafi non presentano. Alcuni termini nei paragrafi sono in un font monospaziato o in corsivo. Tutto questo poggia su una struttura ben precisa di cui fanno parte, appunto, titoli, paragrafi, note, elenchi e box: sono tutti elementi che vengono convenzionalmente indicati come *di blocco*, a cui si contrappongono gli elementi *in linea* che permettono di strutturare (o *marcare*) termini (parole e frasi) all'interno dei vari blocchi (paragrafi, note, elenchi e così via).

SUGGERIMENTO *Sugli elementi di blocco e linea torneremo nel Capitolo 3.*

Questo processo, in un editor di testo, viene gestito con larga autonomia dal programma, al punto che spesso l'utente può disinteressarsene o neppure accorgersene. Quando si apre Word e si comincia a scrivere, il livello di struttura in cui ci trova è il paragrafo (di default nero e in Times New Roman). Quindi possono essere aggiunti titoli e altri elementi e applicati corsivo e grassetto ad alcuni termini, inserite e posizionate immagini e così via. Tutte queste operazioni, oltre che un effetto visivo, hanno un impatto diretto sulla struttura del documento.

NOTA *Ogni editor di testo consente di lavorare sulla struttura di un documento. In Word, il menu* Visualizza/Struttura *e il menu* Stile *permettono, rispettivamente, di visualizzare la struttura del documento e tutti gli stili (intesi qui come la somma dell'elemento strutturale e della relativa presentazione) utilizzati e disponibili (Figure 1.15 e 1.16). Inoltre il menu* Formato/Stili e formattazione *permette di lavorare sugli stili specificandone le qualità strutturali oltre che quelle presentazionali (Figura 1.17).*

ATTENZIONE *La forza di un editor di testo come Word consiste proprio nel rendere trasparenti all'utente i livelli di formattazione dell'informazione, automatizzando il più possibile le operazioni di strutturazione e presentazione di un documento. L'utente può così concentrarsi sul contenuto, applicando elementi di stile che portano con sé anche informazioni di struttura. Un indubbio vantaggio per chi scrive, ma il rovescio della medaglia esiste e non è di poca importanza. Non essendo evidente la struttura del testo su cui si lavora, spesso è facile creare documenti deboli o scorretti proprio a questo livello, e ciò può diventare un problema serio quando il documento deve essere riutilizzato in altri ambiti, per esempio portato su una pagina web o importato in un programma di impaginazione professionale.*

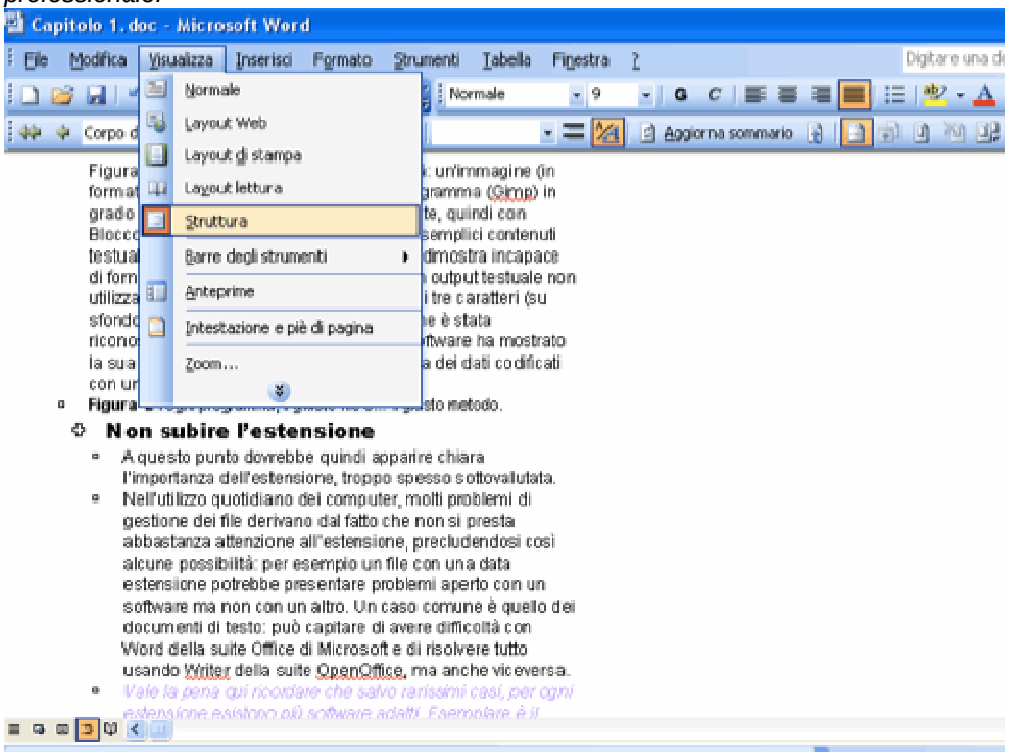


Figura 1.15 Word visualizza la struttura di un documento.

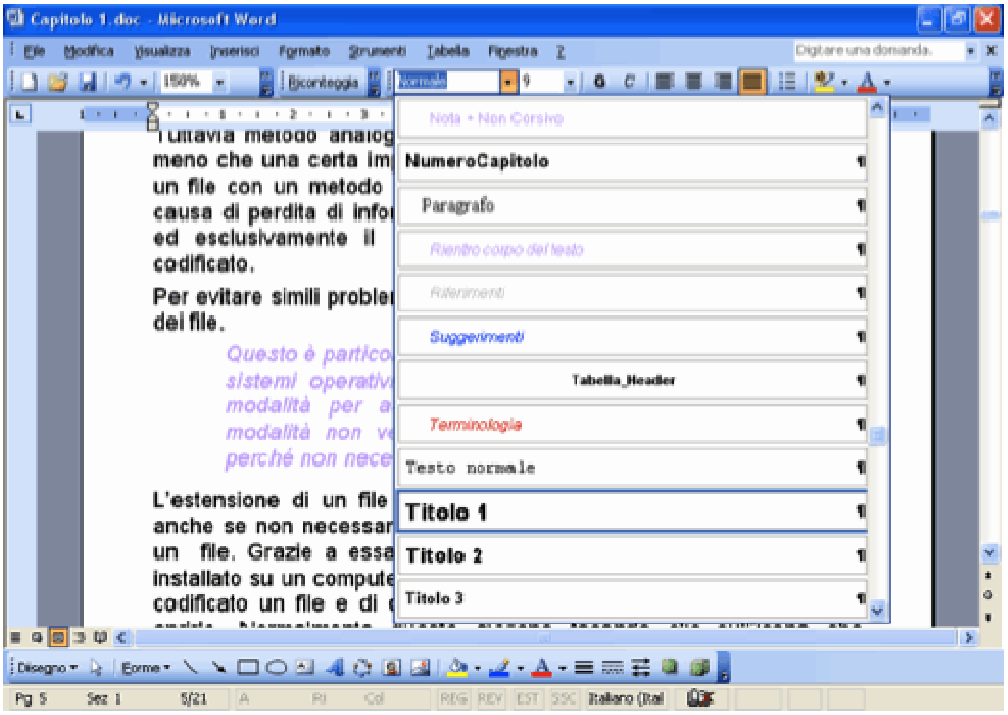


Figura 1.16 L'elenco degli stili disponibili.

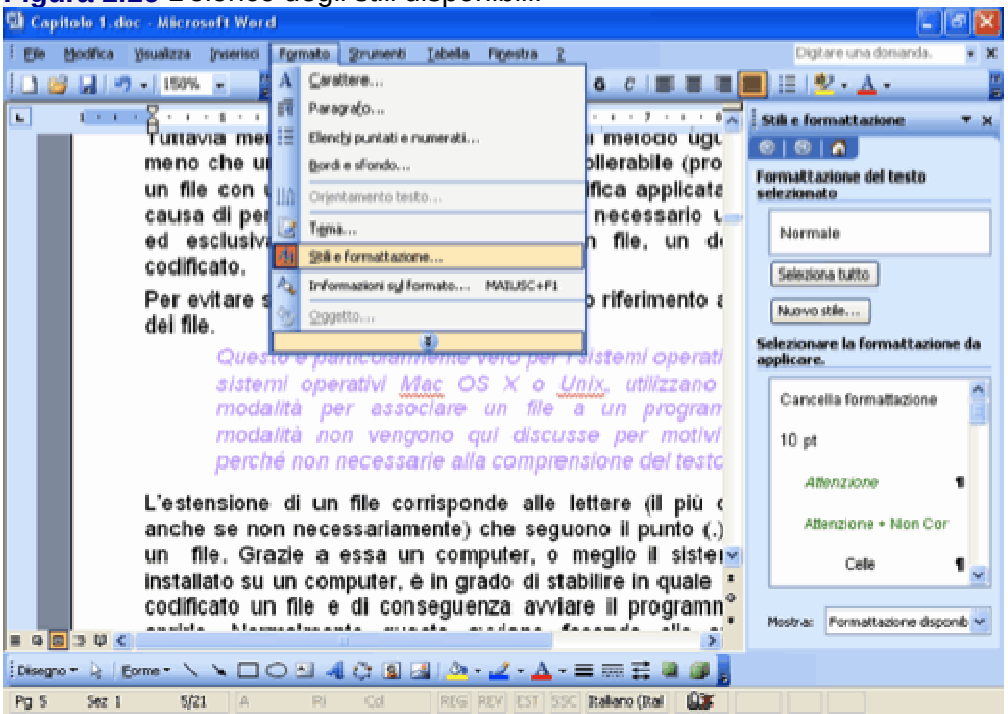


Figura 1.17 L'interfaccia, a destra, che permette di lavorare e modificare gli stili.

Infine, la presentazione di un'informazione può essere direttamente legata al livello del comportamento. È il caso, per esempio, di quei siti web che permettono di scegliere layout della pagina, grandezza del font o colore di sfondo, per andare incontro alle necessità degli utenti con deficit visivi o limitazioni analoghe (Figura 1.18). In questi casi la selezione di un'opzione innesca il cambiamento della pagina la cui presentazione viene modificata senza però alterarne la struttura.

Figura 1.18 Nella testata del sito Diodati.org un link permette di scegliere se visualizzare le pagine con una gabbia a una o due colonne.

Per concludere

In questo capitolo abbiamo introdotto diversi concetti. “Introdurre” è l’unico verbo possibile data la vastità del tema affrontato: come le macchine gestiscono le informazioni. Fondamentale è avere ben chiara l’articolazione dell’informazione su livelli di struttura, comportamento e presentazione. Su di essi torneremo nei prossimi capitoli, dove verranno presentate alcune tecnologie e linguaggi utili alla loro gestione.

Capitolo 2

La lingua della struttura e della semantica

In questo capitolo parliamo di XML, una tecnologia, anzi un linguaggio, che permette di definire in maniera precisa e rigorosa la struttura di un documento arricchendola di valore semantico.

Grazie a XML è possibile creare documenti che si prestano a essere condivisi e riutilizzati in diversi ambiti e da diversi programmi. Insomma è possibile prendere un'informazione, diffonderla e crearne di nuova.

XML presenta alcuni aspetti che ben si adattano alle capacità di analisi che di solito vengono associate alla figura dell'umanista. Ma andiamo con ordine.

NOTA *XML è stato definito dal W3C (*World Wide Web Consortium*, http://www.w3c.it/), l'organo che ha come scopo lo sviluppo e il mantenimento delle tecnologie e degli standard per il Web. La specifica di XML (insieme alle varie traduzioni disponibili), cioè il documento che ne presenta e illustra caratteristiche e funzionalità, è consultabile presso l'indirizzo internet* `http://www.w3.org/XML/`.

La definizione di XML

XML è l'acronimo di *eXtensible Markup Language* (Linguaggio di Marcatura Estensibile).

Un linguaggio di marcatura è così detto perché utilizza dei marcatori per descrivere le caratteristiche logiche di un documento e le eventuali relazioni semantiche tra le informazioni in esso contenute. Grazie ai marcatori un programma opportunamente istruito è in grado di svolgere operazioni su un documento ed elaborarlo.

NOTA *Molti editor supportano e sono in grado di lavorare sulla base di XML.*

Un marcatore, detto anche *tag*, è rappresentato dal simbolo di minore, "<", seguito dal nome del marcatore, seguito a sua volta dal simbolo di maggiore,">" (`<nome_marcatore>`).

Quanto appena detto chiarisce il concetto di "Markup Language". Ma perché "eXtensible"?

La possibilità di estensione di XML è la sua particolarità più importante.

In realtà XML non è un vero linguaggio ma un *metalinguaggio*, cioè un linguaggio che permette di creare nuovi linguaggi. In pratica, a seconda delle necessità di descrizione di un insieme di documenti, è possibile definire un insieme di marcatori specifici. Questo insieme di marcatori è detto *applicazione XML* e le sue regole (cioè le relazioni tra un marcatore e l'altro) sono codificate nelle cosiddette *grammatiche XML*, o DTD, acronimo di *Document Type Definition* (Definizione del Tipo di Documento). Queste grammatiche sono estensibili, vale a dire che di fronte a una nuova esigenza di descrizione è sempre possibile inserire dei nuovi marcatori e specificarne le relazioni nel DTD. Inoltre marcatori di grammatiche diverse possono essere anche utilizzati per marcare, cioè descrivere, un documento comune per il quale un singolo DTD non è sufficiente.

A questo punto, dovrebbe cominciare ad apparire evidente come, proprio per le sue possibilità di estensione, XML sia un potente strumento nelle mani di chi ha la necessità di descrivere informazioni e documenti, rendendoli poi riutilizzabili da terze parti.

NOTA *Si pensi ad archivi e biblioteche, a editori, enti di ricerca, ma in generale a ogni ufficio di qualsiasi azienda: che si tratti di schede bibliografiche, codici minati, libri o fatture, tutto questo dovrebbe poter essere descritto e archiviato, per un successivo recupero e riutilizzo. E se alla base di questa attività c'è la definizione di una grammatica, non è irragionevole pensare che, in questo scenario, un ruolo importante può essere svolto dall'umanista, che proprio attraverso i suoi studi ha sviluppato confidenza con le grammatiche delle ben più*

complicate lingue umane e con le strutture logiche del pensiero.

La sintassi

Come ogni linguaggio che si rispetti, XML ha delle regole sintattiche che vanno assolutamente osservate. Il rischio è quello di produrre documenti imperfetti, contenenti errori tali da pregiudicare la comunicazione, lo scopo principale di qualsiasi tecnologia dedicata alla gestione di informazioni.

I documenti XML sono oggetto di due tipi di verifica che determinano se essi siano ***ben formati*** e ***validi***.

Prima di procedere elencando i criteri che permettono di soddisfare queste due verifiche, dobbiamo però spendere ancora qualche riga per illustrare la sintassi dei tag.

I tag

I tag possono contenere informazioni, informazioni e altri tag o essere vuoti.

Ecco tre esempi:

```
<!-- Esempio 1: tag che contengono informazione -->
<nome_tag>Informazione</nome_tag>

<!-- Esempio 2: tag che contengono informazione e
altri tag -->
<nome_tag1>
     <nome_tag2>Informazione</nome_tag2>
</nome_tag1>

<!-- Esempio 3: tag vuoti -->
<nome_tag></nome_tag>
```

NOTA *Attraverso il tag* `<!-- ... -->` *è possibile inserire dei commenti. Questi commenti vengono ignorati dai programmi che elaborano il documento, ma risultano essere utili per gli autori del documento stesso o per le persone che dovranno lavorarci successivamente. I commenti possono essere usati per evidenziare sezioni di documenti particolarmente lunghi, per inserire delle annotazioni di carattere tecnico o per fissare qualsiasi altro tipo di informazione non strettamente legata ai dati veicolati nel documento XML.*

Negli esempi i tag sono presentati in coppia, tag di apertura e tag di chiusura, nella formula `<nome_tag>...</nome_tag>`.

Questa formula va rispettata ed è una delle principali norme sintattiche di XML: al tag di apertura (`<nome_tag>`), segue il contenuto informativo (se presente), quindi il tag di chiusura (`</nome_tag>`) che si differenzia per la presenza del carattere di slash (/) collocato tra il simbolo di minore e il nome del marcatore. In questo modo è possibile indicare con precisione dove comincia e finisce una marcatura. L'insieme di tag di apertura, contenuto e tag di chiusura è detto ***elemento*** di un documento XML.

Negli esempi sopra ricordati sono quindi elementi:

```
<nome_tag>Informazione</nome_tag>
<nome_tag2>Informazione</nome_tag2>
```

ma anche

```
<nome_tag1>
     <nome_tag2>Informazione</nome_tag2>
</nome_tag1>
```

Quest'ultimo caso mostra come un elemento possa contenere più serie di tag, oltre

all’informazione vera e propria.

L’Esempio 3 mostra invece una coppia di tag vuoti. L’utilizzo e il senso di questa soluzione apparirà più chiaro tra qualche pagina. Qui ci basta aggiungere che in questi casi si parla di ***elementi vuoti*** e che gli elementi vuoti possono essere espressi anche con una sintassi contratta, per cui l’elemento

```
<nome_tag></nome_tag>
```

può essere scritto semplicemente così:

```
<nome_tag />
```

A questo punto possiamo affrontare le altre regole sintattiche di XML.

Documenti ben formati e documenti validi

Un documento XML si dice ben formato (***well formed***) quando rispetta le seguenti condizioni.

- Tutto il documento è compreso tra un unico tag, detto radice (*root*), che lo descrive a livello più alto:

```
<!-- Esempio: documento compreso in un tag root -->
<tag_radice>
…
      <nome_tag1>
          <nome_tag2>Informazione</nome_tag2>
      </nome_tag1>
…
</tag_radice>
<!-- Termine del documento -->
```

- I nomi dei marcatori non cominciano con numeri e non contengono spazi o caratteri speciali (come i caratteri accentati):

```
<!-- Esempio: nomi di marcatori non accettabili -->
<1nome_tag>
<nome tag>
<nomè_tag>
```

- Tutti i tag sono annidati correttamente, cioè si chiudono rispettando l’ordine con cui sono stati aperti e nessun tag è lasciato aperto:

```
<!-- Esempio di errato annidamento dai tag:
nome_tag1 è chiuso prima di nome_tag2-->
<nome_tag1>
     <nome_tag2>Informazione</nome_tag1>
</nome_tag2>

<!-- Esempio di errato annidamento dai tag:
nome_tag2 non è chiuso -->
<nome_tag1>
     <nome_tag2>Informazione
</nome_tag1>
```

- Le coppie di marcatori di apertura e di chiusura sono scritte nel medesimo *case*, maiuscolo o miniscolo:

```
<!-- Esempio: elementi non accettabili per la
violazione del case -->
<nome_tag1>Informazione</NOME_TAG1>
```

```
<Nome_tag1>Informazione</nome_tag1>
<NOME_TAG1>Informazione</nome_tag1>
```

SUGGERIMENTO *Nella scrittura dei marcatori, è preferibile adottare la convenzione di utilizzare sempre il minuscolo.*

Il rispetto di queste regole sintattiche rende un documento ben formato. Se, oltre che essere ben formato, un documento rispetta anche le regole grammaticali di un DTD, allora si dice che il documento è valido (*valid*).

NOTA *È quindi possibile avere un documento ben formato ma non valido, mentre non è possibile avere un documento valido che non sia ben formato.*

La necessità di avere documenti validi, oltre che ben formati, nasce nel momento in cui i documenti devono essere condivisi in ambiti diversi: in questi casi la presenza di una grammatica di riferimento è importante per la comprensione del documento stesso e per una sua successiva elaborazione o estensione. Il DTD garantisce in definitiva che un documento non sia trattato in maniera impropria, introducendo tag o elementi che, sebbene scritti correttamente, potrebbero essere causa di ambiguità di interpretazione da parte dei software preposti all'elaborazione.

TERMINOLOGIA *I software dedicati all'elaborazione dei documenti sono detti* parser*.*

XML in pratica

Dopo questa introduzione teorica è giunto il momento di mostrare quanto detto attraverso un esempio pratico, ovvero l'inizio di questo capitolo a cui è stata applicata una marcatura XML libera (cioè non codificata in nessun DTD) ma allo stesso tempo funzionale alla descrizione del testo. Per maggiore semplicità di lettura, nell'esempio che segue la marcatura è stata evidenziata in neretto. Ecco il risultato:

```
<libro title="L'Umanista informatico">
<!-- …il capitolo 2… -->
  <parte value="2" type="Capitolo">
  <titolo_parte>
La lingua della struttura e della semantica
  </titolo_parte>
  <par>In questo capitolo si parla di <acronym
descrizione="eXtensible Markup Language">XML</
acronym>, una tecnologia, anzi un linguaggio che
permette di definire in maniera precisa e rigorosa
la struttura di un documento arricchendola di valore
semantico. Grazie a <acronym descrizione="eXtensible
Markup Language">XML</acronym> è possibile creare
documenti che si prestano a essere condivisi e
riutilizzati in diversi ambiti e da diversi programmi.
Insomma è possibile prendere un'informazione,
diffonderla e crearne di nuova. <acronym
descrizione="eXtensible Markup Language">XML</acronym>
presenta alcuni aspetti che ben si adattano alle
capacità di analisi che di solito vengono associate
alla figura dell'umanista. Ma andiamo con ordine.</ par>
  <box value="riferimento">
<acronym descrizione="eXtensible Markup
Language">XML</acronym> è stato definito dal <acronym
descrizione="World Wide Web Consortium">W3C</acronym>
```

```
(World Wide Web Consortium, <url>http://www.w3c.it/</
url>), l'organo che ha come scopo lo sviluppo e il
mantenimento delle tecnologie e degli standard per il
Web. La specifica di <acronym descrizione="eXtensible
Markup Language">XML</acronym> (insieme alle varie
traduzioni disponibili), cioè il documento che ne
presenta e illustra caratteristiche e funzionalità, è
consultabile presso l'indirizzo internet
<url>http://www.w3.org/XML/</url>.
  </box>
<titolo1>
La definizione di <acronym descrizione="eXtensible
Markup Language">XML</acronym>
</titolo1>
  <par><acronym descrizione="eXtensible Markup
Language">XML</acronym> è l'acronimo di eXtensible
Markup Language (linguaggio di marcatura
estensibile). Un linguaggio di marcatura è così
detto perché utilizza dei marcatori per descrivere
le caratteristiche logiche di un documento e le
eventuali relazioni semantiche tra le informazioni
in esso contenute. Grazie ai marcatori un programma
opportunamente istruito è in grado poi di svolgere
operazioni su un documento ed elaborarlo.</par>
  <box value="nota">Molti editor supportano e
sono in grado di lavorare sulla base di <acronym
descrizione="eXtensible Markup Language">XML</
acronym>.
  </box>
<!-- …segue il resto del capitolo… -->
  </parte>
<!-- …segue il resto del libro… -->
</libro>
```

Marcatura strutturale

Come si vede il documento è interamente compreso in un tag radice, `<libro>`. In apertura questo marcatore contiene anche un'espressione non ancora incontrata: `title="L'Umanista informatico"`.

Si tratta di un attributo, `title`, a cui è associato un valore, `L'Umanista informatico`. XML prevede infatti la possibilità di inserire attributi nei marcatori, caricandoli di informazione. In questo caso la prima riga del documento permette di scoprire che si è in presenza di un documento di tipo "libro", e che questo libro ha un dato titolo.

SUGGERIMENTO *Un valore viene associato a un attributo attraverso il simbolo di uguale (=). Il valore deve quindi essere compreso tra una coppia di apici doppi o singoli ("", '').*

La stessa informazione avrebbe però potuto anche essere espressa diversamente, tramite un apposito marcatore. Per esempio:

```
<libro>
  <title>Umanista informatico</title>
<!-- …segue il resto del libro… -->
```

```
</libro>
```

La scelta di una modalità o dell'altra è legata alle necessità di descrizione e in un esempio circoscritto come questo ha poco peso. Tuttavia vale la pena notare come l'utilizzo di un attributo permetta di esprimere la stessa informazione in una sintassi più compatta. Non si fatica quindi a comprendere che se il lavoro di marcatura non fosse limitato a una singola pagina ma rivolto a una grande quantità di testi (il catalogo di un editore, il contenuto di una biblioteca, l'archivio di fatture di un'azienda…), optare per una soluzione o per l'altra avrebbe ben altre implicazioni, e comporterebbe oneri di lavoro e vantaggi differenti.

Per prima cosa, più un documento è lungo maggiori sono le risorse di calcolo necessarie per elaborarlo.

SUGGERIMENTO *Questa considerazione, in un momento storico dove le macchine sono sempre più potenti e in grado di elaborare velocemente quantità di dati impressionanti, potrebbe anche passare in secondo piano. Tuttavia tenere presente l'economia di un sistema è sempre una buona cosa: nel mondo dell'informatica (e non solo) le applicazioni migliori sono quelle che permettono di ottenere un risultato con il minor impiego di risorse possibile.*

Grande considerazione merita però il fatto che dovendo lavorare su un'ampia quantità di testi, è necessario utilizzare un'applicazione XML in grado di descriverli tutti senza ambiguità. Per esempio, lavorando alla codifica di testi antichi potrebbe anche capitare di trovarsi di fronte a libri manoscritti dal titolo incerto. In assenza di un titolo, il marcatore `<title>` andrebbe lasciato vuoto, nel rispetto delle regole sintattiche di XML.

Verrebbe però così a mancare un'informazione fondamentale per la corretta archiviazione e il riutilizzo del testo (il titolo è infatti un elemento fondante in qualsiasi catalogo). La stessa *impasse* si avrebbe del resto anche utilizzando l'attributo `title`.

Tuttavia, pensare in termini di attributi consente in certi casi una maggiore flessibilità di descrizione. Si potrebbe infatti prevedere un attributo utilizzabile in casi come questi per connotare comunque il testo in maniera univoca. Una soluzione potrebbe essere fornita dall'introduzione di un attributo id (identificativo) il cui valore potrebbe essere composto da un numero progressivo unito all'anno a cui il documento risale (se noto). L'assenza di una data non sarebbe del resto un problema, essendo l'unicità dell'informazione garantita dal progressivo numerico. Potrebbero quindi esistere libri così marcati:

```
<libro title="" id="0001_1715">
<libro title="" id="0002_1798">
<libro title="" id="0003_XXXX">
```

Un ulteriore grado di precisione potrebbe essere raggiunto spezzando l'informazione inserita in id tra due attributi: `id e anno`. Per cui i tre libri appena ricordati sarebbero così identificati:

```
<libro title="" id="0001" anno="1715">
<libro title="" id="0002" anno="1798">
<libro title="" id="0003" anno="">
```

NOTA *Analogamente all'attributo* `title`*, anche* id *e* anno *potrebbero essere espressi come marcatori. La scelta è sempre influenzata dai fini descrittivi e di riutilizzo dell'informazione. Quando si lavora con grandi basi informative, la definizione o la scelta di un'applicazione XML deve essere oggetto di un'attenta riflessione per evitare di trovarsi di fronte a situazioni critiche. Un buon approccio potrebbe essere quello di delegare ai marcatori tutte le informazioni fondanti un documento, quelle senza le quali esso non potrebbe essere tale (per esempio in un libro il testo dei capitoli), delegando agli attributi possibilità descrittive secondarie, applicabili a casi particolari.*

Tornando all'esempio iniziale, dopo `<libro>` vengono introdotti il marcatore `<parte>` e `<titolo_parte>`.

Come si vede, il primo contiene tutto il resto del capitolo e ha due attributi `value="2"` e `type="Capitolo"`. Questo marcatore comunica in una sola riga che siamo davanti a una macro area del libro, un capitolo, per la precisione il secondo. Vale la pena qui notare la flessibilità di questa soluzione: a type potrebbero essere attribuiti diversi valori, tanti quanti le parti costituenti un libro (Sommario, Premessa, Prefazione, Introduzione, Capitolo, Appendice, Bibliografia, Note, Indice Analitico); value potrebbe avere valore quando necessario (per Capitolo e Appendice, se il testo ne prevede più di una) oppure essere lasciato vuoto (per Introduzione, Prefazione, Premessa); nel caso raro ma non impossibile di due prefazioni, value tornerebbe a essere compilato.

`<titolo_parte>` non prevede invece attributi, ma questo non dovrebbe costituire un problema. In assenza di un valore il marcatore può essere lasciato vuoto, ma la struttura logica del documento non viene meno in quanto delegata al tag `<parte>` e ai suoi attributi.

NOTA *Questi pochi esempi sono già sufficienti per capire funzionalità e scopi degli elementi vuoti: essi permettono di evidenziare l'assenza di un'informazione, pur ricordando che essa è prevista e rilevante in un certo punto del documento. Si potrà anche non conoscere il titolo di un libro, ma questo non vuol dire che il libro non ne abbia uno.*

Marcatura semantica

I tag incontrati finora hanno un alto valore strutturale, cioè permettono di indicare con precisione dove cominciano e finiscono le parti del documento e in che relazione stanno le une con le altre: cosa segue o precede cosa e cosa contiene o è contenuto in cosa. Inoltre portano con sé anche informazione, o meglio *metainformazione* (vale a dire informazione sull'informazione), semantica, in pratica l'indicazione che un testo è un titolo di un capitolo, piuttosto che un capitolo intero.

A questo punto il documento prosegue presentando il contenuto vero e proprio del capitolo, organizzato in paragrafi marcati attraverso il tag `<par>`.

Si potrebbe pensare che la descrizione possa essere a questo livello più che sufficiente, ma come si vede all'interno dei paragrafi fanno la loro comparsa altri marcatori, per esempio `<acronym>` con attributo `descrizione`. In questo caso siamo di fronte a un marcatore prettamente semantico il cui scopo è non tanto descrivere la struttura del testo, quanto delimitare alcune sue parti accomunate dal fatto di essere sigle, per la precisione acronimi.

Nel testo gli acronimi compaiono senza criterio apparente e trovano la loro funzione, o giustificazione, nella frase che li contiene. Non esiste quindi alcun vincolo strutturale, tanto è vero che `<acronym>` può fare la sua comparsa anche in elementi diversi da `<par>`, come si vede dal frammento che segue:

```
<titolo1>
La definizione di ><acronym descrizione="eXtensible
Markup Language">XML</acronym>
</titolo1>
```

dove il tag `<titolo1>` definisce i titoli di primo livello del testo ma allo stesso tempo contiene l'acronimo "XML".

Poco prima è presente però un altro elemento delimitato dal tag `<box>` e ancora una volta dotato di attributo `value`. Si tratta di un elemento con valore sia strutturale sia semantico. Box sono infatti

quei testi del capitolo che si discostano dal flusso, dalla sequenza logica, dei paragrafi permettendo di aprire delle parentesi inerenti ai temi discussi.

I box possono avere del resto diversa valenza semantica; per esempio, in questo libro esistono box per spiegare termini nuovi o poco familiari; suggerire spunti di lettura, approfondimenti o consigli su come comportarsi in una determinata situazione; evidenziare criticità e richiamare l'attenzione su possibili problemi; presentare fonti, cartacee o digitali, dove reperire maggiori informazioni sui temi trattati; introdurre semplici note di approfondimento o curiosità.

L'espressione di tutte queste casistiche è delegata all'attributo `value` che qui può assumere i valori `terminologia`, `suggerimento`, `attenzione`, `riferimento` e `nota`.

All'interno di `<box>` fa la sua comparsa un altro marcatore prettamente semantico, `<url>`, il cui scopo è quello di connotare quelle stringhe di testo che costituiscono un indirizzo internet.

TERMINOLOGIA *URL è l'acronimo di* Uniform Resource Locator *(volendo tentare una traduzione, Identificatore Uniforme di Risorse). Si tratta di un indirizzo che definisce, in maniera univoca e non ambigua, la posizione di una risorsa in Rete. Approfondiremo la trattazione degli URL nel Capitolo 3, paragrafo "Sintassi degli indirizzi".*

Il senso della marcatura

Il piccolo documento XML fin qui discusso ci ha permesso di mettere in luce diverse peculiarità del linguaggio e del suo utilizzo.

Una prima considerazione è che in XML struttura e semantica vanno spesso a braccetto ma a volte la seconda è predominante rispetto alla prima.

La semantica è legata al nome del marcatore e ai relativi attributi, che pertanto meritano particolare attenzione sia in fase di definizione, cioè quando si scrive una grammatica XML, sia in fase di elaborazione, cioè quando si lavora su documenti a cui è applicata una grammatica XML. La metainformazione sul contenuto codificato è delegata a loro. E la metainformazione è l'elemento che rende i documenti XML condivisibili, comunicabili in larga scala. Un esempio aiuterà a chiarire meglio questo, fondamentale, concetto e la potenza di questo linguaggio.

Ecco un testo scritto in greco e privo di marcatura: quasi completamente incomprensibile, a meno di non conoscere il greco:

```
Κ στρα απο οργ
Αλ ξανδρος Μπαρ κκο
8,00 ευρ
Φελτριν λλι
Παγκ σμια οικονομικ κατ σταση
2007
```

Ecco lo stesso testo a cui è stata applicata una marcatura idonea al contenuto descritto. Il valore semantico, la metainformazione aggiunta grazie a XML, è evidente:

```
<libro>
  <titolo>Κ στρα απο οργ</titolo>
  <autore>Αλ ξανδρος Μπαρ κκο</autore>
  <prezzo>8,00 ευρ</prezzo>
  <editore>Φελτριν λλι</editore>
  <collana>Παγκ σμια οικονομικ κατ σταση</collana>
  <anno_pubblicazione>2007</anno_pubblicazione>
</libro>
```

Sebbene il contenuto vero e proprio rimanga intelligibile solo a fronte della conoscenza della lingua greca, le sue parti sono facilmente individuabili e comprensibili: si tratta di un libro di cui vengono elencate alcune informazioni bibliografiche. Se la grammatica XML utilizzata fosse condivisa (per esempio da un gruppo di biblioteche), queste informazioni potrebbero essere processate, elaborate e quindi inserite in un catalogo comune, il tutto senza alcuna ambiguità: nell'elemento `titolo` ci sarebbe sicuramente il titolo del libro, in `autore` il nome dell'autore e così via. Questa breve descrizione, una volta catalogata, potrebbe essere messa a disposizione di tutti gli utenti alla ricerca di informazioni in greco su questo libro.

Con l'inserimento di un altro marcatore, `<id>`, e di un attributo, `lingua`, potrebbe poi essere possibile mettere in relazione tutte le traduzioni disponibili. `<id>` identificherebbe in maniera univoca l'opera, il contenuto, a prescindere dalle sue localizzazioni linguistiche; lingua connoterebbe invece le singole traduzioni, cioè le manifestazioni del libro nelle varie lingue.

L'esempio in greco potrebbe quindi essere riscritto come segue:

```
<libro>
  <!-- il numero 10001 indica che si tratta dell'opera
10001 del catalogo, "el" che si tratta di una versione
in greco -->
  <id lingua="el">10001</id>
  <titolo>Κ στρα απο οργ</titolo>
  <autore>Αλ ξανδρος Μπαρ κκο</autore>
  <prezzo>8,00 ευρ</prezzo>
  <editore>Φελτριν λλι</editore>
  <collana>Παγκ σμια οικονομικ κατ σταση</collana>
  <anno_pubblicazione>2007</anno_pubblicazione>
</libro>
```

mentre il corrispettivo italiano sarebbe

```
<libro>
  <id lingua="it">10001</id>
  <titolo>Castelli di rabbia</titolo>
  <autore>Alessandro Baricco</autore>
  <prezzo>8,00 euro</prezzo>
  <editore>Feltrinelli</editore>
  <collana>Universale economica</collana>
  <anno_pubblicazione>2007</anno_pubblicazione>
</libro>
```

Semantica e possibilità di estensione di XML mostrano così, in questi semplici esempi, tutta la loro potenza.

La scelta della profondità della marcatura merita una nota conclusiva. Quanto bisogna scendere nel dettaglio avvicinandosi alla codifica di un contenuto in XML? Quanto ampia deve essere la grammatica XML su cui lavorare?

La risposta non può essere univoca ed è connessa agli utilizzi prefissi per il documento XML ottenuto. Tuttavia è importante sottolineare che, tanto più una marcatura è atomica e profonda, tanto maggiori sono le possibilità di riutilizzo e distribuzione dei contenuti per usi differenti, e quindi tanto più alta è la potenzialità comunicativa del documento.

La regola può essere così riassunta: la parcellizzazione dell'informazione è direttamente proporzionale alle sue possibilità di riutilizzo e distribuzione, che cresce al crescere del dettaglio della marcatura.

Nella pagine precedenti sono stati discussi i marcatori `<acronym>` e `<url>`. La loro introduzione appare ora più sensata: in un libro la possibilità di poter scendere all'interno dei paragrafi per recuperare in maniera puntuale parole, sigle o parti di frasi, permette di automatizzare la costruzione di indici analitici, glossari o altri tipi di apparati paratestuali, come per esempio la tavola degli indirizzi internet (URL) presenti nel testo.

DTD e filosofia

In precedenza si è più volte accennato ai DTD. Le grammatiche XML sono indispensabili per poter sfruttare appieno tutte le potenzialità di questa tecnologia.

È giunto il momento di approfondire un po' il discorso. Che cosa implica la creazione di un DTD?

Creare un DTD vuol dire ragionare nei termini di scomposizione di un oggetto logico, un discorso, per evidenziarne le parti costituenti.

Da un certo punto di vista si tratta di un procedimento simile al metodo analitico che si apprende studiando la filosofia scolastica medievale, la cui eredità e la cui influenza sono rintracciabili ben oltre il Medioevo, in alcuni filosofi e autori dell'epoca contemporanea.

Lo scopo dei maestri della scolastica era la sistematizzazione del sapere; per conseguire questo fine procedevano scomponendo i problemi più grandi in parti, di cui poi evidenziavano implicazioni e caratteristiche, aggiungendo commenti. Semplificando, potremmo dire che marcavano un contenuto, o per essere più precisi un pensiero. In pratica qualcosa di analogo a ciò che si fa con XML e i suoi marcatori, il più delle volte senza la necessità di raggiungere il livello di dettaglio che la filosofia permette.

Ecco quindi che ancora una volta diviene evidente come chi ha formazione umanistica, filosofica in particolare, non dovrebbe avere problemi ad avvicinarsi a questa tecnologia, in particolare alla sua parte più delicata, vale a dire la definizione di un DTD.

La sintassi di un DTD

Anche i DTD, come i documenti XML, hanno delle regole sintattiche che vanno assolutamente osservate e rispettate. Questa sintassi viene qui solo introdotta per cercare di tradurre in pratica le considerazioni del paragrafo precedente.

RIFERIMENTO *Le peculiarità di XML e le regole delle sue grammatiche sono tali e tante da non poter essere esaurite in un solo capitolo. In questa sede è importante cominciare a prendere confidenza con alcuni concetti fondamentali per avvicinarsi a queste tecnologie. Per chi volesse saperne di più, una ricerca su Google (o altro motore di ricerca) per "XML", "XML guida" o "XML tutorial" fornirà interessanti risorse. Se invece siete alla ricerca di un libro, valutate* XML Pocket *di Massimo Canducci, Apogeo, 2005.*

Come già ricordato il fine di un DTD è specificare le componenti (marcatori, attributi, informazioni, o più in generale elementi) ammesse in un documento XML.

In particolare in un DTD sono presenti regole che sanciscono:

- quali marcatori sono ammissibili in un documento XML, se sono obbligatori o opzionali e il numero di volte che possono essere utilizzati;
- la relazione e la gerarchia tra i vari marcatori, cioè cosa può essere contenuto in una coppia di

marcatori di apertura e chiusura (o meglio in un elemento), se informazioni o altri marcatori;

- gli attributi utilizzabili per ogni elemento e i relativi valori.

In definitiva si tratta di un vero “vocabolario” che insieme ai vocaboli porta con sé anche le regole per il loro corretto uso.

TERMINOLOGIA *File XML costruiti rispettando un determinato DTD vengono genericamente definiti come* applicazioni XML*. Tutti i documenti validi sono quindi applicazioni XML. Questa espressione viene però a volte utilizzata anche per documenti semplicemente ben formati.*

La definizione degli elementi

La prima cosa da sapere è che tutte le regole di un DTD devono essere scritte in un particolare marcatore, in questa forma:

```
<!DOCTYPE libro [
… regole …
]>
```

Nella prima riga, `DOCTYPE` è una parola chiave immutabile, mentre `libro` sta a indicare il tipo di documento e corrisponde al nome del tag radice (`<libro>`).

NOTA *Il DTD presentato nelle prossime pagine si riferisce al documento XML utilizzato come esempio nei paragrafi precedenti.*

Tutte le regole del documento di tipo `libro` andranno quindi a collocarsi tra le parentesi quadre, o meglio tutti i marcatori, gli elementi e i relativi attributi che potranno essere presenti dentro la coppia di marcatori `<libro></libro>` dovranno essere qui opportunamente dichiarati.

Si può quindi passare alla definizione delle regole vere e proprie inerenti gli elementi. In questo caso si utilizza la parola chiave `ELEMENT`, in questa forma:

```
<!ELEMENT nome_marcatore (tipo_regola)>
```

Normalmente si inizia dal tag radice, qui `<libro>`, specificandone il contenuto:

```
<!ELEMENT libro (parte+)>
```

SUGGERIMENTO *Nella scrittura di un DTD è importante fare attenzione alle maiuscole e minuscole: i nomi dei marcatori e attributi devono essere scritti nello stesso case con cui poi andranno utilizzati. Sempre in riferimento all'esempio di questo capitolo, è pertanto corretto dichiarare nella DTD un elemento* `libro` *in quanto il tag* `<libro>` *è stato così impiegato anche nel documento XML di esempio. L'utilizzo di un tag* `<Libro>` *(o* `<LIBRO>`*,* `<libRo>` *e così via) avrebbe invece causato un errore in sede di validazione. Vale la pena anche ricordare che in questo caso si sta procedendo in ordine inverso: normalmente la formulazione di un DTD precede la scrittura di un documento XML basato su di esso.*

Il contenuto delle parentesi tonde indica `che libro` può contenere solo l'elemento parte, una o più volte. Questa possibilità è dichiarata attraverso il simbolo più (+).

SUGGERIMENTO *L'assenza di + (o di altri simboli che verranno introdotti tra poco) indica che l'elemento è obbligatorio e utilizzabile una sola volta.*

A questo punto è necessario dichiarare cosa potrà essere contenuto in parte. Ancora una volta, trattandosi di elemento, è necessario utilizzare la parola chiave `ELEMENT`:

```
<!ELEMENT parte (titolo_parte, par+, box*, titolo1*)>
```

All'interno delle parentesi tonde di questa regola, separati da una virgola, sono elencati gli elementi che possono essere contenuti in parte:

- una sola occorrenza obbligatoria di `titolo_parte`, in quanto si è deciso che ogni parte del libro (sia esso Introduzione, Prefazione, Capitolo o Appendice) debba avere un (e solo un) titolo;

NOTA *Questa regola è puramente esemplificativa e non pretende di essere effettivamente pratica: nella realtà*

capita spesso che prefazioni e introduzioni non abbiano titoli, così come apparati paratestuali come sommari, indici analitici e bibliografie.

- una o più occorrenze di par, cioè paragrafi, senza i quali il libro avrebbe ben poco senso;
- da nessuna a più occorrenze di `box` e `titolo1` che si ritengono qui plausibili ma non strettamente necessari, in quanto un libro potrebbe avere senso compiuto anche senza di essi.

NOTA *La possibilità di utilizzare o meno un elemento è espressa attraverso il simbolo asterisco (*). Volendo è anche possibile dichiarare che un elemento può essere utilizzato una sola volta o nessuna: in questo caso il simbolo da associare all'elemento è il punto interrogativo (?).*

SUGGERIMENTO *Quando un elemento ne contiene altri, si parla di elementi* annidati*. Nella scrittura di un DTD, l'annidamento degli elementi è pertanto una fase importante e deve essere oggetto di particolare attenzione e riflessione: qui si definiscono le relazioni e le gerarchie possibili tra le parti di un documento.*

Questa regola sembra quindi adatta a descrivere il contenuto delle parti di un libro. Tuttavia c'è un errore: secondo la sintassi dei DTD i quattro elementi così dichiarati possono infatti essere utilizzati solo ed esattamente nell'ordine in cui compaiono nella regola (nessuna, una o più volte in base ai simboli associati).

SUGGERIMENTO *Un elenco di elementi separati da virgola e senza alcun simbolo associato indica che essi devono essere tutti presenti, obbligatoriamente una sola volta ed esattamente nell'ordine in cui appaiono nell'elenco stesso.*

Secondo questa regola un elemento `<parte>` potrebbe essere strutturato solo così:

```
<parte>
<!-- Un titolo di parte obbligatorio -->
     <titolo_parte>…</titolo_parte>
<!-- Almeno uno o più paragrafi -->
     <par>…</par>
     <par>…</par>
     <par>…</par>
     <par>…</par>
     …
<!-- Nessuno uno o più box -->
     <box>…</box>
     <box>…</box>
     <box>…</box>
     …
<!-- Nessuno uno o più titoli di livello 1 -->
     <titolo1>…</titolo1>
     <titolo1>…</titolo1>
     …
</parte>
```

Come si vede, questa struttura è troppo rigida e non adatta a descrivere in maniera flessibile il contenuto di un libro: paragrafi, box e titoli devono poter essere disposti nelle varie parti di un libro liberamente, in modo da articolare in maniera ottimale il filo del discorso. La reale struttura dell'esempio da cui si è partiti è infatti:

```
<parte>
     <titolo_parte>…</titolo_parte>
     <par>…</par>
     <box>…</box>
     <titolo1>…</titolo1>
     <par>…</par>
```

```
        <box>…</box>
    </parte>
```

Nei casi un cui è necessario definire un elemento senza però poterne specificare precisamente e rigidamente il contenuto, la sintassi dei DTD mette a disposizione la parola chiave `ANY`. Un elemento così definito può contenere qualsiasi cosa, testo o elementi di ogni tipo e quantità, purché presenti e definiti correttamente nel DTD. La regola può quindi essere riscritta in questo modo:

```
<!ELEMENT parte ANY>
```

Così facendo è possibile strutturare ogni parte di un libro in maniera diversa, secondo gli argomenti trattati e le necessità espressive dell'autore.

Resta il fatto che non è possibile utilizzare elementi non definiti nel DTD. `titolo_parte, par, box` e `titolo1` vanno quindi dichiarati tramite un'apposita regola, tenendo presente che a questo livello di dettaglio gli elementi possono contenere altri elementi ma anche informazioni testuali.

La possibilità di contenere informazioni in forma testuale va dichiarata utilizzando la parola chiave `#PCDATA` (PCDATA significa ***Parsed Character Data*** e indica al parser la presenza di testo generico). Ecco come:

```
<!ELEMENT titolo_parte (#PCDATA)>
<!ELEMENT par (#PCDATA)>
<!ELEMENT box (#PCDATA)>
<!ELEMENT titolo1 (#PCDATA)>
```

SUGGERIMENTO *Utilizzando* `#PCDATA` *è possibile inserire testo composto da qualsiasi carattere a eccezione dei caratteri & ("e" commerciale), < (minore) e > (maggiore). Questi caratteri sono considerati speciali in quanto alla base della sintassi di XML e pertanto non possono essere liberamente inseriti in un testo: il loro utilizzo provoca infatti errori in fase di elaborazione. Per includere questi caratteri in un testo è perciò necessario utilizzare particolari costrutti sintattici detti* entità*, che vanno sostituiti nel testo in corrispondenza del carattere a cui fanno riferimento. Torneremo sulle entità tra qualche pagina. Per ora è sufficiente sapere che:* `&` *è l'entità da utilizzare per il carattere &;* `<` *per <;* `>` *per >. A queste tre entità predefinite in XML se ne aggiungono altre due:* " *per il doppio apice (") e* ' *per il singolo apice (').*

NOTA *Nei testi è sempre obbligatorio l'uso delle entità per "e" commerciale, minore e maggiore. L'utilizzo delle entità per apice doppio e singolo è invece meno vincolante e necessario solo in casi particolari, come si vedrà in seguito. Il più delle volte è infatti possibile includere tranquillamente questi caratteri negli elementi definiti con* `#PCDATA`.

Queste regole soddisfano il fatto che i quattro elementi qui dichiarati contengano testo, ma non tengono conto che essi potrebbero contenere anche altri marcatori, nello specifico `<url>` e `<acronym>`. Le regole vanno pertanto modificate:

```
<!ELEMENT titolo_parte (#PCDATA ¦ url ¦ acronym)*>
<!ELEMENT par (#PCDATA ¦ url ¦ acronym)*>
<!ELEMENT box (#PCDATA ¦ url ¦ acronym)*>
<!ELEMENT titolo1 (#PCDATA ¦ url ¦ acronym)*>
```

Come si vede, gli elementi `url e acronym` sono presenti insieme a `#PCDATA` separati dal simbolo pipe (¦, la barra verticale). Questo carattere permette di creare elenchi di elementi che, all'interno dell'elemento che si dichiara, possono essere utilizzati in alternativa uno all'altro (l'utilizzo di url esclude quindi la possibilità di utilizzare anche `acronym` e così via). Aggiungendo il simbolo asterisco, subito dopo la parentesi tonda di chiusura, queste regole diventano però adatte agli scopi: come ricordato, l'asterisco permette di utilizzare un elemento zero o più volte, per cui

all'interno di ciascun elemento così dichiarato sarà possibile inserire tutto il testo e i marcatori <url> e <acronym> necessari, senza alcun vincolo in termini di numero di occorrenze.

NOTA *Questo esempio mostra come all'interno di una regola possano essere utilizzati più simboli diversi per raggiungere le necessità di descrizione desiderate.*

TERMINOLOGIA *In questi casi si parla di elementi dal contenuto misto. Negli elenchi degli elementi dal contenuto misto, la parola chiave* #PCDATA *deve sempre comparire per prima.*

In precedenza si è detto che un elemento potrebbe anche essere vuoto. Per dichiarare un elemento vuoto è necessario utilizzare la parola chiave EMPTY:

```
<!ELEMENT nome_elemento EMPTY>
```

NOTA *Normalmente gli elementi vuoti vengono usati in aggiunta di attributi.*

SUGGERIMENTO *Sia* ANY *sia* EMPTY *vanno scritti nelle regole senza le parentesi tonde.*

A questo punto, prima di passare alla definizione degli attributi, è utile introdurre la possibilità di definire elenchi di secondo livello all'interno di una regola. Per esempio, la seguente regola

```
<!ELEMENT parte (titolo_parte, par+, box*, titolo1*)>
```

potrebbe essere così ampliata

```
<!ELEMENT parte (titolo_parte, (citazione ¦ epigrafe),
par+, box*, titolo1*)>
```

In questo modo si specifica la possibilità di inserire, dopo il titolo della parte, una citazione o un'epigrafe. Questa possibilità è sintatticamente espressa con una nuova coppia di parentesi tonde, al cui interno trova posto un elenco di secondo livello composto da due nuovi elementi separati dal simbolo pipe per esplicitare che è possibile utilizzare un elemento oppure l'altro, ma non tutti e due contemporaneamente.

NOTA *È possibile scendere in profondità inserendo elenchi di livello inferiore al secondo. Ogni elenco deve sempre essere racchiuso tra parentesi tonde e può utilizzare tutti i simboli fin qui visti (virgole, pipe, asterischi, più, punti interrogativi) nel modo più idoneo a formalizzare la regola che si sta scrivendo.*

La definizione degli attributi

Come ricordato, a un elemento possono essere associati uno o più attributi. Nei marcatori dei documenti XML, gli attributi si manifestano in questa forma: nome_attributo="valore"

```
nome_attributo="valore"
```

Nei DTD gli attributi vanno definiti per ogni elemento utilizzando la parola chiave ATTLIST, in questa forma:

```
<!ATTLIST nome_elemento
    nome_attributo1 tipo_contenuto caratteristiche
    nome_attributo2 tipo_contenuto caratteristiche
    …
>
```

Una sola regola permette così di definire tutti gli attributi per un dato elemento. Per ogni attributo vengono quindi specificati il nome, il tipo di contenuto e le caratteristiche.

Per prima cosa è utile fissare le caratteristiche che un attributo può assumere. Sono tre.

- L'attributo è obbligatorio nell'elemento, quindi deve sempre essere presente. Per esprimere questa condizione si usa la parola chiave #REQUIRED.
- L'attributo è facoltativo nell'elemento, può essere presente o meno. Per esprimere questa condizione si usa la parola chiave #IMPLIED.
- L'attributo ha un valore fisso che deve essere opportunamente dichiarato. Per esprimere

questa condizione si usa la parola chiave `#FIXED` seguita dal valore (`#FIXED "valore"`).

Tre sono anche le tipologie di contenuto specificabili: stringhe di testo, enumerazioni e *token*. Qui vengono presentati solo i primi due, di più facile comprensione e di uso più diffuso.

NOTA *I token come altre peculiarità relative alle definizione degli attributi sono argomenti più complessi che non è possibile esaurire in queste pagine. Per essi si rimanda a testi specifici. Ancora una volta può essere utile svolgere qualche ricerca sul Web, per esempio per "token xml", "token dtd" o più genericamente "dtd xml".*

Per specificare che un attributo può contenere stringhe di testo indefinite si usa la parola chiave `CDATA`.

La definizione dell'attributo descrizione per l'elemento acronym incontrato in precedenza

```
<acronym descrizione="eXtensible Markup
Language">XML</acronym>
```

è pertanto la seguente:

```
<!ATTLIST acronym
    descrizione CDATA #REQUIRED
>
```

Un attributo così definito è pertanto obbligatorio e può contenere qualsiasi valore testuale. Se invece la regola fosse stata così espressa

```
<!ATTLIST acronym
    descrizione CDATA #IMPLIED
>
```

l'attributo sarebbe stato facoltativo, quindi non necessario nel documento XML e ai fini della sua validazione.

SUGGERIMENTO *Come abbiamo detto,* `CDATA` *permette di inserire un testo libero come valore di un attributo. La sintassi degli attributi prevede però che i valori siano rinchiusi in una coppia di apici, singoli o doppi. Se nei testi dei valori compaiono quindi apici singoli o doppi, questi possono provocare problemi in fase di elaborazione, nel caso in cui lo stesso tipo di apice sia utilizzato per racchiudere il valore. In questi casi è quindi necessario utilizzare le entità predefinite* `&quot;` *(per il doppio apice) e* `&apos;` *(per il singolo apice), all'interno del contenuto* `CDATA`*.*

ATTENZIONE *È buona norma utilizzare il doppio apice per racchiudere i valori degli attributi. È più raro che un testo – specialmente se non lungo e articolato, come quello che normalmente costituisce i valori degli attributi – contenga apici doppi, mentre è più comune l'impiego di apici singoli. Così facendo si riduce le possibilità di dover fare ricorso alle entità.*

Gli attributi che permettono enumerazioni sono invece connotati da un elenco di valori finiti validi. L'utilizzo di altri valori provoca quindi errori in fase di elaborazione. Ecco un esempio relativo all'attributo `value` dell'elemento box:

```
<!ATTLIST box
    value (terminologia ¦ suggerimento ¦ attenzione ¦
riferimento ¦ nota) #REQUIRED
>
```

Ancora una volta è il simbolo pipe a specificare la scelta tra i valori elencati. L'attributo è inoltre obbligatorio e quindi deve necessariamente comparire associato a uno dei cinque valori predefiniti.

NOTA *È utile notare come nel caso delle enumerazioni il contenuto dell'attributo sia racchiuso tra parentesi tonde, cosa che non avviene nel caso delle stringhe di testo con* `CDATA`*, utilizzato nelle regole senza parentesi.*

In precedenza si è anche incontrato l'elemento `<parte>` associato agli attributi `value` e `type`:

```
<parte value="2" type="Capitolo">
```

Si è anche detto che a `type` potrebbero essere attribuiti diversi valori, tanti quanti le parti

costituenti un libro (Sommario, Premessa, Prefazione, Introduzione, Capitolo, Appendice, Bibliografia, Note, Indice Analitico), mentre value potrebbe avere valore quando necessario (per esempio per Capitolo e Appendice) oppure essere lasciato vuoto o anche non essere utilizzato. Ecco la regola per formalizzare quanto sopra:

```
<!ATTLIST parte
    value CDATA #IMPLIED
    type (Sommario ¦ Premessa ¦ Prefazione ¦
Introduzione ¦ Capitolo ¦ Appendice ¦ Bibliografia ¦
Note ¦ Analitico) #REQUIRED
>
```

Come si vede, `value` può contenere qualsiasi testo ed è facoltativo. In questo modo è possibile assegnare a una parte del libro un progressivo numerico (per esempio 1, 2, 3… per i capitoli) o letterale (A, B, C… per le appendici) o altro per casi particolari. L'attributo type è invece obbligatorio e i valori ammissibili sono tutti specificati. In questo modo ogni parte del libro sarà sempre riconoscibile per tipo e nel caso di più parti dello stesso tipo queste saranno etichettabili anche per numero o altro progressivo.

La definizione delle entità

Abbiamo già accennato alle entità ricordando le cinque entità predefinite in XML: `&` (&); `<` (<); `>` (>), `&quot;` (“) e `&apos;` (‘). Tali entità sono sempre disponibili e utilizzabili in qualsiasi documento XML.

Esiste però la possibilità di dichiarare nuove entità nei DTD in modo da semplificare e modulare la gestione di alcuni contenuti o informazioni.

Un'entità è di fatto un oggetto logico, normalmente un testo ma volendo anche un frammento di codice XML (o di altro tipo), il cui uso ripetuto è prevedibile all'interno di un documento. In questo modo si semplifica e si rende più economica la gestione delle informazioni di un documento nel quale viene espressa solo l'entità: solo in fase di elaborazione l'entità viene sostituita dal parser con il reale contenuto.

Per dichiarare un'entità si utilizza la parola chiave `ENTITY`, in questa forma:

```
<!ENTITY nome_entità oggetto>
```

Per inserire un'entità all'interno di un documento XML la sintassi è invece

```
&nome_entità;
```

NOTA *L'utilizzo del carattere “e” commerciale (&) nella sintassi delle entità giustifica la necessità di avere un'entità appositamente dedicata a esso utilizzabile nei testi degli elementi e degli attributi XML.*

Un semplice esempio di impiego delle entità si può manifestare nel caso in cui ci si trovi a lavorare su una grande quantità di documenti a firma dello stesso autore. Dovendo inserire in ogni documento il nome dell'autore, potrebbe essere pratico dichiarare nel DTD un'entità:

```
<!ENTITY autore “Fabio Brivio”>
```

Nei vari documenti XML essa potrebbe quindi essere inserita come segue:

```
<documento value='1'>
     <autore>&autore;</autore>
     …
</documento>

<documento value='2'>
```

```
        <autore>&autore;</autore>
        …
    </documento>

    <documento value='3'>
        <autore>&autore;</autore>
        …
    </documento>
```

Così facendo, di fronte alla necessità di modificare il nome dell'autore, per esempio in `Brivio, Fabio`, non sarebbe necessario editare singolarmente ogni documento, ma basterebbe intervenire una sola volta nel DTD

```
<!ENTITY autore "Brivio, Fabio">
```

con un notevole risparmio di tempo e risorse.

Le entità possono essere usate anche per gestire contenuti molto più complessi di una breve stringa, come frammenti di codice o testi strutturati.

In questi casi è possibile dichiarare l'entità come esterna al DTD. La regola dell'entità sarà sempre all'interno del DTD, ma il suo reale contenuto, l'oggetto, sarà in un file esterno che potrebbe anche trovarsi su un'altra macchina diversa da quella dove il DTD è salvato.

La sintassi per dichiarare un'entità esterna è la seguente:

```
<!ENTITY nome_entità SYSTEM indirizzo>
```

Attraverso la parola chiave `SYSTEM` si specifica che l'oggetto dell'entità è reperibile in un file presente a un dato indirizzo, come in

```
<!ENTITY autore SYSTEM "autore.txt">
```

Tra i doppi apici deve quindi essere incluso il nome del file (`autore. txt`) e il relativo indirizzo: può trattarsi di un indirizzo locale (cioè della stessa macchina su cui è presente il DTD, nella forma `Cartella/Sottocartella/File`, si veda il Capitolo 1) o Internet (nella forma `http://server.nome_file`, si veda il Capitolo 3).

SUGGERIMENTO *Nell'esempio appena ricordato il file* `autore.txt` *deve trovarsi nella stessa cartella del DTD, in quanto non sono specificati altri parametri per l'indirizzo: il parser che elaborerà questa istruzione, in assenza di ulteriori informazioni, si limiterà infatti a cercare il file nella cartella in cui si trova il DTD.*

In ogni caso il vantaggio è ancora una volta la semplicità di gestione: grosse parti di codice o testi lunghi possono essere in questo modo separati dal DTD (che guadagna così in semplicità), ma utilizzati quando necessario in un documento XML.

NOTA *È il caso per esempio di lunghi elenchi che vanno ripetuti in più documenti. La possibilità di dichiararli come entità e delegarli a file esterni snellisce molto la loro gestione e gli eventuali aggiornamenti.*

Il DTD di esempio

Alla luce di quanto fin qui visto, il DTD del nostro file XML di esempio potrebbe essere formulato come segue:

```
<!DOCTYPE libro [
<!ELEMENT libro (parte+)>
<!ATTLIST libro
    title CDATA #REQUIRED
    autore CDATA #REQUIRED
>
<!ELEMENT parte ANY>
```

```
<!ATTLIST parte
    value CDATA #IMPLIED
    type (Sommario ¦ Premessa ¦ Prefazione ¦
Introduzione ¦ Capitolo ¦ Appendice ¦ Bibliografia ¦
Note ¦ Analitico) #REQUIRED
>
<!ELEMENT titolo_parte (#PCDATA ¦ url ¦ acronym)*>
<!ELEMENT par (#PCDATA ¦ url ¦ acronym)*>
<!ELEMENT box (#PCDATA ¦ url ¦ acronym)*>
<!ATTLIST box
    value (terminologia ¦ suggerimento ¦ attenzione ¦
riferimento ¦ nota) #REQUIRED
>
<!ELEMENT titolo1 (#PCDATA ¦ url ¦ acronym)*>
<!ELEMENT url (#PCDATA)>
<!ELEMENT acronym (#PCDATA)>
<!ATTLIST acronym
    descrizione CDATA #REQUIRED
>
<!ENTITY autore “Fabio Brivio”>
]>
```

Come si vede, al suo interno elementi, attributi ed entità sono facilmente individuabili grazie alle rispettive parole chiave. Inoltre, per agevolarne la lettura, la definizione degli attributi segue sempre quella dell'elemento nel quale possono comparire. Per completezza e correttezza grammaticale nei confronti del documento XML di esempio, sono state aggiunte le regole per gli elementi `url` e `acronym` e per l'attributo `title` di `libro`. Inoltre è stato aggiunto anche un attributo `autore` per `libro`, che può venire compilato con l'omonima entità vista in precedenza:

```
<libro title=”Umanista informatico” autore=”&autore;”>
```

RIFERIMENTO *Questo esempio di DTD ha valore solo a fini didattici e non pretende di essere una traccia perseguibile per codificare libri o altri testi con XML. Per questo scopo esistono invece due ampi progetti che hanno prodotto due grammatiche XML diverse ma potenti e funzionali: TEI (http://www.tei-c.org/) e DocBook (http://www.docbook.org/). Le linee guida (guideline, i manuali, o specifiche, che ne descrivono i set di elementi e le modalità di impiego) sono liberamente disponibili e consultabili sui relativi siti.*

Grammatiche interne ed esterne

A questo punto le funzioni e i meccanismi delle grammatiche XML o DTD dovrebbero apparire più chiare. Quello che resta da vedere è come associare una grammatica al documento XML che la utilizza.

Esistono due modalità.

- Inserire la grammatica nel file XML.
- Collegare la grammatica al file XML.

Nel primo caso la grammatica è *interna* al documento XML, nel secondo caso *esterna*.

Per inserire un DTD all'interno di in un documento XML è sufficiente collocarlo immediatamente prima del tag root di apertura, in questo modo:

```
<!DOCTYPE libro [
… regole …
]>
<libro>
```

```
… documento XML …
</libro>
```

Il parser che elaborerà il documento avrà in questo modo in un solo file tutte le informazioni per validare o meno il documento.

Questa soluzione è però poco pratica nel caso di DTD e documenti XML lunghi, o in presenza di diversi documenti XML basati sulla stessa grammatica.

In questi casi è preferibile inserire il DTD in un file a parte (`con estensione .dtd`) e collegarlo ai documenti XML tramite un apposito marcatore, anch'esso posizionato prima del tag root di apertura:

```
<!DOCTYPE libro SYSTEM indirizzo_DTD>
```

Ancora una volta viene impiegata la parola chiave `SYSTEM` seguita dall'indirizzo della grammatica che, come per le entità esterne, può risiedere sulla stessa macchina su cui è presente il documento XML o in Rete:

```
<!DOCTYPE libro SYSTEM "esempio.dtd">
```

NOTA *Valgono qui le stesse considerazioni fatte per la sintassi delle entità esterne.*

Il parser potrà in questo modo recuperare il DTD prima di proseguire nell'elaborazione.

Creare un documento XML o un DTD

Fino a questo punto si è parlato di XML e DTD in termini di sintassi e funzionalità. Resta da capire come creare documenti di questo tipo e in che modo utilizzarli.

Scrivere un documento XML o un DTD non necessita di particolari strumenti. Si tratta di documenti di testo quindi qualsiasi editor di testo è adatto. Si può utilizzare Blocco note di Windows o editor professionali specifici dotati di numerose funzionalità utili per semplificare il lavoro, tra cui – per esempio – la possibilità di controllare se un documento è ben formato o valido.

ATTENZIONE *Tra gli editor professionali vale la pena ricordare XMLSpy di Altova (`http://www.altova.com/`) e XMLmind di Pixware (`http://www.xmlmind.com/xmleditor/`). Il primo è a pagamento (è però possibile scaricare una versione demo utilizzabile per 30 giorni), il secondo è gratuito nella versione Personal, non utilizzabile per scopi commerciali e dotato di meno funzionalità rispetto alla versione Professional (a pagamento), ma adatto a chi desidera cominciare a lavorare con XML.*

La Figura 2.1 presenta il documento XML utilizzato come esempio in questo capitolo (con grammatica inclusa), riscritto con Blocco note.

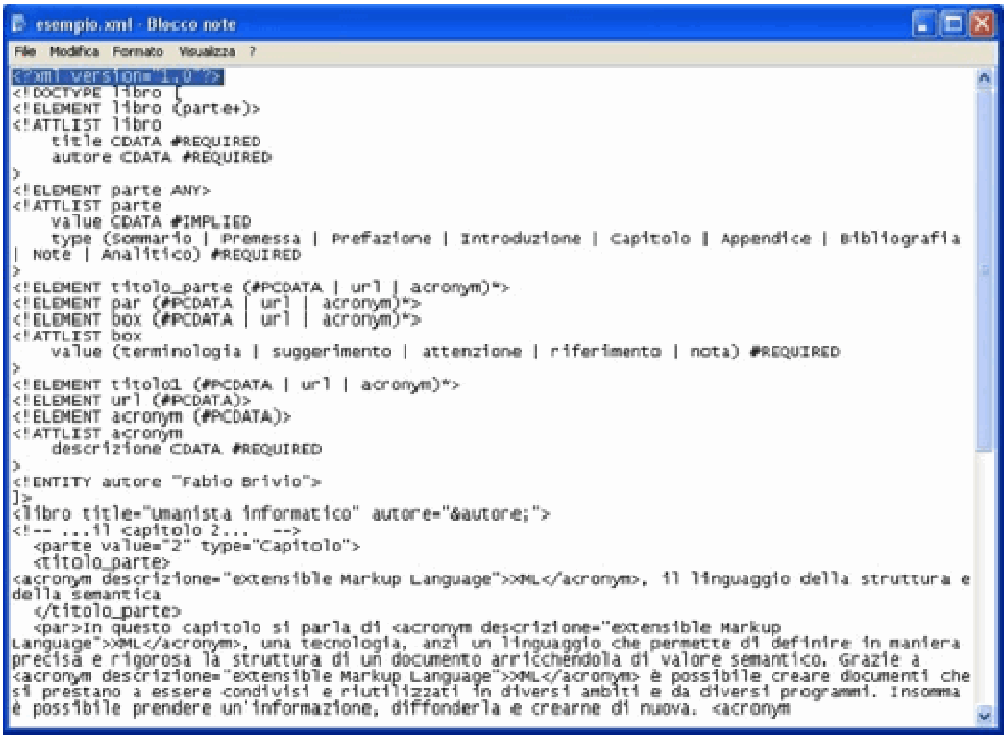


Figura 2.1 L'editor di testo Blocco note permette di scrivere documenti XML.

NOTA *Alla prima riga è stata aggiunta un'istruzione* `<?xml version="1.0"?>` *il cui scopo è quello di comunicare al software che dovrà elaborare il documento a quale versione di XML esso fa riferimento. La presenza di questa istruzione non è obbligatoria, ma è consigliabile soprattutto se si lavora su progetti complessi: in questo modo si riducono le ambiguità di interpretazione.*

Una volta terminata la scrittura, è necessario salvare il documento specificando che si tratta di un file XML, in modo che i software che poi dovranno utilizzarlo siano in grado di riconoscerlo correttamente. L'operazione è molto semplice: è sufficiente selezionare la voce *Salva* dal menu *File* e inserire nella casella *Nome file* della finestra di dialogo *Salva con nome* il nome che si desidera assegnare al documento seguito dall'estensione `.xml`; quindi bisogna specificare *Tutti i file* nella casella *Salva come*, poi selezionare *Unicode* nella casella *Codifica* e infine premere *Salva* (Figura 2.2).

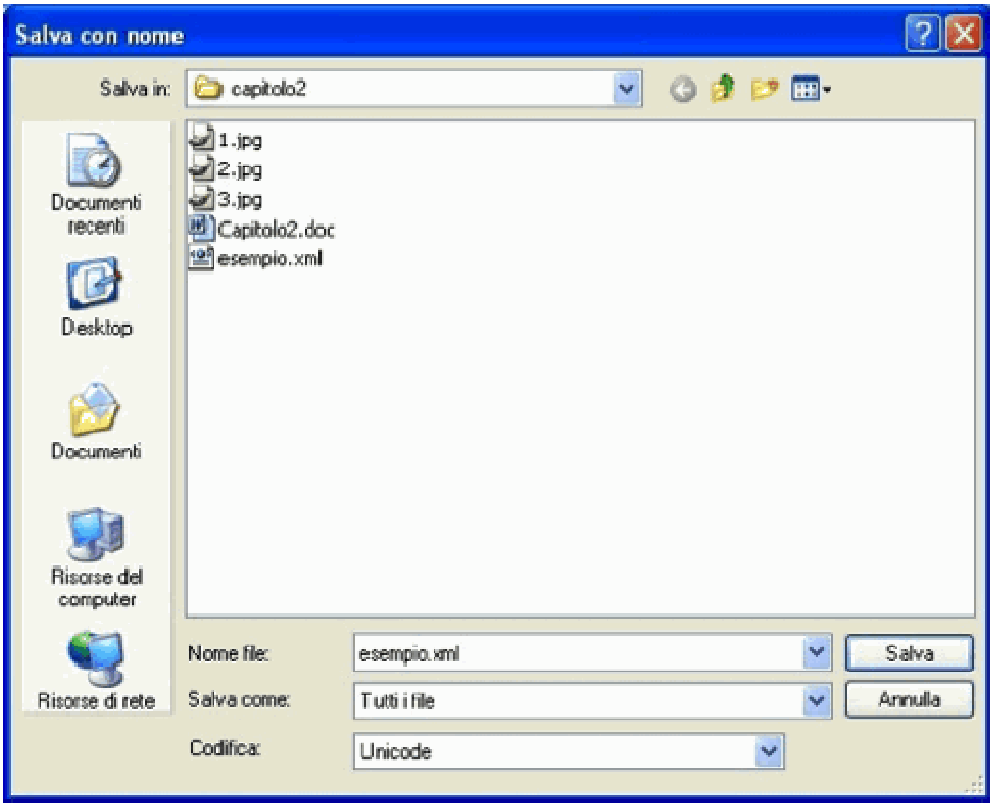


Figura 2.2 Il salvataggio di un documento nel formato XML.

SUGGERIMENTO *Questa procedura vale anche per i DTD. L'unica differenza è che l'estensione da assegnare deve essere in questo caso* `.dtd`.

Il documento viene così salvato nella cartella di destinazione: se l'operazione è andata a buon fine, il sistema operativo associa al nuovo file un'icona caratteristica, in modo che sia distinguibile anche visivamente (Figura 2.3).

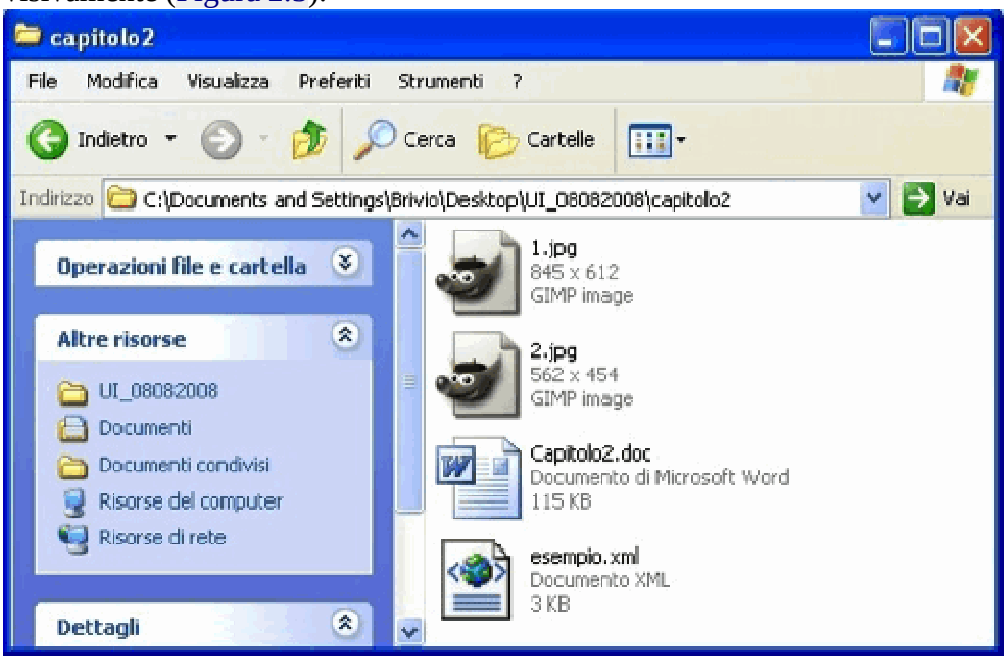


Figura 2.3 Un'icona caratterizza il file XML appena creato.

Interpretare la struttura di un file XML

È interessante e utile visualizzare il contenuto di un file XML attraverso un comune browser, come Internet Explorer o Mozilla Firefox: basta utilizzare la funzione *Apri file* nel menu *File* del browser o semplicemente trascinare l'icona del documento XML in una finestra del browser.

TERMINOLOGIA *Il* browser *è il programma che normalmente si utilizza per navigare tra siti e pagine web. Un browser permette però di visualizzare anche altre tipologie di documenti come XML, file di immagini (JPG, GIF, PNG) o file di testo (TXT o PDF). In pratica può svolgere le funzioni di un reader, permettendo la visualizzazione di un contenuto ma non la modifica. Di browser e documenti web si parlerà meglio nel Capitolo 3.*

Il browser permette di visualizzare la struttura di un documento XML evidenziandone la gerarchia (Figura 2.4).

NOTA *Si può notare come il browser, elaborando il documento, abbia correttamente riconosciuto l'entità* `&autore;`*, sostituendola con il relativo valore nell'attributo* `autore` *del tag* `libro`*.*

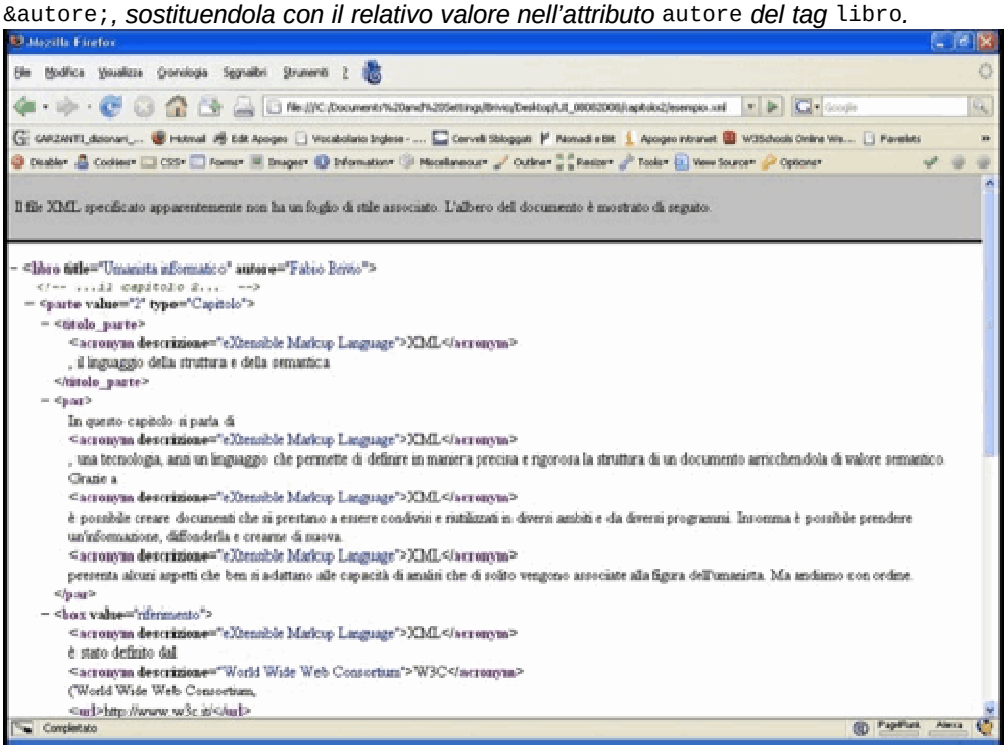


Figura 2.4 Un file XML aperto con il browser Mozilla Firefox.

SUGGERIMENTO *Il browser mostra il documento XML ma non il DTD, in quanto è stato programmato per comportarsi in questo modo in presenza di documenti con estensione* `.xml`*. Il browser quindi non si limita a mostrare il documento nella sua struttura, ma ne offre una prima, grezza, presentazione. Sulla presentazione delle informazioni in relazione alle pagine web si tornerà nel Capitolo 3.*

ATTENZIONE *Per visualizzare la reale struttura di un documento attraverso un browser è necessario selezionare* Visualizza > Sorgente pagina *(Figura 2.5).*

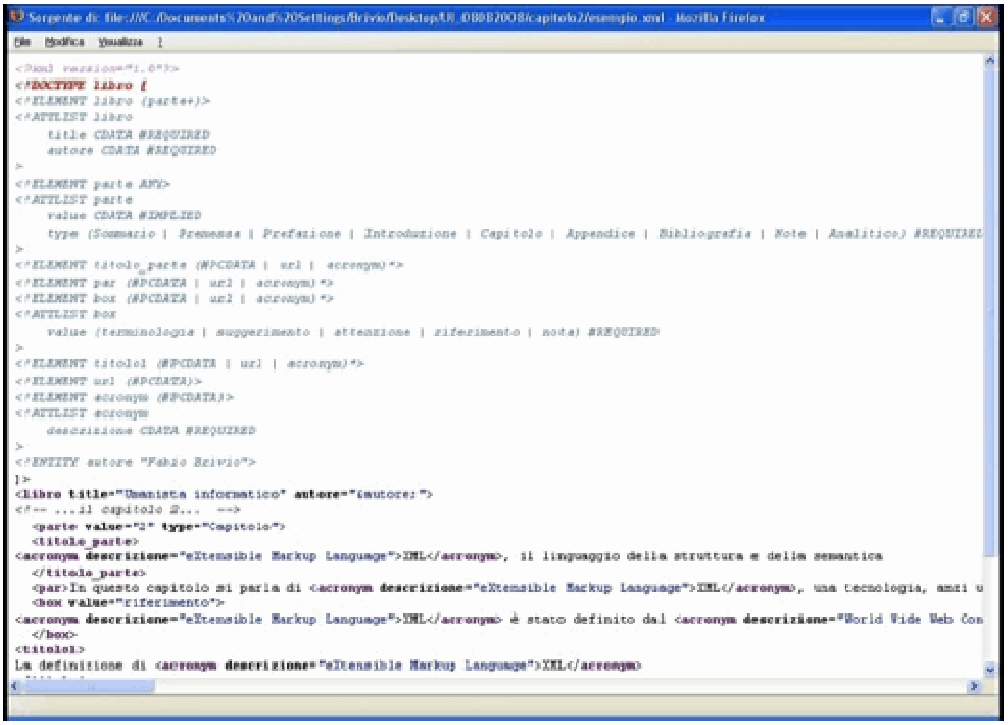


Figura 2.5 La struttura (XML e DTD) del documento visualizzata tramite il browser Firefox.

La struttura dei documenti XML viene spesso descritta utilizzando due metafore: quella dell'albero e quella della famiglia.

Utilizzando la metafora dell'albero, si può immaginare un documento XML come una radice (il tag root) da cui si dipartono vari rami (elementi), ognuno dei quali può articolarsi in altri rami e così via fino ad arrivare alle foglie (informazioni o tag vuoti).

Utilizzando la metafora della famiglia, i vari elementi assumo di volta in volta il valore di genitore (quando un elemento ne contiene altri), figlio (quando un elemento è contenuto in un altro) o fratello (per gli elementi di pari livello gerarchico). Seguendo questa metafora si può dire, sempre facendo riferimento all'esempio utilizzato in questo capitolo, che `<parte>` è genitore di `<titolo_parte>` e allo stesso tempo figlio di `<libro>`; ma anche che `<titolo_parte>`, `<par>`, `<box>` e `<titolo1>` sono tutti fratelli, in quanto presenti nel documento allo stesso livello: essi sono analogamente figli di `<parte>` che per tutti ricopre il ruolo di genitore.

Condivisione e riutilizzo delle informazioni

All'inizio di questo capitolo abbiamo detto che le informazioni organizzate in documenti XML sono facilmente condivisibili. Questa possibilità è garantita da due fattori.

Per prima cosa, XML è ormai uno standard nel mondo della comunicazione digitale e pertanto non esiste sistema operativo che non sia in grado di lavorare con file XML e DTD.

NOTA *Anche i più diffusi software per l'ufficio (come per esempio Microsoft Word o Excel), come la quasi totalità dei programmi applicativi oggi presenti sul mercato, offrono la possibilità di salvare i file in formato XML o di esportare in XML file contenenti metainformazioni (per esempio, il popolare iTunes utilizza XML per esportare i dettagli – artista, titolo brano, album e così via – dei brani musicali inseriti nella Libreria). Per fare questo si basano su DTD appositamente sviluppati.*

In secondo luogo, i file XML e DTD non sono altro che file di testo composti semplicemente da informazione a cui è associata una marcatura. Ne deriva che anche in assenza di editor e reader

specifici è possibile accedere al loro contenuto utilizzando un programma in grado di leggere questo tipo di file, come Blocco note di Windows. Inoltre, i file di testo sono utilizzati e supportati da qualsiasi macchina o sistema operativo e garantiscono accessibilità e fruibilità dei contenuti su qualsiasi computer e in qualsiasi ambiente informatico.

La questione della condivisione delle informazioni codificate in file XML è quindi chiara. Un po' più complessa è invece la problematica inerente il riutilizzo delle informazioni così strutturate per la quale è necessario utilizzare un parser specifico. Ma prima è necessario fare un passo indietro.

RIFERIMENTO *Per quanto riguarda l'utilizzo di XML come formato di scambio dati si faccia riferimento anche al Capitolo 4, paragrafo "Importare ed esportare dati".*

Abbiamo visto come attraverso l'utilizzo di XML e dei suoi marcatori sia possibile concentrare l'attenzione sulla struttura di un documento, lasciando da parte tutto quello che riguarda la sua forma o presentazione.

In un documento XML tutte le parti (titoli, paragrafi, note e così via) sono distinguibili e individuabili grazie al marcatore che li racchiude.

È anche importante sottolineare come niente in un documento XML ne vincoli l'uso, cioè il modo in cui verrà fruito dagli utenti. Per esempio, un documento XML può diventare una pagina di un libro o una pagina web, indifferentemente. Dal punto di vista di XML questo è irrilevante; la rilevanza si manifesta solo quando si decide la forma attraverso la quale l'informazione andrà distribuita.

Per esempio, questo frammento di codice

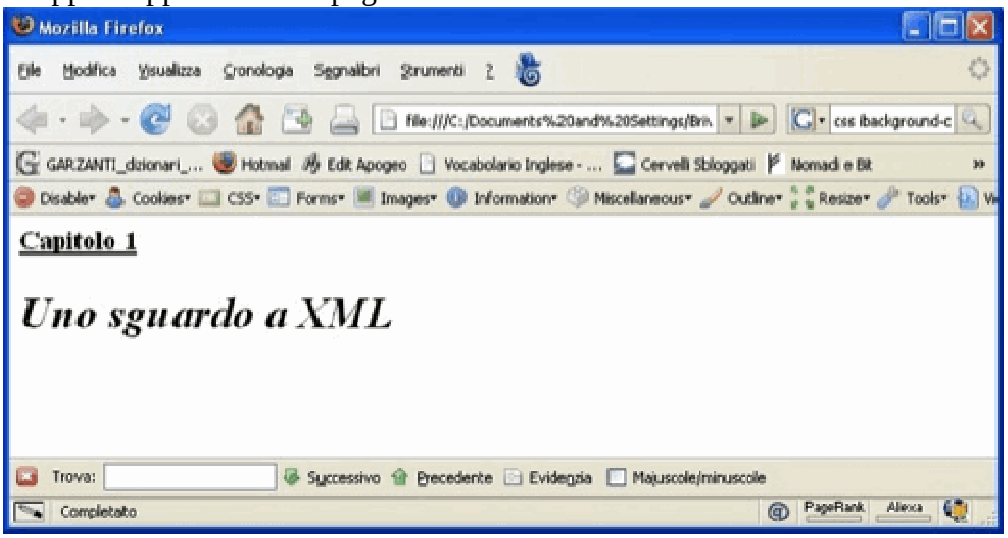
```
<sezione>Capitolo 1</sezione>
<titolo>Uno sguardo a XML</titolo>
```

può diventare parte di una pagina di un libro, con questo aspetto

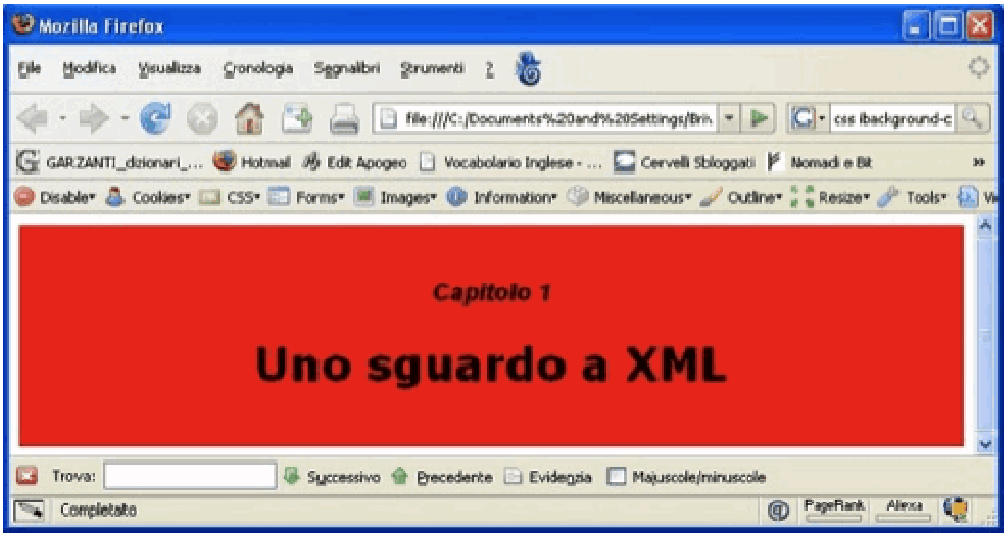
Capitolo 1

Uno sguardo a XML

oppure apparire in una pagina web così



o ancora così



o con ogni font, corpo carattere, colore ed effetto grafico che si riterrà più utile, più adatto o semplicemente più bello. Volendo potrebbe anche essere presentato in maniera adatta allo schermo di un dispositivo mobile come un telefono cellulare o essere convertito in un file sonoro, per esempio in formato MP3.

In ogni caso la struttura XML originaria non subirà modifiche (a meno che queste non siano volute, per esempio per sfruttare l'estensibilità di XML) e a essa si potrà sempre tornare e fare riferimento per studiare nuove applicazioni e utilizzi dell'informazione.

Le possibilità di riutilizzo di un dato una volta che è stato codificato con XML sono in teoria infinite, in pratica legate solo alle necessità di impiego per cui è stato creato e, a un livello più vicino alle macchine, al tipo di parser utilizzabile o disponibile.

Trasformare XML in qualcos'altro

Per utilizzare un'informazione contenuta in un file XML è necessario utilizzare un parser. Un parser è un programma in grado di interpretare ed elaborare i vari elementi di un documento per poi svolgervi determinate operazioni.

In presenza di un documento `libro` come quello presentato in questo capitolo, un parser potrebbe per esempio essere programmato per:

- individuare il tag `parte` e `titolo_parte`;
- estrarre i relativi contenuti testuali (anche quelli degli attributi);
- ricodificare questi contenuti in maniera tale che siano inseribili in una pagina web;
- salvare quanto prodotto in un nuovo file.

TERMINOLOGIA *Nella terminologia informatica, questa sequenza di operazioni è detta* algoritmo *(si veda il Capitolo 5).*

Il risultato finale sarebbe una pagina web contenente un semplice sommario del libro analizzato. Niente dell'originaria struttura XML sarebbe così mantenuto, ma a essa si potrebbe sempre tornare, per esempio per elaborare il contenuto dei capitoli e farne delle anteprime, per il Web o per qualsiasi altro media.

In pratica, il compito di un parser è *processare* l'informazione.

L'utilizzo e la creazione di un parser rientrano nella sfera della programmazione informatica in senso stretto. È possibile processare un documento XML con qualsiasi tipo di linguaggio di

programmazione (come PHP, Perl, Python, C e così via).

RIFERIMENTO *Di programmazione e linguaggi di programmazione si tornerà a parlare nel Capitolo 5.*

Tuttavia è anche possibile fare ricorso a XSL (***Extensible Stylesheet Language***), un linguaggio simile per forma e sintassi a XML che permette di creare file (noti anche come ***fogli stile***) il cui scopo non è però la strutturazione delle informazioni, bensì la definizione delle regole di formattazione da applicare all'informazione.

Un file di questo tipo può essere collegato a un documento XML utilizzando un'istruzione simile a quella vista per i DTD esterni, quindi sarà compito di un parser (questa volta un programma scritto in PHP, Perl o altro linguaggio) interpretare queste istruzioni e fornire il relativo esito, o ***output***.

Così facendo, i momenti della programmazione delle operazioni da svolgere sulle informazioni e della loro effettiva attuazione vengono separati, con non trascurabili vantaggi in termini di semplicità di gestione del flusso di lavoro.

NOTA *XSL permette di lavorare sulle informazioni contenute nei file XML per ottenere documenti di diverso tipo come pagine web, file di testo (*`.txt`*), altri documenti XML o file PDF.*

Per concludere

In questa sede non è possibile affrontare le problematiche connesse alla sintassi e all'utilizzo di XSL: qui è sufficiente prendere confidenza con alcuni concetti inerenti la corretta strutturazione delle informazioni con XML e le fasi del flusso di lavoro che ne consentono la fruizione in ambiti diversi.

Comprendere e valutare correttamente gli scopi per cui un dato è salvato e strutturato è di fondamentale importanza nella società dell'informazione. Errori di valutazione possono compromettere progetti a lungo termine o tradursi in costi non sempre sostenibili. Saper gestire e organizzare dati e contenuti in maniera proficua è quindi una capacità importante.

XML offre una soluzione, ma non l'unica soluzione. Per questo nel Capitolo 4 verranno introdotte le basi di dati.

Nel prossimo capitolo parleremo invece del Web e dei suoi linguaggi. Come apparirà chiaro, la conoscenza della sintassi di XML semplificherà non poco la comprensione di quanto stiamo per vedere.

Capitolo 3

Le lingue del Web

Internet ha radicalmente cambiato il modo in cui le informazioni vengono archiviate, distribuite e fruite. Oggi nessuno può fare a meno della Rete e del Web, e questo è vero soprattutto nel mondo della lavoro. Una quantità indeterminabile di dati è affidata a Internet e finisce sugli schermi dei computer in forma di parole, immagini, suoni e video. Ogni giorno.

La capacità di orientarsi in Rete è quindi imprescindibile in un mondo dove la comunicazione è una necessità vitale.

In questo capitolo parliamo della Rete e soprattutto del Web, introducendone logiche, linguaggi e sintassi.

Ma prima è necessaria una precisazione.

Il Web non è Internet

Molto spesso il termini "Web" (o "WWW") e "Internet" (o "Rete") vengono usati come sinonimi. Niente di più errato.

Internet – "la rete delle reti" – è una grande "ragnatela" di computer sparsi per il mondo e connessi in diversi modi. Le modalità in cui questi computer comunicano sono differenti e basate su diversi protocolli.

Per *protocollo* si intende l'insieme delle regole che guidano lo scambio di informazioni tra macchine diverse. Ogni protocollo è pensato per consentire servizi specifici, quindi esistono, per esempio, protocolli per scambiare messaggi via posta elettronica, trasferire file, comunicare via chat, condividere e scaricare file multimediali (audio e video) e navigare tra ipertesti.

RIFERIMENTI *Un protocollo fondamentale per Internet è il* TCP *(*Transmission Control Protocol*). Senza di esso la Rete non potrebbe esistere come la conosciamo. Ai fini di questo libro non è necessario comprendere il TCP, ma chi volesse saperne di più può cominciare consultando su Wikipedia* `http://it.wikipedia.org/wiki/Transmission_Control_Protocol`.

NOTA *La particolarità di Internet è quella di non avere un centro: la sua struttura è costruita in modo tale che anche in caso di danneggiamento di una sua parte, la Rete rimane attiva e funzionante. Non è cioè possibile "spegnere" Internet, ma è possibile spegnere alcuni dei computer che ne fanno parte, o impedire la comunicazione tra due o più macchine (anche solo limitatamente ad alcuni servizi).*

È proprio per la navigazione tra ipertesti che il Web (o WWW, acronimo di *World Wide Web*) fa il suo ingresso su Internet, dove occupa uno spazio ben preciso: quello degli ipertesti, appunto.

In pratica il Web è una rete di computer sui quali sono presenti documenti ipertestuali. Questi computer comunicano tra loro utilizzando uno specifico protocollo – HTTP (*HyperText Transfer Protocol*, Protocollo di Trasferimento per Ipertesto) – e non sono al di fuori di Internet, ma ne fanno parte a tutti gli effetti.

ATTENZIONE *Nella rete Internet non tutte le macchine ospitano documenti ipertestuali. Paragonare Internet al Web è perciò scorretto: il Web è solo una delle possibilità offerte dalla Rete e non necessariamente una macchina connessa a Internet dà accesso a pagine ipertestuali (dette anche pagine web). È del resto anche vero che oggi Internet e il Web convivono in stretta simbiosi, per cui è facile trovare siti web che consentono di utilizzare le modalità comunicative proprie di altri protocolli. È il caso dei servizi di posta elettronica come Hotmail, Gmail o Yahoo! a cui si accede via Web, ma che in maniera del tutto trasparente per l'utente, utilizzano specifici protocolli per l'invio e la ricezione dei messaggi. E questo è solo un esempio.*

HTTP è quindi l'elemento che garantisce la comunicazione nel Web, ma gli ipertesti non potrebbero esistere in quanto tali senza il cosiddetto *link*, o collegamento ipertestuale, il ponte di congiunzione tra una pagina web e l'altra. E il successo del Web è legato alla semplicità con cui è possibile creare un link, come vedremo più avanti affrontando la sintassi della lingua con cui sono scritte le pagine web.

SUGGERIMENTO *Si potrebbe in un certo senso dire che il Web è un gigantesco ipertesto reticolare dove vige il "diritto a linkare": a prescindere dal punto di partenza, attraverso percorsi inimmaginabili è possibile raggiungere qualsiasi pagina.*

Breve storia del Web

Il Web come lo conosciamo nasce agli inizi degli anni Novanta del secolo scorso, nei laboratori del CERN (*Consiglio Europeo per la Ricerca Nucleare*) di Ginevra ed è legato al nome di Tim Berners-Lee.

Fu lui a sistematizzare il linguaggio HTML grazie al quale ancora oggi sono scritte le pagine web e a implementare la prima comunicazione efficace via HTTP tra due computer. Era il 25 dicembre 1990.

TERMINOLOGIA *In informatica si fa un ampio uso del termine* implementare*, un neologismo derivato dal verbo inglese* to implement*. Con esso si intende l'attività di sviluppo di un programma o di un dispositivo o più semplicemente la realizzazione pratica di un progetto. Per esempio, si può implementare un sito web, un programma per la contabilità o, per rifarsi al Capitolo 2, un DTD dopo che ne sono state definite le caratteristiche logiche.*

Tuttavia concettualmente il Web ha origini più lontane e si lega ad altri due nomi: Vannevar Bush e Ted Nelson.

Il primo, nel 1945, pubblicò un articolo dove teorizzava una macchina (il Memex) in grado di archiviare e catalogare testi e documenti di varia natura in una struttura reticolare. Il suo scopo era la sistematizzazione del sapere, in particolare delle conoscenze scientifiche verso le quali avvertiva una tendenza dispersiva imputabile alla grande quantità di studi e ricerche che alimentavano lo scibile delle varie scienze.

Il secondo coniò il termine "ipertesto" e fu a capo del Progetto Xanadu (1960), il cui scopo era proprio lo sviluppo di ipertesti all'interno di reti di computer.

Tim Berners-Lee ha così raccolto l'eredità e le suggestioni di questi suoi predecessori, trasformandole in realtà.

Agli inizi del XXI secolo ha poi cominciato a diffondersi l'espressione *Web 2.0*. Non si tratta di qualcosa di diverso dal Web di Berners-Lee, bensì di una sua naturale evoluzione in termini di utilizzo e possibilità da parte degli utenti, per i quali è sempre più facile passare dallo stato di fruitori "passivi" di ipertesti, a quello di parte attiva e produttiva del WWW. È il fenomeno dei cosiddetti *user generated content* (i contenuti generati dagli utenti).

TERMINOLOGIA *Nel mondo della programmazione informatica è consuetudine connotare le evoluzioni dei programmi con un numero di* release *(o versione), che ne indica la maturità. Più il numero è alto più il programma è maturo e completo. Le applicazioni in fase di sviluppo hanno di solito una release 0.X, e sono nella cosiddetta fase* beta*, la fase di test. Quando lo sviluppo raggiunge un primo livello di completezza e affidabilità, l'applicazione viene rilasciata come 1.0. L'espressione "Web 2.0" indica così che il World Wide Web ha raggiunto un secondo livello di maturità.*

A questo si deve aggiungere un'esperienza del Web più interattiva e coinvolgente, nella quale i siti

assomigliano per comportamento e funzionalità sempre più alle applicazioni installate sul computer, al punto che non si parla più di sito web ma di *applicazione web*.

RIFERIMENTI *Esempi di Web 2.0 possono essere visti in Google: questo famoso motore di ricerca ha da tempo differenziato la sua offerta arrivando a fornire servizi analoghi a quelli di editor di testi come Word – si veda Google Docs (*http://docs.google.com/*) – o servizi di mappe ricche di interattività (*http://maps.google.it/*). Il tutto rigorosamente online, sul Web.*

TERMINOLOGIA *La crescente potenza di calcolo dei computer e l'offerta sempre più capillare di connessioni a banda larga sono due degli ingredienti su cui ha potuto proliferare il Web 2.0.*

Nel mondo informatico le applicazioni web 2.0 vengono spesso definite come RIA (*Rich Internet Application*). RIA sono quindi applicazioni "ricche", in grado di offrire all'utente una migliore esperienza, gratificandolo con innovative possibilità di interazione, facendogli quasi dimenticare di essere sul Web e non alle prese con un programma installato sul computer.

NOTA *Nello sviluppo di applicazioni web, come di soluzioni tecnologiche in generale, la tendenza è quella di creare prodotti software e hardware per l'utilizzo dei quali non sia necessario apprendere nuovi schemi o procedure. Gli architetti dell'informazione, il cui compito è organizzare le strutture di accesso alle informazioni (una pagina web, i menu di un cellulare, il catalogo di una biblioteca, la planimetria delle corsie di un supermercato e così via), tendono a riprodurre quanto più possibile modalità d'uso di un servizio ormai consolidate, per evitare che l'utente si senta smarrito, per esempio entrando in un nuovo supermercato o accendendo un nuovo cellulare. Per approfondire queste tematiche, un buon testo di partenza è* Architettura dell'informazione *di Luca Rosati, Apogeo, 2007.*

In pratica il Web 2.0 ha mantenuto la promessa intrinseca nel DNA nel Web: scardinare il modello dei media tradizionali dove un emittente "impone" i suoi contenuti a tanti riceventi passivi, creando un ambiente dove tutti possono essere allo stesso tempo emittenti (o meglio editori) e fruitori attivi di contenuti.

RIFERIMENTI *Un libro per comprendere le origini del Web e l'idea di comunicazione libera e democratica di Tim Berners-Lee è* L'architettura del nuovo Web*, scritto dallo stesso Berners-Lee e pubblicato in Italia da Feltrinelli (2001).*

Il Web 2.0 è insomma il "vecchio" Web, semplicemente più potente, più ricco, più interessante e più facile; questo mentre già si comincia a sentir parlare Web 3.0…

L'architettura del Web

Il Web è quindi una rete di computer su cui sono ospitati documenti ipertestuali. La comunicazione tra queste macchine è regolata dal protocollo HTTP.

All'interno della rete i computer assumono poi il ruolo di *client* o di *server*: i client sono i computer (e i relativi programmi) da cui si parte per esplorare il Web, mentre i server sono computer (e i relativi programmi) che ospitano i siti o ogni altro documento disponibile.

SUGGERIMENTO *Usando una metafora, si può vedere una biblioteca (intesa come edificio, personale, scaffali, libri, schedari e così via) come un server a cui gli utenti, cioè i client, possono inoltrare la richiesta di uno o più libri.*

TERMINOLOGIA *Poiché la funzione di un server è ospitare documenti e risorse, questo viene anche detto* host*.*

In pratica succede che un client effettua la richiesta di una pagina web, questa richiesta viene trasferita tramite HTTP a un server che la elabora e, se rileva tra i suoi archivi il documento richiesto, lo restituisce al client sempre via HTTP.

ATTENZIONE *Non solo il Web, ma tutta la rete Internet è basata su un'architettura di tipo* client/server*.*

Sintetizzando, il client fa una domanda e il server risponde. Questa è la logica su cui è basata la

comunicazione in Rete (Figura 3.1).

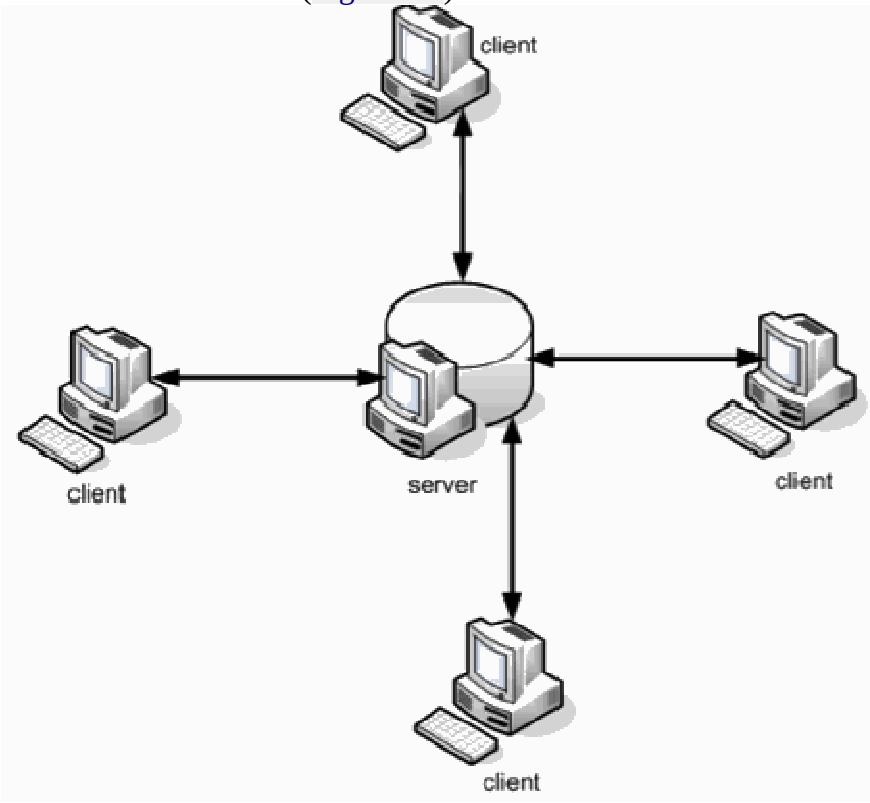


Figura 3.1 Diversi client effettuano richieste a un server che risponde.

ATTENZIONE *Nella terminologia informatica* client *e* server *sono sia le macchine fisiche da cui partono domande e risposte, sia i programmi che permettono di svolgere questa comunicazione.*

Il client web per antonomasia, cioè il programma attraverso cui è possibile interrogare i server e leggerne le risposte, è il *browser* (dal verbo inglese *to browse*, "sfogliare").

Sul mercato esistono diversi browser (tra i più famosi Internet Explorer di Microsoft, Mozilla Firefox, Safari, Opera e Chrome di Google) ma tutti svolgono esattamente la stessa funzione: permettere all'utente di navigare sul Web, tra ipertesti.

NOTA *Programmi server web famosi sono invece Apache HTTP Server e Internet Information Services (IIS) di Microsoft.*

Utilizzare un browser è semplicissimo: si muove il puntatore del mouse su una pagina web aperta e si fa clic su un link per richiedere un nuovo documento al server. In alternativa è anche possibile effettuare direttamente una richiesta, digitando un indirizzo internet nella barra degli indirizzi del browser (Figura 3.2). Proprio qui dobbiamo ora concentrare l'attenzione per cogliere le peculiarità della sintassi degli indirizzi nella comunicazione via Internet.

Figura 3.2 Dettaglio della finestra del browser: in evidenza un indirizzo internet.

Sintassi degli indirizzi

Per prima cosa è necessario introdurre il concetto di URL (*Uniform Resource Locator*) o URI (*Uniform Resource Identifier*): con questi acronimi, ormai usati come sinonimi, ci si riferisce alla

stringa di caratteri che connota in maniera univoca una risorsa in Internet.

URL o URI sono quindi gli indirizzi internet che il browser invia come richiesta al server.

Come ricordato, su Internet sono presenti documenti di vari formati: pagine web, immagini, suoni, video, file PDF, DOC, TXT e così via. A prescindere dal formato, tutti questi documenti possono essere consultati solo effettuando una richiesta sintatticamente corretta.

La struttura sintattica di un URL si compone di varie parti (qui sotto evidenziate in grassetto), alcune delle quali opzionali:

`protocollo://server/file?query_string`

Protocollo

Specifica il protocollo da utilizzare per la richiesta e deve essere seguito dai caratteri "due punti, slash, slash" (`://`). Nell'uso dei browser normalmente il protocollo è HTTP, ma potrebbe anche essere HTTPS o FTP (come vedremo tra poco). Ecco come questi protocolli devono essere espressi: `http://`, `https://`, `ftp://`.

NOTA *Quando il protocollo non viene specificato, i browser completano automaticamente l'URL utilizzando il protocollo HTTP.*

ATTENZIONE *HTTPS (*HyperText Transfer Protocol Secure*) è una variazione del protocollo HTTP sviluppata per consentire comunicazioni protette e sicure tra client e server, quindi difficili da intercettare da terze parti non autorizzate. Questo protocollo viene utilizzato, per esempio, sui server che ospitano applicazioni di e-commerce e che quindi richiedono all'utente di inserire dati come i dettagli della carta di credito.*

Server

Specifica a quale server effettuare la richiesta e normalmente è costituito da un nome di dominio. Un dominio è una sezione logica della rete Internet identificata da un nome e composta da una o più sottoreti. I nomi di domini sono formati da più parti che insieme permettono di identificare in modo univoco un computer connesso a Internet. I domini sono quindi divisi in livelli diversi. Nella sintassi dei nomi di dominio ogni livello è separato dall'altro da un punto (.) e a destra è presente il livello più importante o più alto (Figura 3.3).

Ecco un esempio: `www.apogeoeditore.com/`

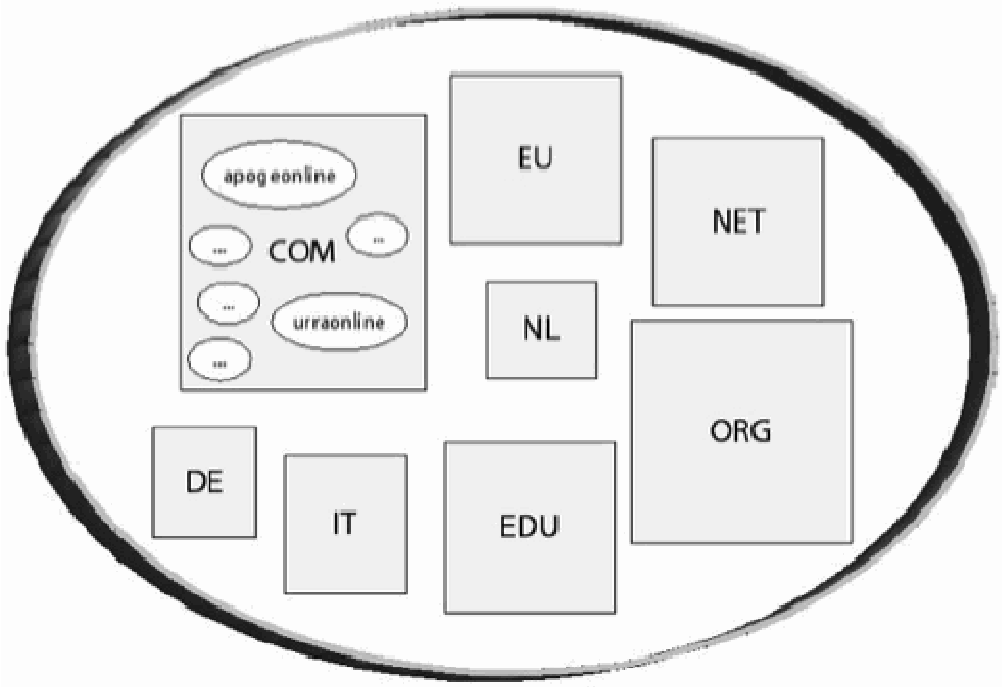


Figura 3.3 Schema della rete Internet: al suo interno i domini di primo livello si articolano in diversi domini di livello inferiore.

Partendo da destra, cioè dal livello più importante, `com` è il dominio di primo livello che indica la sezione della rete sotto cui risiede `apogeoeditore`; `apogeoeditore` è la denominazione vera e propria o dominio di secondo livello, mentre `www` è il prefisso normalmente associato alle risorse disponibili nel Web.

NOTA *Oltre a* `com`*, altri diffusi domini di primo livello sono* `net`, `org` *e i domini nazionali o geografici come per esempio* `it` `nl` `de` *(per Italia, Olanda e Germania) ed* `eu` *per l'Unione Europea.*

ATTENZIONE *In alcuni casi è anche possibile omettere il prefisso* `www` *e venire comunque indirizzati alla pagina ricercata.*

Esistono anche domini di terzo livello: esempi si ritrovano in siti di grandi dimensioni, dove una eventuale localizzazione geografica è spesso indicata attraverso un dominio di terzo livello (come in `it.wikipedia.org`).

Il nome di dominio dovrebbe infine sempre essere fatto seguire dal carattere slash (`/`), anche se questo è rigidamente necessario solo nel caso in cui l'URL continui con i dettagli del file richiesto.

NOTA *Ci si potrebbe chiedere come si possa essere sicuri che, una volta digitato un nome di dominio, la richiesta venga inoltrata alla macchina che effettivamente ospita il sito desiderato: qui basti sapere che l'associazione nome di dominio/macchina è gestita e garantita dai DNS (*Domain Name System*). Si tratta di sistemi che fanno da intermediari tra client e server occupandosi proprio di individuare il server a cui è associato un dato dominio. Semplificando, i DNS sono una sorta di mappa stradale a cui le macchine fanno riferimento per instradare correttamente richieste e risposte.*

File

Si tratta di una parte opzionale e se presente specifica in quale cartella del filesystem del server è presente un dato file o risorsa.

Questa parte dell'URL può essere composta da più elementi, tanti quante sono le cartelle da "percorrere" per arrivare a quella che contiene il file desiderato. Per ogni cartella va indicato il nome esatto (senza tralasciare eventuali lettere maiuscole). Dopo il nome di ogni cartella deve essere

presente uno slash (/):

```
varie/cataloghi/informatica.pdf
```

Nell'esempio qui riportato il file `informatica.pdf` è quindi contenuto all'interno della cartella `cataloghi`, che a sua volta risiede nella cartella varie. Questo percorso viene in gergo chiamato *path* e graficamente può essere rappresentato come nella Figura 3.4.

Figura 3.4 Rappresentazione grafica del path di una risorsa su un server.

NOTA *La tassonomia di un filesystem è stata introdotta nel Capitolo 1.*

ATTENZIONE *La configurazione dei server prevede che essi consentano o meno di visualizzare l'elenco dei file contenuti in una cartella. Se ciò non è impedito, è quindi possibile editare manualmente un path (togliendo per esempio il nome del file) e provare a "curiosare" tra i contenuti delle varie cartelle utilizzando il browser (Figura 3.5). Si tratta solo di un piccolo "trucco", ma questa possibilità mostra come il Web possa essere esplorato in vari modi e non solo seguendo i link presenti in una pagina o affidandosi ai motori di ricerca.*

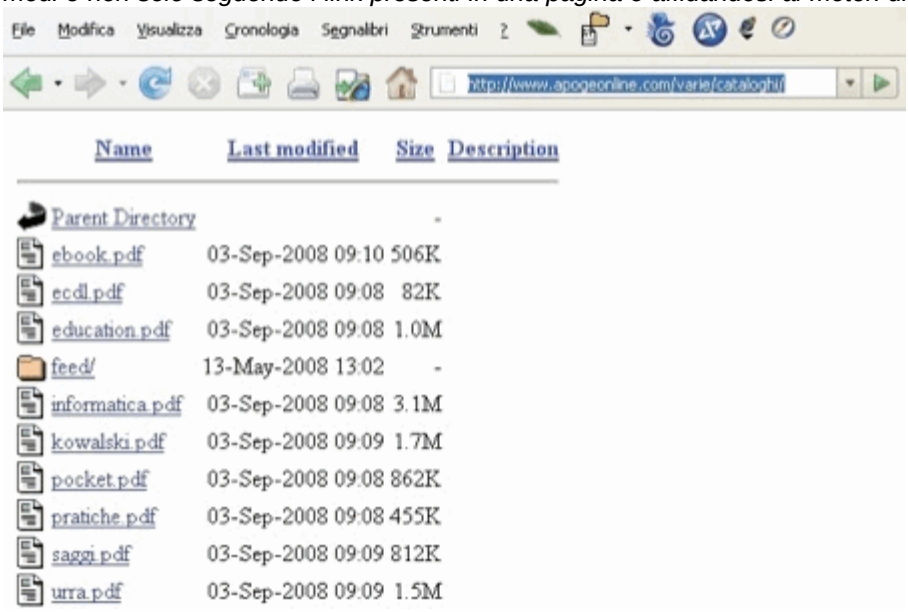


Figura 3.5 Editando un path è a volte possibile visualizzare tutti i contenuti di una cartella.

Come vedremo tra poche pagine, i path vengono utilizzati nella scrittura di documenti HTML per richiamare risorse o creare collegamenti ipertestuali. In quest'ambito, i path possono essere espressi in maniera *assoluta* o *relativa*.

Un path assoluto è un URL completo, nel quale cioè sono espressi sia il nome di dominio (quindi il server), sia il percorso della risorsa.

Un path relativo è invece utilizzato quando la risorsa da richiamare si trova sullo stesso server del documento HTML che lo richiama. In questo caso si omette il nome di dominio e si esprime l'indirizzo della risorsa con una sintassi abbreviata che permette al browser di recuperarla.

Ecco le regole e le precisazioni necessarie per scrivere correttamente dei path relativi.

Per prima cosa bisogna sapere che, in presenza di un path relativo, il browser per recuperare la risorsa prende come punto di riferimento l'URL della cartella in cui si trova la pagina HTML che

richiama la risorsa. In relazione a questa cartella, l'indirizzo relativo gli deve comunicare dove spostarsi, cioè se risalire a una cartella superiore o scendere a una cartella inferiore, o ancora se rimanere nella cartella in cui si trova. Se il path relativo è composto solo dal nome del file (per esempio `catalogo.pdf`), il browser cercherà la risorsa nella cartella in cui si trova.

Se il path relativo è composto dal nome del file preceduto da una o più cartelle separate dal simbolo slash (per esempio `cartella/cartella/catalogo.pdf`), il browser le interpreterà come sottocartelle della cartella in cui si trova, e le seguirà scendendo alla ricerca della risorsa.

Infine, se il path relativo è composto dal nome del file (ed eventualmente da una o più cartelle separate dal simbolo slash) preceduto dai caratteri "punto punto slash" (per esempio `../cartella/cartella/catalogo.pdf`), il browser per prima cosa risalirà di una cartella rispetto alla cartella in cui si trova e da lì si muoverà alla ricerca della risorsa scendendo attraverso le varie sottocartelle indicate.

SUGGERIMENTO *È possibile indicare al browser di risalire due o più cartelle semplicemente ripetendo la sequenza* `../`*. Per esempio* `../../catalogo.pdf` *indica al browser di risalire due cartelle rispetto a quella in cui si trova per recuperare il file* `catalogo.pdf`*.*

Query string

Anche questa parte è opzionale e il suo scopo è quello di passare, oltre alla richiesta di un documento, dei parametri che il server utilizzerà ed elaborerà per fornire il documento stesso.

Per comprendere l'importanza di questi parametri è necessario introdurre il concetto di pagine web *statiche* e *dinamiche*.

Le pagine web statiche sono documenti che esistono in quanto tali e che il server non fa altro che recuperare e restituire al client. Questi documenti sono immutabili, cioè possono essere richiesti da una a un numero indefinito di volte, ma non cambiano mai. Gli esempi più comuni sono i file con estensione `.html` (di cui ci occuperemo tra poco).

Le pagine web dinamiche non esistono effettivamente sui server. Esse sono costruite di volta in volta in base a dei parametri che vengono forniti al server attraverso la query string (traducibile come "stringa di interrogazione"). Ricevuti questi parametri, il server li passa a un programma che li utilizza per creare una pagina *ad hoc*, servita poi al client.

TERMINOLOGIA *Da questo punto di vista ha perfettamente senso riferirsi agli URL che contengono parametri con il termine* query *(interrogazione). È giusto però anche notare che tutti gli URL inviati a un server sono del resto query, in quanto corrispondono a richieste di documenti, statici o dinamici non fa differenza.*

Utilizzi di pagine dinamiche sono molto diffusi e frequenti. Un esempio tipico si trova in tutti i servizi di webmail (come Hotmail, Gmail, Yahoo! e così via). Quando un utente si registra per accedere alla casella di posta elettronica, i parametri *nome utente* e *password* vengono passati al server che li elabora in modo da restituire una pagina dove sono presenti solo i messaggi di quel dato utente. La stessa cosa succede in tutti i siti dove è richiesta una registrazione ma anche in siti di grosse dimensioni, come quelli di librerie o prenotazione di voli aerei, dove prima di effettuare o meno un acquisto è possibile effettuare ricerche: in questi casi i parametri possono essere il titolo del libro o l'aeroporto e le date di partenza e arrivo.

Le pagine dinamiche possono avere estensioni di vario tipo, in base al tipo di linguaggio con cui è scritto il programma utilizzato per elaborare i parametri e restituire la pagina richiesta.

RIFERIMENTI *Nel Web, uno dei linguaggi più diffusi è PHP e i file della pagine dinamiche così creati hanno*

estensione `.php`*. Questo linguaggio verrà introdotto nel Capitolo 5.*

I parametri della query string servono quindi per la generazione delle pagine web dinamiche. Non esistono limiti al numero dei parametri ma anche per essi è prevista una specifica sintassi.

Per prima cosa la query string deve essere introdotta dal simbolo ?. Ogni parametro deve quindi essere composto da una coppia di elementi costituita dal nome del parametro e dal valore associato (utilizzando il simbolo = per l'associazione). Infine ogni parametro deve essere separato dal simbolo &. Per cui una query string così composta:

```
?nome_parametro1=valore1&nome_parametro2=valore2
```

permette di passare al server due parametri con i rispettivi valori.

Possiamo vedere un esempio pratico di quanto fin qui illustrato nella Figura 3.6, che mostra la sezione di una pagina dei risultati fornita da Google dopo aver immesso come termini di ricerca **fabio brivio libri**.

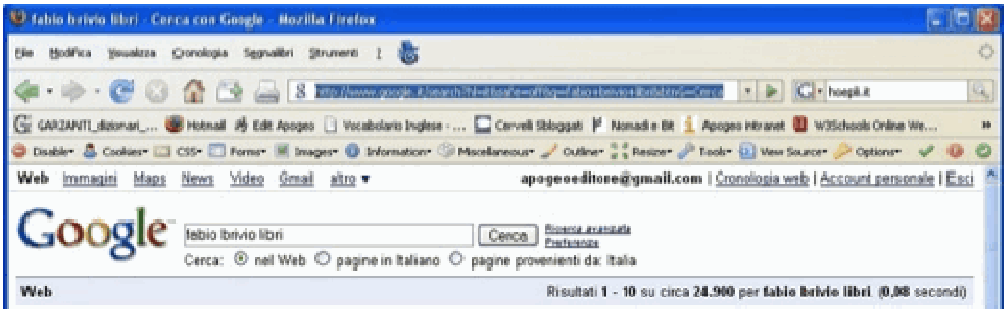


Figura 3.6 Esempio di URL con query string.

TERMINOLOGIA *Normalmente i valori dei parametri vengono raccolti attraverso campi, caselle e aree di testo, e inviati al server premendo un apposito pulsante. Come si vedrà tra poche pagine, l'insieme di questi elementi viene detto* form*.*

Ecco l'URL della pagina i cui elementi ora dovrebbero essere più comprensibili:

`http://www.google.it/search`**`?hl=it&safe=off&q=fabio+brivio+libri&btnG=Cerca`**

La parte a destra del punto interrogativo contiene quattro parametri e con un po' di attenzione non è difficile capire di che tipo di informazioni si tratta e il loro scopo. Vediamole coppia per coppia.

- `hl=it:` qui la presenza della particella `it` è abbastanza chiarificatrice, questo parametro passa infatti la lingua di riferimento che verrà utilizzata per elaborare la risposta da inviare al client.
- `safe=off:` questo parametro è abbastanza esplicito, "safe" (sicurezza) è impostato su "off" (spento); è quindi ragionevole pensare che nell'elaborazione della risposta non verranno prese misure di sicurezza di alcun tipo.

NOTA *Per quanto riguarda Google,* safe *permette di attivare (valore* `on`*) o meno un filtro (SafeSearch) per i contenuti pornografici o offensivi.*

- `q=fabio+brivio+libri:` anche questo parametro è di facile comprensione, si tratta dei termini da ricercare che vengono passati al server associati a un parametro di nome q scelto probabilmente dai programmatori di Google come abbreviazione di "query".

NOTA *L'Appendice di questo libro è dedicata all'utilizzo di Google e alla sintassi delle query nei motori di ricerca.*

- `btnG=Cerca:` questo parametro indica il pulsante utilizzato per lanciare la ricerca, in questo caso *Cerca*; questo appare più evidente se si considera btnG come la contrazione di

"buttonGoogle".

NOTA *Lanciando la ricerca con un altro pulsante la risposta sarebbe stata differente; per esempio con il pulsante* Mi sento fortunato *il server di Google non risponde con una pagina dei risultati, ma direttamente con la pagina del primo risultato valido per Google.*

I parametri delle query string vanno osservati con attenzione in quanto possono essere editati direttamente nell'URL: con un po' di esperienza e di fortuna è in questo modo spesso possibile semplificare e gestire con maggiore precisione l'invio delle richieste al server.

ATTENZIONE *Poiché editando a mano i valori dei parametri è possibile ottenere dal server risposte inaspettate e non previste dagli stessi programmatori (a volte anche critiche per la sicurezza di un sito), nello sviluppo web è possibile specificare se la query string possa essere o meno visibile all'interno degli URL. Per fare questo esistono due modalità di invio dei parametri al server:* `get` *e* `post`*. Il primo permette la visualizzazione dei parametri nell'URL, il secondo la vieta. Su di essi torneremo più avanti quando affronteremo i form HTML.*

Sintassi delle richieste FTP

Come abbiamo ricordato, utilizzando un browser normalmente il protocollo di riferimento è HTTP. Tuttavia alcuni browser (come Internet Explorer di Microsoft) possono essere utilizzati anche per gestire la comunicazione tra client e server con altri protocolli, tra cui FTP (*File Transfer Protocol*, protocollo di trasferimento file).

Questo protocollo è particolarmente utile e diffuso in quanto consente di trasferire con facilità file di medie e grosse dimensioni tra computer diversi.

Un client, per comunicare via FTP con un server, utilizza di solito un programma specifico che permette di visualizzare e navigare nella struttura del filesystem sia del computer client, sia del computer server, e quindi di scegliere da quale e in quale cartella trasferire i file (Figura 3.7).

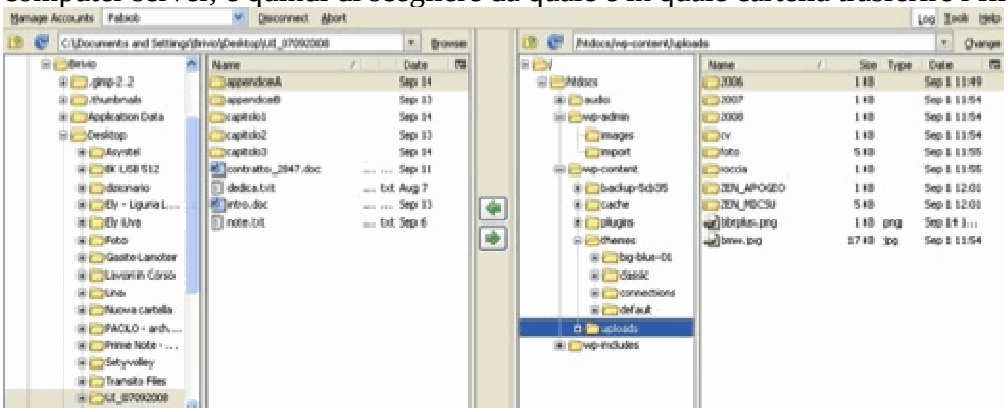


Figura 3.7 Interfaccia di un programma client FTP: i filesystem dei computer client e del server sono visibili insieme.

Come ricordato, la comunicazione client/server via FTP è spesso possibile anche utilizzando un browser nella cui barra degli indirizzi deve essere digitato un URL con una sintassi molto simile a quella fin qui vista per le richieste HTTP:

```
protocollo://utente:password@server
```

Per il protocollo vale quanto precedentemente detto. In questo caso però esso sarà `ftp://` e dovrà sempre essere espresso: diversamente, il browser invierà l'URL come una richiesta HTTP, producendo un errore.

Dopo il protocollo va inserita la coppia utente e password. Per ragioni di sicurezza, l'accesso via FTP ai server non è infatti pubblico e pertanto sono necessarie delle credenziali che vengono fornite

dall'amministratore del server stesso. Utente e password vanno inseriti separati dal simbolo : e vanno fatti seguire dal simbolo @ (la chiocciola, nota anche come "at"). La chiocciola introduce il server, o host, da contattare.

ATTENZIONE *Per i server FTP vale quanto detto per i server HTTP. In questo caso però non si incontra mai il prefisso* `www`*, ma a volte è presente il prefisso* `ftp`*.*

Una richiesta come quella che segue

`ftp://fabio:86321@ftp.fabiob.eu`

inviata tramite browser, si potrebbe pertanto parafrasare così: "Apri una connessione via FTP, presso il server `ftp.fabiob.eu`, per l'utente `fabio` con password `86321`".

Tuttavia esprimere in un URL dati sensibili con utente e soprattutto password non è consigliabile per motivi di sicurezza. Per questo è possibile tralasciare questi dati inviando al server una semplice richiesta di connessione, così:

`ftp://ftp.fabiob.eu`

Una volta contattato, il server risponderà richiedendo le credenziali di accesso che potranno essere questa volta inserite e inviate in sicurezza tramite appositi campi (Figura 3.8).

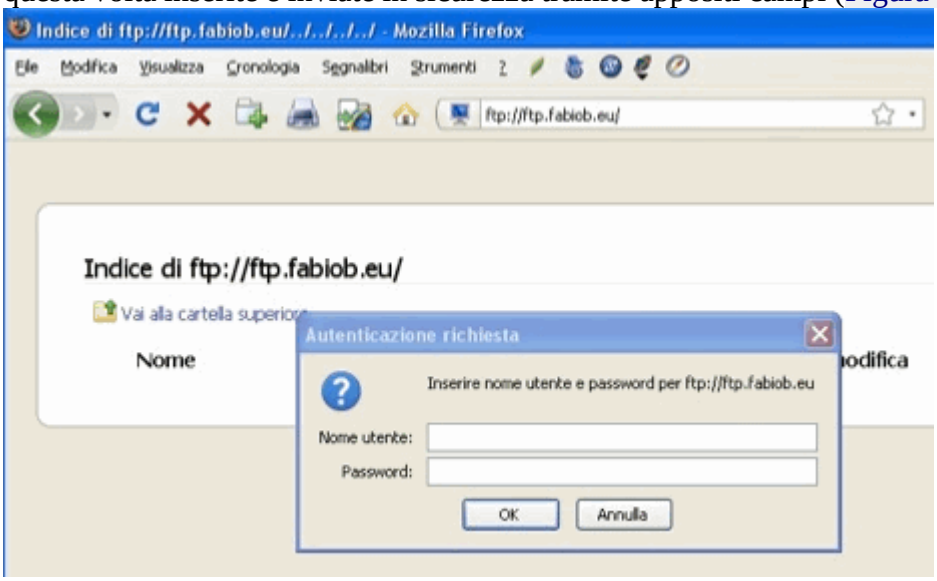


Figura 3.8 Apertura sicura di una connessione FTP tramite browser.

I linguaggi delle pagine web: la struttura e la presentazione

Nelle pagine precedenti abbiamo visto le particolarità e la sintassi della comunicazione in Rete. È giunto ora il momento di affrontare i linguaggi (e la relativa sintassi) con cui è scritta la maggior parte dei documenti web. Nei prossimi paragrafi parleremo di HTML e CSS: il primo permette di strutturare i contenuti delle pagine web, mentre il secondo consente di definire l'aspetto con cui questi contenuti verranno presentati sullo schermo di un computer, ma non solo.

ATTENZIONE *Per sperimentare quanto presenteremo nelle prossime pagine e quindi per scrivere e salvare documenti HTML o CSS, è sufficiente avere a disposizione un normale editor di testo, per esempio Blocco note di Windows. Le modalità di salvataggio sono le stesse già viste per XML, con la differenza che l'estensione da specificare deve essere diversa:* `.html` *o* `.css`*. (nella casella* Codifica *non è inoltre necessario specificare* Unicode*, ma può essere lasciato il valore di default* ANSI*). Una volta salvata sul computer, la pagina HTML può essere subito testata visualizzandola con un browser: il documento verrà rappresentato correttamente e integralmente ma, in assenza di una connessione a Internet, tutti i riferimenti esterni (link e immagini) non saranno funzionanti. Per mettere online (cioè per pubblicare sul Web) i documenti così creati sarà invece necessario*

caricare i file su un server, preferibilmente attraverso un accesso FTP.

La lingua della struttura del Web

Potrebbe apparire strano ma, sebbene sia in continua evoluzione, il Web da sempre parla, o meglio si sviluppa, attorno a un solo linguaggio: HTML (*HyperText Markup Language*, Linguaggio di Marcatura per Ipertesti).

Sviluppata da Tim Berners-Lee agli inizi del 1990, questa tecnologia ha subìto diverse evoluzioni nel corso degli anni, assestandosi però nel 1999 su quella che viene definita la specifica HTML 4.01, ovvero la grammatica di riferimento per chi vuole scrivere documenti da pubblicare nel World Wide Web.

RIFERIMENTI *La versione originale inglese di questo importante documento è visibile sul sito del W3C, all'indirizzo* `http://www.w3.org/TR/ html401/`*. Una traduzione italiana è invece disponibile all'indirizzo* `http://www.diodati.org/w3c/html401/cover.html`.

HTML è quindi un linguaggio di marcatura e in quanto tale è per molti aspetti simile a XML, da cui però si discosta per due importanti motivi.

- HTML è composto da un insieme già definito di tag o marcatori (corredati di attributi), ideali a soddisfare la maggior parte delle esigenze di descrizione di un testo per il Web. Tutti i tag ammessi da HTML sono elencati nella specifica 4.01.
- Sebbene nella scrittura di codice HTML sia auspicabile il rispetto delle regole sintattiche viste per XML (la chiusura di ogni tag e il corretto annidamento degli elementi), è possibile scrivere documenti HTML anche con errori di forma, in quanto i browser sono molto "permissivi" verso questo tipo di violazioni e riescono il più delle volte a compensare eventuali errori presenti nel codice di una pagina web.

NOTA *La tolleranza dei browser è uno dei motivi che ha consentito – e che consente – anche agli autori meno esperti, o meno attenti, di pubblicare con successo pagine e documenti sul Web.*

ATTENZIONE *I browser riuniscono due funzionalità estremamente importanti: con un browser è possibile navigare, e "sfogliare", le cartelle di un computer visualizzando diverse tipologie di contenuti (file PDF, immagini, documenti XML e così via), ma soprattutto il browser interpreta (o meglio processa) le pagine web (vale a dire il codice HTML e CSS ricevuto dal server) per mostrarle in maniera leggibile all'utente. La resa visiva di una pagina può variare da browser a browser a seconda del* motore di rendering *utilizzato. Il motore di rendering è il cuore del browser: si tratta di un parser il cui scopo è quello di renderizzare, cioè tradurre in output, le istruzioni scritte in linguaggio HTML e CSS.*

XML + HTML = XHTML

Come appena ricordato, è possibile scrivere codice HTML con errori di forma. In questo modo si creano però documenti il cui riutilizzo in altri ambiti potrebbe essere difficoltoso. La possibilità di riutilizzo di un contenuto è invece intrinseca nei documenti correttamente scritti in XML, cioè ben formati e validi.

Per garantire la massima condivisione dei documenti web, il W3C ha così sviluppato una variante di HTML in XML: XHTML (*eXtensible Hypertext Markup Language*).

XHTML è in pratica la grammatica HTML ripensata nel completo rispetto delle regole sintattiche di XML: tra HTML e XHTML non esistono quindi grosse differenze per quanto riguarda i tag utilizzabili, il set di elementi è quasi il medesimo, ma in XHTML essi devono essere necessariamente scritti senza errori di forma.

NOTA *XHTML non permette l'uso di tag stilistici o di presentazione, delegando la definizione dello stile di una*

pagina web esclusivamente ai CSS. Sui tag HTML torneremo nel prossimo paragrafo.

In questo modo i documenti HTML sono processabili al pari di tutti gli altri documenti XML e di conseguenza riutilizzabili ed estensibili, con indubbi guadagni in termini di possibilità di condivisione delle informazioni.

RIFERIMENTI *La specifica XHTML è consultabile presso l'indirizzo internet* `http://www.w3.org/TR/xhtml1/`. *Esiste anche una traduzione italiana all'indirizzo* `http://www.w3c.it/traduzioni/xhtml1-it.html`.

Grammatica HTML in pillole

È giunto il momento di introdurre un sottoinsieme di tag e attributi HTML la cui conoscenza permette di scrivere pagine web strutturate e complete, pronte per essere pubblicate. Come vedremo la scrittura di codice HTML non presenta grosse difficoltà.

Gli elementi HTML possono essere sommariamente divisi in tre categorie: semantici (predisposti a dare informazioni sul tipo di contenuto descritto come il titolo, la lingua, l'autore e così via), strutturali (in grado di specificare dove comincia e finisce un titolo, un paragrafo, una lista e così via) e stilistici (dedicati alla definizione dello stile da applicare a una parte del testo, come grassetto, corsivo, sottolineato e così via).

ATTENZIONE *È preferibile utilizzare il meno possibile i marcatori stilistici, delegando le regole di questo tipo ai CSS, su cui torneremo più avanti.*

Un seconda divisione, importante, degli elementi HTML è tra elementi di *blocco* e *in linea*. L'inserimento dei primi provoca un'interruzione nel flusso di un documento, per cui l'elemento che segue un elemento di blocco viene visualizzato andando a capo, su una nuova riga. L'inserimento dei secondi non provoca invece nessuna interruzione nel flusso di un documento, quindi dopo un elemento in linea può essere presente, sulla stessa riga, un altro elemento.

Un'altra differenza tra queste due tipologie di elementi è relativa al contenuto: entrambi possono contenere testo, ma un elemento di blocco può contenere anche altri elementi, sia di blocco sia in linea, mentre un elemento in linea può contenere solo altri elementi in linea.

NOTA *La comprensione delle differenze tra elementi di blocco e in linea è fondamentale per la definizione dell'aspetto di una pagina web con i CSS.*

Un esempio tipico di elemento HTML di blocco è il tag `<p>`, usato per marcare i paragrafi. La Figura 3.9 mostra come un browser interpreta e visualizza questo frammento di codice:

```
<p>Paragrafo 1</p><p>Paragrafo 2</p><p>Paragrafo 3</p>
```

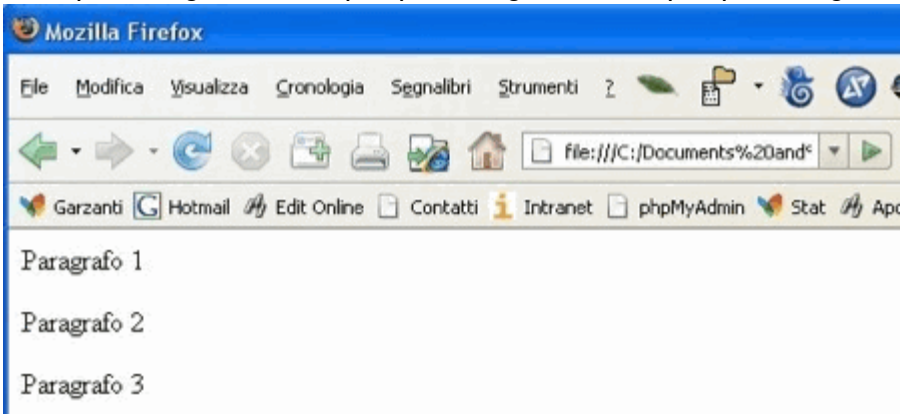


Figura 3.9 Gli elementi di blocco provocano interruzioni nel flusso di un documento HTML.

Per ogni paragrafo viene creata una nuova riga, a prescindere dalla quantità di testo contenuto.

La Figura 3.10 mostra invece il comportamento di un elemento in linea, il tag stilistico `<b>` (il cui

scopo è marcare in grassetto porzioni di testo), inserito all'interno di un elemento di blocco:

```
<p>In questo paragrafo è presente un elemento <b>in
linea</b> evidenziato in grassetto.</p><p>Paragrafo
2</p>
```

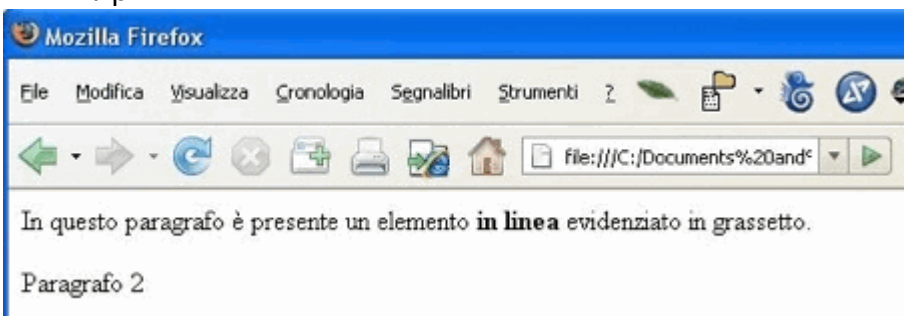


Figura 3.10 Gli elementi in linea non influenzano il flusso di un documento HTML.

La presenza dell'elemento `<b>` non ha il minimo effetto sul flusso del primo paragrafo, che si interrompe, creando un blocco e quindi una nuova riga, solo con il tag di chiusura `</p>`.

NOTA *Esiste anche una terza tipologia di elementi, quella degli elementi* rimpiazzati *che incontreremo più avanti, parlando degli elementi HTML per la gestione delle immagini.*

Possiamo ora passare in visione il set di tag – e i più comuni attributi – con cui è scritta la maggior parte delle pagine web.

NOTA *La sintassi degli attributi HTML è analoga a quella degli attributi XML:* `nome_attributo="valore_attributo"`*. Lo stesso vale per i commenti che in HTML ricoprono la stessa funzione svolta in XML: inserire nel codice informazioni che non verranno interpretate dal parser. Un commento HTML va quindi racchiuso tra i marcatori* `<!--` *e* `-->`*.*

Intestazione e corpo

Una pagina web si compone di due parti fondamentali: un'intestazione, marcata dal tag `<head>`, e un corpo, marcato dal tag `<body>`. Questi due elementi sono compresi a loro volta nel tag `<html>` che indica l'inizio e la fine di un documento HTML:

```
<html>
    <head>
    …
    </head>
    <body>
    …
    </body>
 </html>
```

L'intestazione contiene il titolo del documento e alcune informazioni semantiche. Il tag più importante, che non dovrebbe mai mancare, è `<title>` che marca appunto il titolo del documento:

```
  <head>
   <title>Titolo del documento HTML</title>
 </head>
```

Il contenuto del tag `<title>` non viene visualizzato dai browser nel corpo della pagina, ma nella barra del titolo del browser: `<title>` è infatti il titolo del documento HTML, differente dagli altri titoli che possono essere presenti nel corpo della pagina (Figura 3.11).

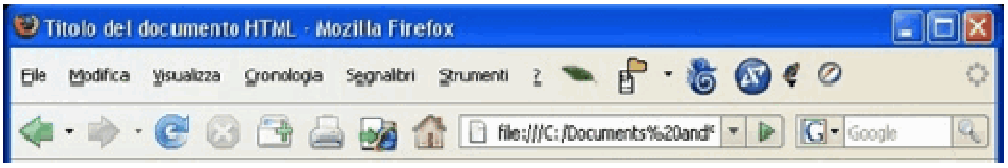


Figura 3.11 Il titolo di una pagina web visualizzato da un browser.

ATTENZIONE *Il contenuto di* `<title>` *è una delle prime cose che gli spider dei motori di ricerca notano (si veda l'Appendice di questo libro) e registrano e per questo dovrebbe essere compilato nella maniera più chiara e precisa possibile; un buon titolo deve comunicare esattamente il contenuto di un documento web.*

Sempre nell'intestazione possono poi essere presenti (in numero variabile) due elementi vuoti, cioè privi di contenuto testuale ma accompagnati da attributi: `<link>` e `<meta>`. Torneremo sul tag `<link>` più avanti, affrontando i CSS, mentre per quanto riguarda il tag `<meta>` è qui sufficiente sapere che il suo scopo è fornire delle informazioni sul documento ai parser che analizzeranno la pagina (come l'autore, delle parole chiave, una descrizione breve o la lingua di riferimento). Queste informazioni non vengono quindi visualizzate dal browser e sono del tutto trasparenti agli utenti.

TERMINOLOGIA *I tag* `<meta>` *contengono quindi le informazioni* sul *documento, non le informazioni* del *documento. Si tratta a tutti gli effetti di informazioni su informazioni, vale a dire metainformazioni.*

Possiamo quindi passare ai tag che compongono il corpo di una pagina. Li affronteremo in relazione allo scopo, partendo dai tag per marcare i titoli.

Titoli

In HTML i titoli possono essere marcati attraverso sei marcatori: `<h1>` `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`. Il numero ne indica il peso: 1 per i titoli di primo livello, 6 per quelli di sesto. Il livello 1 è quello di maggior rilevanza, il titolo principale, dove dovrebbe essere dichiarato senza ambiguità l'argomento della pagina.

SUGGERIMENTO *Non ci sono limiti al numero di titoli inseribili in un documento, ma è consigliabile avere un solo titolo* `<h1>`*: in questo modo sia gli utenti, sia i motori di ricerca potranno comprendere meglio il contenuto della pagina.*

La Figura 3.12 mostra come un browser rappresenta questo frammento di codice:

```
<body>
    <h1>Titolo 1</h1>
    <h2>Titolo 2</h2>
    <h3>Titolo 3</h3>
    <h4>Titolo 4</h4>
    <h5>Titolo 5</h5>
    <h6>Titolo 6</h6>
</body>
```

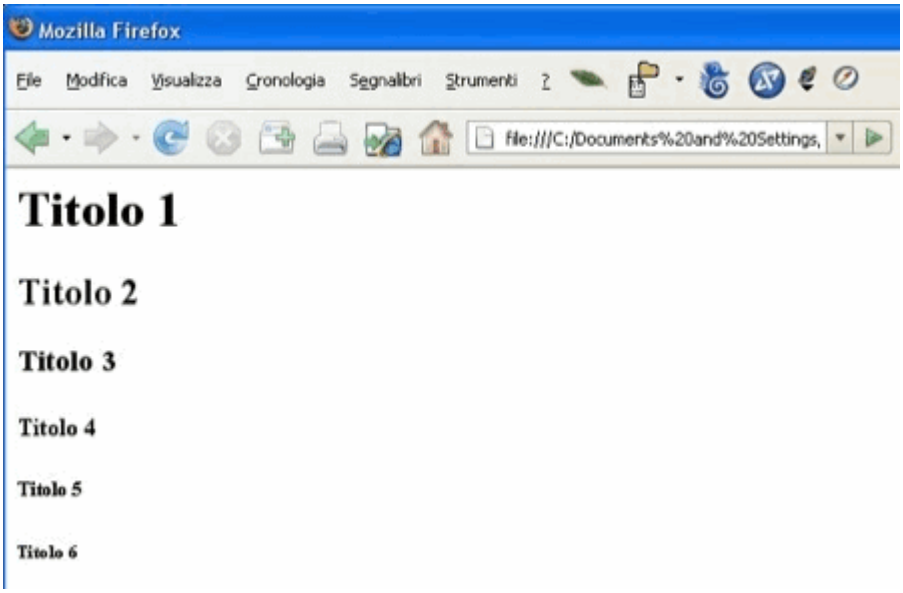


Figura 3.12 I titoli in HTML.

Tutti i titoli sono elementi di blocco.

Paragrafi e sezioni di testo

Abbiamo già incontrato il tag `<p>`, il cui scopo è marcare singoli paragrafi; qui introduciamo anche il tag `<div>` fondamentale in quanto permette di articolare un documento in sezioni.

A differenza di altri elementi (per esempio lo stesso `<p>` o i titoli), `<div>` non ha alcun valore semantico preciso (infatti i browser rappresentano i testi marcati con `<div>` in maniera analoga a quelli marcati con `<p>`): esso semplicemente racchiude una sezione di testo ed essendo `<div>` un elemento di blocco può contenere qualsiasi altro elemento, di blocco o in linea.

L'uso di questo elemento è molto frequente con gli attributi id o class che in HTML sono usati per aggiungere valore semantico a un elemento.

ATTENZIONE *In una pagina è possibile avere più elementi con lo stesso valore di* `class`*, ma un solo elemento per ogni valore di* `id`*. La funzione di* `id` *è infatti connotare in maniera univoca un elemento, mentre la funzione di* `class` *è connotare in maniera comune un insieme di elementi. Per aggiungere un medesimo valore semantico a più elementi di una pagina è quindi necessario usare l'attributo* `class`*. Come vedremo affrontando i CSS, l'uso degli attributi* `id` *e* `class` *è fondamentale per la presentazione di una pagina web.*

L'importanza di `<div>` è in relazione alla limitata possibilità di descrizione di HTML che permette di connotare le parti di un documento solo per la loro funzione (quindi come titoli, paragrafi, elenchi e così via). Grazie a questo elemento una pagina web può essere strutturata da un punto di vista semantico in maniera più profonda e coerente.

Ecco un esempio di utilizzo di `<div>`:

```
<div id="Libro">
  <p id="autore">Fabio Brivio</p>
  <p id="titolo">l'Umanista Informatico</p>
  <p class="capitolo">Capitolo 1 L'informazione tra
forma e formattazione</p>
  <p class="capitolo">Capitolo 2 La lingua della
struttura e della semantica</p>
  <p class="capitolo">Capitolo 3 Le lingue del Web
</p>
```

```
    <p class="capitolo">…</p>
  </div>
```

Come si vede, la sezione `Libro` contiene diversi paragrafi ma id viene usato una sola volta per tipo di valore: "autore" e "titolo" sono infatti unici per tutto un libro. La coppia `class="capitolo"` si ripete invece più volte, in quanto i capitoli di un libro sono diversi ma possono essere accomunati in un'unica classe.

La Figura 3.13 mostra la visualizzazione di questo codice in un browser: in assenza di informazioni di presentazione la presenza dell'elemento `<div>` non ha apparentemente alcuna conseguenza sulla resa a video.

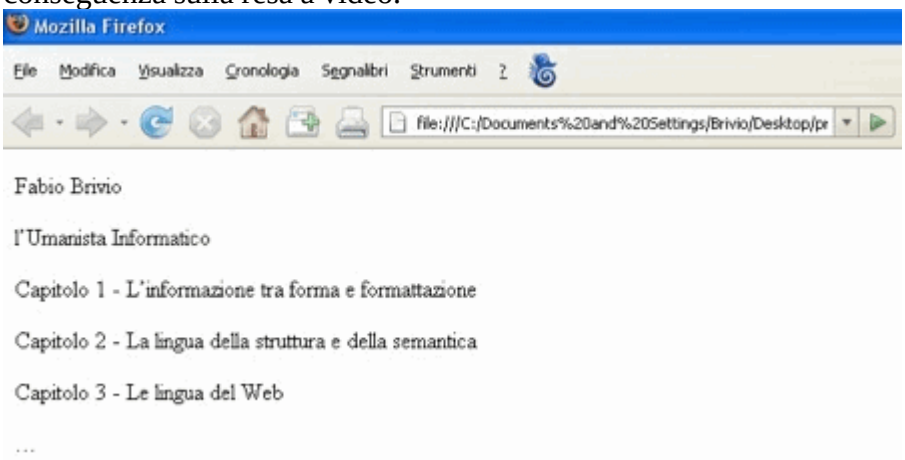


Figura 3.13 Il valore semantico aggiunto dall'elemento `<div>` non è visibile a video.

La necessità di marcare sezioni di testo in una maniera specifica può manifestarsi anche all'interno di un blocco, su una o poche parole. In questi casi, se non si vuole creare un nuovo blocco, è possibile utilizzare l'elemento `<span>`, il corrispondente di `<div>` in linea. L'inserimento di `<span>` in un testo non ne determina l'interruzione e non ha effetti sulla resa visiva. Ecco un semplice esempio del suo utilizzo:

```
<p><span class="acronimo">HTML</span> e <span
class="acronimo">XML</span> sono due linguaggi di
marcatura pensati per agevolare la pubblicazione e la
condivisione delle informazioni in Rete.</p>
```

Nel caso in cui sia invece necessario inserire un'interruzione all'interno del flusso di un blocco, senza però voler ricorrere a un nuovo blocco, è possibile utilizzare il tag vuoto `<br/>` (dall'inglese *break row*), la cui funzione è proprio quella di creare interruzioni di riga forzate.

Ecco un esempio e la relativa visualizzazione in un browser (Figura 3.14):

```
<p>Questo paragrafo andrà a capo qui<br/> e riprenderà
su nuova riga.</p>
```

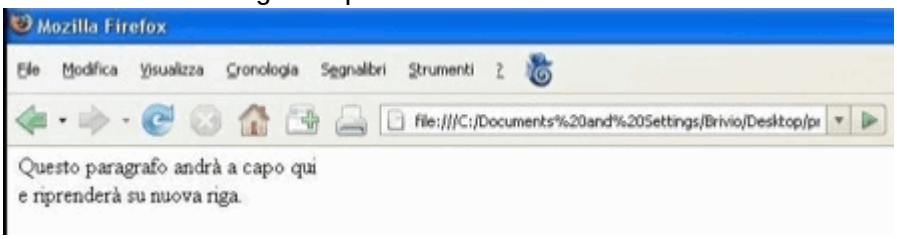


Figura 3.14 Un utilizzo comune del tag `<br>`.

Elenchi

Spesso in un documento è necessario organizzare le informazioni in elenchi o liste. HTML utilizza il tag `<ul>` per marcare elenchi non ordinati e `<ol>` per quelli numerati. I singoli punti di una lista devono invece essere marcati con il tag `<li>`.

ATTENZIONE *L'utilizzo di* `<ul>` *e* `<ol>` *implica necessariamente l'uso di* `<li>`*, e viceversa: differentemente la lista non sarebbe interpretata, e rappresentata, come tale.*

```
<h3>Elenco non ordinato</h3>
<ul>
   <li>Elemento</li>
   <li>Elemento</li>
   <li>Elemento</li>
</ul>
<h3>Elenco ordinato</h3>
<ol>
    <li>Elemento</li>
    <li>Elemento</li>
    <li>Elemento</li>
</ol>
```

Come mostra la Figura 3.15, anche `<ul>` e `<ol>` (e gli elementi `<li>`) si comportano come blocchi.

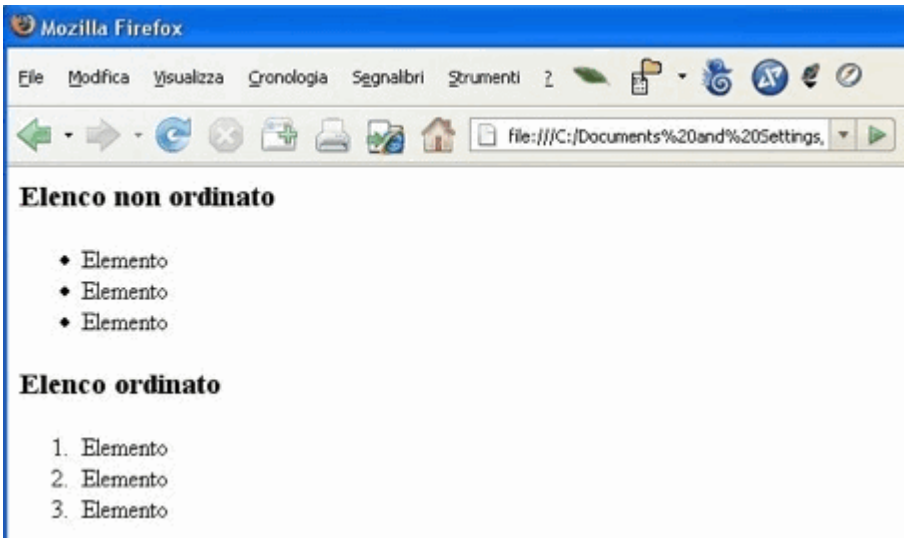


Figura 3.15 Gli elementi degli elenchi ordinati vengono numerati automaticamente dal browser.

Immagini

Il Web non sarebbe così bello e interessante senza immagini ad arricchirne i contenuti per scopi informativi e decorativi.

HTML gestisce le immagini come risorse esterne che vengono richiamate e quindi rappresentate in un documento attraverso il tag `<img>` corredato dall'attributo `src` a cui è assegnato come valore l'indirizzo (URL) dove risiede l'immagine che si desidera richiamare.

L'indirizzo di un immagine può essere assoluto o relativo.

- Se l'immagine risiede su un server diverso da quello che ospita la pagina, allora è necessario indicare un URL assoluto, cioè completo di nome di dominio e path della risorsa.
- Se invece pagina e immagine sono presenti sullo stesso server, allora è possibile indicare un URL relativo, cioè il solo path della risorsa espresso secondo delle apposite regole.

RIFERIMENTI *Si veda in questo capitolo, nel paragrafo "Sintassi degli indirizzi", i sottoparagrafi "Server" e soprattutto "File".*

Il tag `<img>` non prevede contenuto testuale: si tratta quindi di un marcatore vuoto il cui scopo è indicare dove nel testo deve essere collocata un'immagine e dove reperirla.

Questo tag non è classificabile né come elemento di blocco, né in linea: `<img>` fa infatti parte di una terza tipologia di elementi, quella degli elementi *rimpiazzati*.

Questi elementi sono così detti perché non contengono un vero contenuto ma semplicemente comunicano al browser dove reperirlo e in quale punto del documento inserirlo.

Dal punto di vista del codice HTML i tag `<img>` funzionano quindi come segnaposti: il contenuto, cioè l'immagine, non è realmente incluso nel documento, ma viene richiamato dal browser al momento di mostrare la pagina all'utente. In altre parole, il browser "rimpiazza" il tag con la relativa immagine.

Ecco un frammento di codice che utilizza il tag `<img>` per richiamare un'immagine esterna, utilizzando un indirizzo assoluto:

```
<p>Questa immagine viene richiamata dal server che
ospita il sito Apogeonline.com.</p>
<img src="http://www.apogeonline.com/img/apogeo-marchio.gif" />
```

La Figura 3.16 ne mostra la visualizzazione in un browser.

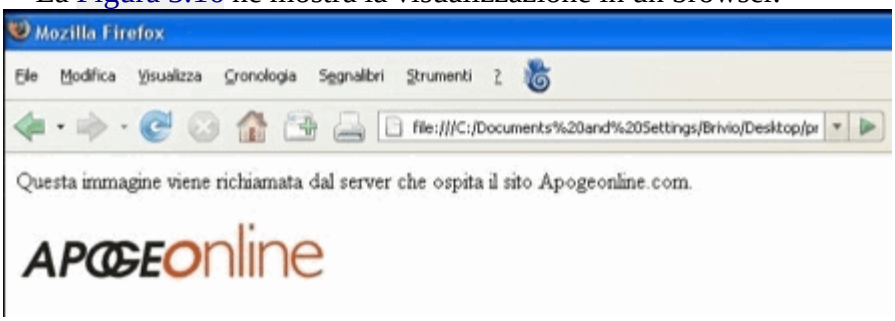


Figura 3.16 L'immagine viene richiamata per essere mostrata nel documento esattamente nel punto richiesto.

ATTENZIONE *La pratica di richiamare nelle pagine di un sito web immagini ospitate su server esterni senza un'esplicita autorizzazione è da molti considerata scorretta; in questo modo viene infatti sfruttato il server che ospita l'immagine, ma non quello che ospita la pagina web.*

Link

La possibilità di includere nel documento risorse esterne al documento stesso è tipica degli ipertesti, così come quella di creare collegamenti (link) tra documenti diversi. Il tag da utilizzare è in questo caso `<a>` (da *anchor*, "àncora" in inglese) unito all'attributo href a cui è assegnato come valore l'indirizzo (URL) del documento che si desidera "linkare". Un testo così marcato diviene pertanto attivabile con un clic del mouse e permette all'utente di accedere a un nuovo documento.

NOTA *È possibile creare link verso qualsiasi risorsa presente sul Web: un documento HTML, un'immagine, un file sonoro, un PDF e così via. A seconda del tipo di file linkato, questo sarà aperto dal programma adatto a gestirlo o a visualizzarlo sul computer dell'utente.*

Anche per i link l'indirizzo del documento può essere assoluto o relativo: è quindi possibile linkare un documento sia interno, sia esterno al server che ospita la pagina da cui parte il link.

Ecco un esempio e la relativa rappresentazione in un browser (Figura 3.17):

```
<p>Maggiori <a href="http://www.apogeonline.com/
```

```
libri/9788850328475/scheda”>dettagli sul libro
l’Umanista Informatico</a> sono disponibili sul sito
Apogeonline.com.</p>
```

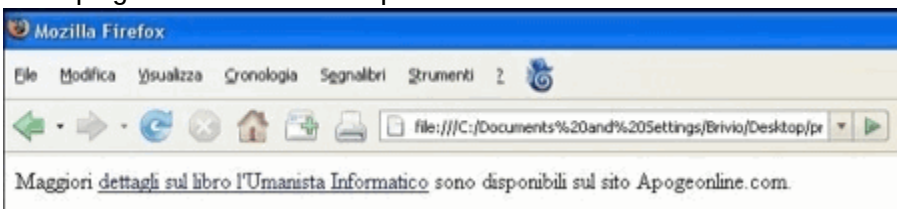


Figura 3.17 In un browser i link vengono mostrati di default con il testo blu e sottolineato.

SUGGERIMENTO *È buona norma utilizzare per i link testi esplicativi e descrittivi della risorsa a cui si rimanda, evitando il più possibile espressioni generiche. Da evitare sono formule del tipo “fai clic* `<a href='… '>`qui`</a>`*”: soluzioni di questo tipo impoveriscono il codice HTML di valore semantico e non vanno a favore di una sua corretta indicizzazione da parte dei motori di ricerca, molto attenti invece al contenuto dei tag* `<a>`*. Sull’indicizzazione delle pagine web da parte dei motori di ricerca torneremo nell’Appendice.*

I link costruiti con il tag `<a>` sono elementi in linea.

Tabelle

In un documento può essere necessario inserire dati in una griglia tabellare. Per questo scopo è possibile utilizzare diversi tag combinati tra loro. Eccoli.

- `<table>` marca una tabella nel suo insieme.
- `<thead>` marca le righe di una tabella che ne costituiscono l’intestazione.
- `<tbody>` marca le righe di una tabella che ne costituiscono il corpo.
- `<tfoot>` marca le righe di una tabella che ne costituiscono il piede.
- `<tr>` marca ogni singola riga di una tabella.
- `<th>` marca ogni singola cella dell’intestazione di una tabella.
- `<td>` marca ogni singola cella di una tabella, a eccezione di quelle dell’intestazione.

ATTENZIONE *Non tutti i tag appena elencati devono essere sempre presenti in una tabella HTML. L’utilizzo o meno di* `<thead>` *e di* `<tfoot>` *dipende dalla presenza o meno di intestazione e piede. I browser sono del resto istruiti a rappresentare correttamente intestazioni, corpo e piede delle tabelle anche in assenza di* `<thead>`, `<tbody>` *e* `<tfoot>`*; è sufficiente che le celle siano marcate con* `<th>` *o* `<td>` *(l’utilizzo dei tag* `<thead>`, `<tbody>` *e* `<tfoot>` *rende però le tabelle più corrette da un punto di vista semantico e formale). In definitiva in una tabella devono sempre essere presenti i tag* `<table>`, `<tr>`, `<td>` *(o* `<th>` *per le intestazioni).*

Ecco un semplice esempio dell’utilizzo di questi tag; nel tag `<table>` è inserito l’attributo border con valore 1: questo attributo permette di regolare lo spessore delle linee che compongono la griglia di una tabella, rendendola più o meno visibile. La Figura 3.18 ne mostra la relativa visualizzazione in un browser:

```
<table border=”1”>
   <thead>
     <tr>
     <th>Intestazione 1</th>
     <th>Intestazione 2</th>
     <th>Intestazione 3</th>
     </tr>
   </thead>
   <tbody>
     <tr>
     <td>Riga 1 - Valore 1</td>
```

```
        <td>Riga 1 - Valore 2</td>
        <td>Riga 1 - Valore 3</td>
        </tr>
        <tr>
        <td>Riga 2 - Valore 1</td>
        <td>Riga 2 - Valore 2</td>
        <td>Riga 2 - Valore 3</td>
        </tr>
        <tr>
        <td>Riga 3 - Valore 1</td>
        <td>Riga 3 - Valore 2</td>
        <td>Riga 3 - Valore 3</td>
        </tr>
      </tbody>
  </table>
```

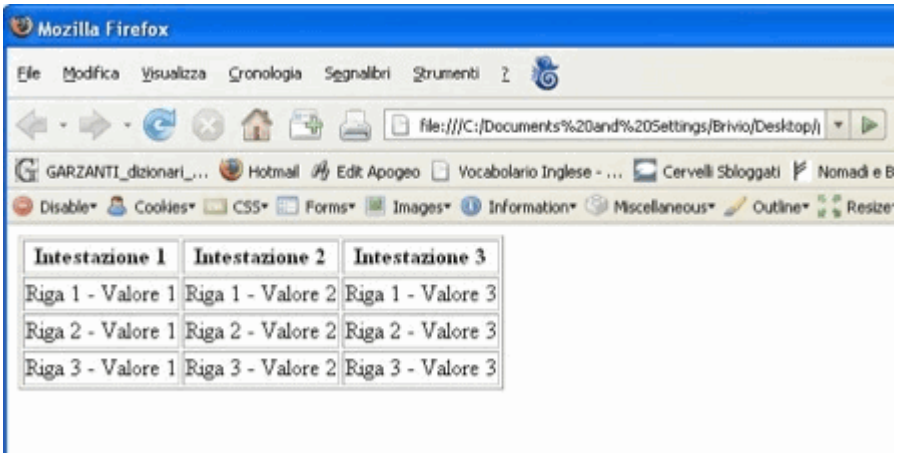


Figura 3.18 Una semplice tabella HTML.

ATTENZIONE *La possibilità intrinseca nelle tabelle di organizzare rigidamente i contenuti in griglie è stata per molti anni oggetto di abusi da parte di molti sviluppatori web, che strutturavano l'impaginazione a colonne di una pagina proprio attraverso questi elementi HTML. Tale prassi era certo in grado di fornire esiti apparentemente funzionali, ma era assolutamente scorretta dal punto di vista semantico e inoltre rendeva il codice HTML più complicato e difficile da gestire. Oggi questa pratica è fortunatamente deprecata e in disuso, e l'impaginazione di una pagina web viene delegata ai CSS (come si vedrà tra poco) in grado di lavorare con efficacia sul posizionamento dei* `<div>` *e degli altri elementi di blocco.*

Form

Parlando della sintassi degli URL e delle query string abbiamo fatto riferimento ai form. Si tratta di un insieme di elementi il cui scopo è quello di raccogliere informazioni dall'utente e quindi trasmetterle al server. Caselle di testo, menu a tendina, caselle di controllo (*checkbox)* e pulsanti d'opzione (*radio button*), aree di testo: la comunicazione, o meglio l'invio di dati dall'utente, il client, al server passa attraverso queste strutture o interfacce. La digitazione del testo di una e-mail o dei termini da ricercare con una motore di ricerca avviene proprio attraverso form, opportunamente configurati e integrati in una pagina web (Figura 3.19).

Figura 3.19 Un esempio noto di utilizzo di form: la pagina principale di Google.

Tutti gli elementi dei form sono rimpiazzati, quindi una volta che il browser li incontra provvede a inserire nella pagina la relativa interfaccia. Vediamoli nel dettaglio.

La struttura dei form non è molto complessa: essi si compongono normalmente di uno o più campi per la raccolta dei dati e di un pulsante di invio (o conferma) attraverso il quale i dati vengono inoltrati al server. Tutti questi elementi devono essere compresi dal tag `<form>`, al quale deve essere sempre associato l'attributo action.

Come ricordato parlando delle pagine web dinamiche, i dati dei form vengono elaborati dal server prima di restituire una risposta al client: lo scopo dell'attributo action è quindi specificare l'indirizzo della pagina a cui i dati devono essere inviati dopo che l'utente ha premuto il pulsante di conferma. Anche in questo caso è possibile utilizzare indirizzi assoluti o relativi.

Inoltre a `<form>` può essere associato l'attributo method che può assumere i valori di get o `post`. Un attributo `method` impostato a get rende visibili i dati nella query string dell'URL, cosa che non avviene con `post`.

NOTA *In assenza dell'attributo method i dati vengono inviati di default in modalità* `get`.

Ecco un esempio di utilizzo del tag `<form>`:

```
<form action="prova_form.php" method="get">
<!-- Campi di raccolta dati -->
</form>
```

RIFERIMENTI *Un semplice esempio di utilizzo di form è presente nel Capitolo 5, paragrafo "Variabili predefinite"*

Non essendoci ancora istruzioni per i campi di raccolta dati, la visualizzazione di questo codice in un browser non mostrerebbe nulla: il tag `<form>` marca semplicemente l'inizio e la fine di un form e specifica dove i dati devono essere inviati, esso non viene quindi rimpiazzato e ha solo un valore semantico e strutturale.

Vediamo quindi il primo elemento per la raccolta dati: `<input>`.

Si tratta di un elemento vuoto che, una volta associato all'attributo `type`, permette di creare diverse strutture di raccolta dati.

Ecco i principali valori di `type` i relativi esiti.

- text crea una semplice casella di testo.
- password crea una casella di testo specifica per digitare la password, i cui caratteri non verranno mostrati durate la digitazione ma sostituiti da asterischi.

- checkbox crea caselle di controllo utili per raccogliere informazioni del tipo “Sì/No”.
- radio crea pulsanti d’opzione utili per raccogliere un’unica preferenza tra scelte multiple.
- hidden crea un campo dati che però non viene mostrato all’utente ma rimane solo a livello di codice HTML (è utile per immagazzinare temporaneamente in una pagina dei valori che non servono tanto all’utente, ma al server per eventuali ulteriori elaborazioni).
- `reset` crea un pulsante che permette di svuotare i campi di un form, annullando così tutti i valori digitati o le scelte selezionate.
- `submit` crea un pulsante per l’invio dei dati al server.

Oltre all’attributo type, a tutti i tag `<input>` (ma in generale a tutti gli elementi dei form) deve essere associato un attributo `name`, il cui scopo è quello di definire il nome del parametro che si invia al server.

ATTENZIONE *Un’importante eccezione si manifesta con i tag* `<input>` *di tipo* `submit` *e* `reset`*: in questo caso l’uso di* `name` *non è necessario mentre è utile l’uso dell’attributo* value *il cui valore specifica l’etichetta che deve apparire sul pulsante (*Invia*,* Conferma*,* Cancella *e così via).*

Ecco un frammento di codice esemplificativo e la sua rappresentazione in un browser con i campi già compilati (Figura 3.20):

```
<form action=”prova_form.php” method=”get”>
<p>Digita il nome
<input type=”text” name=”nome” />
</p>
<p>Digita il cognome
<input type=”text” name=”cognome” />
</p>
<p>Fascia d’età:
meno di 18 anni
<input type=”radio” name=”anni” value=”Under18” />
- più di 18 anni
<input type=”radio” name=”anni” value=”Over18” />
</p>
<p>Voglio ricevere aggiornamenti da questo sito
<input type=”checkbox” name=”aggiornamenti” />
</p>
<p>Autorizza il trattamento dei tuoi dati
<input type=”checkbox” name=”autorizzazione” />
</p>
<input type=”submit” value=”Invia dati” />
</form>
```

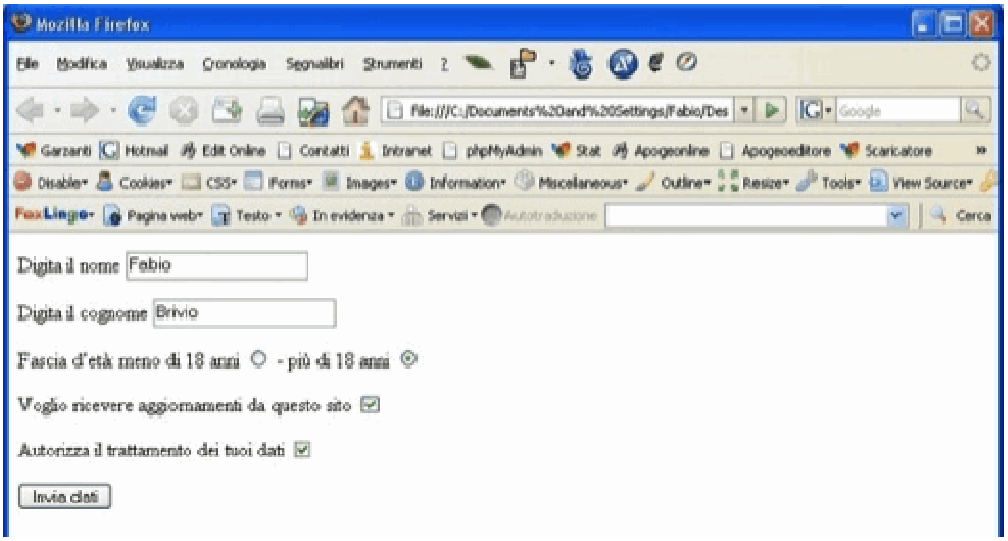


Figura 3.20 Un form composto da due campi testo, due pulsanti d'opzione e due caselle di controllo.

Dobbiamo notare come il valore dell'attributo `name` sia differente per tutti i campi del form, a eccezione dei pulsanti d'opzione (`radio`): come ricordato, il loro scopo è quello di raccogliere un'unica preferenza tra scelte multiple, pertanto, selezionato un valore il browser deseleziona ogni altro pulsante `radio` a cui è associato il medesimo valore di `name`. Questo significa che tra tutti i pulsanti radio accomunati dallo stesso nome, è possibile attivarne uno solo alla volta. Inoltre, a ogni pulsante radio deve essere sempre associato un attributo value compilato con il relativo valore (nell'esempio `"Under18"` e `"Over18"`): il valore del pulsante selezionato verrà inviato al server.

Ben diverso è l'utilizzo dei campi di tipo `checkbox`: ogni `checkbox` è infatti un campo a sé caratterizzato da un valore di name. Lo scopo di questi campi è quello di permettere di esprimere una scelta di tipo "Accesso/Spento" di fronte a una specifica situazione. Anche in questo caso dovrebbe essere assegnato un attributo `value` compilato con il valore che deve essere inviato al server in caso di selezione. Tuttavia, in assenza di un attributo `value`, le caselle di controllo funzionano ugualmente inviando al server un valore booleano di tipo `"On/1"` in caso di selezione, e nessun valore o `"Off/0"` in caso contrario.

TERMINOLOGIA *I valori* booleani *prendono il nome dal matematico G. Boole, il teorizzatore dell'algebra binaria. Nei sistemi binari gli unici numeri possibili sono 1 (vero) e 0 (falso) e tutti i passaggi logici, quindi tutta la comunicazione, si svolge attraverso complesse sequenze e combinazioni di questi due numeri.*

SUGGERIMENTO *Per selezionare di default un campo* `radio` *o* `checkbox` *è possibile utilizzare l'attributo* `checked`*. Questo attributo non prevede né permette valori: basta la sua presenza per far sì che una casella sia preselezionata.* `checked` *è quindi utile nel caso in cui si voglia suggerire all'utente una data scelta, per rendere la compilazione del form più veloce.*

Per quanto invece riguarda i campi di tipo `text`, vale la pena ricordare che anche qui è possibile inserire un attributo `value` il cui valore apparirà di default nella casella: l'utente potrà comunque modificarlo in seguito.

Fatte queste osservazioni, consideriamo ora la query string generata dal form della Figura 3.20:

```
prova_form.php?nome=Fabio&cognome=Brivio&anni=Over18&a
ggiornamenti=on&autorizzazione=on
```

Ogni valore raccolto dai campi del form viene passato al server nella sintassi precedentemente

vista: il valore dell'attributo `name` appare come nome del parametro, alla sua destra c'è il simbolo di uguale e quindi il valore del parametro. Per i campi radio il valore passato è quello dell'attributo `value`, mentre per i campi `checkbox` attivati viene passato il valore on non essendoci, in questo caso, altri valori specificati tramite l'attributo `value`.

Possiamo ora vedere altri due elementi che, insieme a `<input>`, permettono di creare form funzionali alle diverse esigenze di raccolta dati: `<select>` e `<textarea>`.

L'utilizzo del tag `<select>` permette di creare dei menu di scelta, detti "a tendina". A differenza di `<input>`, questo elemento non è mai vuoto: al suo interno devono sempre essere presenti almeno due (o più) elementi `<option>` che a loro volta devono contenere i singoli valori tra cui scegliere. Ecco un esempio e la sua rappresentazione in un browser (Figura 3.21):

```
 <form action="prova_form.php" method="get">
Scegli un valore tra quelli in elenco
<select name="elenco">
<option value="1">Valore 1</option>
<option value="2">Valore 2</option>
<option value="3">Valore 3</option>
<option value="4">Valore 4</option>
<option value="5">Valore 5</option>
</select>
<input type="submit" value="Invia dati" />
</form>
```

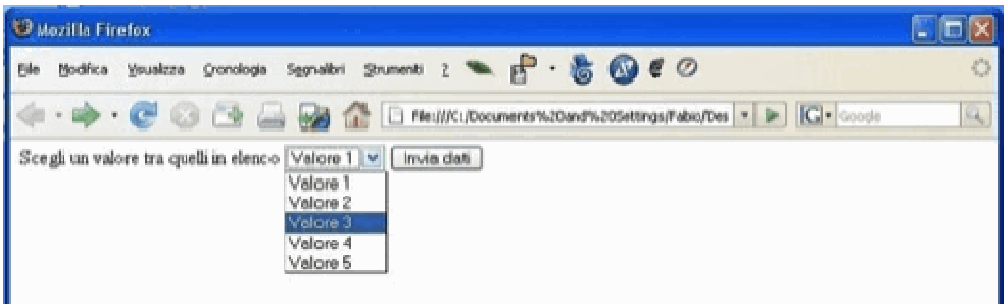


Figura 3.21 Un classico menu a tendina.

Anche qui è necessaria una precisazione: l'attributo `name` deve sempre essere associato solo al tag `<select>` e non ai singoli `<option>`, nei quali invece deve essere sempre presente l'attributo value. Al parametro name verrà quindi associato solo il valore dell'opzione selezionata.

Passiamo infine all'elemento `<textarea>`, la cui funzione è creare dei campi in grado di raccogliere testi lunghi (per esempio quelli di un messaggio e-mail). Ancora una volta si tratta di un elemento non vuoto la cui particolarità consiste nel fatto che un eventuale valore di default deve essere digitato tra il marcatore di apertura e di chiusura. Ecco un esempio e la relativa rappresentazione in un browser (Figura 3.22):

```
<form action="prova_form.php" method="get">
<p>Area per i messaggi di testo</p>
<textarea name="messaggio" rows="10" cols="40">
Digita qui il tuo messaggio
</textarea>
<br/>
<input type="submit" value="Invia dati" />
```

```
</form>
```

NOTA *A volte è utile definire la dimensione della casella di testo. Gli attributi* `rows` *e* `cols` *associati a* `<textarea>` *servono proprio a questo: il primo permette di specificare il numero di righe (quindi la dimensione verticale), il secondo la larghezza (quindi la dimensione orizzontale).*

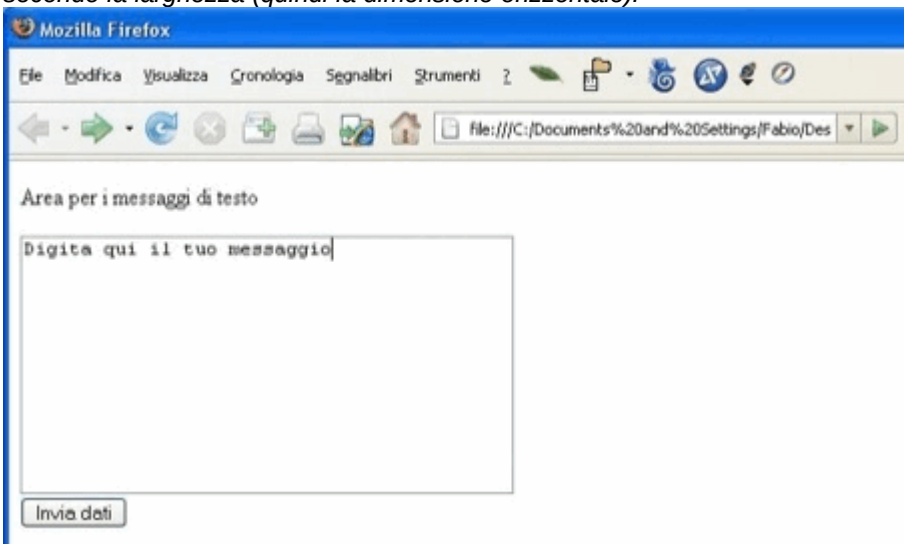


Figura 3.22 Una comune area per messaggi di testo.

Enfasi: grassetti e corsivi

Tutti i tag fin qui visti permettono di descrivere la struttura di un documento. A volte però è utile poter lavorare sulla struttura di una singola frase, enfatizzando alcune parole o marcandole in maniera più precisa. HTML mette per questo a disposizione alcuni marcatori specifici. Sono tutti elementi in linea e qui vengono ricordati solo i più diffusi.

- `<em>` aggiunge enfasi e i termini così marcati vengono visualizzati in corsivo in un browser.
- `<strong>` aggiunge maggiore enfasi e i termini così marcati vengono visualizzati in grassetto in un browser.
- `<cite>` marca testi come citazioni che vengono visualizzate in corsivo in un browser.

La differenza tra `<em>` e `<cite>` non consiste quindi nella rappresentazione che il browser ne fa, ma nel valore semantico: da una parte l'enfasi, dall'altra la citazione.

SUGGERIMENTO *Esistono anche i tag* `<i>` *e* `<b>`*, che permettono di applicare rispettivamente corsivo e grassetto senza però connotare il testo di enfasi. Si tratta di tag stilistici o "di presentazione", in grado quindi di lavorare solo sullo stile del testo, senza aggiungere valore semantico ai testi marcati. Altri due tag "di presentazione" abbastanza diffusi sono* `<small>` *e* `<big>`*, che permettono di riprodurre il testo con un corpo di carattere più piccolo o più grosso rispetto a quello degli altri termini della frase. È comunque preferibile utilizzare questi tag il meno possibile delegando tutte le caratteristiche di stile ai CSS, come vedremo nelle prossime pagine.*

Studiare il codice di una pagina

Gli elementi fin qui presentati permettono di creare pagine web complete e ben strutturate. Tuttavia i tag HTML inclusi nella specifica 4.01 sono davvero molti.

Non avrebbe del resto senso studiarli tutti a priori: alcuni trovano veramente scarse possibilità di impiego. Per migliorare la conoscenza di HTML, un utile suggerimento è invece studiare il codice sorgente delle pagine web, magari quelle di siti più famosi o che catturano l'attenzione per qualche particolare funzionalità.

Ogni browser permette di visualizzare il codice HTML della pagina che sta mostrando: basta

selezionare la voce *Sorgente pagina* (o *Vedi HTML*, o semplicemente *HTML*, la dicitura varia da browser a browser) nel menu *Visualizza*.

Per imparare i trucchi e le regole della programmazione in HTML, osservare il codice delle pagine web è uno dei metodi più pratici, senza dimenticare che su questo linguaggio è disponibile online molta documentazione: una rapida ricerca con Google per la query "HTML guida", o "HTML tutorial" fornisce un elenco interessante di risorse liberamente consultabili.

RIFERIMENTI *Se invece siete alla ricerca di un testo cartaceo, valutate* HTML 4.01 Pocket *di Gabriele Gigliotti, Apogeo, 2004.*

La lingua della presentazione del Web

È giunto il momento di esaminare il linguaggio che permette di definire l'aspetto delle pagine web: CSS (*Cascading Style Sheets*, Fogli Stile a Cascata).

Tutte le caratteristiche stilistiche (come per esempio il colore, la dimensione, la posizione, il font, lo sfondo e così via) di ogni elemento HTML possono essere definite attraverso delle regole espresse con il linguaggio CSS.

Anche questo linguaggio è una tecnologia sviluppata dal W3C, la cui specifica di riferimento è attualmente CSS2.

RIFERIMENTI *La versione originale inglese di questo documento è visibile sul sito del W3C, all'indirizzo* `http://www.w3.org/TR/CSS2/`. *Una traduzione italiana è invece disponibile all'indirizzo* `http://www.diodati.org/w3c/css2/cover.html`.

I CSS sono in pratica dei file (con estensione `.css`) che contengono le regole di stile per gli elementi HTML di un documento. L'espressione "a cascata" indica invece una particolarità fondamentale di queste regole: l'*ereditarietà*.

Parlando di XML abbiamo ricordato come la struttura e la gerarchia di un documento possano essere descritte in termini di relazioni di parentela genitori/figli. Lo stesso concetto si applica a tutti i linguaggi di marcatura e quindi anche a HTML, dove ogni elemento è padre o figlio di un altro. I CSS lavorano proprio su queste relazioni, per cui le regole di stile che si applicano a un elemento genitore ricadono a cascata su tutti i suoi elementi figli.

Per esempio, volendo definire il font con cui presentare tutto il contenuto testuale di un documento, questa regola CSS può essere scritta in relazione al solo tag `<body>`, che come ricordato contiene tutto il contenuto di una pagina web. Per effetto dell'ereditarietà il font definito per `<body>` sarà applicato a tutti i suoi figli, in questo caso tutti gli elementi del documento.

Nel caso in cui sia poi necessario definire un font diverso per un altro elemento (per esempio un titolo), basterà scrivere una regola specifica che annullerà l'effetto dell'ereditarietà. Poiché è possibile scrivere regole per ogni elemento di una pagina e poiché queste regole potrebbero anche essere in conflitto una con l'altra, un secondo principio fondamentale nei CSS è la *vicinanza*. In pratica a essere effettivamente applicata è sempre la regola più vicina a un elemento.

Tornando all'esempio di qualche riga fa, una regola per definire il font di un titolo `<h1>` (ma anche per il font di un `<div>` che contiene a sua volta il titolo `<h1>`) ha più peso (e quindi viene applicata) di quella definita per il font di `<body>`, che proprio perché padre di tutti gli elementi è anche l'elemento più lontano.

SUGGERIMENTO *Proprio per questo motivo nella scrittura dei CSS è utile definire per gli elementi più alti in*

gerarchia quelle regole che si desidera siano comuni e applicate a tutti gli elementi di una pagina web. In questo modo si semplifica la scrittura del foglio stile, che rimane più pulito e semplice da gestire o modificare.

Chiarito come i CSS operano sugli elementi HTML, prima di procedere all'illustrazione della loro sintassi, vediamo come possono essere associati a un documento HTML.

Un foglio stile può essere associato a una pagina HTML in tre modi.

- *In linea*. Le singole regole CSS vengono inserite nell'elemento su cui operano (quindi nel codice HTML) attraverso l'attributo HTML `style`. Si tratta di una modalità poco pratica che appesantisce il codice HTML e rende poco agevole la modifica e la gestione delle informazioni di stile.
- *Incorporati*. Le regole CSS sono specificate tutte nell'elemento HTML `<head>`, racchiuse nel tag `<style>`. Rispetto a quella in linea, questa modalità permette una gestione più chiara delle regole CSS ma ha in sé lo svantaggio di dover ripetere le regole in ogni singolo documento HTML e questo è poco pratico quando si lavora su siti composti da più pagine: ogni modifica di stile deve in questo modo essere ripetuta in ogni documento HTML con un notevole aumento dei tempi di gestione.
- *Esterni*. Questa modalità è fortemente raccomandata soprattutto quando il foglio stile deve essere applicato a più pagine web o a un intero sito. Tutte le regole CSS stanno in un file con estensione `.css` che viene associato a un documento HTML attraverso il tag `<link>` inserito nell'elemento `<head>` della pagina.

Il tag `<link>` è un elemento vuoto, e poiché permette di richiamare documenti di diverso tipo (non solo CSS) quando viene utilizzato per i richiamare i fogli stile, a esso vanno associati alcuni attributi.

- `href` specifica l'URL (relativo o assoluto) del foglio stile.
- `type`, con valore `text/css`, specifica il tipo di documento che `<link>` richiama, qui un foglio stile.
- `media` specifica il tipo di periferica a cui sono destinate le regole del foglio stile, per esempio il monitor di un computer o lo schermo di un cellulare, ma anche una stampante o un browser vocale;

ATTENZIONE *Con i fogli stile non si specifica solamente l'aspetto che un documento HTML avrà sullo schermo di un computer una volta visualizzato da un browser, ma anche la sua resa in stampa o sullo schermo di un cellulare o su altri media. Questo significa che la stessa informazione, una volta strutturata in HTML, può essere distribuita attraverso altre periferiche semplicemente predisponendo uno specifico foglio stile e modificando il valore dell'attributo* media*. Appare quindi ancora più evidente l'importanza di un buon uso dei CSS e del loro posizionamento al di fuori delle pagine HTML: solo così è possibile ridurre i tempi di gestione e sviluppo e amplificare al massimo la distribuzione delle informazioni. Tra i valori che* `media` *può assumere ricordiamo* `screen` *(monitor di un computer),* print *(stampante),* `handheld` *(dispositivi palmari come telefoni cellulari) e* `braille` *(dispositivi con risposta tattile di tipo braille).*

- `rel`, con valore `stylesheet`, specifica che si tratta del foglio stile da utilizzare di default per il media specificato (questo attributo è necessario perché è possibile avere più fogli stile per ogni media).

Ecco un esempio di utilizzo del tag `<link>`:

```
<head>
    <link href="stili.css" type="text/css"
```

```
 media=”screen” rel=”stylesheet” />
 </head>
```

Questa istruzione richiama in una pagina HTML il foglio stile `stili.css` che verrà applicato al documento quando questo sarà visualizzato sul monitor di un computer. Inoltre ogni modifica al foglio stile avrà effetto su tutte le pagine HTML che contengono questa istruzione `<link>`, con una notevole riduzione dei tempi di gestione e sviluppo del sito.

A questo punto possiamo passare alla sintassi delle regole CSS, che verranno introdotte utilizzando come pagina web di test il codice HTML riprodotto di seguito e dove tre blocchi (`<div>`) sono distinti dal valore dell’attributo `id`, mentre i paragrafi sono connotati da un attributo `class`:

```
    <head>
     <link href=”stili.css” type=”text/css”
media=”screen” rel=”stylesheet” />
</head>
<body>
<h1>Lorem ipsum</h2>
<div id=”blocco1”>
<p class=”testo”>Lorem ipsum dolor sit amet,
consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis
aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.</p>
</div>
<div id=”blocco2”>
<p class=”testo”>Lorem ipsum dolor sit amet,
consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis
aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur.  Excepteur
sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.</p>
</div>
<div id=”blocco3”>
<p class=”testo”>Lorem ipsum dolor sit amet,
consectetur adipisicing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis
aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.</p>
</div>
```

```
</body>
```

Grammatica CSS in pillole

In assenza di un foglio stile, questo codice HTML viene rappresentato in un browser come nella Figura 3.23. Ogni browser è infatti predisposto di default per offrire una presentazione standard delle pagine HTML. Questo vuol dire che, in assenza di altre informazioni di stile (cioè di istruzioni CSS), il browser utilizza un semplice foglio stile per mostrare i contenuti codificati in HTML. Se così non fosse, sullo schermo apparirebbero le istruzioni HTML nude e crude, e usufruire dei contenuti non sarebbe molto agevole.

NOTA *Normalmente questo foglio stile standard mostra i testi con un font graziato, come il Times New Roman, in colore nero su sfondo bianco, mentre i link vengono presentati con il medesimo font ma in blu e con l'aggiunta dell'effetto sottolineato. I browser permettono di modificare queste impostazioni predefinite nel pannello* Opzioni*, accessibile dal menu* Strumenti *(Figura 3.24).*

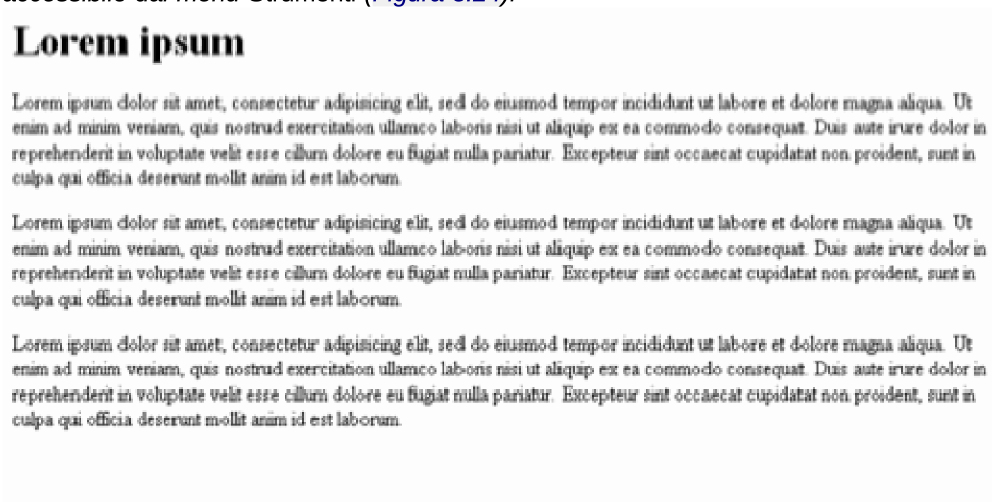


Figura 3.23 Rappresentazione standard di una pagina HTML in un browser.

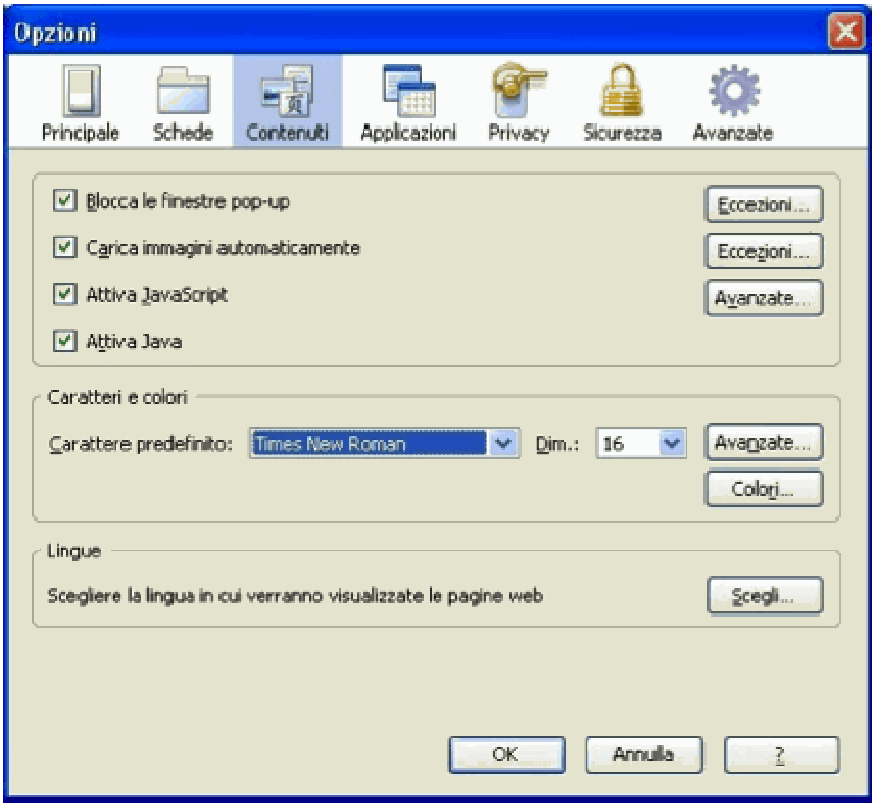


Figura 3.24 La scheda Contenuti del pannello Opzioni di Firefox permette di intervenire sui caratteri e i colori predefiniti del browser.

Questa presentazione dei contenuti a opera del solo browser, sebbene funzionale, è molto limitata e rigida, ma gestendola attraverso i CSS è possibile ottenere delle visualizzazioni alternative dei medesimi contenuti HTML, molto più funzionali, originali e creative.

SUGGERIMENTO *Per rendersi conto della potenza dei CSS è utile visitare la galleria di* css Zen Garden *(disponibile in Italiano all'indirizzo* `http://www.csszengarden.com/tr/italiano/`*). Si tratta di un progetto ideato da Dave Shea che per primo ha messo online una singola pagina HTML e un relativo foglio stile. Quindi ha proposto ai visitatori del suo sito di sviluppare nuovi fogli stile per la sua pagina e di inviarglieli. In questo modo ha creato una galleria che ospita numerose versioni della pagina originaria: ogni versione ha uno specifico foglio stile ma tutte lavorano sul medesimo codice HTML. Il risultato complessivo è spettacolare, non tanto per il numero di versioni ma per la varietà di scelte stilistiche: a seconda del foglio stile il contenuto viene proposto in maniera differente, e sebbene sia sempre presente identico, a volte passa in secondo piano, offuscato dalla potenza espressiva del foglio stile che lo presenta.*

Per questo è necessario definire delle regole di stile per i vari elementi HTML, cioè le istruzioni scritte all'interno dei file `.css`.

La sintassi di una regola CSS è la seguente:

```
selettore { proprietà: valori; }
```

Il selettore indica l'elemento o gli elementi HTML su cui deve essere applicata la regola, quindi all'interno di parentesi graffe trovano spazio – separate dal simbolo di due punti (`:`) – una o più coppie di proprietà (cioè l'aspetto stilistico su cui si vuole intervenire, a sinistra) e relativi valori (a destra) – seguite dal simbolo di punto e virgola (`;`).

NOTA *Alcune proprietà ammettono anche più valori contemporaneamente.*

Ecco una semplice regola e il relativo effetto sul codice HTML di esempio (Figura 3.25):

```
body { font-family: Arial; }
```

Il selettore indica che questa regola deve operare sull’elemento <body> (e a cascata, per via dell’ereditarietà, su tutto il suo contenuto). La proprietà font-family consente di specificare il font e come valore è impostato Arial. Il risultato è la presentazione di tutto il documento HTMLs con il font Arial.

A questo punto possiamo passare a esaminare meglio la sintassi dei selettori e a introdurre alcune proprietà e i relativi valori. In particolare vedremo delle proprietà il cui uso permette di controllare il posizionamento degli elementi HTML, intervenendo così sull’“impaginazione” di una pagina web.

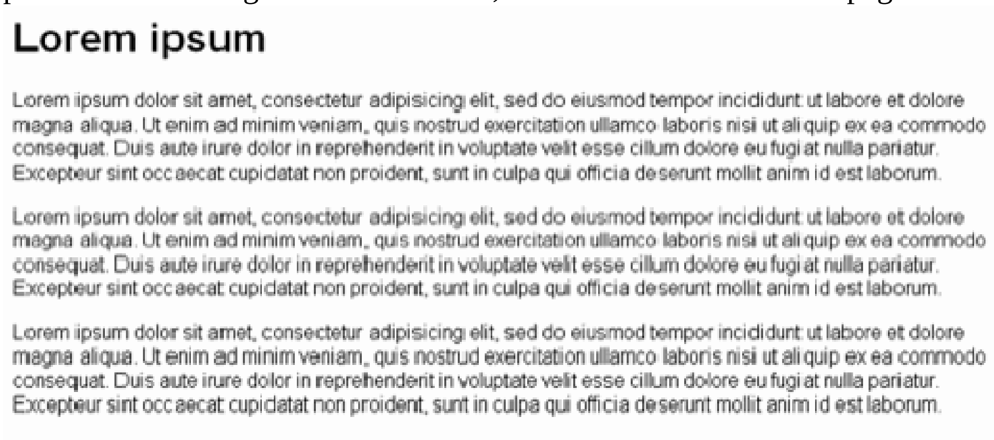


Figura 3.25 La proprietà font-family viene applicata a tutto il documento.

RIFERIMENTI *Per motivi di spazio non è possibile trattare qui tutte le caratteristiche dei CSS. Quanto segue è sufficiente per cominciare a lavorare con questa tecnologia, comprendendone in linea di massima le istruzioni. Molte particolarità (come le eccezioni al principio dell’ereditarietà) non possono però essere qui esaminate e quindi chi desidera approfondire l’argomento deve fare riferimento alla numerosa documentazione presente online, oppure, come libro, a* CSS Pocket *di Gianluca Troiani, Apogeo, 2006.*

Selettori

Come ricordato, il selettore indica l’elemento o gli elementi HTML su cui deve essere applicata la regola di stile.

La sintassi più comune dei selettori prevede la semplice indicazione del nome dell’elemento HTML (senza i simboli di maggiore e minore), per esempio:

```
h1 { proprietà: valori; }
div { proprietà: valori; }
p { proprietà: valori; }
```

TERMINOLOGIA *I selettori che utilizzano gli elementi HTML vengono detti* selettori di tipo*.*

Queste regole si applicano rispettivamente a tutti gli elementi <h1>, <div> e <p>.

È anche possibile scrivere una regola comune a più elementi, esplicitando un elenco di selettori separati da virgola:

```
h1, div, p { proprietà: valori; }
```

Questa regola si applica a tutti gli elementi <h1>, <div> e <p>.

Inoltre il simbolo asterisco (*) può essere utilizzato come selettore universale per applicare una regola a tutti gli elementi di una pagina:

```
* { proprietà: valori; }
```

I selettori possono quindi utilizzare gli attributi HTML id e class, che come ricordato permettono di identificare in maniera univoca un singolo elemento (id) o una classe di elementi (class).

In questo modo è possibile scrivere regole che utilizzano selettori in grado di operare direttamente sulle classi o gli identificatori coinvolti.

Un selettore di classe viene espresso attraverso il valore dell'attributo `class` preceduto da un punto (`.`):

```
.nome_classe { proprietà: valori; }
```

Questa regola opera su tutti gli elementi a cui è associato un attributo `class` con valore `nome_classe`.

Per gli identificatori, invece, il selettore viene espresso attraverso il valore dell'attributo `id` preceduto dal simbolo cancelletto (#):

```
#nome_id1 { proprietà: valori; }
#nome_id2 { proprietà: valori; }
#nome_id3 { proprietà: valori; }
```

Queste regole operano rispettivamente sugli elementi a cui è associato un attributo `id` con valore `nome_id1`, `nome_id2` e `nome_id3`.

È anche possibile combinare la sintassi dei selettori di tipo, classe e identificatori per creare delle regole che hanno effetto solo su insiemi ristretti di elementi. Ecco due semplici esempi:

```
p.nome_classe { proprietà: valori; }
div#nome_id1 { proprietà: valori; }
```

La prima regola ha effetto solo sugli elementi `<p>` a cui è associato un attributo `class` con valore `nome_classe`. La seconda regola opera invece sull'elemento `<div>` a cui è associato un attributo id con valore `nome_id1`.

Infine è possibile scrivere selettori anche in riferimento alle relazioni di parentela degli elementi HTML: le regole così espresse possono essere molto accurate e restrittive nell'applicazione.

La sintassi in questo caso prevede di indicare due o più selettori separati da uno spazio. Il selettore a destra dello spazio è l'elemento figlio, quello a sinistra il padre. I selettori possono essere espressi per tipo, classe o identificatore:

```
body #nome_id1 .nome_classe { proprietà: valori; }
```

Questa regola opera solo sugli elementi a cui è associato un attributo `class` con valore `nome_classe`, contenuti all'interno di un elemento a cui è associato un attributo `id` con valore `nome_id1`, a sua volta contenuto in un elemento `<body>`.

Ecco un utilizzo pratico e il suo effetto sul documento HTML di esempio (Figura 3.26):

```
body #blocco1 .testo {
font-style: italic;
font-weight: bold;
}
```

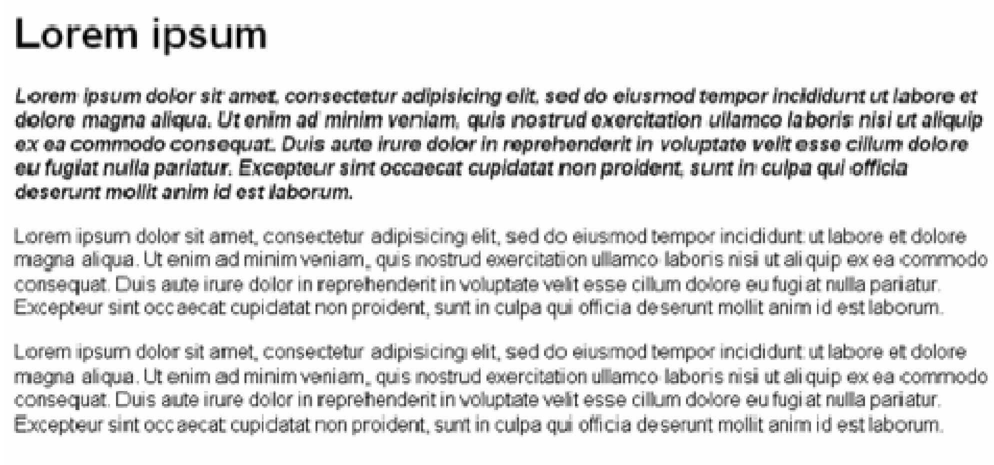


Figura 3.26 Applicazione di una regola CSS che sfrutta le relazioni di parentela degli elementi HTML.

Attraverso le proprietà `font-style` (con valore `italic`) e `font-weight` (con valore `bold`) il solo testo del primo paragrafo viene presentato in corsivo e grassetto.

Valori

Prima di passare in visione le più comuni proprietà CSS, osserviamo i tipi di valori che esse permettono.

Esistono diversi tipi di valori adatti alle varie proprietà e alcune proprietà ammettono più tipi di valori e più valori contemporaneamente.

Ecco un elenco dei valori possibili più comunemente impiegati.

- *Numeri*. Sono ammessi numeri interi (`1`, `2`, `3` e così via) e decimali con la parte decimale separata da un punto (`1.5`, `2.7`, `11.15` e così via).
- *Lunghezze*. Si tratta di numeri (interi o decimali, positivi o negativi) a cui è associata un'unità di misura (tra il numero e l'unità di misura non devono essere presenti spazi). Le unità di misura possono essere ***assolute*** o ***relative***. Tra le unità di misura assolute ricordiamo centimetri (`cm`), millimetri (`mm`), pollici (in, 1 pollice è uguale a 2.54 centimetri), punti (`pt`, 1 punto è uguale a 1/72 di pollice). Le unità di misura relative permettono invece di esprimere una dimensione in relazione a un riferimento. Tra di esse è utile ricordare gli `em` e i pixel (`px`). Gli em lavorano prendendo come riferimento la dimensione del carattere puntato dal selettore (per esempio 2em indica una lunghezza pari a due volte la dimensione – qualunque essa sia – del carattere in uso nell'elemento selezionato o nel suo genitore, se l'elemento selezionato non contiene testo). I px sono invece relativi alla risoluzione del monitor (per esempio 800×600 px o 1024×768 px) e alla sua dimensione (per esempio 13, 15 o 17 pollici). Inoltre è possibile esprimere dimensioni anche in valori percentuali (`%`). La percentuale è in relazione all'elemento genitore dell'elemento puntato dal selettore.

TERMINOLOGIA *Il* pixel *è l'unità minima di cui si compongono le immagini digitali e singolarmente non è distinguibile a occhio nudo. Nei CSS è possibile esprime in pixel le dimensioni di testi, margini, blocchi, bordi e in generale di tutti gli elementi presenti in una pagina web.*

NOTA *Nello sviluppo di fogli stile è preferibile utilizzare unità di misura relative. In questo modo il contenuto potrà adattarsi con maggiore facilità a dispositivi di output diversi (monitor di computer, cellulari, stampanti) o a risoluzioni di monitor differenti.*

- *URL*. Questa tipologia di valori è usata per richiamare risorse esterne al file CSS (come un'immagine da utilizzare nello sfondo di un elemento). In questo caso il valore vero e proprio, cioè l'URL (sia esso relativo o assoluto), deve essere espresso attraverso l'operatore `url` e racchiuso tra parentesi tonde (per esempio **`url`**`(../ URL_relativo)` o **`url`**`(http://URL_assoluto)`).

ATTENZIONE *Gli URL relativi devono essere espressi in relazione alla posizione del foglio stile e non al file HTML su cui il foglio di stile si applica. Per la sintassi degli URL relativi, si veda in questo capitolo, nel paragrafo "Sintassi degli indirizzi", i sottoparagrafi "Server" e soprattutto "File".*

- *Colori*. È possibile definire il colore del testo, dello sfondo o dei bordi di un elemento, esprimendolo come valore in vari modi. Il più semplice è attraverso l'utilizzo di parole chiave che altro non sono che il nome inglese dei principali colori (`red, black, green, blue, yellow, white, grey, purple` e così via). Questa modalità è senza dubbio intuitiva ma consente di utilizzare un insieme ristretto di colori (sedici per la precisione). Per ovviare a questa limitazione e utilizzare colori con una particolare tonalità bisogna ricorrere alla modalità RGB (*Red-Green-Blue*), che consente di definire un colore sulla base delle sue componenti di rosso, verde e blu. Esistono diverse sintassi per questo; qui ne ricordiamo solo una, quella che prevede l'utilizzo dell'operatore rgb. Le tre componenti di rosso verde e blu vanno dichiarate all'interno di parentesi tonde, separate da virgole secondo questo modello: `rgb(R,G,B)`. Ogni colore può assumere un valore da 0 (assenza di colore) a `255` (massima carica di colore). Per esempio, il colore rosso pieno può essere dichiarato con la parola chiave red o con la formula `rgb(255,0,0)`; portando il verde a 100 si otterrebbe invece una tonalità di arancio (`rgb(255,100,0)`) e così via.

RIFERIMENTI *Esistono diversi siti web che permettono di testare una combinazione RGB prima di utilizzarla in un foglio stile. Uno tra i tanti è RGB Hexadecimal to Decimal Converter, all'indirizzo* `http://www.psyclops.com/tools/rgb/`.

- *Parole chiave*. Abbiamo appena visto come le proprietà dei colori siano definibili attraverso parole chiave. Altre proprietà ammettono l'uso di parole chiave, per esempio la proprietà `font-family` ammette come parole chiave il nome del font che si desidera utilizzare. Incontreremo altre parole chiave più avanti, affrontando le singole proprietà.

NOTA *Nelle regole CSS, i valori vengono normalmente espressi senza apici (singoli o doppi) che li delimitino. Ci sono però alcune eccezioni a questo, ovvero le parole chiave composte da più termini, come vedremo tra poco nel caso della proprietà* `font-family`.

Proprietà

Passiamo ora in rassegna le più comuni proprietà che i CSS mettono a disposizione. Nelle pagine che seguono esse verranno elencate per ambito di utilizzo. Partiamo dalle proprietà applicabili sul testo.

Testo

È possibile intervenire su molte qualità dei testi variandone l'aspetto in maniera differente. Il tipo di font, il peso, la decorazione, l'allineamento, la dimensione, il colore sono tutti controllabili e definibili attraverso le proprietà che seguono.

- `font-family` permette di definire il tipo di carattere, cioè il font. Il valore è in questo caso

il nome del font (Arial, Times New Roman, Courier New e così via). È possibile assegnare a questa proprietà più valori (separati da virgola); se un font ha un nome composto da più parole (come Times New Roman), è preferibile scriverlo comprendendolo tra apici. Il browser tenta di applicare il primo valore dichiarato, cioè il font immediatamente a destra dei due punti (`:`): se questo non è presente sul computer dell'utente, viene preso in considerazione il secondo e così via. Nel caso in cui nessun font sia disponibile viene applicato il font di default.

SUGGERIMENTO *La tipologia di font disponibile su un computer varia da macchina a macchina. È quindi utile indicare sempre un font alternativo universalmente usato, come per esempio il Times New Roman (per i caratteri graziati) o l'Arial (per i caratteri a bastone).*

- `font-style` permette di definire come un font deve essere presentato. I valori più comunemente utilizzati sono `italic` (per il corsivo) e `normal` (per presentare il testo senza effetti).
- `font-weight` permette di definire il peso del font. I valori più comunemente utilizzati sono `bold` (per ottenere un effetto grassetto) e `normal` (per presentare il testo senza particolari pesi).

SUGGERIMENTO *Il valore* `normal` *è assegnato di default a queste due e ad altre proprietà. Non è quindi necessario specificarlo, a meno che non lo si voglia usare per annullare l'effetto a cascata di una proprietà impostata per un valore differente su un elemento genitore. La stessa considerazione vale anche per gli altri valori di default (per esempio* `none` *che vedremo tra poco per la proprietà* `textdecoration`*).*

- `font-size` permette di definire la dimensione del font. I valori più facilmente utilizzabili sono in questo caso lunghezze, preferibilmente relative e percentuali.
- `color` permette di definire il colore del testo. I valori sono le parole chiave per i colori o le componenti RGB.
- `text-align` permette di definire l'allineamento del testo, che può essere allineato a sinistra, a destra, centrato o giustificato. I valori relativi sono le quattro parole chiave `left`, `right`, `center` e `justify`.
- `text-indent` permette di definire il rientro della prima riga di un blocco di testo, cioè l'indentazione. I valori utilizzabili sono le lunghezze, preferibilmente relative e percentuali.
- `text-decoration` permette di definire il tipo di decorazione da applicare al testo. I valori ammessi sono le parole chiave none (nessuna decorazione, valore applicato di default), `underline` (testo sottolineato), `overline` (testo sovrastato da una linea), `linethrough` (testo barrato) e `blink` (testo lampeggiante).
- `text-transform` permette di definire il *case* (maiuscolo o minuscolo) del testo. I valori ammessi sono le parole chiave `none` (nessuna impostazione), `capitalize` (la prima lettera di ogni parola viene portata in maiuscolo), `lowercase` (tutto il testo viene portato in miniscolo) e `uppercase` (tutto il testo viene portato in maiuscolo).

Il foglio stile che segue mostra l'utilizzo di queste proprietà. Possiamo vedere la sua applicazione nella Figura 3.27.

```
body {
font-family: Arial;
}
```

```
h1 {
text-transform: uppercase;
text-align: center;
}
#blocco1 {
font-style: italic;
font-family: ‘Times new Roman’;
font-size: 1.5em;
text-align: left;
}
#blocco2 {
font-weight: bold;
color: rgb(255,0,0);
font-family: ‘Courier New’;
text-align: right;
text-decoration: underline;
}
#blocco3
{
text-indent: 1.5em;
text-align: justify;
text-decoration: line-through;
}
```

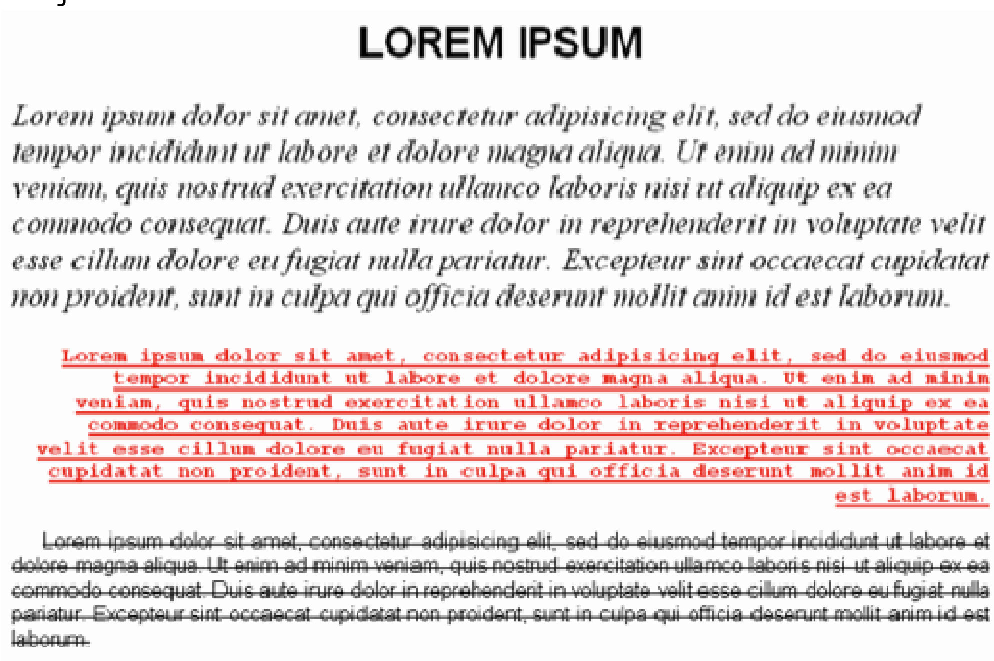


Figura 3.27 Poche regole CSS cambiano radicalmente l’aspetto e la leggibilità di un contenuto.

Come si vede, su tutto il `<body>` è applicato un font Arial, ma questo appare solo nel titolo `<h1>` e nel terzo `<div>` in quanto i blocchi con il valore id settato a `blocco1` e `blocco2` hanno una specifica proprietà `font-family` che imposta il font rispettivamente a Times New Roman e Courier New.

Il testo del titolo viene presentato in maiuscolo per effetto della proprietà `text-transform` e tutti i blocchi presentano un allineamento diverso in quanto il valore di `text-align` è sempre

differente.

Il primo `<div>` appare quindi con il testo in corsivo e di dimensioni maggiori per effetto delle proprietà `font-style` e `font-size`.

Il testo del secondo `<div>` appare invece in grassetto, in rosso e sottolineato. Sono gli effetti delle proprietà `font-weight`, `color` e `text-decoration`.

Il testo del terzo `<div>` presenta altre qualità stilistiche: la prima riga è indentata ma tutto il testo è barrato, come se fosse cancellato; sono gli effetti delle proprietà `text-indent` e `text-decoration`.

Queste presentazioni, seppur semplici e (volutamente) confuse, sono già indicative della potenza dei CSS: basta l'applicazione di poche regole e il medesimo contenuto acquista o perde di leggibilità ed evidenza (o credibilità).

Tipo degli elementi

Abbiamo già ricordato che gli elementi HTML si distinguono principalmente in due tipi: in linea e di blocco.

È possibile variare il tipo di elemento utilizzando la proprietà `display`. Questa proprietà supporta quattro valori: `block`, `inline`, `list-item` e `none`.

- `block` trasforma un elemento in un elemento di blocco.
- `inline` trasforma un elemento in un elemento in linea.
- `list-item` trasforma un elemento in un elemento di lista, o meglio in un blocco preceduto da un marcatore grafico tipico degli elenchi (per esempio il *bullet*, il comune "pallino" che si incontra negli elenchi non ordinati).
- `none` rende un elemento invisibile. Un elemento così trattato rimane presente nel codice HTML ma non ha alcun peso nella visualizzazione della pagina. Tutti gli altri elementi vengono mostrati come se l'elemento con la proprietà `display` impostata a `none` non esistesse.

Ecco un esempio di utilizzo di queste proprietà e il loro effetto sulla presentazione della pagina di esempio (Figura 3.28):

```
body {
font-family: Arial;
}
h1 {
display: none;
}
div, p {
display: inline;
}
#blocco3 {
margin-left: 2em;
display: list-item;
}
```

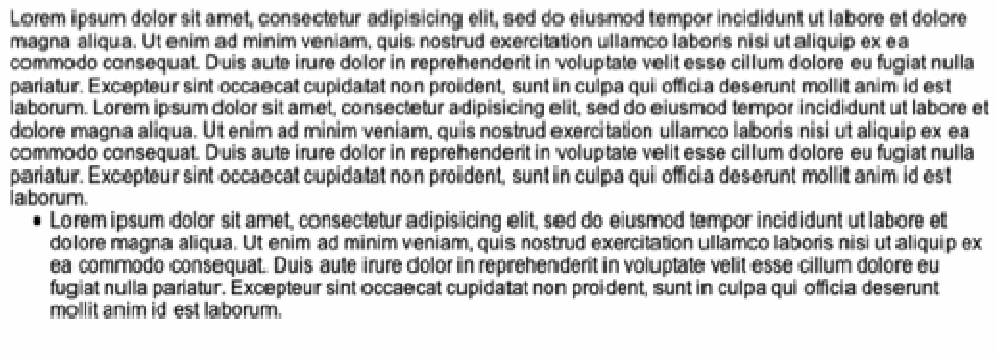


Figura 3.28 Modifica della presentazione del tipo di elemento.

L'applicazione di questo foglio stile ha come effetto quello di non mostrare il titolo `<h1>`. Quindi tutti gli elementi `<div>` e `<p>` vengono gestiti non come elementi di blocco, ma in linea. I primi due paragrafi appaiono così fusi in un unico flusso. Il `<div>` e il `<p>` del terzo blocco non sono invece toccati da questa regola. Su di esso viene infatti applicata una regola definita appositamente (il selettore riferisce infatti all'id con valore `blocco3`) che annulla l'effetto a cascata della regola precedente e mostra il blocco come un elemento di lista.

NOTA *Sulla proprietà* `margin-left` *torneremo tra poco. Per ora basti sapere che essa è qui necessaria per fare in modo che il marcatore di elenco, il bullet, venga visualizzato. Diversamente esso non sarebbe stato visibile, in quanto posizionato dal browser fuori dalla gabbia (cioè dalla dimensione orizzontale) dell'elemento, che in questo caso corrisponde a quella della pagina.*

La possibilità di intervenire sul tipo di elemento potrebbe apparire poco utile in un primo momento. Poiché gli elementi vengono definiti durante la strutturazione di un documento HTML, perché dovrebbe essere necessario variarne il tipo in sede di presentazione?

I motivi possono essere diversi e tutti hanno a che fare con scelte di comunicazione.

Per esempio, potrebbe essere utile nascondere dei paragrafi quando il documento viene visualizzato su un dispositivo mobile come un cellulare, dotato di uno schermo piccolo e quindi non adatto alla lettura di testi troppo lunghi. In questo modo si potrebbe creare una versione ridotta di una pagina dedicata a chi accede al Web non comodamente seduto davanti al computer ma con in mano un telefono.

Oppure, sapendo che i motori di ricerca durante l'indicizzazione delle pagine (si veda l'Appendice di questo libro) prestano attenzione al contenuto dei titoli, si potrebbe ritenere opportuno dare ad alcuni testi il peso semantico di un titolo (attraverso uno dei sei marcatori `<h>`), ma in sede di presentazione si potrebbe valutare come più adatta una scelta stilistica che mostri alcuni titoli in linea, senza quindi interruzioni di flusso con i testi che seguono.

La stessa valutazione potrebbe essere fatta al contrario, per dare maggior evidenza stilistica a elementi che semanticamente nascono come in linea.

Infine potrebbe essere necessario, nello sviluppo di un foglio stile complesso, prevedere che alcuni elementi cambino di tipo, in modo da avere più margine di azione sul documento e seguire più facilmente un'idea creativa apparentemente non adattabile alla reale struttura del codice HTML che, è bene ricordarlo, in progetti complessi viene spesso scritto da altre persone attente principalmente alla struttura e alla semantica del documento e non alla sua presentazione finale.

Dimensioni dei blocchi

Le proprietà che seguono si applicano solo agli elementi di blocco e ne consentono la gestione per quanto riguarda le dimensioni.

Ogni blocco è definibile per quanto riguarda altezza, larghezza, bordo, *padding* (cioè la distanza tra contenuto e bordo) e margine (cioè la distanza tra il bordo e gli altri elementi della pagina).

Queste caratteristiche compongono il cosiddetto *box model* (Figura 3.29) e a ognuna di esse possono essere applicate diverse proprietà.

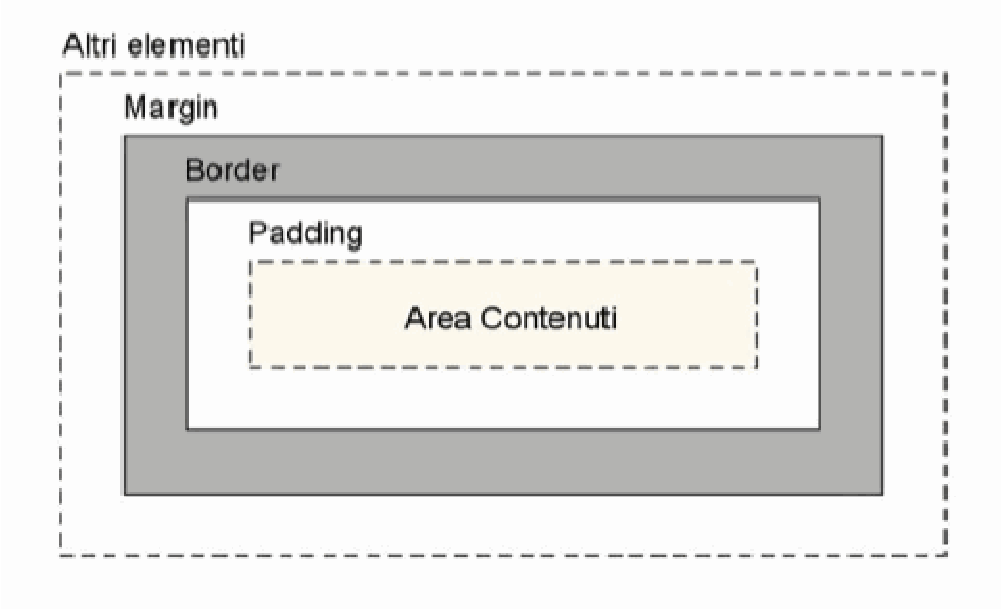


Figura 3.29 Il box model degli elementi di blocco.

Secondo il box model, un elemento di blocco si compone quindi (dall'interno all'esterno):

- di un'area dei contenuti veri e propri (definibile in altezza e larghezza);
- del *padding*, cioè un'area (vuota) circostante l'area dei contenuti e che termina in prossimità del bordo;
- di un bordo;
- di un margine, cioè un'area (vuota) circostante il bordo, al di fuori della quale trovano spazio gli altri elementi della pagina.

La reale dimensione di un elemento di blocco è data quindi dalla somma di tutte queste caratteristiche e non, come potrebbe essere spontaneo pensare, dalla dimensione della sola area dei contenuti.

ATTENZIONE *La comprensione del box model è fondamentale per lavorare in maniera corretta sul dimensionamento e sul posizionamento degli elementi di blocco.*

Vediamo ora le proprietà che permettono di definire le caratteristiche di un blocco.

- `widht` permette di definire la larghezza dell'area dei contenuti. I valori ammessi sono lunghezze relative, assolute e percentuali. Se non viene specificato, il contenuto dell'elemento può espandersi orizzontalmente per tutta l'area che l'elemento genitore riserva ai contenuti dei suoi elementi figli.

NOTA *Di default i browser lasciano espandere i contenuti per tutta l'ampiezza della finestra. Questo equivale in pratica a un valore di* `100%` *per la proprietà* `width` *di tutti gli elementi.*

- `height` permette di definire l'altezza dell'area dei contenuti. I valori ammessi sono lunghezze relative assolute e percentuali. Se non viene specificato, l'altezza dell'elemento è quella minima necessaria per mostrarne il contenuto.
- `padding` permette di definire la dimensione del padding. I valori ammessi sono lunghezze relative o assolute, e percentuali. È possibile specificare da uno a quattro valori. Se viene specificato un solo valore, questo si applica a tutti e quattro i padding (superiore, destro, inferiore, sinistro). Nel caso di due valori, il primo si applica ai padding superiore e inferiore, il secondo ai padding destro e sinistro. Nel caso di tre valori, il primo si applica al padding superiore, il secondo ai padding destro e sinistro, il terzo al padding inferiore. Nel caso di quattro valori, essi si applicano in senso orario, cioè il primo al padding superiore, il secondo al padding destro, il terzo al padding inferiore e il quarto al padding sinistro.

NOTA *Molte proprietà del box model condividono la modalità di applicazione di più valori appena vista per la proprietà* padding*.*

- `border` permette di specificare lo spessore, lo stile e il colore del bordo. Ognuna di queste qualità necessita uno specifico valore. Lo spessore ammette valori espressi come lunghezze (per esempio px). Lo stile viene invece controllato attraverso parole chiave. Le più comuni sono `solid` (riga continua), `dotted` (riga punteggiata), `dashed` (riga tratteggiata) e `none` (nessun bordo, valore di default); meno comuni ma comunque utilizzabili sono `double` (due linee continue), `groove` (effetto "scavato"), `ridge` (effetto "sbalzato"), `inset` (effetto bassorilievo) e `outset` (effetto altorilievo). Il colore viene quindi definito con le parole chiave per i colori o le componenti RGB.

NOTA *Se non viene specificato alcun colore per il bordo, questo viene presentato con il colore applicato al testo del contenuto del blocco, cioè con il colore definito dalla proprietà* `color`*.*

SUGGERIMENTO *I valori della proprietà* `border` *non vanno necessariamente indicati in un ordine fisso. Spessore, stile e colore di un bordo sono gestiti da valori di tipo differente e il browser non ha quindi ambiguità nella loro corretta applicazione.*

NOTA *È anche possibile definire un singolo bordo (superiore, destro, inferiore o sinistro) utilizzando le apposite proprietà* `border-top`*,* `border-right`*,* `border-bottom` *e* `border-left`*. I valori ammessi sono i medesimi della proprietà* `border`*.*

- `margin` permette di definire la dimensione del margine. I valori ammessi sono lunghezze relative, assolute e percentuali. È possibile specificare da uno a quattro valori in maniera analoga alla proprietà `padding`.

NOTA *È anche possibile definire un singolo margine (superiore, destro, inferiore o sinistro) utilizzando le apposite proprietà* `margin-top`*,* `margin-right`*,* `margin-bottom` *e* `margin-left`*.*

Ecco un esempio di utilizzo di queste proprietà e il loro effetto sulla presentazione della pagina di esempio (Figura 3.30):

```
body {
font-family: Arial;
width: 100%;
}
h1 {
padding: 30px;
border: 1px solid black;
margin: 20px;
```

```
}
#blocco1 p {
width: 80%;
padding: 30px;
border: 1px dotted black;
margin-top: 0px;
margin-left: 20px;
margin-bottom: 20px;
}
#blocco2 p {
width: 60%;
padding: 30px;
border: 1px dashed black;
margin-top: 0px;
margin-left: 40px;
margin-bottom: 20px;
}
#blocco3 p {
width: 40%;
padding: 30px;
border: 1px solid black;
margin-top: 0px;
margin-left: 60px;
margin-bottom: 20px;
}
```

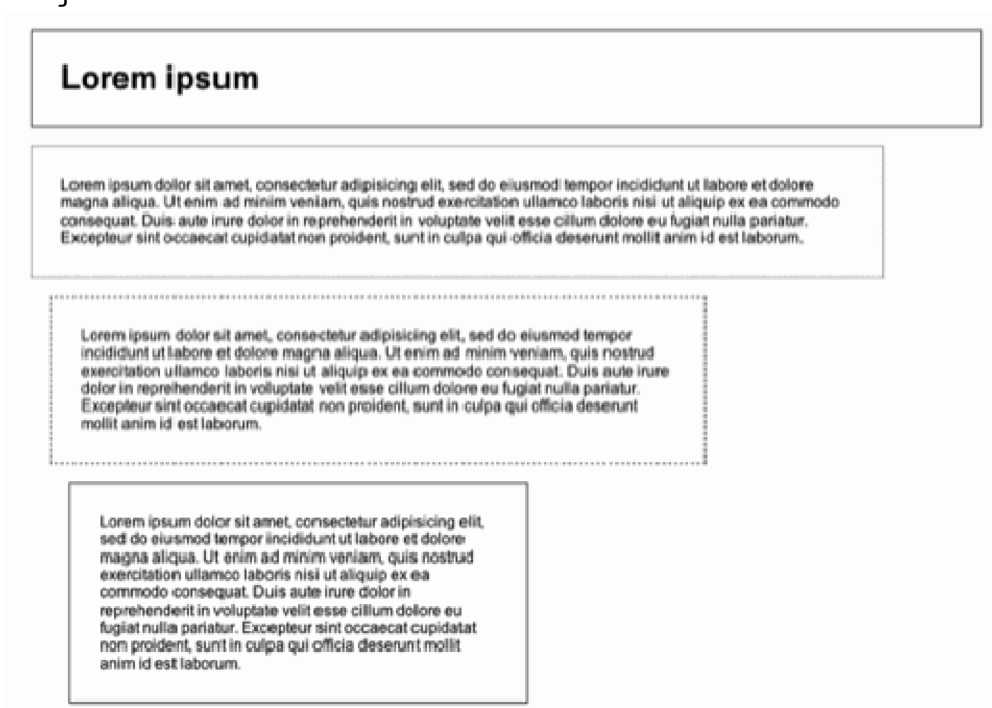


Figura 3.30 Applicazione delle proprietà del box model.

La larghezza dell’elemento `<body>` è impostata al `100%` e questo valore ricade a cascata su tutti gli elementi figli. Quindi il titolo `<h1>` mostra tre caratteristiche del box model: il padding intorno al contenuto, il bordo e il margine sono evidenti e impostati con specifiche caratteristiche comuni ai

quattro lati del blocco.

Il padding è inoltre impostato con valore `30px` in tutte le regole.

Appare evidente come tra i vari elementi sia costante un margine delle medesime dimensioni: `20px`. Si tratta del margine inferiore (`margin-bottom`), in quanto il margine superiore è impostato sempre a `0px` (a eccezione dell'elemento `<h1>` i cui margini sono tutti impostati a `20px`).

La larghezza dei tre elementi paragrafi è impostata con valori percentuali decrescenti (80, 60 e 40% in riferimento all'elemento padre `<body>`): in questo modo i blocchi si restringono proporzionalmente sull'asse orizzontale e si allungano di conseguenza sull'asse verticale. La proprietà `height` non è mai impostata, quindi l'altezza dei blocchi è quella minima per mostrare il contenuto. Infine la proprietà `margin-left` (impostata a 20, 40 e 60 pixel) allontana progressivamente i blocchi dal margine sinistro. Questo "movimento" appare evidente prendendo come riferimento i bordi, impostati con tre stili differenti.

Infine è utile notare come nelle proprietà che ammettono più valori questi siano espressi uno dopo l'altro separati solo da uno spazio e senza altri delimitatori come la virgola (che invece è necessaria, come abbiamo visto, per separare i vari valori della proprietà `font-family`).

Sfondi

Nella configurazione di default, normalmente i browser mostrano i contenuti testuali in nero su uno sfondo bianco. Alcune proprietà permettono di definire sfondi alternativi, con colori e immagini.

- `background-color` permette di definire il colore di sfondo di un elemento. Il valori ammessi sono le parole chiave per i colori o le componenti RGB.

ATTENZIONE *Negli elementi di blocco l'area di sfondo comprende l'area dei contenuti e il padding, ovvero tutto quello che sta all'interno dei bordi. Anche gli elementi in linea possono avere uno sfondo che però comprende la sola area dei contenuti.*

- `background-image` permette di applicare allo sfondo un'immagine. Il valore ammesso è in questo caso un URL. È anche possibile utilizzare la parola chiave `none`, per specificare che nessuna immagine va applicata.
- `background-repeat` permette di controllare la ripetizione verticale e orizzontale dell'immagine che di default viene collocata una prima volta nell'angolo superiore sinistro dell'elemento e quindi ripetuta orizzontalmente e verticalmente fino a saturare tutta l'area. I valori ammessi sono le parole chiave `repeat-y` (l'immagine viene ripetuta solo verticalmente), `repeat-x` (l'immagine viene ripetuta solo orizzontalmente) e `no-repeat` (l'immagine viene inserita una sola volta).
- `background-position` permette di definire il posizionamento iniziale dell'immagine. I valori ammessi sono lunghezze, relative o assolute, e percentuali oppure le parole chiave `left`, `center` e `right` per l'asse orizzontale e `top`, `center` e `bottom` per l'asse verticale. Questa proprietà ammette due valori contemporaneamente, il primo specifica il posizionamento sull'asse orizzontale, il secondo sull'asse verticale. Per esempio, `background-position: right bottom` indica che l'immagine deve essere collocata sullo sfondo nell'angolo inferiore destro del blocco.
- `background-attachment` permette di definire il comportamento dell'immagine in base ai

contenuti e alla finestra del browser. I valori ammessi sono le parole chiave `fixed` e `scroll`: la prima rende un'immagine fissa in relazione alla finestra del browser ("scrollando" la pagina i contenuti scorreranno mentre l'immagine sarà sempre visibile); la seconda fa in mondo che l'immagine occupi sempre la posizione assegnatale nell'elemento (pertanto l'immagine scorrerà insieme ai contenuti durante lo scroll della pagina).

Ecco un esempio di utilizzo di queste proprietà e il loro effetto sulla presentazione della pagina di esempio (Figura 3.31):

```
body {
font-family: Arial;
width: 100%;
background-image: url(bonsai.gif);
background-attachment: fixed;
background-position: right bottom;
background-repeat: no-repeat;
}
h1 {
padding: 30px;
border: 1px solid black;
margin: 20px;
background-color: grey;
}
#blocco1 p {
width: 80%;
padding: 30px;
border: 1px dotted black;
margin-top: 0px;
margin-left: 20px;
margin-bottom: 20px;
background-image: url(fiore.gif);
}
#blocco2 p {
width: 60%;
padding: 30px;
border: 1px dashed black;
margin-top: 0px;
margin-left: 40px;
margin-bottom: 20px;
background-image: url(fiore.gif);
background-position: right top;
background-repeat: repeat-y;
}
#blocco3 p {
width: 40%;
padding: 30px;
border: 1px solid black;
margin-top: 0px;
margin-left: 60px;
margin-bottom: 20px;
background-image: url(fiore.gif);
```

```
background-position: left bottom;
background-repeat: no-repeat;
}
```

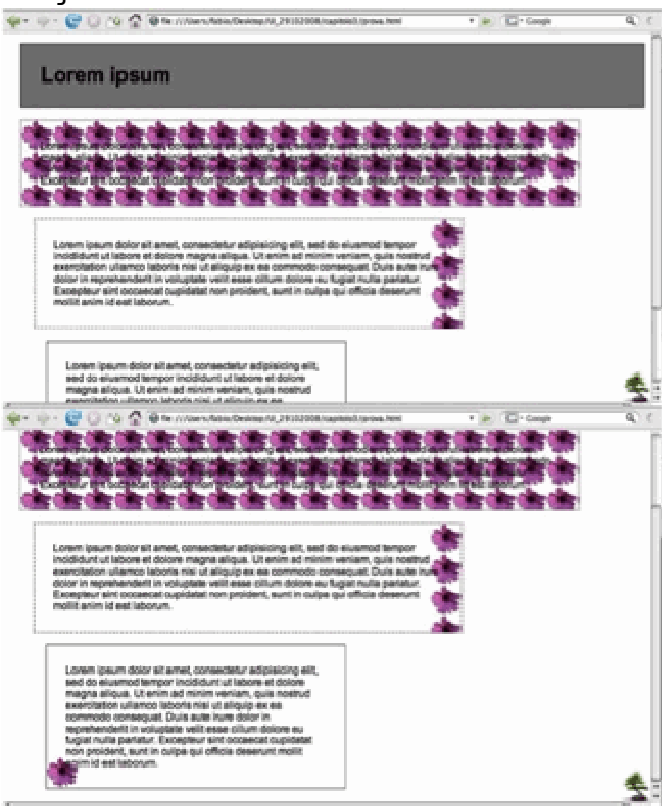


Figura 3.31 Effetti delle proprietà per la gestione degli sfondi.

L'elemento `<h1>` appare con uno sfondo grigio, mentre nei tre blocchi che seguono è presente un'immagine di sfondo (`fiore.gif`): nel primo blocco in assenza di controlli specifici essa viene ripetuta per tutta l'area di sfondo; nel secondo blocco viene posizionata nell'angolo superiore destro e da lì ripetuta per tutto l'asse verticale; nel terzo blocco viene posizionata nell'angolo inferiore sinistro e mai ripetuta.

Le finestre del browser mostrate nella Figura 3.31 chiariscono quindi l'effetto della proprietà `background-attachment` applicata all'elemento `<body>`. L'immagine (`bonsai.gif`) viene posizionata in maniera fissa nell'angolo inferiore destro della finestra del browser, non dell'elemento `<body>`. L'immagine è infatti presente nella medesima posizione in entrambe le figure: scrollando la pagina essa rimane fissa mentre il testo alla sua sinistra scorre.

Posizionamento dei blocchi

Le ultime proprietà che andiamo a esaminare sono quelle che permettono di intervenire sul posizionamento degli elementi. Queste proprietà consentono di variare il flusso degli elementi di una pagina ottenendo risultati estremamente differenti: usate con le proprietà del box model permettono di ridisporre i contenuti e disegnare nuove griglie di impaginazione, ideali per il tipo di messaggio e al tipo di comunicazione che si vuole realizzare.

NOTA *È bene sottolinearlo: in qualsiasi medium l'efficacia della comunicazione non è delegata al solo il contenuto ma anche al modo in cui esso viene presentato. Nei giornali cartacei come nelle pagine web, l'impaginazione di*

un contenuto è un elemento fondamentale per la sua comunicazione.

I blocchi, come abbiamo visto finora, di default si posizionano uno sotto l'altro, nell'esatto ordine in cui sono presenti nel codice HTML.

Per modificare questo posizionamento è possibile utilizzare le proprietà `position` e `float`.

ATTENZIONE *Le proprietà che seguono sono considerate e illustrate in relazione al loro uso su elementi di blocco. Tuttavia alcune di esse sono applicabili anche a elementi in linea. Il posizionamento degli elementi in linea può però causare un cambiamento di tipo, per cui l'elemento viene considerato come elemento di blocco.*

La proprietà `position` ammette quattro valori, le parole chiave `static` (posizionamento di default), `relative` (posizionamento relativo), `absolute` (posizionamento assoluto), `fixed` (posizionamento fisso) e – a eccezione del caso `position: static` – deve essere accompagnata da due altre proprietà: `top o bottom` (per collocare il blocco in riferimento all'asse verticale) e `left o right` (per collocare il blocco in riferimento all'asse orizzontale).

NOTA *Le proprietà* `top`, `bottom`, `left` *e* `right` *ammettono come valori lunghezze relative, assolute e percentuali. È anche possibile esprimere lunghezze con valori negativi.*

Il *posizionamento relativo* permette di definire il posizionamento di un blocco senza che questo abbia effetti sugli altri blocchi che precedono o seguono.

Ecco una sua applicazione e l'effetto sulla presentazione della pagina di esempio (Figura 3.32):

```
body {
font-family: Arial;
width: 100%;
}
h1 {
padding: 30px;
border: 1px solid black;
margin: 20px;
}
#blocco1 p {
padding: 30px;
border: 1px dotted black;
margin-top: 0px;
margin-left: 20px;
margin-bottom: 20px;
background-color: grey;
position: relative;
left: 100px;
top: 100px;
}
#blocco2 p {
padding: 30px;
border: 1px dashed black;
margin-top: 0px;
margin-left: 20px;
margin-bottom: 20px;
}
#blocco3 p {
padding: 30px;
border: 1px solid black;
margin-top: 0px;
```

```
margin-left: 20px;
margin-bottom: 20px;
}
```

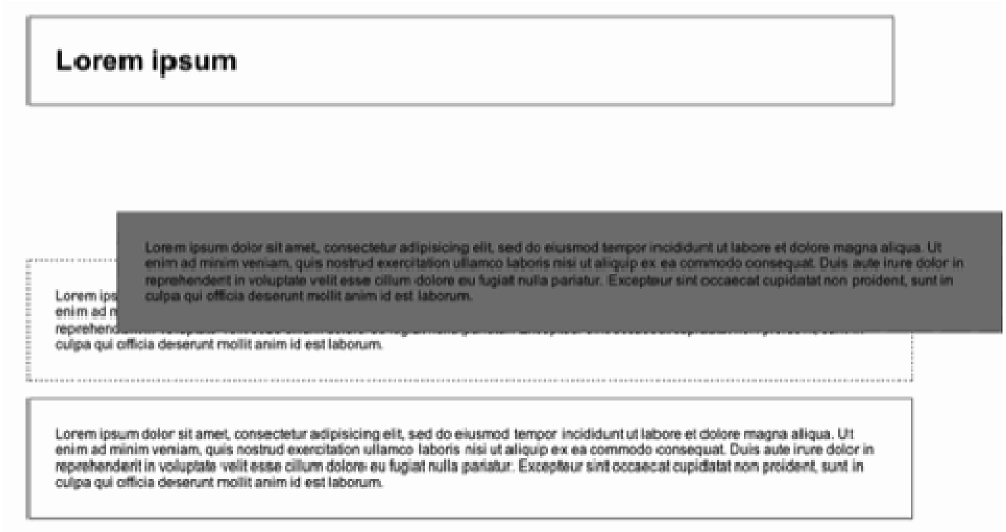


Figura 3.32 Posizionamento relativo di un elemento.

Tutti gli elementi sono stati dimensionati a una larghezza del 100%, in modo da poterne osservare meglio il comportamento in relazione al primo paragrafo su cui è stato anche applicato uno sfondo grigio. Il posizionamento relativo su questo blocco ha l’effetto di allontanarlo di 100 pixel dal suo margine superiore e sinistro. Il blocco si muove quindi in relazione alla posizione che avrebbe normalmente abbassandosi fino a sovrapporsi parzialmente al `<div>` che segue, il quale non viene però influenzato da questo riposizionamento e si dispone secondo il normale flusso.

Il posizionamento relativo permette quindi di variare la posizione dei blocchi in maniera sensibile, senza però alterare la posizione degli altri elementi, a meno che su di essi non sia applicata un’apposita regola di posizionamento. La Figura 3.33 mostra la completa inversione dell’ordine di posizionamento del primo e del secondo paragrafo.

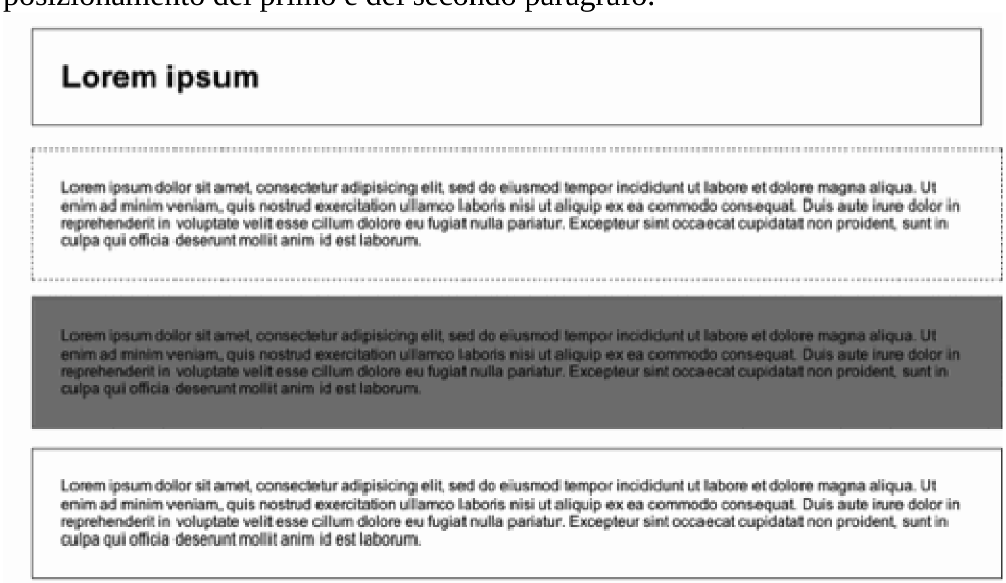


Figura 3.33 Il posizionamento del primo e del secondo paragrafo viene invertito.

Questo effetto è ottenuto attraverso un posizionamento relativo sui due blocchi impostato come

segue:

```
#blocco1 p {
position: relative;
top: 155px;
}
#blocco2 p {
position: relative;
bottom: 150px;
}
```

Il primo paragrafo si abbassa in relazione al suo margine superiore fino a occupare lo spazio del secondo, che a sua volta si alza in relazione al suo margine inferiore fino a occupare lo spazio del primo. Questo esempio illustra inoltre come sia possibile posizionare i blocchi anche in assenza di indicazioni per uno dei due assi (in questo caso quello orizzontale, proprietà `left` e `right`). Le proprietà non espresse vengono intese con valore nullo.

Passiamo ora al *posizionamento assoluto*. Esso permette di definire il posizionamento di un blocco slegandolo completamente dal flusso della pagina. Un blocco così definito non viene considerato come parte della pagina e si posiziona in riferimento all'elemento che lo contiene. Gli altri elementi si comportano e si dispongono come se esso non fosse presente nel codice HTML.

SUGGERIMENTO *Nel posizionamento assoluto è buona norma specificare sempre le dimensioni dell'elemento in modo da controllarlo con maggiore precisione.*

Il codice che segue applica un posizionamento assoluto sul primo blocco della pagina di esempio. Il risultato è visibile nella Figura 3.34.

```
#blocco1 {
position: absolute;
left: 200px;
top: 400px;
width: 400px;
}
```

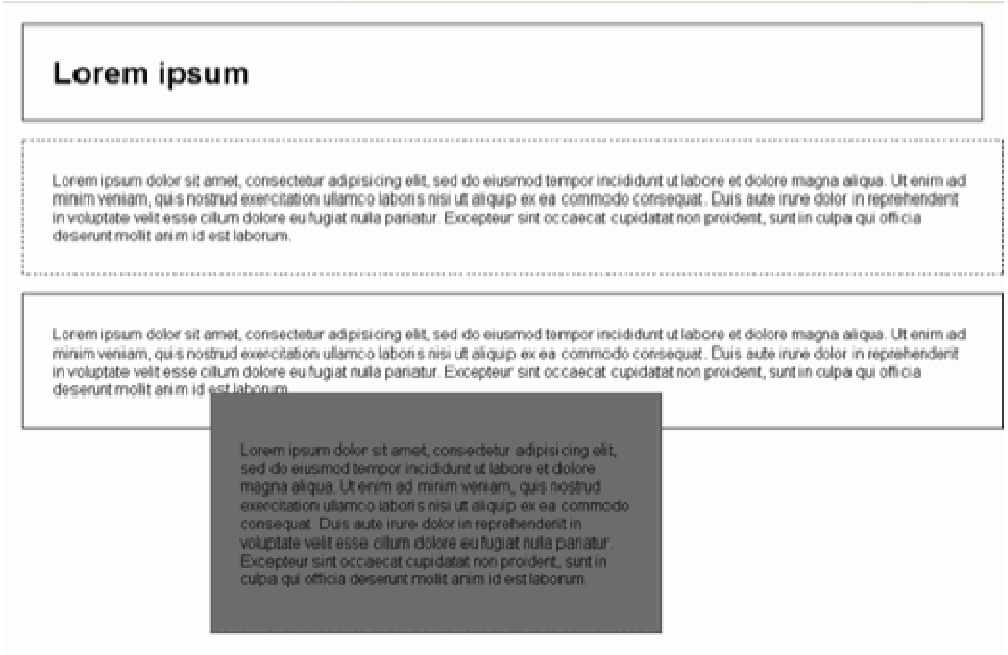


Figura 3.34 Posizionamento assoluto di un elemento.

Per effetto di questo posizionamento l'elemento (la cui larghezza è stata impostata a 400 pixel e che ricordiamo è nel codice HTML il primo dei tre `<div>`) si colloca a 200 pixel da sinistra e 400 pixel dall'alto rispetto all'elemento che lo contiene (l'elemento `<body>`), in parziale sovrapposizione e più in basso del terzo `<div>`. Gli altri elementi si dispongono invece seguendo il normale flusso, ma come se il primo `<div>` non esistesse: infatti il secondo blocco appare immediatamente dopo il titolo, e il terzo blocco immediatamente dopo il secondo, "scalando" quindi entrambi di una posizione.

Il *posizionamento fisso* permette infine di definire la posizione di un blocco in riferimento alla finestra del browser. L'effetto è quindi analogo a quello visto per gli sfondi con il valore `fixed` della proprietà `background-attachment`, con la differenza che "scrollando" la pagina i contenuti scorreranno di fianco a un elemento (che rimane pertanto sempre visibile) e non a un'immagine di sfondo.

ATTENZIONE *Anche con il posizionamento fisso l'elemento viene slegato dal flusso della pagina, esattamente come avviene con il posizionamento assoluto. La differenza principale nei due posizionamenti è quindi il riferimento: la finestra del browser per il posizionamento fisso, l'elemento contenitore per il posizionamento assoluto.*

Modifichiamo il codice dell'esempio precedente variando il posizionamento da assoluto a fisso. Il risultato è visibile nella Figura 3.35.

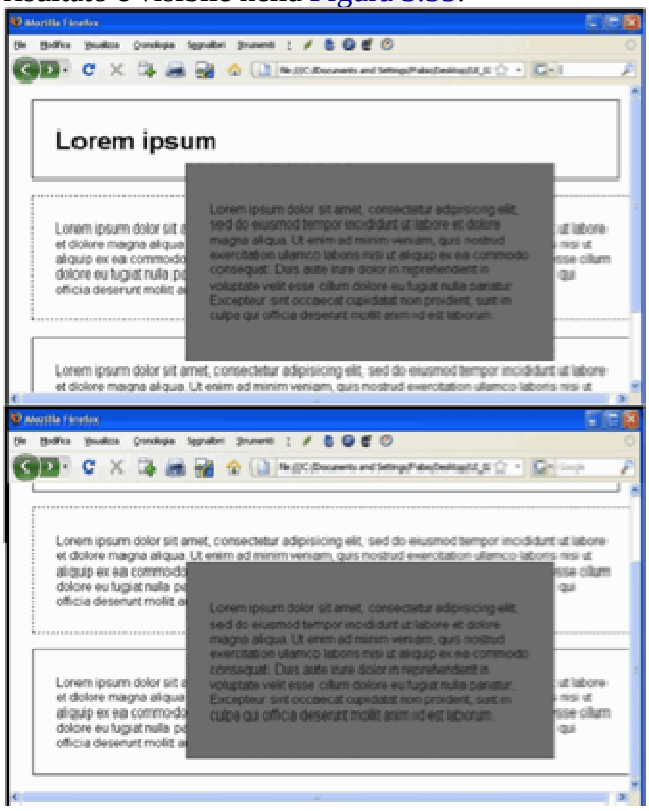


Figura 3.35 Posizionamento fisso di un elemento.

```
#blocco1 {
position: fixed;
left: 200px;
top: 100px;
width: 400px;
```

```
}
```

Ancora una volta gli elementi della pagina si comportano come se il primo `<div>` non esistesse. Esso però appare saldamente posizionato al centro della finestra del browser (a 200 pixel da sinistra e 100 pixel dall'alto) e lì rimane anche durante lo scorrimento verticale della pagina.

I posizionamenti fin qui visti sono tutti ottenibili con la proprietà `position`. La proprietà `float` a cui abbiamo fatto riferimento in precedenza permette invece un ulteriore tipo di posizionamento, detto *flottante*.

La prima e più importante particolarità di questo posizionamento consiste nel fatto che un elemento flottante si posiziona in un flusso influenzando il posizionamento del contenuto degli altri elementi.

ATTENZIONE *È bene sottolinearlo: a essere influenzato è il contenuto e non l'elemento che lo contiene. Per quanto riguarda il flusso degli elementi (intendendo gli elementi come l'insieme di margine, bordo, padding e area contenuti), anche gli elementi flottanti vengono infatti slegati dal normale flusso della pagina. Questo vuol dire che il posizionamento flottante di un elemento non ha effetti sui margini, i bordi e il padding degli elementi che lo circondano, che si comportano come se esso non fosse presente.*

Inoltre, non è possibile indicare riferimenti precisi sul posizionamento del blocco per quanto riguarda l'asse verticale e orizzontale.

Un blocco così gestito si sposta (o *flotta*) a destra o a sinistra all'interno dell'elemento che lo contiene, influenzando il posizionamento dei contenuti degli altri elementi presenti.

La proprietà `float` ammette come valori le parole chiave `none` (valore di default), `left` (spostamento a sinistra) e `right` (spostamento a destra).

Quando un elemento flotta a sinistra, il contenuto degli altri elementi si dispone alla sua destra; all'opposto, quando un elemento flotta a destra il contenuto degli altri elementi si dispone alla sua sinistra.

Ecco un'applicazione di questa proprietà al primo `<div>` della pagina di esempio. Il risultato è visibile nella Figura 3.36.

```
#blocco1 {
margin-left: 0px;
margin-right: 20px;
float: right;
width: 200px;
}
```

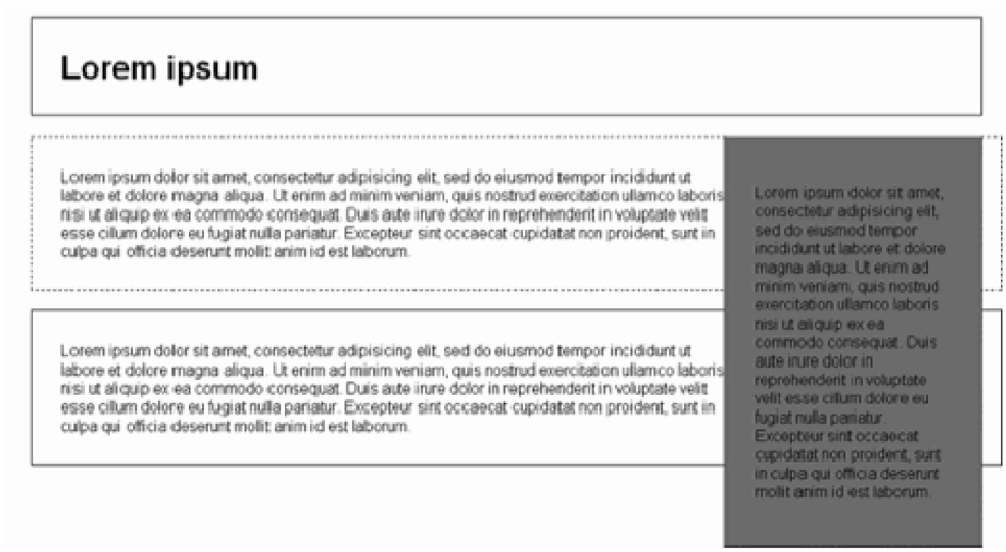


Figura 3.36 Posizionamento flottante di un elemento.

Il blocco – la cui larghezza è fissata a 200 pixel – si sposta a destra all'interno dell'elemento contenitore (`<body>`) con un margine destro di 20 pixel sufficiente per vedere come bordi e padding degli altri elementi non siano toccati dal movimento flottante del primo `<div>`. Infatti osservando i bordi destri degli altri elementi, possiamo notare che essi si comportano come se il primo `<div>` non fosse presente. Lo stesso non si può però dire per il loro contenuto, che viene influenzato dal movimento e scorre alla sinistra del blocco flottante, sfiorandone il bordo. Questo avviene perché il suo margine sinistro ha valore nullo (0 pixel): assegnando un valore positivo (per esempio 20 pixel) a questo margine, i contenuti degli altri elementi si distanziano dal blocco flottante (Figura 3.37).

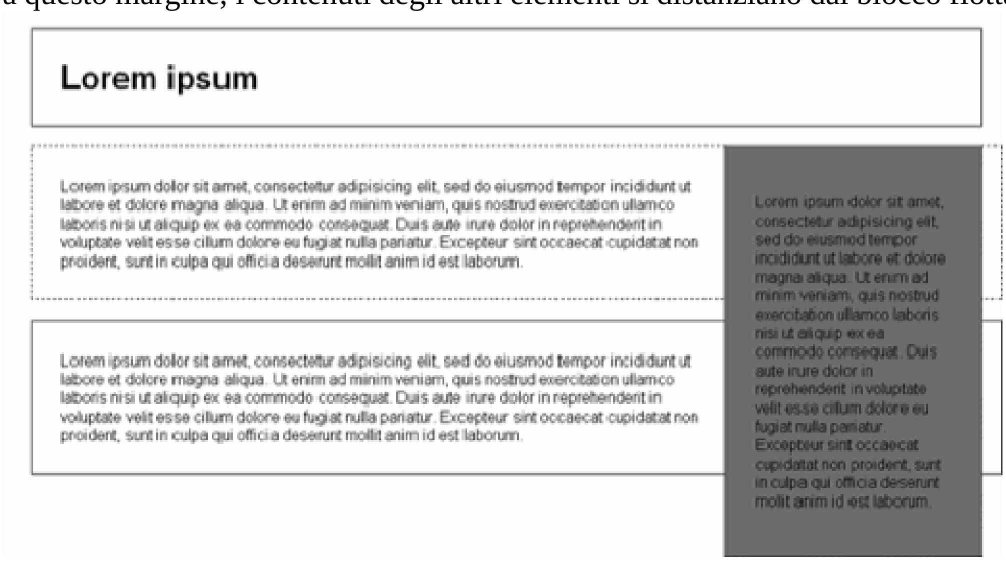


Figura 3.37 Le distanze tra un blocco flottante e i contenuti che gli scorrono attorno devono essere definite sul blocco su cui è applicata la proprietà float.

La proprietà float permette di ottenere con discreta facilità risultati particolari come l'impaginazione a colonne di un documento HTML.

Nell'esempio che segue i tre i `<div>` – ognuno con una larghezza impostata a 250 pixel – flottano a sinistra uno sull'altro. La proprietà `margin-right` (impostata a 20 pixel nei primi due) mantiene i contenuti testuali separati. Il risultato ottenuto è un'impaginazione a tre colonne (Figura 3.38):

```
#blocco1 p {
padding: 30px;
border: 1px dotted black;
background-color: grey;
margin-left: 20px;
margin-right: 20px;
float: left;
width: 250px;
}
#blocco2 p {
padding: 30px;
border: 1px dashed black;
margin-left: 0px;
margin-right: 20px;
float: left;
width: 250px;
}
#blocco3 p {
padding: 30px;
border: 1px solid black;
margin-left: 0px;
float: left;
width: 250px;
}
```

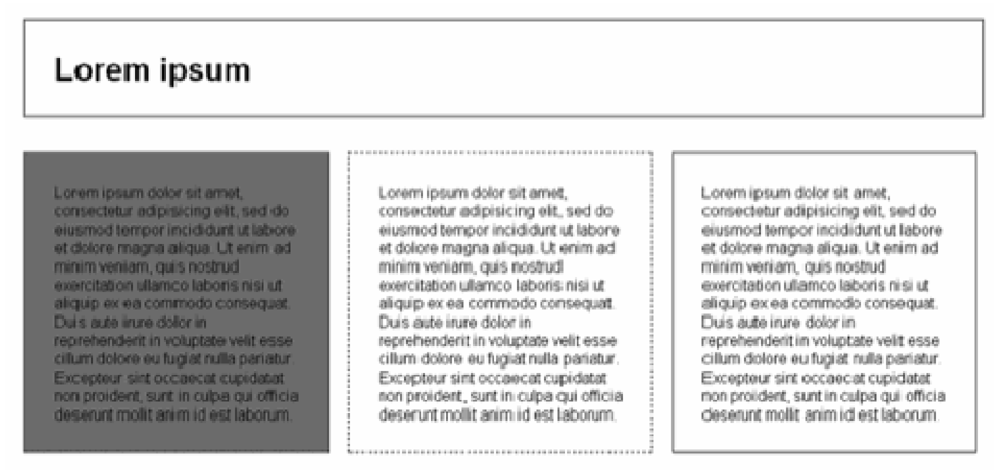


Figura 3.38 Impaginazione a tre colonne di una pagina web ottenuta grazie alla proprietà float.

Prima di concludere, è necessario accennare anche alla proprietà clear la cui funzione è quella di interrompere l'effetto di un posizionamento flottante. Essa può quindi essere applicata agli elementi che seguono un blocco flottante in modo che i loro contenuti non scorrano intorno al blocco ma secondo il normale flusso.

Questa proprietà ammette come valori quattro parole chiave: `none` (valore di default), `left` (annulla l'effetto della proprietà `float` con valore `left`), `right` (annulla l'effetto della proprietà `float` con valore `right`) e `both` (annulla l'effetto della proprietà `float` con valore sia `left`, sia `right`).

Problemi nell'utilizzo dei CSS

Quanto fin qui visto costituisce una panoramica discretamente completa della grammatica dei CSS. Una volta imparate alcune proprietà e fissati i concetti di elementi di blocco e in linea, scrivere un foglio di stile non è certo difficile.

Tuttavia la difficoltà è dietro l'angolo, o meglio nei browser.

Non tutti i browser implementano e supportano i CSS allo stesso modo: alcune proprietà vengono riconosciute e applicate solo da certi browser, altre sono interpretate in maniera differente.

Critico è in particolar modo il comportamento del browser Internet Explorer, versioni 5 e 6, ma anomalie si incontrano anche in altri browser.

NOTA *Un famoso e importante difetto di Internet Explorer 5 è la scorretta implementazione del box model.*

Le difficoltà maggiori si incontrano sul posizionamento dei blocchi, soprattutto in applicazioni elaborate: una scelta di impaginazione può funzionare perfettamente su un browser e presentare vistose anomalie o peggio ancora "sballare" completamente su un altro.

Scrivere fogli di stile funzionali su tutti i browser non è comunque impossibile, ma per questo è necessaria un'ottima conoscenza dei CSS (le cui particolarità – è bene ricordarlo ancora – non sono state e non possono essere esaurite in queste poche pagine) unita alla comprensione di alcune prassi di sviluppo che permettono di rendere un foglio stile "a prova di browser".

Strumenti di analisi e sviluppo

Per quanto detto sopra rimandiamo a testi e letture più specifici. Qui invece segnaliamo due strumenti gratuiti molto utili per avvicinarsi allo sviluppo web e allo studio dei CSS.

Entrambi sono estensioni del browser Mozilla Firefox (`http://www.mozilla-europe.org/it/`) che, oltre a essere gratuito e open source, offre un ottimo supporto dei CSS ed è quindi l'ambiente ideale da cui partire per lo studio di questa tecnologia.

DOM Inspector , attivabile dal menu *Strumenti*, permette di analizzare la pagina visualizzata evidenziando la struttura HTML e le varie proprietà CSS applicate sui singoli elementi (Figura 3.39).

NOTA *A essere mostrate non sono solo le proprietà scritte nelle regole del file* `.css` *associato al documento, ma anche quelle applicate di default dal browser.*

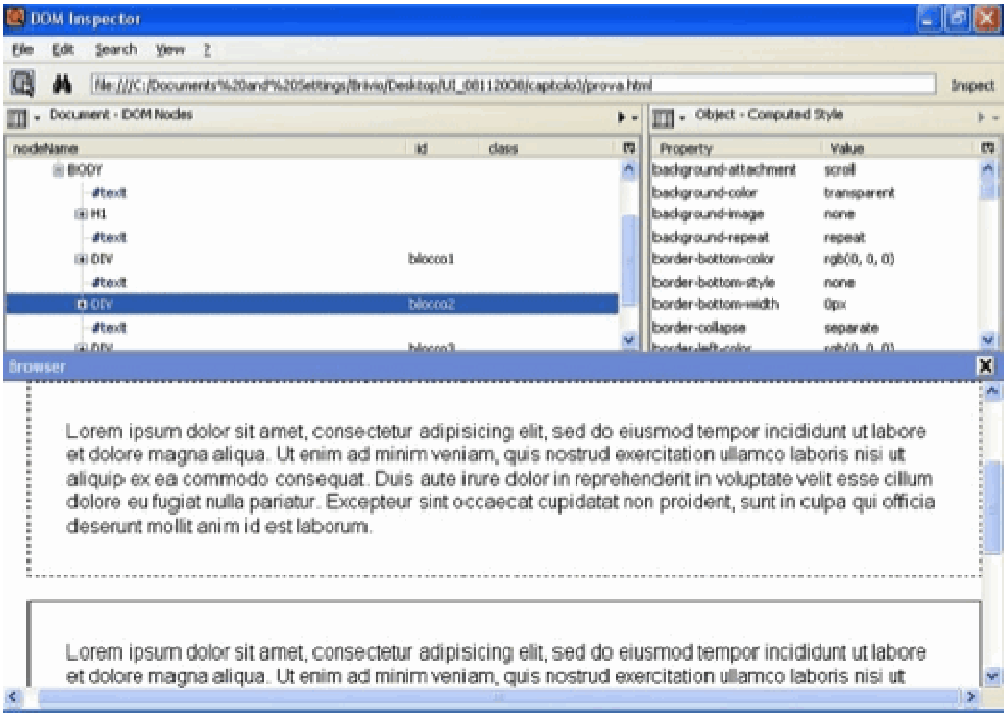


Figura 3.39 DOM Inspector in azione: nel quadrante superiore sinistro viene mostrata la struttura della pagina. Alla sua destra le proprietà assegnate all'elemento selezionato. Sotto la pagina analizzata.

Web Developer Toolbar è invece una barra di strumenti pensati per lo sviluppatore web. Alcuni di questi strumenti sono dedicati ai fogli stile e sono accessibili dal menu *CSS*. È possibile per esempio vedere il foglio stile di una pagina, disabilitarlo e anche editarlo virtualmente per valutare i risultati ottenibili prima ancora di salvare le modifiche nel file `.css` (Figura 3.40).

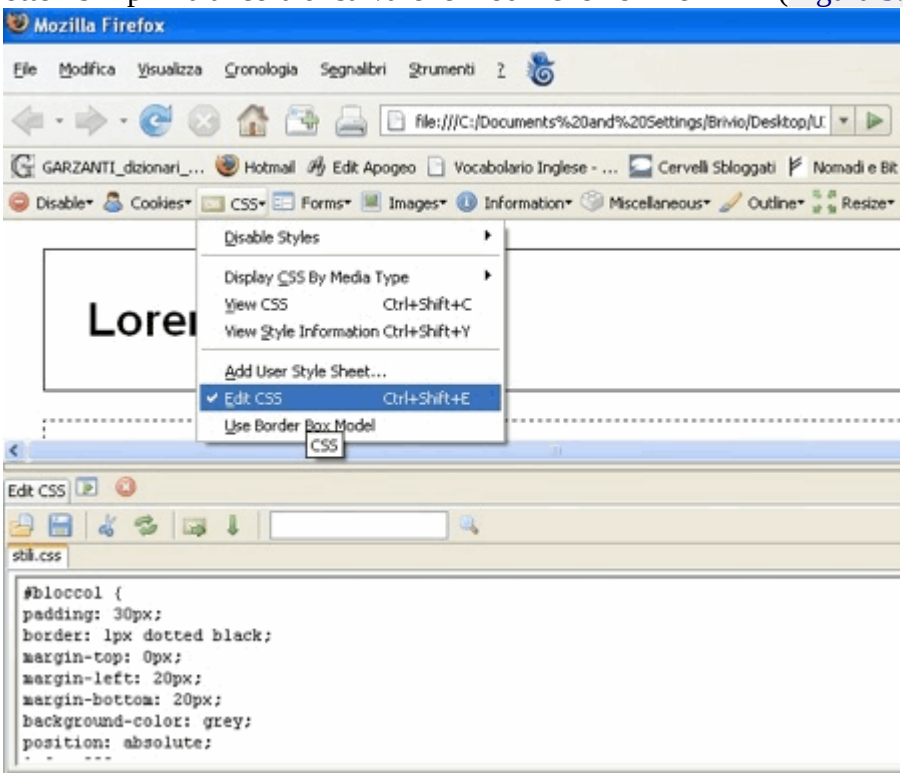


Figura 3.40 Il menu CSS di Web Developer Toolbar con attiva la funzionalità Edit CSS.

RIFERIMENTI *È possibile scaricare queste estensioni ricercandole dalla pagina* Componenti aggiuntivi di Firefox *(*https://addons.mozilla.org/it/firefox/*) oppure direttamente agli indirizzi* https://addons.mozilla.org/it/firefox/addon/6622 *(DOM Inspector) e* https://addons.mozilla.org/it/fire-fox/addon/60 *(Web Developer Toolbar).*

Fogli di stile e word processor

Le modalità con cui i CSS operano sulla presentazione delle pagine web non sono circoscritte al "pacchetto" browser/HTML/CSS.

Nel Capitolo 1 abbiamo introdotto i tre livelli dell'informazione facendo anche riferimento a word processor come Microsoft Word: Word, come qualsiasi altro editor di testo, permette di lavorare sulla struttura e sulla presentazione delle informazioni. E la maniera con cui gli editor di testo (e anche i software di impaginazione) gestiscono i contenuti ha molti punti di contatto con quanto visto per i CSS.

Per rendersene conto basta aprire il menu *Formato* di Word ed esplorare le varie possibilità di intervento offerte: *Carattere*, *Paragrafo*, *Bordi*, *Sfondo* e così via, le stesse incontrate illustrando le proprietà CSS.

Per esempio, dalla finestra *Paragrafo* (Figura 3.41) è possibile intervenire sui margini superiore e inferiore (voce *Spaziatura*) e destro e sinistro (voce *Rientri*); oppure dalla finestra *Bordi e sfondo* è possibile specificare spessore, tipo e colore di un bordo da applicare a uno o a più elementi del documento.

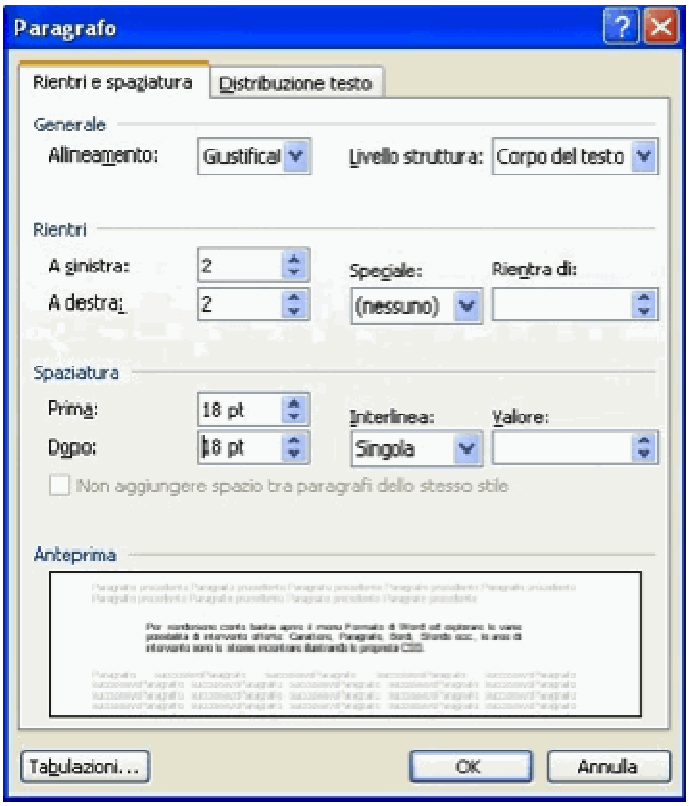


Figura 3.41 La finestra Paragrafo di Microsoft Word.

Il box model trova quindi applicazione e rimane valido anche nel più comune documento di testo, a prescindere dal software utilizzato per gestirlo, sia esso Word, un altro editor di testo o un software

specifico per l'impaginazione (come Adobe InDesign).

Comprendere il box model permette quindi un uso migliore di Word (come di altri editor), non passivo e viziato dall'apparenza, ma attivo e consapevole.

E le affinità non si fermano a questo. Anche nel flusso e nel posizionamento dei testi, i word processor distinguono tra elementi di blocco e in linea. La differenza è che gli elementi di blocco (negli editor come Word) sono definiti generalmente "paragrafi". Quindi è anche possibile intervenire sul posizionamento dei singoli paragrafi o delle immagini. Per esempio, la scheda *Layout* della finestra *Formato immagine* permette di scegliere tra diversi posizionamenti (Figura 3.42). I nomi non diranno niente, ma i risultati ottenibili sono affini a quanto già visto. La Figura 3.43 mostra che cosa si può ottenere scegliendo come stile dell'immagine *Incorniciato*: un posizionamento flottante dell'immagine.

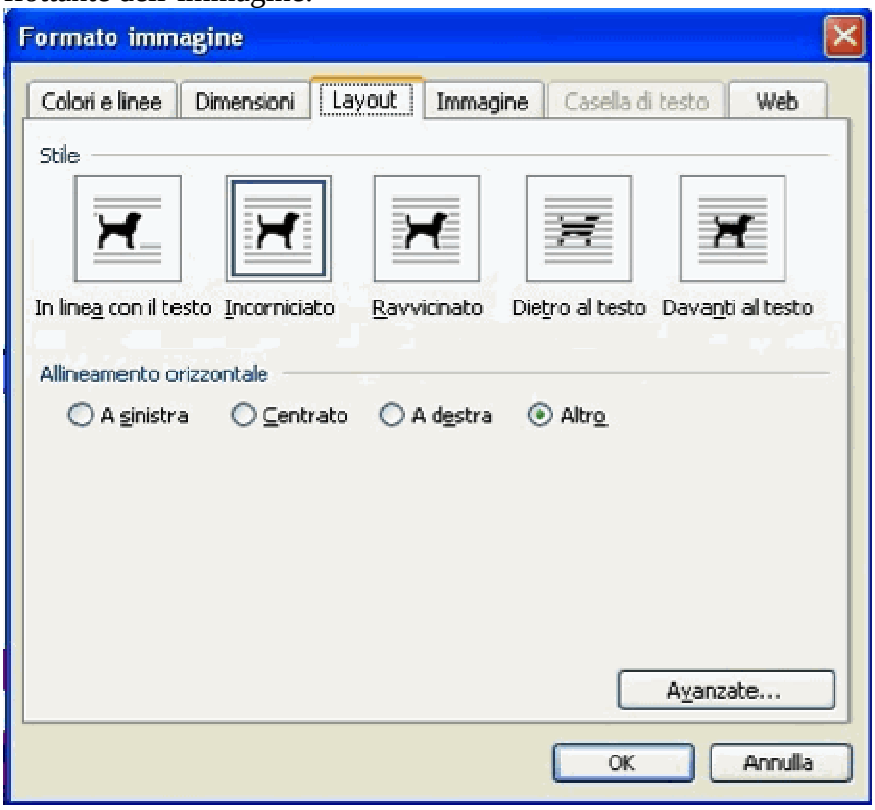


Figura 3.42 Definizione del tipo di posizionamento di un'immagine.

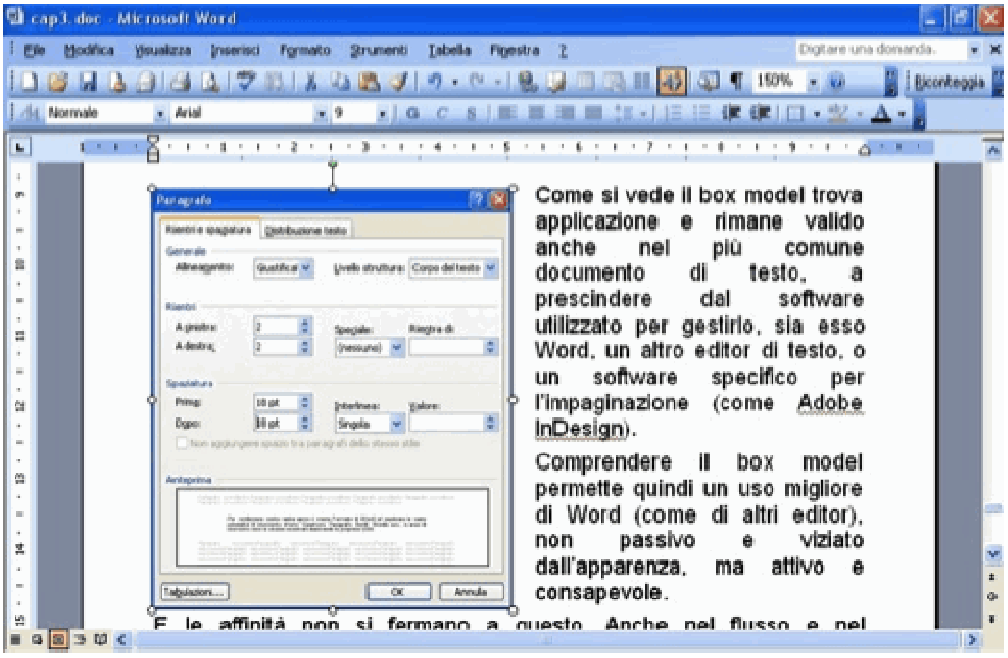


Figura 3.43 Posizionamento flottante di un'immagine in un documento di Word.

Liberarsi dalla forma dell'informazione

Il primo vantaggio del testo digitale è senza dubbio la sua flessibilità, da cui deriva una maggiore possibilità di utilizzo, o riutilizzo, del contenuto.

L'operazione più comune con un testo digitale è senza dubbio il copia e incolla.

Alla luce di quanto fin qui detto dovrebbe però ormai essere chiaro che a essere copiato non è solo il contenuto, ma anche la sua struttura e la sua forma. Questo è sia un vantaggio sia uno svantaggio, dipende dal contesto di lavoro.

Se si copia e si incolla dallo stesso documento (per esempio per spostare un paragrafo), le informazioni di stile entro cui si opera sono comuni e condivise, quindi normalmente non si hanno problemi.

Se invece si copia un testo da una pagina web per incollarlo in un editor di testo (come il più volte citato Word), in esso verranno importate anche le informazioni di stile e ciò potrebbe dare origine a delle anomalie o a dei problemi di formattazione, a volte anche di difficile gestione.

Per essere sicuri di non sporcare stilisticamente un documento bisogna quindi ripulire dagli stili il testo che si desidera importare.

Un modo pratico – anche se poco ortodosso – consiste nell'incollare una prima volta il testo copiato in un editor come Blocco note che, come ricordato nel Capitolo 1, lavora su testi non formattati, quindi privi di informazioni di struttura e presentazione. Quindi il testo può essere ricopiato e incollato dove serve: le uniche informazioni di stile saranno quelle del documento di destinazione, e al testo verrà automaticamente assegnato lo stile in uso nel paragrafo dove verrà collocato.

Syndication: linguaggi per la distribuzione delle informazioni

Al termine di questo lungo capitolo accenniamo ad alcuni linguaggi che da diversi anni stanno

ridefinendo le modalità di fruizione dei contenuti nel Web. Essi fanno parte di un fenomeno ampio che prende il nome di *syndication*.

Questo termine inglese un po' misterioso nasconde una grande rivoluzione nel modo di fruire i contenuti. Applicato al Web può essere tradotto con "distribuzione". La *web syndication* è prima di tutto questo: la distribuzione di contenuti o flussi di informazioni (*feed*) in una forma adatta a essere riutilizzata in formati e ambiti diversi dal sito che li distribuisce.

Ma c'è di più. Fino ad alcuni anni fa, chi utilizzava il Web come fonte informativa doveva tornare costantemente sul sito di interesse per verificare la presenza di news e aggiornamenti.

La syndication ha invertito questa tendenza permettendo di verificare la presenza di aggiornamenti interessanti senza dover tornare alla fonte.

Questo si traduce in un indubbio vantaggio quando le fonti da monitorare, magari più volte al giorno, sono molte.

Nella pratica ciò avviene attraverso la sottoscrizione di un *feed*, cioè del flusso di informazioni rilasciato da un sito.

Da un punto di vista tecnico un feed non è altro che un file scritto utilizzando una specifica applicazione XML (si veda il Capitolo 2). Questo file contiene le ultime notizie presenti su un sito in ordine temporale decrescente: l'ultima novità è in testa all'elenco.

NOTA *Non esiste un limite alle notizie che possono apparire in un feed, ma normalmente si incontrano feed di 10 o 20 news. Questa scelta è dettata sia da una questione di economia di gestione dei feed (che rimangono così più leggeri e semplici da processare), sia dal principale scopo della syndication, ossia favorire la distribuzione di contenuti aggiornati, non di tutti i contenuti possibili.*

Sono stati sviluppati diversi formati per i feed, tutti funzionali all'esigenza di distribuzione dei contenuti, ma i più noti sono RSS e Atom: ecco perché si sente parlare di feed RSS e feed Atom.

Scoprire se un sito distribuisce un feed è semplice: normalmente nella pagina principale è presente un'icona specifica (Figura 3.44) o una sua variante.

Figura 3.44 L'icona più diffusa per indicare la disponibilità di un feed.

Una volta individuata l'icona, è sufficiente selezionarla con un clic del mouse per aprire nel browser il feed stesso.

NOTA *La rappresentazione che i browser danno dei feed non è sempre uguale: alcuni si limitano a mostrare il documento nel linguaggio in cui è scritto (RSS, Atom e così via), altri ne offrono una rappresentazione più amichevole e intelligibile (Figura 3.45).*

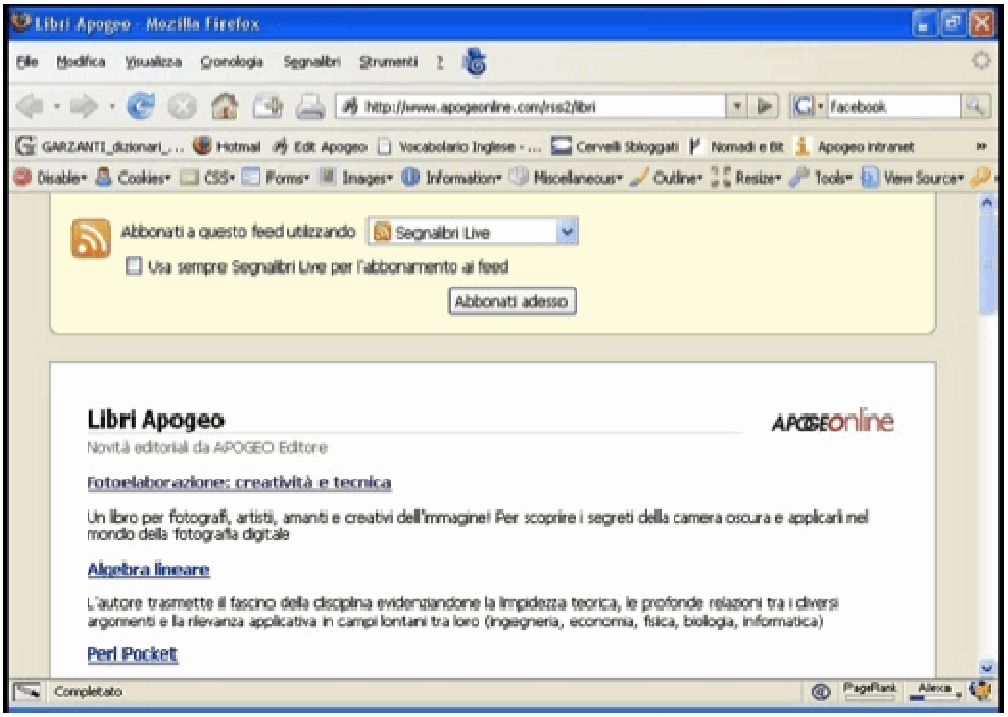


Figura 3.45 Un feed aperto con il browser Firefox.

Per utilizzare le potenzialità dei feed esistono appositi programmi noti come *lettori* (*reader*) o *aggregatori* di feed. Questi programmi permettono di creare collezioni di feed diversi, consultabili con facilità e velocità. L'operazione di aggiunta di un feed a una collezione è detta, come accennato poco fa, *sottoscrizione*.

SUGGERIMENTO *Qualche ricerca con Google, per esempio per "aggregatori feed", fornisce un'ampia panoramica di soluzioni accessibili. Una buona soluzione gratuita è Feed Reader (*`http://www.feedreader.com/`*), di cui è disponibile anche una localizzazione in italiano (*`http://www.feedreader.com/translations/Italian/`*). In alternativa è possibile utilizzare anche il browser Firefox, che integra una funzionalità detta Segnalibri Live ottima per gestire e consultare feed (Figura 3.46).*

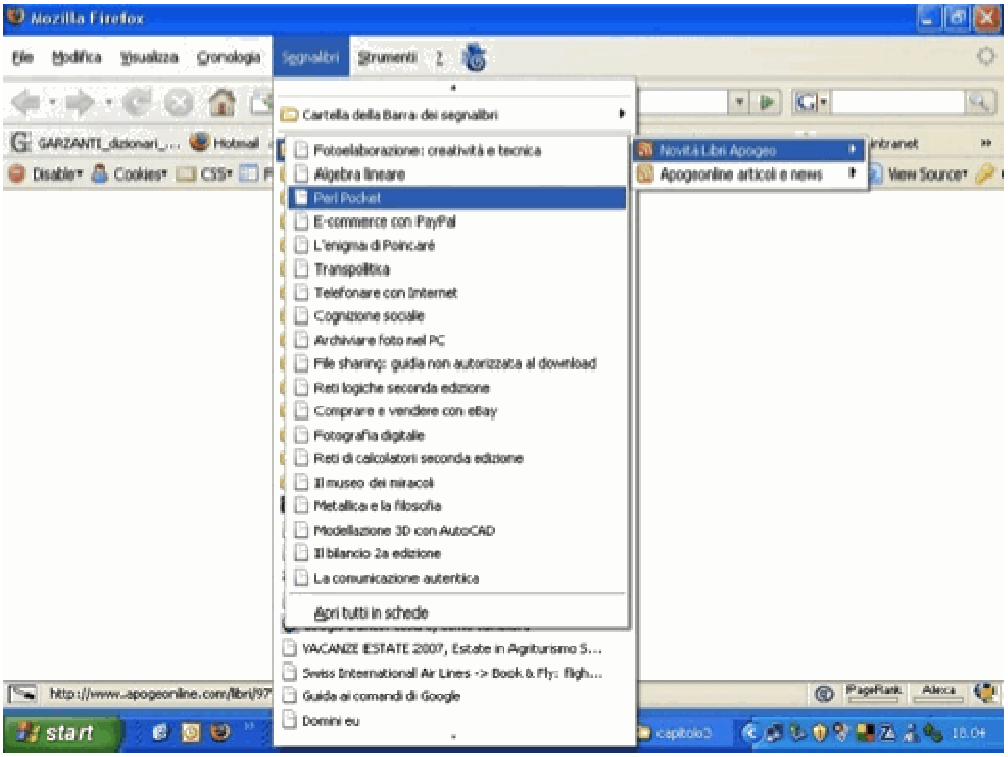


Figura 3.46 Segnalibri Live di Firefox: un feed appare tra i normali segnalibri ma posizionandosi su di esso con il puntatore del mouse mostra immediatamente i titoli delle news contenute. Con un clic si apre quindi la pagina della news di interesse.

Rss e Atom

Abbiamo detto che i feed sono file XML. Per essere più precisi, i feed sono quindi file il cui contenuto è strutturato secondo una specifica applicazione XML. Esistono diverse applicazioni XML per i feed, ma le più note sono RSS o Atom. Riallacciandoci al Capitolo 2, un feed RSS e un feed Atom per essere tali devono quindi essere scritti nel rispetto delle grammatiche RSS e Atom, o meglio dei DTD RSS e Atom.

RIFERIMENTI *La tecnologia RSS è attualmente giunta alla versione 2.0, la cui specifica ufficiale è consultabile all'indirizzo* `http://www.rssboard.org/rss-specification`*; una sua traduzione italiana è invece disponibile presso* `http://rss.specifiche.it/2.0/`*. La tecnologia Atom è attualmente giunta alla versione 1.0 e la sua specifica ufficiale è consultabile all'indirizzo* `http://tools.ietf.org/html/rfc4287`*.*

Non è possibile qui entrare nel dettaglio di queste grammatiche, ma osservando il frammento di feed che segue è possibile fare alcune osservazioni utili a comprendere meglio la syndication:

```
<channel>
  <title>…</title>
  <link>…</link>
  <description>…</description>
  <item>
    <title></title>
    <link></link>
    <pubDate></pubDate>
    <description></description>
  </item>
  <item>
  <!-- … -->
```

```
    </item>
  </channel>
```

Quello mostrato è il cuore e l'anima di un feed. Per prima cosa bisogna osservare come tutti i tag siano correttamente aperti e chiusi, nel rispetto della sintassi XML. Il primo tag `<channel>` contiene tutto il feed: i flussi di news oggetto della syndication sono quindi assimilabili a un canale di comunicazione.

NOTA *Un sito può distribuire anche più feed, ognuno dedicato a un tipo diverso di news. Per esempio, un quotidiano online potrebbe distribuire feed di politica, sport, cronaca, intrattenimento e così via fino a coprire tutta la tipologia di notizie trattate.*

Figli diretti di `<channel>` sono i tag `<title>`, `<link>`, `<description>` e `<item>`. I primi tre definiscono il canale e contengono un titolo (che a volte coincide col nome del sito), un URL (del sito e della sezione del sito da cui il feed è distribuito) e una descrizione breve dei contenuti. L'elemento `<item>` è invece dedicato alla singola news. Un feed contiene quindi un `<item>` per ogni news e ogni news è a sua volta definita per titolo, URL, data di pubblicazione e descrizione.

Su questa semplice struttura si basa in sintesi la syndication, anche se in realtà i tag delle specifiche RSS e Atom sono molti più di quelli qui presentati. Tuttavia attraverso questi pochi elementi è possibile già creare e distribuire feed funzionali ed efficaci.

La syndication è quindi un ottimo esempio di sinergia tra tecnologie: la flessibilità di XML e la sua capacità di operare su ogni piattaforma permettono di organizzare i contenuti in feed facilmente distribuibili e fruibili. La distribuzione si basa del resto sul protocollo HTTP e quindi può avvenire attraverso un normale sito web visualizzato da un comune browser.

Un discorso a parte meriterebbe la produzione del feed stesso che può essere generato da parser differenti in grado di lavorare su diverse sorgenti di dati. Per ora però ci fermiamo qui. Avremo modo di ritornare su questi aspetti della gestione delle informazioni nei prossimi due capitoli.

Per concludere

Questo lungo capitolo sulle lingue del Web ha messo in luce come l'informazione sia in esso gestita attraverso tecnologie implementate per lavorare in modo specifico sulla struttura, sulla presentazione e sulla comunicazione del dato.

La separazione dei livelli di struttura e presentazione è apparsa qui evidente e necessaria: solo così è possibile gestire in maniera economica siti di ogni dimensione e complessità, raggiungendo il maggior numero di utenti.

Nel prossimo capitolo parleremo invece di database, cioè archivi il cui scopo è semplicemente organizzare le informazioni al fine di renderle disponibili per scopi differenti, dalla semplice consultazione di un dato particolare, alla creazione di pagine web dinamiche o alla generazione di feed. Come vedremo, i database sono "struttura del dato" allo stato puro.

Capitolo 4

La lingua dei database

Nel Capitolo 2 abbiamo parlato della possibilità di organizzare dati utilizzando il linguaggio XML. Abbiamo anche detto che XML è una tecnologia tutto sommato giovane, quantomeno più giovane del Web. Tuttavia l'organizzazione delle informazioni è sempre stata uno dei principali scopi per cui le macchine sono state concepite.

Prima dell'entrata in scena di XML questa esigenza era gestita in altri modi. I *database* sono nati proprio per organizzare grandi quantità di dati e ancora oggi sono irrinunciabili quando le informazioni sono numerose e di diverso tipo.

NOTA *XML, in virtù della sua capacità di operare su ogni piattaforma, è un ottimo formato per lo scambio di dati strutturati, ma quando la mole di informazioni diviene importante e si differenzia, la gestione dei dati passa attraverso la loro archiviazione in database.*

Nello specifico i database sono archivi la cui struttura è concepita per organizzare e gestire determinati tipi di dati, consentendone per esempio l'inserimento, la modifica, l'aggiornamento e la cancellazione.

Possono quindi esistere database per contenere le informazioni su pezzi di ricambio d'auto o libri, voli e tariffe aeree o capi di abbigliamento. Questi database avranno strutture diverse, adatte a descrivere il tipo di dato (di volta in volta differente), ma tutti potranno essere gestiti nel medesimo modo, secondo la logica dei database e gli strumenti che i programmi mettono a disposizione per operare su una base di dati.

Anche nel caso dei database esistono programmi server e client e la comunicazione tra i due avviene nella stessa modalità che abbiamo descritto per il Web: il client effettua una richiesta (per esempio l'inserimento di un nuovo dato), il server la esegue e restituisce un risultato (un messaggio di conferma dell'avvenuto inserimento). Il protocollo utilizzato per la comunicazione tra client e server di database potrà variare, ma se il client è un'applicazione web (come molto spesso avviene), allora richiesta e risposta saranno inviate tramite il protocollo HTTP e il risultato sarà visualizzabile con un browser: un ulteriore esempio di come il Web sia il crocevia non solo di dati diversi, ma anche di diverse modalità – o protocolli – di comunicazione.

RIFERIMENTI *Esistono diverse soluzioni server e client per database. Tra i più diffusi server segnaliamo MySQL (`http://www-it.mysql.com`), potente e soprattutto gratuito in quanto sviluppato sotto licenza open source. MySQL viene spesso utilizzato con il client phpMyAdmin (`http://www.phpmyadmin.net`), distribuito sotto la medesima licenza: si tratta di un'applicazione web che una volta installata su un server web offre tutti gli strumenti utili alla gestione del database, dalla sua creazione, alla definizione degli utenti, fino al pieno controllo dei dati (Figura 4.1). Un altro famoso client è Access di Microsoft: si tratta di un applicativo desktop (cioè da installare sul computer da cui si desidera lavorare), abbastanza semplice da usare e dotato di un'interfaccia grafica per certi versi affine a quella degli altri applicativi della suite Office. Con Access è anche possibile creare piccoli database direttamente sul computer in uso, ma come ogni programma Microsoft è a pagamento. Ritornando ai server, non si può non ricordare Oracle (`http://www.oracle.com`), la scelta preferita dalle grandi industrie.*

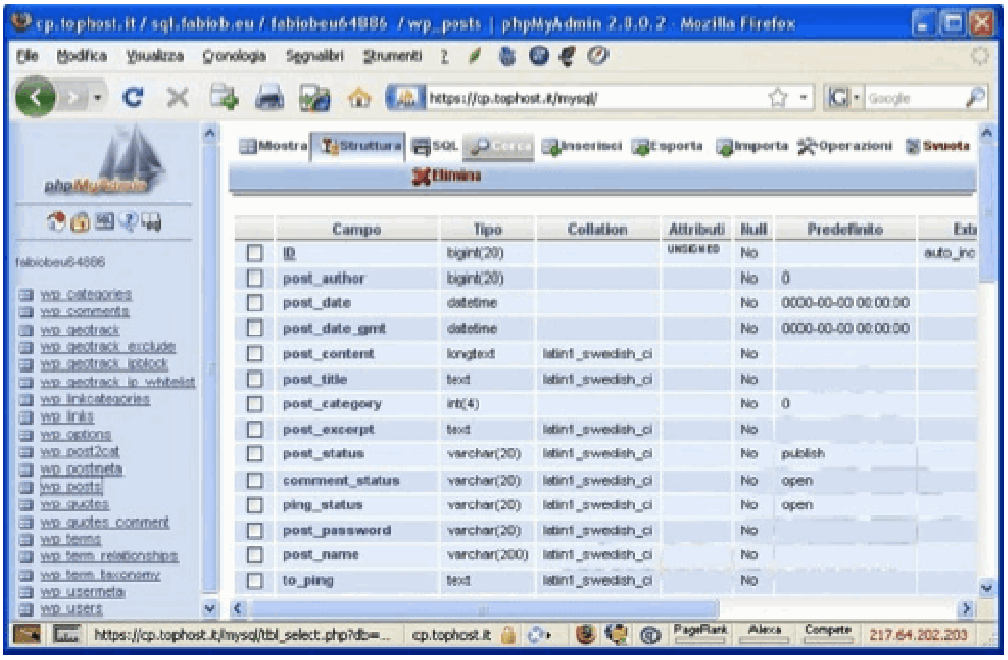


Figura 4.1 L'interfaccia di phpMyAdmin: in alto sono visibili alcuni degli strumenti disponibili.

Introduzione alla struttura e alla logica dei database

I database oggi in uso sono i cosiddetti *database relazionali*. Sono così chiamati perché permettono di organizzare i dati formulando delle relazioni tra loro.

Questo concetto è fondamentale e per capirlo dobbiamo soffermarci un attimo sulla struttura cardine di ogni database: la *tabella*.

A un livello molto semplice i database non sono altro che insiemi di tabelle al cui interno sono organizzati i dati.

NOTA *In virtù di quello che diremo tra poco, anche una sola tabella può, teoricamente, essere considerata un database. Tuttavia un database composto da una sola tabella avrebbe poco senso: non è necessario dotarsi di un server di database per gestire dati che stanno in una sola tabella. I database sono invece fondamentali per gestire dati compositi, suddivisi in più tabelle.*

Ogni tabella è composta da righe, o *record*, ed è suddivisa in colonne, o *campi* (Figura 4.2). Ogni riga contiene quindi una specifica informazione descritta attraverso i campi, che possono essere visti anche come gli *attributi* della tabella. Ecco un semplice esempio di tabella, nominata `Libri`.

ISBN	Autore	Titolo	Pagine	Prezzo
9788850325177	Eungénie Vegleris	Manager con la filosofia	224	13
9788850327171	Umberto Galimberti	Il segreto della domanda	224	13
9788850322985	Gerd B. Achenbach	Del giusto nel falso	192	13
9788850326723	Andrea Vitullo	Leadership riflessive	256	13

Figura 4.2 La struttura delle tabelle di un database non ha niente di diverso dalle "normali" tabelle: anche qui i dati sono organizzati in righe e colonne.

NOTA *Nei database ogni tabella deve avere un nome che la identifichi in maniera univoca.*

Nell'esempio appena riportato ogni riga contiene delle informazioni su un singolo libro, e cioè il codice ISBN (il codice identificativo internazionale utilizzato per i libri), l'autore, il titolo, le pagine, il prezzo: ognuno di questi valori è inserito nella relativa colonna.

In questo modo le informazioni sono messe in relazione tra di loro all'interno di una tabella.

SUGGERIMENTO *I campi di una tabella dovrebbero essere organizzati nella maniera più atomica possibile.*

Questo avviene quando un'informazione all'interno di un campo non può essere ulteriormente scomposta. Nel nostro esempio questa condizione non viene rispettata dal campo Autore*, che potrebbe essere ulteriormente scomposto in* Nome *e* Cognome*. La strutturazione atomica, o granulare, di tutte le tabelle di un database, ne permette uno sfruttamento pieno ed efficace. Differentemente il database è comunque utilizzabile ma alcune analisi sui suoi dati possono essere limitate o compromesse.*

TERMINOLOGIA *La granularità del dato è solo uno dei criteri che rendono un database perfettamente strutturato. Chi lavora sulla gestione e sull'implementazione di basi di dati si trova spesso nella necessità di ottimizzare tabelle mettendo in pratica questi criteri. In questo caso si parla di* normalizzazione di un database*.*

Come abbiamo però detto, i database sono composti da più tabelle, ognuna delle quali contiene informazioni specifiche. Per esempio, un negozio di libri potrebbe aver bisogno di tabelle per gestire i clienti e i relativi ordini, gli sconti, le giacenze e quindi la disponibilità dei titoli, gli editori da cui si rifornisce e così via. Tutte queste informazioni sono in relazione una con l'altra e quindi le tabelle del database che le contengono – o meglio i record di queste tabelle – dovranno relazionarsi tra di loro. Solo attraverso queste relazioni sarà possibile sapere, per esempio, che un certo libro appartiene al catalogo di un editore, è disponibile in una certa quantità ed è stato venduto in un dato numero di copie, magari con uno sconto particolare.

Ma come sono possibili queste relazioni?

Per rispondere a questa domanda dobbiamo introdurre un altro concetto, quello di *chiave* di una tabella.

La chiave di una tabella è un campo i cui valori sono diversi in ogni record. Il valore del campo chiave distingue quindi in maniera univoca ogni riga e rende possibile l'instaurarsi di relazioni tra tabelle. Una chiave così intesa è detta *chiave primaria*.

SUGGERIMENTO *Ogni tabella dovrebbe sempre avere una chiave primaria e non dovrebbero mai esistere duplicati di record (cioè righe esattamente identiche per tutti i valori dei campi: è sufficiente che un solo campo sia diverso per differenziare due record). Quando un'informazione è duplicata in maniera identica in due o più record si crea una ridondanza, l'informazione archiviata diviene sporca e la leggibilità e l'utilizzo del dato vengono limitati o compromessi. L'applicazione di chiavi primarie contribuisce a scongiurare questa eventualità. Individuare la chiave primaria di una tabella e assicurarsi che sia stata correttamente applicata è consigliabile quando si comincia a lavorare sui dati di un database esistente, soprattutto se di medie e grandi dimensioni.*

Nel nostro esempio l'unico campo che può svolgere il ruolo di chiave primaria è ISBN: ogni libro ha un codice `ISBN` univoco che permette di distinguerlo dagli altri. Gli altri campi non possono essere chiavi della tabella, in quanto un autore potrebbe essere tale anche per un altro libro e due o più libri potrebbero avere lo stesso titolo, numero di pagine e prezzo.

ATTENZIONE *Esiste anche la possibilità di connotare in maniera univoca i record attraverso chiavi costituite dai valori di più campi. In questo caso si parla di* chiave composta*. L'utilizzo di una chiave composta non è comune, ma è comunque utile sapere che esiste questa eventualità.*

Torniamo quindi alle relazioni tra tabelle. Partendo dalla chiave primaria di una tabella è possibile instaurare relazioni con altre tabelle a patto che un campo di queste tabelle sia anch'esso compilato con i valori della chiave primaria della prima tabella.

TERMINOLOGIA *All'interno di una relazione, i valori della chiave primaria di una tabella utilizzati nella tabella correlata svolgono il ruolo e sono definiti* chiave esterna*.*

In altri termini, la chiave primaria in una tabella (tabella A) è quindi condizione essenziale per l'instaurarsi una relazione con una seconda tabella (tabella B). Tale relazione non può però avvenire se nelle tabella B non esiste un campo che contiene i medesimi valori della chiave primaria della

tabella A. La Figura 4.3 chiarisce meglio questo fondamentale concetto.

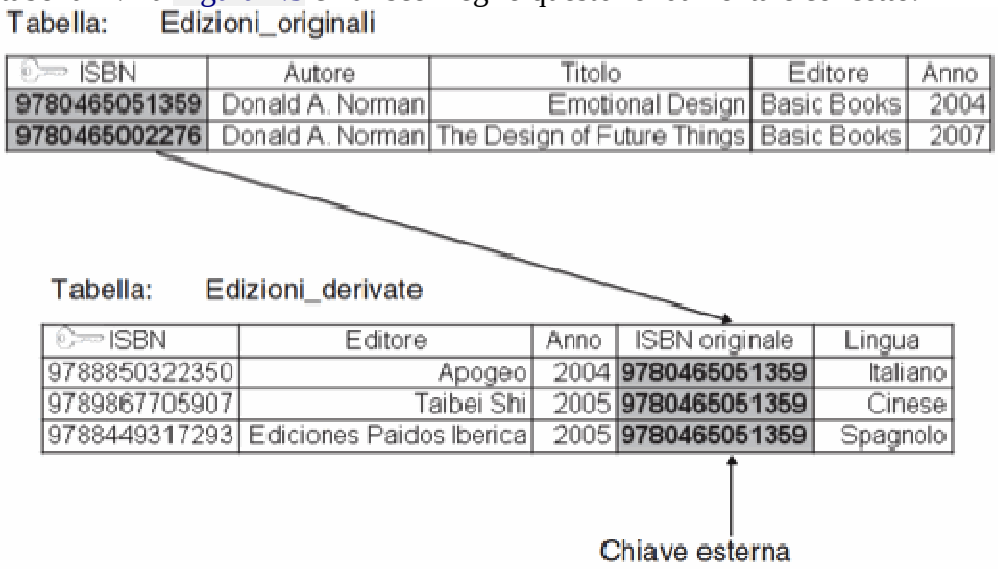


Figura 4.3 Il ruolo di una chiave primaria e di una chiave esterna in una relazione tra due tabelle.

Le tabelle di questo esempio hanno lo scopo di tracciare la relazione tra un'edizione originale di un libro e le sue edizioni derivate, cioè le sue possibili traduzioni in altre lingue.

La tabella `Edizioni_originali` contiene quindi alcune informazioni sul libro: l'autore, il titolo, l'editore e l'anno di pubblicazione. Ogni record è, ancora una volta, distinto dal codice ISBN, che svolge il ruolo di chiave primaria.

La tabella `Edizioni_derivate` contiene invece le informazioni sulle traduzioni: l'editore, la lingua e l'anno di pubblicazione. Anche qui il ruolo di chiave primaria è svolto dal codice ISBN, ma attenzione: in questo caso si tratta del codice dell'edizione tradotta, mentre nella colonna `ISBN_originale` è presente il codice ISBN della tabella `Edizioni_originali`, che svolge il ruolo di chiave esterna della relazione e che quindi permette di risalire ai dati dell'edizione da cui una traduzione è derivata.

NOTA *Vale la pena sottolineare che poiché i valori di* `ISBN_originale` *non sono univoci, in quanto un libro può essere tradotto in più lingue, questo campo non può svolgere il ruolo di chiave primaria nella tabella* `Edizioni_derivate`*.*

I tipi di dato

Compresa la struttura cardine dei database, soffermiamoci ora un momento sul tipo di dato che può essere inserito nelle varie tabelle.

Senza scendere nel dettaglio, è qui sufficiente sapere che i campi dei database possono essere compilati con:

- *stringhe*, cioè sequenze alfanumeriche di caratteri, in pratica del testo;
- *numeri* interi, decimali, positivi e negativi;
- *date*, composte da giorni, mesi, anni ma anche ore, minuti e secondi;
- valori *booleani*, cioè `1` e `0` (vero o falso, acceso o spento, `true` o `false`).

NOTA *Questi sono i principali tipi di dato con cui sono più frequentemente compilate le tabelle di un database. Ne esistono comunque altri e ogni tipo – in sede di creazione o modifica di una tabella – può essere definito in modo diverso (per esempio stringhe di soli 32 caratteri, numeri interi di 4 cifre, date nel formato* ANNO/MESE/GIORNO> *e così via).*

I database prevedono poi la possibilità di assegnare a un campo il valore di NULL in assenza di altro valore. In pratica possono esistere record in cui non tutti i campi hanno un valore (per esempio un libro di cui non si conosce l'editore). NULL permette di esprimere questa condizione di "non valore". È bene sottolineare che NULL non è uguale a una stringa vuota (cioè composta da zero caratteri), al valore numerico 0, alla data 00/00/0000, o al valore booleano 0; semplicemente NULL "non è", e per questo non può essere assimilato ai tipi di dato sopra ricordati.

RIFERIMENTI *L'importanza di questa differenza apparirà più chiara tra qualche pagina, quando vedremo la selezione di risultati attraverso l'utilizzo di alcuni operatori specifici.*

Relazioni tra tabelle

Passiamo ora in esame i tipi di relazione che possono instaurarsi tra due tabelle nel momento in cui sono dotate di chiavi primarie e campi che possono svolgere il ruolo di chiavi esterne.

Esistono tre tipi di relazioni: *uno a uno*, *uno a molti* e *molti a molti*.

La relazione di tipo *uno a uno* si ha quando un record di una tabella può essere correlato a un solo record in un'altra tabella.

Nella realtà una relazione di questo tipo può essere tranquillamente espressa in una sola tabella, con tutti i dati interessati inseriti tra i campi di uno stesso record: un record esprime già di per sé una relazione di dati. Se però i campi sono molti o i dati di diverso tipo, allora per semplicità di gestione si può scegliere di dividerli in più tabelle. Per esempio, il database di un editore potrebbe avere una tabella `Libri` (con i campi `Titolo`, `Autore`, `Pagine` e così via) e una tabella Giacenza (con la quantità di copie in giacenza in magazzino). Un libro può avere solo una giacenza e quindi questo valore potrebbe tranquillamente essere un campo della tabella `Libri`, ma considerando che le informazioni della tabella `Libri` sono normalmente gestite dalla redazione, mentre le giacenze dai responsabili del magazzino, potrebbe avere senso separare l'informazione lasciando a ogni reparto il compito di gestire una tabella: alla redazione `Libri`, al magazzino la Giacenza. In un secondo momento attraverso la chiave primaria ISBN sarà sempre possibile mettere in relazione le informazioni.

La relazione di tipo *uno a molti* si ha quando ogni record di una tabella può essere relazionato a nessuno, uno o molti record in un'altra tabella, i cui record hanno però un solo record di riferimento nella prima tabella.

Un esempio può essere visto nella Figura 4.3: ogni libro della tabella `Edizioni_originali` può essere relazionato a zero, uno o più libri nella tabella `Edizioni_derivate` in quanto un libro può essere tradotto in nessuna, una o molte lingue. Ma allo stesso tempo ogni libro della tabella `Edizioni_derivate` può essere relazionato a un solo libro nella tabella `Edizioni_originali`, in quanto un libro può avere solo una edizione originale, la prima, da cui vengono derivate tutte le traduzioni (Figura 4.4).

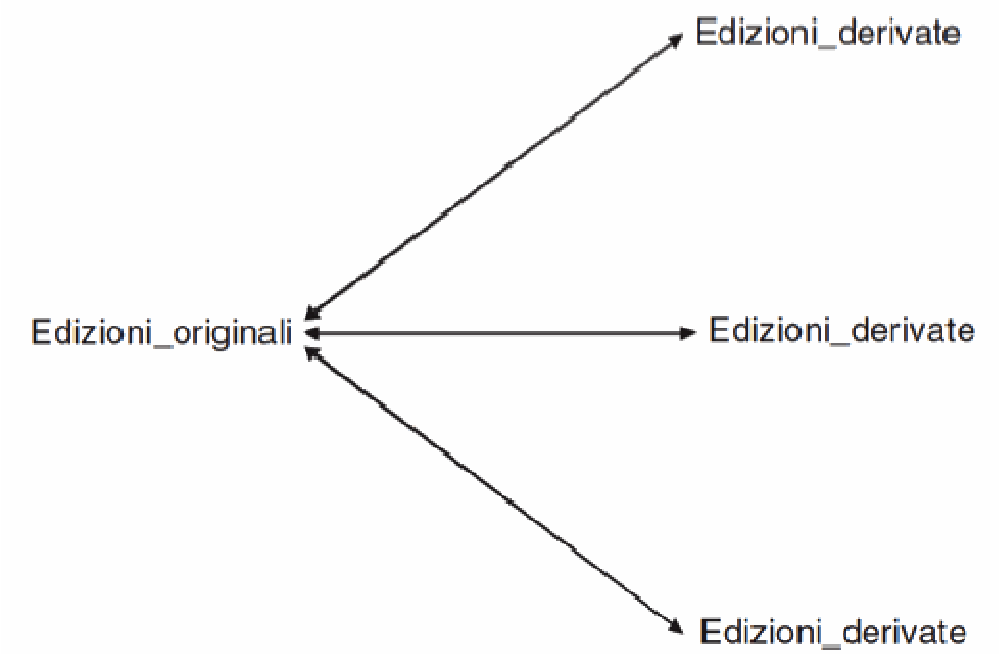


Figura 4.4 Relazione uno a molti: solo i record di una tabella possono avere molte relazioni.

Infine, la relazione di tipo *molti a molti* si ha quando ogni record in entrambe le tabelle può avere molti record relazionati. Possiamo trovare un esempio nelle relazioni tra due ipotetiche tabelle `Libri e Argomenti`. Un libro può avere uno o più argomenti e ogni argomento può essere associato a uno o più libri. La particolarità di questa relazione consiste nel fatto che nei database essa viene implementata attraverso una terza tabella, detta di *raccordo* o *congiunzione*.

La tabella `Libri` avrà una chiave primaria sul campo `ISBN` mentre la tabella `Argomenti` potrà essere composta da due soli campi e cioè `Descrizione` (l'argomento vero e proprio, per esempio, "Storia", "Filosofia", "Informatica" e così via) e `Codice` (un codice numerico univoco, per esempio 1001, 1002, 1003 e così via) che svolgerà il ruolo di chiave primaria. La tabella di raccordo, `Libri_Argomenti`, potrà così essere composta da due soli campi (ISBN e Codice) e la relazione molti a molti scomposta in due relazioni uno a molti instaurate tra la tabella `Libri` e la tabella `Libri_Argomenti`, e tra la tabella Argomenti e – di nuovo – la tabella `Libri_Argomenti`.

In pratica ogni libro potrà essere relazionato a uno o molti record nella tabella `Libri_Argomenti`, mentre ogni record di questa tabella avrà una sola relazione nella tabella `Libri`. Analogamente ogni argomento potrà essere relazionato a uno o molti record nella tabella `Libri_Argomenti`, e ogni record di questa tabella avrà una sola relazione nella tabella `Argomenti` (Figura 4.5).

Figura 4.5 Una relazione molti a molti viene scomposta in due relazioni uno a molti con l'aiuto di una terza tabella.

In questo modo la relazione molti a molti viene completamente rispettata e lavorare sui dati coinvolti diviene più semplice: la tabella di raccordo esplicita la relazione tra le due tabelle sulle quali è possibile concentrarsi di volta in volta per gestire le informazioni, aggiungendo, cancellando e modificando libri e argomenti.

Importare ed esportare dati

Lavorando sulle basi di dati, una delle operazioni che capita spesso di dover svolgere è l'importazione o l'esportazione di dati.

Ancora prima di introdurre il linguaggio dei database, è utile aprire una parentesi sulle possibilità naturalmente derivate dal fatto che l'elemento fondante ogni base di dati non è altro che una tabella.

In virtù di questo, nei database è possibile importare ed esportare dati in diversi formati di file, purché a struttura tabellare. Tanto per ricordarne alcuni: fogli di calcolo di Excel, file composti da tabelle HTML e naturalmente XML.

NOTA *La lista potrebbe continuare includendo file* `.txt` *con i dati separati da tabulazione o file* `.csv` *(estensione il cui acronimo significa* Comma Separated Values*), che altro non sono che file di testo semplice in cui i dati sono separati da un carattere definito dall'utente (normalmente la virgola). Per questi due tipi di file, la tabulazione e il carattere di separazione permettono al parser utilizzato di individuare senza ambiguità i dati contenuti nei vari campi dei record.*

Per quanto riguarda i file XML, alla luce di quanto visto nel Capitolo 2, dovrebbe apparire chiaro come una tabella possa essere facilmente riproposta attraverso questo linguaggio. In pratica il nome della tabella diviene l'elemento radice e ogni record un suo elemento figlio, che a sua volta contiene tanti elementi quanti sono i campi della tabella.

Ecco la riproposizione in XML della struttura della tabella vista nella Figura 4.2:

```
<Libri>
  <Libro>
    <ISBN>9788850327171</ISBN>
    <Autore>Umberto Galimberti</Autore>
    <Titolo>Il segreto della domanda</Titolo>
    <Pagine>224</Pagine>
    <Prezzo>13</Prezzo>
  </Libro>
  <Libro>
  <!-- Altri elementi/record -->
  </Libro>
</Libri>
```

NOTA *In pratica i file XML possono anche essere visti come database testuali che godono però di tutti i vantaggi propri di XML, tra cui quello – non trascurabile – di essere condivisibili a prescindere della disponibilità di un server e di un client specifici.*

Ma come è possibile praticamente importare ed esportare dati?

Nel Capitolo 1 abbiamo detto che gli editor permettono di lavorare sui dati. Nel caso dei database i client svolgono anche le funzioni di editor, permettendo quindi non solo di interrogare i server dove risiedono le basi di dati, ma anche di intervenire sul dato stesso, modificandolo, cancellandolo e anche esportandolo o importandolo.

I client integrano quindi funzionalità di importazione ed esportazione, come mostra l'esempio visibile nella Figura 4.6.

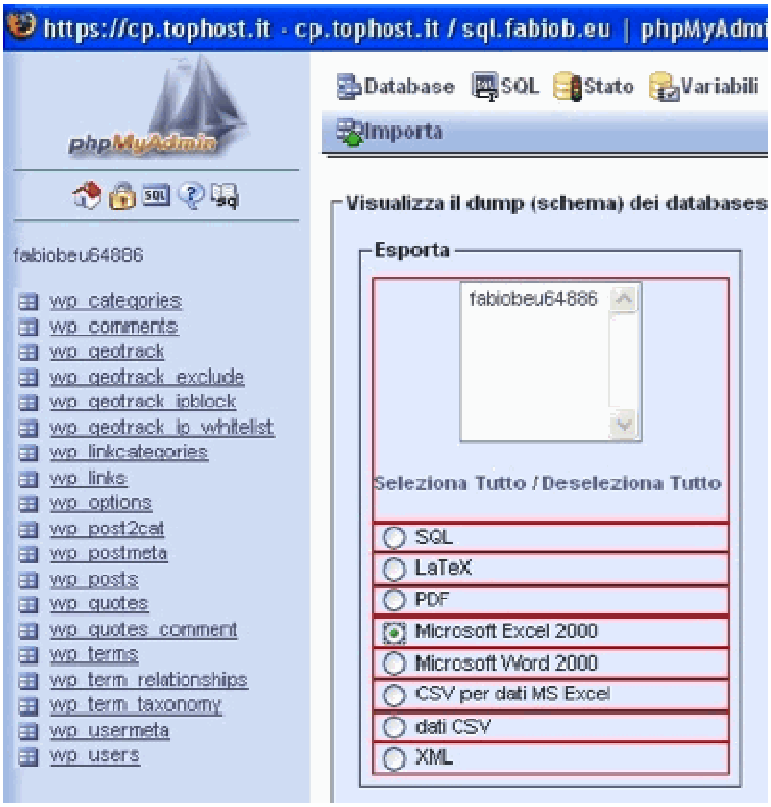


Figura 4.6 A destra, le funzionalità di esportazione di phpMyAdmin.

Comunicare con i database

Fino a questo punto abbiamo accennato ai criteri che dovrebbe rispettare un database e ai tipi di relazioni che possono esistere tra i dati. Queste informazioni dovrebbero essere sufficienti per cominciare a lavorare sulle basi di dati utilizzando il loro linguaggio, che vedremo tra poco.

Lo scopo di questo capitolo non è insegnare a progettare, creare e gestire database relazionali *ex-novo*, ma semplicemente imparare a comunicare con essi. Saper estrarre l'informazione che serve, quando serve, è un'abilità non trascurabile nel mondo del lavoro.

Tuttavia è bene sottolineare che la progettazione della struttura di un database è frutto di riflessione accurata e che più che conoscenze informatiche in senso stretto sono qui necessarie conoscenze di logica: per utilizzare un'informazione in maniera creativa e costruttiva è prima di tutto necessario scomporla e organizzarla in una maniera coerente e rispettosa delle relazioni che possono esistere tra i vari dati che la compongono.

Il linguaggio dei database

Tutti i database relazionali supportano e riconoscono il linguaggio SQL (*Structured Query Language*). Si tratta di un linguaggio molto ampio attraverso cui è possibile non solo interrogare un database, ma anche modificare, inserire e cancellare dati, come tabelle o intere basi di dati.

Nelle prossime pagine vedremo come utilizzarlo per svolgere interrogazioni su una o più tabelle contemporaneamente, ma anche per aggiornare, modificare e cancellare record. In questo modo introdurremo le basi della sua sintassi che si compone di *comandi*, *clausole*, *parole chiave* e *operatori*.

RIFERIMENTI *Chi volesse approfondire lo studio di SQL può fare riferimento alla numerosa documentazione*

disponibile in Rete, oppure, come libro, a SQL Pocket *di Marco Ferrero, Apogeo, 2004.*

L'utilizzo combinato di comandi, clausole e operatori permette di creare istruzioni SQL, normalmente definite *query*.

NOTA *Si noti come il concetto di "query" sia comune a tutti i sistemi informatici o informativi. Solo per rimanere nei casi visti in questo libro, una query è una richiesta inviata da un client a un server, un'istruzione SQL e un'interrogazione a un motore di ricerca.*

Come si vedrà, nella sintassi SQL sono ricorrenti termini inglesi. Originariamente esso doveva infatti assomigliare più a un linguaggio naturale che a un linguaggio informatico. Anche se questo progetto, un po'troppo ambizioso, non si è realizzato, SQL, almeno in alcune semplici query, ricorda un linguaggio umano e ciò costituisce senza dubbio un vantaggio per il suo apprendimento.

Query di selezione

Il comando su cui costruire query per selezionare dati da una tabella è `SELECT`.

NOTA *Per convenzione i comandi, le clausole e gli operatori SQL verranno scritti in maiuscolo, in modo da distinguerli dagli altri elementi di una query (cioè i nomi di campi e tabelle). Nella pratica è però possibile scriverli sia in minuscolo, sia in maiuscolo. Per quanto riguarda invece i nomi delle tabelle e di campi, essi possono essere scritti rispettando o meno il* case *(maiuscolo o minuscolo) a seconda del tipo di database su cui si lavora. Alcuni database prestano infatti importanza al case, altri no: è comunque buona norma abituarsi a scrivere le query rispettando il case dei nomi di campi e tabelle.*

Questo comando va usato con la clausola `FROM`, seguendo questa sintassi:

```
SELECT campi_tabella FROM tabella
```

La query si apre con il comando `SELECT` seguito dall'elenco dei campi dai quali si vuole estrarre il valore, quindi FROM specifica su quale tabella la query deve agire. Ecco tre esempi riferiti alla tabella `Edizioni_derivate` della Figura 4.3.

RIFERIMENTI *Da qui in avanti le tabelle visibili nella Figura 4.3 saranno il riferimento per tutte le query di esempio.*

```
SELECT Editore FROM Edizioni_derivate
SELECT * FROM Edizioni_derivate
SELECT ISBN, Editore, Lingua FROM Edizioni_derivate
```

La prima query estrae tutti i valori contenuti nel solo campo Editore.

La seconda query utilizza invece il simbolo asterisco (`*`) per estrarre tutti i valori di tutti i campi: in pratica l'intero contenuto della tabella. Infine, la terza query estrae tutti i valori contenuti nei campi `ISBN`, `Editore` e `Lingua`. L'elenco dei campi viene scandito dalla virgola e i valori sono recuperati e presentati secondo questo elenco, che non deve necessariamente coincidere con la reale struttura della tabella: in pratica è possibile effettuare estrazioni di dati secondo un ordine arbitrario (per esempio `Editore`, `Lingua`, `ISBN`, come mostrato nella Figura 4.7).

Figura 4.7 Una query (in alto) eseguita tramite phpMyAdmin: nei risultati (in basso) l'ordine dei campi non rispecchia quello effettivo della tabella.

SUGGERIMENTO *La virgola non deve essere presente tra l'ultimo campo dell'elenco e la clausola* `FROM`.

Alias

SQL permette anche di cambiare i nomi dei campi nei risultati delle query. Questo è possibile attraverso la parola chiave AS. Ecco un esempio:

```
SELECT ISBN AS Codice_libro, Lingua
FROM Edizioni_derivate
```

Questa query estrae tutti i valori contenuti nel campo `Lingua` e `ISBN`, ma quest'ultimo viene presentato nei risultati con l'etichetta `Codice_ libro`.

NOTA AS *agisce solo sui risultati e non sulla tabella;* AS *non cambia quindi il nome ai campi di una tabella, ma semplicemente li mostra con un'intestazione diversa tra i risultati di una query. La sua utilità si avverte in presenza di campi con un nome criptico, oppure nel caso (che vedremo nelle prossime pagine) di utilizzo di funzioni all'interno di una query. È possibile utilizzare* AS *anche su tutti i campi coinvolti in una query.*

Ordinamento dei risultati

Un migliore controllo dei risultati può essere fatto introducendo un ordinamento. Per questo è possibile utilizzare la clausola `ORDER BY`, in combinazione con le parole chiave `ASC` o `DESC`, in questo modo:

```
SELECT ISBN, Editore, Lingua FROM Edizioni_derivate
ORDER BY Editore DESC
```

Questa query elenca i risultati ordinati per i valori del campo `Editore` in ordine alfabetico decrescente (Z-A). Con `ASC` si sarebbe ottenuto l'ordinamento opposto (A-Z).

NOTA *In generale* `ASC` *ordina dal più piccolo al più grande e* `DESC` *dal più grande al più piccolo.*

In pratica in questo modo è possibile ordinare i risultati in base ai valori di un campo che svolge il ruolo di *chiave di ordinamento*.

ATTENZIONE *La chiave di ordinamento può essere espressa con il nome di un campo o con un suo eventuale alias.*

La chiave di ordinamento può anche essere composta da più campi, separati da virgola. Ecco come:

```
ORDER BY Editore, Anno DESC
```

In questo caso i risultati vengono prima ordinati per `Editore` e quindi ogni gruppo di record riferiti a un singolo editore viene ordinato per Anno, sempre secondo un criterio decrescente.

Funzioni

Effettuando una query di selezione è anche possibile utilizzare delle funzioni che permettono di ottenere come risultato non il dato puro, ma elaborato.

Esistono diversi tipi di funzioni in grado di operare su un singolo dato o su un insieme di dati (per esempio tutti i valori di una colonna).

TERMINOLOGIA *Le funzioni che lavorano su più dati vengono dette di* aggregazione *e restituiscono un singolo dato ricavato dall'insieme dei dati oggetto di aggregazione, mentre quelle che lavorano su un singolo dato sono dette* scalari*.*

SQL prevede diverse funzioni "native", mentre altre – specifiche – sono implementate e supportate dai singoli server di database (MySQL, Oracle e così via). Questo vuol dire che a seconda del tipo di database con cui si ha a che fare è possibile utilizzare oltre alle funzioni SQL native anche altre funzioni proprie del database in uso.

Funzioni di aggregazione SQL native di uso comune sono per esempio `SUM` (per calcolare la somma dei valori di un campo) e `AVG` (per calcolare la media dei valori di un campo). Queste funzioni si dimostrano utili su campi che contengono dati numerici e la loro sintassi è la seguente:

```
SELECT SUM(campo) AS tot FROM tabella
SELECT AVG(campo) FROM tabella
```

Il risultato in questo caso non sarà una serie di record ma solo la somma o la media dei valori del campo indicato. Inoltre, nel caso della prima query, la somma calcolata sarà presentata con l'etichetta tot; l'impiego di funzioni è un altro caso in cui gli alias ritornano utili per migliorare la leggibilità dei risultati.

ATTENZIONE *Tra le altre funzioni di aggregazione SQL native vale la pena qui ricordare* `MIN` *e* `MAX` *(per estrarre il valore minimo o massimo di un campo) e* `COUNT` *(per ottenere il numero complessivo dei risultati di una query, come vedremo nel prossimo paragrafo).*

Esempi di funzioni scalari SQL native – questa volta applicabili su campi che contengono stringhe – sono invece `LOWER` (per convertire in minuscolo tutti i caratteri di un campo) e `UPPER` (per convertire in maiuscolo tutti i caratteri di un campo). La loro sintassi è analoga a quella appena vista:

```
SELECT UPPER(campo) FROM tabella
SELECT LOWER(campo) FROM tabella
```

Il risultato di queste due query saranno tutti i valori del campo selezionato convertiti in maiuscolo e in minuscolo, elencati uno dopo l'altro (in pratica in forma *scalare*).

RIFERIMENTI *Un elenco più esauriente delle funzioni SQL può essere consultato all'indirizzo* `http://www.w3schools.com/sql/sql_functions.asp`.

Raggruppamento dei risultati

In alcuni casi è utile raggruppare i risultati in base al valore di un campo. In questi casi è possibile utilizzare la clausola `GROUP BY`.

La sua sintassi è semplice e del tutto analoga a `ORDER BY`:

```
GROUP BY criterio_raggruppamento
```

ATTENZIONE *Il criterio di raggruppamento può essere espresso con il nome di un campo o con un suo eventuale alias.*

Riprendendo la tabella `Edizioni_derivate`, potrebbe essere per esempio utile sapere quante traduzioni ha avuto un singolo libro. Il criterio di raggruppamento è quindi il codice ISBN dell'edizione originale, mentre per calcolare i numero di record in cui compare il medesimo ISBN originale è necessario utilizzare la funzione `COUNT`:

```
SELECT ISBN_originale, Count(ISBN_originale) AS
Traduzioni FROM Edizioni_derivate
GROUP BY ISBN_originale
```

Questa query seleziona i codici ISBN originali ma allo stesso tempo li aggrega, e per ogni aggregazione conta le occorrenze, che presenta quindi con l'etichetta `Traduzioni`. Il risultato è visibile nella Figura 4.8.

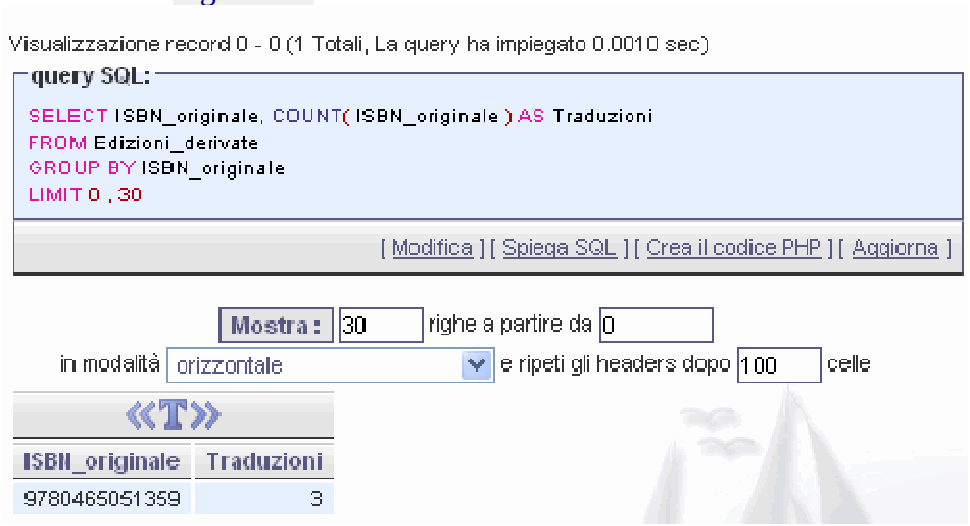


Figura 4.8 Selezionando, conteggiando e raggruppando per il medesimo campo, si ottiene il numero di occorrenze per ogni singolo valore del campo in oggetto.

SUGGERIMENTO *In query con più di un campo di selezione, la funzione* `COUNT` *può essere utilizzata solo in presenza della clausola* `GROUP BY`*. Quest'ultima può essere omessa quando* `COUNT` *è l'unico oggetto della selezione (per esempio* `SELECT COUNT(*) FROM tabella`*, che restituisce il numero di righe di una tabella).*

È importante notare che query di questo tipo hanno senso solo se il criterio di raggruppamento è lo stesso della selezione. Per esempio questa query

```
SELECT Editore, Count(ISBN_originale) AS Traduzioni
FROM Edizioni_derivate GROUP BY ISBN_originale
```

sebbene sintatticamente corretta, produrrebbe un risultato falso:

Editore	Traduzioni
Apogeo	3

Un rapido sguardo alla tabella `Edizioni_derivate` basta per verificare che Apogeo compare una sola volta. Che cosa è successo?

Il conteggio e il raggruppamento sono stati eseguiti per `ISBN_originale`, ma il primo campo elencato nella query è `Editore`, e poiché l'istruzione SQL contiene una funzione di aggregazione viene restituito come risultato un solo record composto da due campi: nel secondo – `Traduzioni` – è presente il risultato del conteggio, mentre per il primo l'istruzione SQL ha recuperato il valore del campo `Editore` presente nel primo record della tabella, quindi ha effettuato il conteggio e il raggruppamento, terminando il suo compito come previsto dalla specificità della funzione di aggregazione.

Filtri sui risultati

Le query che abbiamo visto finora, a prescindere dal tipo di selezione, operano sempre su tutti i record di una tabella.

A volte è però necessario effettuare selezioni mirate, in base a criteri particolari per vedere solo i record che soddisfano determinate condizioni.

Per dichiarare le condizioni a seconda delle quali la query deve avere effetto, SQL mette a disposizione la clausola `WHERE`. Ecco la sua sintassi:

```
SELECT campi_tabella FROM tabella WHERE condizioni
```

NOTA *Le clausole* `GROUP BY` *e* `ORDER BY` *se presenti, vanno collocate dopo le condizioni di* `WHERE`*. Se entrambe le clausole sono presenti,* `GROUP BY` *deve sempre precedere* `ORDER BY`*.*

Le condizioni vengono quindi espresse attraverso determinati operatori.

Operatori

SQL mette a disposizione tutta una serie di operatori che si rivelano utili sia per svolgere operazioni sui campi, sia per esprimere le condizioni in cui la query deve operare.

Esistono tre categorie di operatori: *di confronto, logici* e *aritmetici*.

Gli *operatori di confronto* permettono di esprimere condizioni sulla base di uguaglianze e disuguaglianze tra i valori dei campi. La tabella che segue riporta i più comuni.

Operatore	Significato
=	uguale a
!=	diverso da
<	minore di
>	maggiore di
<=	minore o uguale di/a
>=	maggiore o uguale di/a

Ecco un esempio del loro utilizzo:

```
SELECT * FROM Edizioni_derivate WHERE Anno='2005'
```

Dopo la clausola `WHERE` viene dichiarato il campo sui cui valori deve essere effettuato il controllo, quindi l'operatore e il valore di controllo: la query appena vista restituisce solo i record dove `Anno` ha valore 2005.

Altri due operatori di confronto molto utili sono `LIKE` e `BETWEEN…AND`.

`LIKE` permette di esprimere una condizione di somiglianza tra due valori testuali. In pratica questo operatore viene utilizzato quando il valore di confronto non è finito: è il caso di porzioni di stringa. Per esempio, si potrebbe voler vedere tutti i libri pubblicati da un dato editore, senza però essere sicuri del nome esatto dell'editore. Ecco come:

```
SELECT * FROM Edizioni_derivate WHERE Editore LIKE
"Apo%"
```

Questa query seleziona tutti i record dove le lettere iniziali dei nomi degli editori corrispondono alla stringa `Apo`. Il simbolo di percentuale (`%`) svolge il ruolo di jolly e sostituisce un numero indeterminato di caratteri.

Per cercare una stringa in qualsiasi punto dei valori di un campo, sarebbe invece stato necessario utilizzare due `%`, in questo modo:

```
SELECT * FROM Edizioni_derivate WHERE Editore LIKE
"%pog%"
```

Questa query restituisce tutti i record dove nel nome dell'editore appare la stringa `pog` all'inizio, alla fine o in mezzo.

ATTENZIONE *Volendo cercare una stringa solo negli ultimi caratteri di un valore, la sintassi di* `LIKE` *sarebbe di nuovo con un solo* `%`*,* LIKE "%geo"*.*

NOTA *L'utilizzo di* `LIKE` *è possibile anche con il simbolo underscore (_), che svolge il ruolo di jolly limitatamente a un singolo carattere. Ecco un semplice esempio:* `LIKE "A_o_eo"`*.*

SUGGERIMENTO *È importante sottolineare che l'operatore* `LIKE` *funziona solo se utilizzato o con il simbolo* `%` *o con il simbolo* `_`*.*

L'operatore `BETWEEN…AND` permette invece di esprimere una condizione in relazione a un intervallo di valori finiti. In questo caso servono due valori di riferimento, minimo e massimo. Ecco un esempio:

```
SELECT * FROM Edizioni_derivate WHERE Anno BETWEEN
2004 AND 2007
```

Questa query recupera tutti i record in cui l'anno di pubblicazione è compreso fra il 2004 (incluso) e il 2007 (incluso).

Nel caso di campi con valori `NULL` non è possibile utilizzare nessuno degli operatori di confronto fin qui visti. Come abbiamo ricordato, `NULL` non corrisponde a un valore nullo, ma semplicemente all'assenza di valore. Sui campi `NULL` gli operatori di confronto non possono quindi operare perché non esiste un valore finito da confrontare. È comunque possibile utilizzare `WHERE` in riferimento ai campi con valore `NULL` grazie agli operatori `IS NULL` e `IS NOT NULL`, che permettono di verificare se un valore è o meno `NULL`. Ecco un esempio:

```
SELECT * FROM Edizioni_derivate WHERE Anno IS NOT NULL
```

Passiamo ora agli *operatori logici*. Questi operatori permettono di esprimere condizioni articolate, utilizzando nella stessa query diversi operatori di confronto.

Gli operatori logici di cui si fa maggiore uso sono `AND` e **OR**.

NOTA *Gli operatori logici possono essere ripetuti più volte all'interno di una query per legare due o più condizioni.*

L'operatore `AND` permette di specificare due (o più) condizioni che devono essere soddisfatte e quindi vere affinché un record sia incluso tra i risultati.

L'operatore `OR` permette di specificare due (o più) condizioni delle quali almeno una deve essere soddisfatta e quindi vera affinché un record sia incluso tra i risultati.

Ecco tre esempi:

```
SELECT * FROM Edizioni_derivate WHERE Editore LIKE
"%pog%" AND Anno=2004
SELECT * FROM Edizioni_derivate WHERE Editore LIKE
"%pog%" OR Editore LIKE "%bei%" OR Editore LIKE
"%dos%"
SELECT * FROM Edizioni_derivate WHERE
(Editore="Apogeo" AND Anno=2004) OR Lingua="Cinese"
```

La prima query restituisce solo i record che soddisfano entrambe le condizioni legate dall'operatore `AND`.

La seconda query restituisce i record che soddisfano una delle tre condizioni legate dai due operatori `OR`.

Infine la terza query restituisce solo i record che soddisfano entrambe le condizioni legate dall'operatore `AND` e racchiuse tra parentesi tonde o che soddisfano la condizione legata dall'operatore `OR`.

NOTA *Le parentesi tonde permettono di esprimere condizioni complesse utilizzando contemporaneamente gli*

operatori `AND` *e* `OR`*.*

Concludiamo accennando agli *operatori aritmetici*. Questi operatori permettono di eseguire calcoli all'interno di una query. Gli operatori che SQL mette a disposizione sono i classici `+` (operatore di somma), – (operatore di sottrazione), `*` (operatore di moltiplicazione) e `/` (operatore di divisione). Tutti e quattro possono essere utilizzati sia sui campi della tabella che seguono `SELECT`, sia nelle condizioni introdotte da `WHERE`, come in questo esempio:

```
SELECT campo1*2 FROM tabella WHERE campo2-100>0
```

Questa query seleziona il valore campo1 moltiplicato per 2, solo quando il valore di `campo2` meno `100` è maggiore di `0`.

Selezione di dati da due tabelle

Finora abbiamo visto come effettuare query di selezione su una sola tabella.

In precedenza abbiamo però visto che tra due tabelle possono esistere tre tipi di relazioni.

Come, quindi, eseguire query in grado di restituire risultati composti da campi di due tabelle tra cui esiste una relazione?

È possibile creare query di selezione su due tabelle relazionate utilizzando la clausola `JOIN`. Ecco la sua sintassi:

```
SELECT campi_tabella
FROM tabella1 JOIN tabella2
ON tabella1.campo_chiave_primaria = tabella2.campo_
chiave_esterna
WHERE condizioni
```

TERMINOLOGIA *Spesso ci si riferisce a query di questo tipo utilizzando il maschile al posto del femminile. Per cui sono di uso abbastanza comune espressioni tipo "eseguire un JOIN" per indicare una query che esegue una selezione su due tabelle attraverso la clausola* `JOIN`*.*

Per prima cosa notiamo come la clausola `JOIN` leghi le due tabelle tra cui esiste una relazione, per cui il `SELECT` viene eseguito:

```
FROM tabella1 JOIN tabella2
```

Quindi la parola chiave `ON` permette di esplicitare i campi delle due tabelle che svolgono il ruolo di chiave primaria e chiave esterna:

```
ON tabella1.campo_chiave_primaria = tabella2.campo_
chiave_esterna
```

L'associazione di questi due campi viene fatta grazie al simbolo = e i nomi dei due campi devono sempre essere preceduti dal nome della tabella a cui appartengono e da un punto (`.`).

ATTENZIONE *Un'istruzione SQL contenente un* `JOIN` *potrebbe quindi essere letta così: "Seleziona (*`SELECT`*) questi campi dalla tabella1 e dalla tabella2 (*`JOIN`*) prendendo come riferimento (*`ON`*) questo campo della tabella1 e questo campo della tabella2 che contengono valori uguali (=)".*

Far precedere il nome del campo da un punto e dal nome della tabella è consigliabile (ma non obbligatorio) anche per i campi oggetto di `SELECT` o delle altre clausole SQL (`WHERE, GROUP BY, ORDER BY`). Questa sintassi diventa però obbligatoria in presenza di campi con nome identico nelle tabelle su cui il `JOIN` viene eseguito: diversamente, la query non potrebbe essere eseguita in quanto nell'impossibilità di distinguere su quale campo effettuare una selezione, un controllo, un raggruppamento o un ordinamento.

SUGGERIMENTO *La sintassi dei* `JOIN` *è comune a tutti i tipi di relazioni tra tabelle, sia che si tratti di relazioni uno a uno o uno a molti. Non esiste alcuna differenza a livello di istruzione SQL. La differenza si manifesta nei*

risultati che rispecchieranno la relazione. Per cui nel caso di relazione uno a uno per ogni record della tabella1 si otterrà un solo risultato composto dai dati presi dalla tabella1 e dalla tabella2; mentre nel caso di relazione uno a molti per ogni record della tabella1 si otterranno N *risultati, tanti quanti sono i record a esso relazionabili nella tabella2.*

NOTA *Nelle query, dopo la clausola* `JOIN` *e la parola chiave* `ON` *possono essere inseriti gli altri costrutti di SQL (*`WHERE`, `GROUP BY`, `ORDER BY`*) come precedentemente visto.*

I tipi di JOIN

Nella pratica i `JOIN` si distinguono in tre tipi:

- `INNER JOIN`
- `LEFT OUTER JOIN`
- `RIGHT OUTER JOIN`

NOTA *È possibile scrivere query con più di un* `JOIN` *(anche di tipo diverso) per ottenere risultati composti da campi di tre, quattro o più tabelle tra cui esiste una qualche relazione. Per esempio, se tra la* tabellaA *e la* tabellaB*, e tra la* tabellaB *e la* tabellaC *esistono due diverse relazioni, è possibile scrivere una query con un* `JOIN` *tra* `A` *e* B*, seguito da un* `JOIN` *tra* `B` *e* `C`*. In questo caso di parla di triplo* `JOIN`*. I* `JOIN` *multipli sono però di più difficile gestione e per questo non vengono trattati in queste pagine. Per l'approfondimento della loro sintassi si rimanda a testi specifici.*

Gli `INNER JOIN` considerano e quindi restituiscono tra i risultati solo le righe di due tabelle tra le quali esiste una relazione. In pratica, se un record nella *tabella1* non trova un corrispettivo nella *tabella2*, esso viene automaticamente escluso dai risultati. Un `INNER JOIN` è quindi la scelta perfetta per vedere solo ed esclusivamente le relazioni esistenti tra i record di due tabelle.

`I LEFT OUTER JOIN` e i `RIGHT OUTER JOIN` permettono invece di vedere tutti i record di una tabella a prescindere dal fatto che un record abbia o meno una relazione con uno o più record nell'altra tabella.

In pratica questi `JOIN` considerano e restituiscono tra i risultati sempre tutti i record della tabella che sta a sinistra (`LEFT OUTER`) o a destra (`RIGHT OUTER`) della relazione, sia che essi siano associati o meno a record nell'altra tabella.

Ma come stabilire qual è la tabella di destra o di sinistra, cioè la tabella su cui il `JOIN` pone l'"accento"?

Per farlo basta spostare l'attenzione nella query sull'ordine con cui vengono introdotte le due tabelle:

```
FROM tabella1 JOIN tabella2
```

Qui si esplicita la relazione: la prima tabella dopo il `FROM (`*`tabella1`*`)` è quella che sta a sinistra della relazione, la seconda – quella che segue la clausola `JOIN (`*`tabella2`*`)` – è quella che sta a destra della relazione.

Un paio di esempi chiariranno quanto abbiamo appena visto. Ancora una volta le tabelle di riferimento sono `Edizioni_derivate ed Edizioni_originali`.

Immaginiamo di voler sapere quali libri sono stati tradotti, da quali editori e in quali lingue. La query sarebbe questa:

```
SELECT Edizioni_originali.ISBN AS 'ISBN originale',
Titolo,
Edizioni_derivate.Editore AS 'Editore traduzione',
Lingua
FROM
```

```
Edizioni_originali INNER JOIN Edizioni_derivate
ON
Edizioni_originali.ISBN = Edizioni_derivate.ISBN_
originale
```

NOTA *La sintassi SQL permette di scrivere query anche su più righe, andando a capo arbitrariamente, per esempio per separare le varie parti di una query, rendendola meglio leggibile.*

In questo caso è sufficiente un `INNER JOIN`: i libri che non sono stati mai tradotti non compariranno tra i risultati, mentre per ogni traduzione saranno riportati l'`ISBN` originale, il titolo originale, l'editore che ha pubblicato la traduzione e la lingua della traduzione (Figura 4.9).

Figura 4.9 Risultati di un INNER JOIN.

NOTA *Si noti come nei campi oggetto di selezione sia stato necessario specificare il nome della tabella per* `ISBN` *ed* `Editore`*: questi due campi sono infatti presenti in entrambe le tabelle.*

Se volessimo invece vedere tutte le edizioni originali e le eventuali traduzioni, dovremmo modificare la query come segue:

```
SELECT Edizioni_originali.ISBN AS 'ISBN originale',
Titolo,
Edizioni_derivate.Editore AS 'Editore traduzione',
Lingua
FROM
Edizioni_originali LEFT OUTER JOIN Edizioni_derivate
ON
Edizioni_originali.ISBN = Edizioni_derivate.ISBN_
originale
```

Il `LEFT OUTER JOIN` prende come riferimento la tabella a sinistra della relazione (`Edizioni_originali`), ne riporta tutti i record e se trova relazioni nella tabella `Edizioni_derivate` recupera i dati, altrimenti lascia vuoti i campi `Editore e Lingua` (Figura 4.10).

Figura 4.10 Risultati di un LEFT OUTER JOIN.

ATTENZIONE *Alcuni database (per esempio MySQL) supportano una sintassi di* `JOIN` *leggermente diversa. Il termine* `OUTER` *diventa opzionale, per cui è possibile inserire nelle query semplicemente la formula* `LEFT JOIN` *o* `RIGHT JOIN`*.*

Query di modifica, inserimento e cancellazione

A questo punto passiamo velocemente in rassegna i comandi SQL che permettono di effettuare modifiche, inserimenti e cancellazioni di dati.

Il comando su cui costruire query per modificare i record di una tabella è `UPDATE`. La sua sintassi è la seguente:

```
UPDATE tabella SET campo_tabella = valore WHERE
condizioni
```

Dopo il comando `UPDATE` viene inserito il nome della tabella oggetto di modifica, quindi la clausola `SET` introduce l'elenco dei campi che devono essere modificati e i relativi valori (assegnati con il simbolo =). Ogni coppia campo/valore deve essere separata da una virgola.

Una considerazione va fatta sulla clausola `WHERE`: nelle query di modifica serve per specificare le condizioni in cui la modifica deve essere applicata. In sua assenza la modifica viene eseguita su tutti i record della tabella.

NOTA *Da un punto di vista sintattico l'utilizzo di* `WHERE` *nelle query di modifica è identico a quanto visto per le query di selezione.*

Ecco un paio di esempi:

```
UPDATE Edizioni_originali SET Autore = "Norman, Donald
A." WHERE Autore = "Donald A. Norman"
UPDATE Edizioni_originali SET Autore = "Donald, Norman
A.", Editore = "Basic Book Publishing" WHERE ISBN =
9780465051359
```

La prima query rileva nella tabella `Edizioni_originali` tutte le occorrenze di Donald A. `Norman` nel campo Autore e le modifica con `Norman, Donald A.`.

La seconda query rileva invece i record con uno specifico valore del campo `ISBN` e una volta individuati ne modifica i valori dei campi `Autore ed Editore`.

ATTENZIONE *Come abbiamo visto,* `ISBN` *è la chiave primaria della tabella* `Edizioni_originali`*: questo vuol dire*

che la query agirà al massimo su un solo record. Per modificare un record specifico utilizzare come criterio la chiave primaria è la scelta migliore e più sicura.

Passiamo ora all'inserimento dei dati. Il comando su cui costruire query per inserire nuovi record in una tabella è `INSERT`. La sua sintassi è la seguente:

```
INSERT tabella (campi) VALUES (valori)
```

Dopo il comando `INSERT` viene inserito il nome della tabella oggetto di inserimento, quindi – tra parentesi tonde – l'elenco dei campi e l'elenco dei valori per ogni campo, separati dalla clausola `VALUES`.

SUGGERIMENTO *Campi e relativi valori devono essere elencati nel medesimo ordine. Campi e valori devono essere separati da una virgola.*

Ecco un esempio:

```
INSERT Edizioni_originali
(ISBN, Autore, Titolo, Editore, Anno)
VALUES
(9788884909848, "Enrico Brizzi", "Bastogne",
"Baldini&Castoldi", 2006)
```

Questa query inserisce nella tabella `Edizioni_originali` un nuovo record: i cinque campi della tabella sono elencati nella prima parentesi, mentre i relativi valori seguono nel medesimo ordine nella seconda parentesi, che è preceduta dalla clausola `VALUES`. È interessante notare come i valori testuali (cioè le stringhe) siano racchiusi tra doppi apici: si tratta di un accorgimento sul cui senso torneremo nel Capitolo 5.

Esaminiamo infine il comando `DELETE`, che permette di cancellare record da una tabella. La sua sintassi è la seguente:

```
DELETE FROM tabella WHERE condizioni
```

Notiamo subito come dopo il comando sia presente al clausola `FROM` prima della tabella su cui l'eliminazione deve avere effetto. In realtà la presenza di `FROM` è facoltativa nel linguaggio SQL standard, ma poiché in alcuni database (come MySQL) esso è diventato obbligatorio e visto che rende la query meglio leggibile, si è preferito qui lasciarlo in evidenza. La clausola `WHERE` specifica invece, come di consueto, le condizioni in cui la query deve essere eseguita, cioè i record da cancellare.

SUGGERIMENTO *In assenza di* `WHERE` *un'istruzione di* `DELETE` *ha l'effetto di cancellare tutti i record di una tabella, lasciandola completamente vuota. Lo svuotamento di una tabella non equivale però alla sua cancellazione: la sua struttura rimane intatta e pronta per essere ripopolata di altri record.*

ATTENZIONE *Anche nelle query di cancellazione, volendo eliminare un solo record il criterio migliore da utilizzare è la chiave primaria.*

Ecco un esempio:

```
DELETE FROM Edizioni_originali WHERE ISBN =
9788884909848
```

Questa query ha l'effetto di cancellare il record precedentemente inserito.

I file .sql

In questo capitolo abbiamo visto come utilizzare `SQL` per scrivere alcuni tipi di query. Abbiamo però anche ricordato che `SQL` permette di svolgere diversi tipi di operazioni sui database. Esistono comandi e istruzioni `SQL` che consentono di creare, cancellare, modificare, ottimizzare e copiare interi database e tabelle.

Tutte queste istruzioni possono essere scritte ed eseguite direttamente da un client, inserite in un programma o anche salvate in un file con estensione .sql per essere utilizzate ed eseguite in un secondo momento.

I file `.sql` non sono quindi altro che file di testo composti da query. Il loro utilizzo non è comune: qualsiasi client è dotato di un semplice editor attraverso il quale è possibile scrivere ed eseguire query e questo rende poco pratico salvare le query in file a parte prima di eseguirle.

Tuttavia a volte è utile effettuare copie di un database o di una tabella, semplicemente per averne un backup o per trasferirlo altrove, e in questi casi la possibilità di avere tutto in un file `.sql` non è da trascurare.

TERMINOLOGIA *La copia di una tabella o di un database viene detta* dump*.*

Prima di tutto si tratta di file nel formato nativo di SQL, quindi ogni client è in grado di riconoscerli e utilizzarli e ogni database li supporta. Inoltre, visto che il contenuto non è altro che il testo di cui sono composte le istruzioni SQL, questi file sono all'occorrenza modificabili con qualsiasi editor di testo e allo stesso tempo molto leggeri: un fattore da non sottovalutare quando tabelle e database contano diverse migliaia di record. La Figura 4.11 mostra la tabella `Edizioni_derivate` dopo che è stata esportata come file `.sql` e aperta in un editor: come si vede sono presenti sia le istruzioni per creare la tabella, sia le query di inserimento dei dati precedentemente esportati.

La Figura 4.12 compara invece il peso della medesima tabella in formato SQL ed Excel: praticamente il doppio. Questa differenza è dovuta al fatto che Excel colloca il dato in una struttura tabellare aggiungendovi anche delle informazioni di stile. E struttura e stile hanno un peso.

Immaginatevi cosa questo avrebbe significato se al posto di una tabella di tre righe e cinque campi, si fosse trattato di un intero database con decine di tabelle compilate con migliaia di campi.

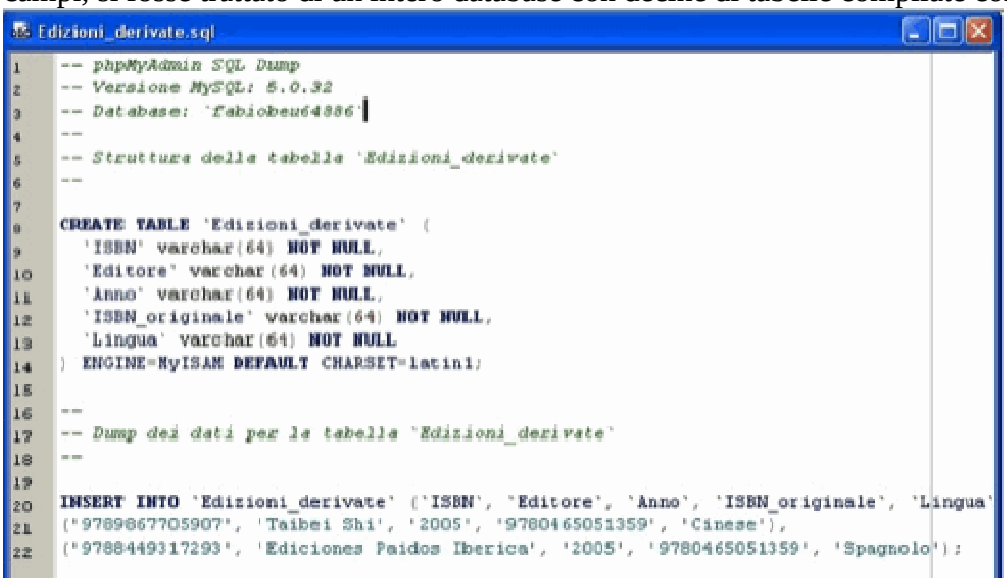


Figura 4.11 Il contenuto di un file `.sql`.

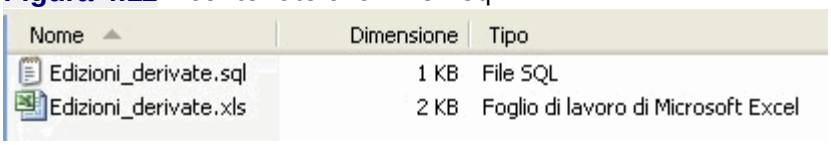


Figura 4.12 Il peso dell'informazione varia a seconda del formato.

Per concludere

In questo capitolo abbiamo parlato dell'organizzazione dei dati in database e della loro gestione, con particolare attenzione all'aspetto del loro recupero. Tutte le operazioni su un database possono essere svolte attraverso un client, tuttavia quando l'attività su di esso si fa intensa e cresce il numero degli utenti che necessita di consultare i dati contenuti nelle sue tabelle, allora diventa più economico e pratico affidarsi a dei programmi scritti proprio per semplificare e personalizzare l'attività di consultazione e gestione dei dati.

Nel prossimo capitolo parleremo proprio di programmi e linguaggi di programmazione, apparentemente uno degli argomenti più ostici per un umanista.

Capitolo 5

I linguaggi di programmazione: ovvero dire alle macchine cosa fare

Nei capitoli precedenti abbiamo visto come agire sulla struttura delle informazioni. Abbiamo esaminato in particolare tre soluzioni: il linguaggio XML (ideale per lo scambio di dati tra sistemi diversi), il linguaggio HTML (per la pubblicazione sul Web) e i database (che con il linguaggio SQL sono la scelta migliore per la gestione di grandi quantità di informazioni).

Abbiamo anche più volte ricordato come la forza di queste soluzioni sia la possibilità di riutilizzare i dati per scopi e in ambiti differenti, per poter creare nuova informazione e raggiungere il maggior numero di utenti.

Nel Capitolo 1 abbiamo anche introdotto il livello del comportamento delle informazioni: su questo livello bisogna operare per individuare i dati nelle strutture in cui sono salvati, estrarli e riorganizzarli in base a esigenze di comunicazione diverse, e per fare questo si deve utilizzare un *linguaggio di programmazione*.

Nei capitoli precedenti abbiamo incontrato tre tipi di linguaggi:

- di marcatura (XML e HTML), il cui scopo è marcare le informazioni aggiungendo valore semantico;
- di stile (CSS), il cui scopo è definire l'aspetto con cui le informazioni devono essere presentate;
- per le query (SQL), il cui scopo è gestire le informazioni contenute in un database.

Un linguaggio di programmazione è qualcosa di diverso. Un linguaggio di programmazione non definisce la struttura o l'aspetto delle informazioni, ma semplicemente permette di pianificare le operazioni che potranno essere svolte sull'informazione.

Queste operazioni possono essere di diverso tipo e anche impattare sugli altri livelli dell'informazione. Per esempio, con un linguaggio di programmazione è possibile trasformazione un file XML in un file HTML; oppure cambiare il colore di un elemento in base a una scelta dell'utente; o modificare, inserire, cancellare record da un database quando si verificano determinate condizioni; o più semplicemente recuperare un'informazione dalla struttura dati in cui è archiviata e presentarla in un certo modo all'utente che ne ha fatto richiesta (esattamente quello che fa il motore di ricerca Google).

Esistono diversi linguaggi di programmazione, ognuno caratterizzato da una propria sintassi. Tutti sono però basati su alcuni componenti specifici (che vedremo tra qualche pagina). Imparare un linguaggio di programmazione vuol quindi dire capire come scrivere in maniera grammaticalmente corretta questi componenti.

RIFERIMENTI *Per una panoramica introduttiva sulle tipologie dei linguaggi di programmazione, un punto di partenza può essere:* http://it.wikipedia.org/wiki/Linguaggio_di_programmazione.

Algoritmi e codice sorgente

Scendiamo ora un po' più nel dettaglio.

Con un linguaggio di programmazione è possibile scrivere un programma in grado di svolgere

determinate operazioni per conseguire un risultato, cioè un *algoritmo*.

Pensare a un programma come a un algoritmo vuol dire scomporre un problema nei singoli passaggi che permettono di arrivare alla sua soluzione. Ogni passaggio viene espresso con la sintassi del linguaggio di programmazione usato e prende uno o più dati che gli sono forniti dal passaggio precedente per restituire al passaggio seguente un nuovo dato, e così via fino alla soluzione.

In un programma ogni passaggio deve quindi essere inserito nel punto esatto in cui dovrà essere eseguito: la sequenza delle operazioni da svolgere è fondamentale e non permette ambiguità. Il rischio è un errore e il non conseguimento del risultato.

NOTA *Ancora una volta siamo di fronte a un procedimento di scomposizione e riduzione: lo abbiamo già visto nel Capitolo 2 parlando di XML e della scrittura dei DTD in relazione alla descrizione della struttura delle informazioni, ma anche nel Capitolo 4 in riferimento alla granularità delle informazioni nelle tabelle di un database; qui lo incontriamo nuovamente in relazione alle operazioni da svolgere sulle informazioni. Le considerazione fatte in precedenza valgono pertanto anche qui: un umanista, in particolare chi ha alle spalle studi filosofici e di logica, ha tutte le qualità per avvicinarsi e operare con profitto all'ambito della programmazione informatica.*

Pensiamo al file XML del Capitolo 2 e immaginiamo di voler creare una pagina web contenente i titoli di ogni capitolo. Il programma per fare questo dovrà prevedere almeno questi passaggi: accedere e leggere i dati contenuti nel file XML, individuare i marcatori che racchiudono i titoli dei capitoli, per ogni marcatore copiare il contenuto e quindi inserirlo in un file HTML marcato con i tag HTML scelti allo scopo, salvare il nuovo file HTML.

Questa sequenza di passaggi è un algoritmo ideale a raggiungere lo scopo prefisso. Esso potrà essere scritto, o meglio implementato, con diversi linguaggi, ma qualsiasi implementazione (se ben scritta) produrrà il medesimo risultato e conterrà i passaggi nel medesimo ordine.

Questo vuol dire che programmatori diversi possono implementare i vari passaggi con linguaggi diversi, attraverso le cui sintassi è possibile esprimere le istruzioni utili allo svolgimento di ogni passaggio.

Queste istruzioni vengono materialmente scritte attraverso editor di testo (preferibilmente editor specifici, in grado cioè di distinguere i vari elementi sintattici di un'istruzione) e assumono la forma di *righe di codice*. L'insieme di tutte le righe che compongono un programma viene detto *codice sorgente*.

Una volta scritto, il codice potrà essere salvato in file che avranno un'estensione differente a seconda del tipo di linguaggio usato.

TERMINOLOGIA *Si parla di* codice sorgente *anche in relazione a file scritti con linguaggi non necessariamente di programmazione. Espressioni come "sorgente HTML", "sorgente XML" o "sorgente CSS" sono molto utilizzate per indicare l'insieme delle istruzioni o regole presenti in un file* `.xml`, `.html` *o* `.css`.

NOTA *Esistono molti editor sia specifici per la programmazione in un dato linguaggio, sia in grado di lavorare con linguaggi differenti. Qui ricordiamo solo due soluzioni, entrambe gratuite: HTMLKit (*`http://www.htmlkit.com/`*) e GNU Emacs (*`http://www.gnu.org/software/emacs/`*). Entrambi sono indicati per scrivere file utilizzando linguaggi di marcatura, linguaggi di programmazione ma anche CSS e SQL. Oltre a diverse funzionalità per velocizzare le fasi di scrittura e controllo del codice, questi editor sono in grado di riconoscere la sintassi dei linguaggi ed evidenziare le diverse parti del codice, agevolandone così la lettura e la comprensione.*

Linguaggi compilati e linguaggi interpretati

A questo punto apriamo una parentesi per introdurre le due famiglie in cui si dividono i linguaggi di programmazione. Una volta scritto, il codice è pronto per essere eseguito dalla macchina, ma

come può una macchina riconoscerne le istruzioni e quindi metterle in atto?

Come vedremo meglio tra qualche pagina, i linguaggi di programmazione sono sintatticamente pensati per essere scritti e compresi dagli uomini. Tuttavia devono essere poi eseguiti dalle macchine. Il problema è quindi fare in modo che le macchine riconoscano ed eseguano le istruzioni scritte dal programmatore.

L'estensione permette alla macchina di riconoscere il tipo di file, dopodiché il contenuto di un file deve essere reso in una forma comprensibile alle macchine, in un *linguaggio macchina*.

TERMINOLOGIA *Nella programmazione si è soliti riferirsi ai linguaggi di programmazione nei termini di linguaggi* ad alto livello *per indicare che sono comprensibili dall'uomo. Un linguaggio adatto alle macchine è invece detto* di basso livello*.*

Nel Capitolo 3 abbiamo accennato al motore di rendering che nei browser permette di interpretare le istruzioni HTML e CSS offrendone all'utente una rappresentazione leggibile.

Qualcosa del genere esiste anche nella programmazione, con la differenza che a dover usufruire delle istruzioni in una forma leggibile non è un essere umano ma una macchina.

L'"equivalente" dei motori di rendering per i linguaggi di programmazione sono gli *interpreti* e i *compilatori*. A seconda che un programma sia tradotto in una forma leggibile – e quindi eseguibile – dalle macchine attraverso un interprete o un compilatore si parla perciò di linguaggi *interpretati* o *compilati*.

Interpreti e compilatori sono quindi parser e svolgono in pratica lo stesso compito: traducono le istruzioni scritte in un linguaggio di programmazione ad alto livello in un linguaggio macchina a basso livello. Ma lo fanno in maniera differente.

Gli interpreti "risiedono" sulla macchina che deve eseguire i programma (per esempio un server). Il programmatore non deve fare altro che caricare il file del programma sulla macchina quindi, ogni volta che il programma viene richiesto (per esempio per creare una pagina web dinamica), l'interprete lo controlla prima di tradurlo alla macchina che lo esegue.

Differentemente, con i linguaggi compilati, un programmatore, dopo aver scritto il codice, deve passarlo a un programma compilatore che si occupa di controllarlo e tradurlo in un linguaggio macchina. Questa versione del programma compilato viene poi passata alla macchina per essere eseguita.

Nel primo caso il programma deve essere controllato e tradotto ogni volta e questo ha un costo maggiore in termini di tempo e risorse impiegate dalla macchina. Il programmatore non si deve però preoccupare d'altro se non di scrivere il codice che potrà poi essere eseguito su qualsiasi macchina sia installato l'interprete.

NOTA *In pratica esiste un interprete specifico per ogni linguaggio di questo tipo.*

TERMINOLOGIA *I programmi scritti con i linguaggi interpretati sono detti* portabili*: possono cioè essere portati ed eseguiti su macchine diverse con facilità, basta che sia presente l'interprete.*

Un programma compilato viene invece eseguito in tempi notevolmente minori, in quanto il suo contenuto è già stato controllato e tradotto. Tuttavia ogni volta che si effettua una modifica al sorgente è necessaria una nuova compilazione per renderla operativa e questo implica tempi di sviluppo maggiori. Inoltre il programma potrà essere eseguito solo su sistemi operativi e dispositivi hardware identici a quelli su cui è stato compilato; diversamente dovrà essere ricompilato per la

nuova piattaforma.

In pratica linguaggi interpretati e compilati presentano vantaggi e svantaggi differenti che vanno soppesati quando si decide di sviluppare un nuovo programma. Se si prevede di dover intervenire spesso su programmi di dimensioni contenute e destinati a svolgere operazioni non particolarmente complicate e dispendiose in termini di risorse di calcolo, allora è preferibile optare per un linguaggio interpretato.

Se invece si deve sviluppare un programma complesso in grado di lavorare con e su grosse quantità di dati, allora la scelta di un linguaggio compilato può essere azzeccata.

NOTA *Nello sviluppo di siti e applicazioni web dinamiche si utilizzano di norma linguaggi interpretati. Tra i più famosi PHP, Perl e Python. Una pagina web non contiene mai enormi quantità di dati ed è spesso necessario tornare su di essa per effettuare piccole correzioni o migliorie. Inoltre in questo modo si ha la garanzia di poter spostare tutta l'applicazione da un server a un altro senza grossi problemi; basterà assicurarsi che l'interprete sia disponibile o eventualmente installarlo.*

NOTA *Anche sui server di database è presente un interprete specifico il cui compito è tradurre in linguaggio macchina le istruzioni SQL.*

I componenti di un linguaggio di programmazione

Passiamo ora in rassegna i componenti di un linguaggio di programmazione. Si tratta di costrutti sintattici che opportunamente combinati permettono di implementare un dato algoritmo. Anche se ogni linguaggio prevede una propria sintassi per esprimere questi componenti, essi hanno sempre il medesimo obiettivo.

Variabili

Si tratta di strutture il cui scopo è memorizzare un singolo dato (come un numero, una stringa di testo, una data o un valore booleano).

Una variabile può memorizzare un solo dato alla volta, per cui la memorizzazione di un dato in una variabile implica la cancellazione di qualsiasi altro dato precedentemente memorizzato.

Funzioni

Le funzioni sono comandi in grado di svolgere specifiche elaborazioni sui dati al fine di restituire un risultato (per esempio individuare una specifica porzione di una stringa o estrarre la radice quadrata di un numero). Una funzione può lavorare su uno o più valori per restituire un risultato e questi valori possono o meno essere inseriti in una variabile. Per esempio, una funzione il cui scopo è dividere per due un numero, avrebbe due valori in ingresso: il numero da dividere (inserito in una variabile) e il numero 2.

Una funzione può essere più o meno complessa, cioè lavorare su più variabili e prevedere più passaggi prima di restituire un risultato: in pratica ogni funzione è a sua volta un algoritmo utilizzabile per implementare algoritmi più complessi (i programmi).

TERMINOLOGIA *Spesso le funzioni vengono anche dette* routine *o* subroutine*.*

Ogni linguaggio di programmazione mette a disposizione numerose funzioni dette ***native,*** cioè già pronte per essere utilizzate. Tuttavia è sempre possibile scrivere nuove funzioni adatte a uno scopo particolare. Ogni funzione – sia essa nativa o creata ***ad hoc*** – ha un nome e per utilizzarla basta richiamarla inserendo il suo nome nel punto del programma dove deve operare.

NOTA *Il vantaggio di una funzione è quindi quello di consentire lo svolgimento di una serie di operazioni semplicemente richiamando il nome della funzione che le prevede.*

Una funzione può essere utilizzata diverse volte in un programma semplicemente richiamandola, l'importante è che essa sia stata definita una prima volta; le funzioni native sono già definite nel linguaggio, mentre le funzioni scritte per un particolare programma devono essere definite nel programma stesso o in un file esterno che il programma richiama.

NOTA *Ogni linguaggio di programmazione mette a disposizione delle funzioni il cui scopo è unicamente richiamare file esterni. Per individuare e richiamare un file queste funzioni utilizzano un URL (si veda il Capitolo 3) che gli viene passato come valore.*

TERMINOLOGIA *Molto spesso si usa il termine* librerie di funzioni *per riferirsi a un insieme di funzioni messe a disposizione dal linguaggio stesso (quindi native) o inserite in un file da un programmatore.*

Le funzioni sono in pratica gli strumenti che un programmatore può utilizzare per implementare un algoritmo e raggiungere un risultato.

Operatori

Si tratta di simboli che permettono di svolgere delle operazioni sui dati contenuti o meno nelle variabili. Gli operatori più noti sono gli operatori aritmetici (per svolgere somme, sottrazioni, moltiplicazioni e divisioni) e gli operatori di confronto (per valutare un dato in relazione a un altro in termini di uguaglianza o disuguaglianza).

RIFERIMENTI *Si veda anche nel Capitolo 4 il paragrafo "Operatori".*

Un'operazione tra due o più dati svolta tramite operatori viene anche detta *espressione*.

Strutture di controllo

Si tratta di costrutti che permettono di governare e controllare il flusso di un programma in base al verificarsi o meno di determinate condizioni (un esempio classico è il login a un servizio di webmail: il programma esegue prima un controllo su nome utente e password e se li riconosce come validi allora mostra il contenuto della mailbox, altrimenti un messaggio notifica un errore di riconoscimento e chiede all'utente di riprovare).

Array

Si tratta di strutture dati complesse che permettono di memorizzare più dati contemporaneamente.

In pratica, mentre una variabile può contenere un solo dato per volta, un array può contenere un numero indefinito di dati.

NOTA *Un array può contenere dati di diverso tipo, come numeri e stringhe, ma anche altri array: in questo caso viene detto* multidimensionale*.*

Ogni dato inserito in un array è connotato da un indice (di solito numerico).

È quindi possibile recuperare, modificare e cancellare i dati contenuti in un array attraverso gli indici. Inoltre ogni linguaggio mette a disposizione dei costrutti per leggere e scandire tutti gli elementi inseriti in un array.

TERMINOLOGIA *Per* elemento *si intende in questo caso la coppia* indice/dato*.*

NOTA *Il primo elemento inserito in un array può avere indice 0 o 1, a seconda del tipo di linguaggio utilizzato.*

La comprensione di un array è più facile se si pensa a esso come a un "casellario". Immaginiamo per esempio un array contenente gli indirizzi di posta elettronica degli iscritti a una newsletter. Un array di questo tipo potrebbe essere facilmente collocato in una struttura come quella visibile nella Figura 5.1.

Indice	Valore
0	fabio@gmail.com
1	marco@hotmail.com
2	costa@yahoo.it
3	paolo@alice.it
4	enrico@hotmail.it
...	...

Figura 5.1 Un array può essere visto con una struttura tabellare.

Un'altra visualizzazione più vicina a ciò che la macchina vede è invece mostrata nella Figura 5.2.

```
Array
(
    [0] => fabio@gmail.com
    [1] => marco@hotmail.com
    [2] => costa@yahoo.it
    [3] => paolo@alice.it
    [4] => enrico@hotmail.it
)
```

Figura 5.2 La vista del contenuto di un array.

In ogni caso a ogni valore è sempre associato un indice univoco che ne permette la sicura individuazione.

Il linguaggio PHP

Proviamo ora a vedere in pratica quanto fin qui detto.

Per farlo ci baseremo sul linguaggio PHP di cui vedremo la sintassi per i componenti sopra elencati.

I motivi di questa scelta sono tre.

- PHP è un linguaggio abbastanza semplice da imparare e trova largo impiego nello sviluppo web, per cui comprenderne i rudimenti vuol dire mettersi nelle condizione di capire e lavorare su diverse applicazioni web.
- La sua utilizzazione è libera e non legata a vincoli commerciali: per utilizzare questo linguaggio non sono necessarie autorizzazioni, sia che si sviluppi un'applicazione commerciale, sia che si crei il proprio sito personale.
- Su PHP esiste una vasta documentazione in Rete e lo stesso linguaggio fornisce numerose funzioni native che semplificano l'attività di sviluppo (molte funzioni sono dedicate all'utilizzo del database MySQL, per cui l'integrazione di query SQL nel codice PHP non solo è possibile ma è anche agevolata). Queste funzioni, come tutte le caratteristiche del linguaggio, sono ben documentate sul sito ufficiale di PHP (`http://it.php.net/`) di cui diverse sezioni sono anche tradotte in italiano.

RIFERIMENTI *Non è possibile qui fornire altro che un'introduzione al linguaggio, ma chi ne volesse approfondire lo studio può fare riferimento, oltre alla documentazione appena ricordata, anche a* PHP 5 Pocket *di Massimo Canducci, Apogeo, 2004.*

Per cominciare

Fissiamo ora alcuni concetti fondamentali.

I file PHP hanno estensione .php: il codice contenuto in un file con questa estensione, una volta caricato su una macchina dove è presente l'interprete PHP, quando richiesto viene eseguito ed elabora una risposta da restituire all'utente.

Di norma queste risposte sono pagine web, cioè composte da istruzioni HTML e CSS: PHP nasce appositamente per lo sviluppo di pagine web dinamiche e in assenza di indicazioni specifiche il contenuto di un risultato restituito da un programma PHP viene interpretato come una pagina web e aperto con un browser.

ATTENZIONE *È tuttavia possibile utilizzare PHP per creare anche altri tipi di documenti (come file PDF, Excel, PNG e così via): in questo caso è però necessario specificare nel programma che il risultato (l'*output*) sarà differente da una pagina web. Per fare questo bisogna inserire una riga di codice contenente la funzione* header *opportunamente compilata per il tipo di file scelto:* `header("Contenttype: application/pdf");` *specifica per esempio che il risultato restituito non è una pagina web ma un file PDF.*

Un `.php` contiene quindi righe di codice PHP, ma può contenere anche istruzioni in altri linguaggi, come HTML e CSS.

Per far sì che l'interprete possa distinguere le istruzioni PHP dalle altre istruzioni, è necessario racchiudere tutti i blocchi di codice PHP all'interno dei marcatori `<?php` e `?>`. Questa è la *prima regola sintattica* del linguaggio. Ecco un esempio:

```
<?php
// Istruzioni PHP…
// …
?>
Istruzioni HTML o CSS
<?php
// Altre Istruzioni PHP
?>
```

Le istruzioni PHP serviranno per elaborare la parte dinamica della pagina, mentre le istruzioni presenti al di fuori dei marcatori `<?php` e `?>` comporranno le parti statiche, cioè immutabili, della pagina (per esempio, nel caso di una casella di posta elettronica, la sezione della barra degli strumenti per comporre un messaggio, normalmente comune a tutti gli utenti).

ATTENZIONE *La sequenza di due slash (`//`) permette di inserire commenti all'interno del codice PHP. Si noti che questi commenti smettono di essere tali andando a capo: in pratica ogni riga di commento deve essere preceduta da `//`.*

La *seconda regola sintattica* è che ogni singola istruzione PHP deve terminare con un punto e virgola (`;`), altrimenti l'interprete segnalerà un errore.

NOTA *Il concetto di* istruzione *apparirà più chiaro osservando gli esempi delle prossime pagine.*

A questo punto possiamo passare alla sintassi delle componenti del linguaggio, a partire dalle variabili.

La sintassi delle variabili

In PHP è possibile dichiarare e quindi creare variabili utilizzando il simbolo `$`. Ogni variabile valida deve cioè avere un nome preceduto dal simbolo del dollaro.

TERMINOLOGIA *Una variabile viene creata quando l'interprete del programma la incontra per la prima volta. Scrivere una nuova variabile in un programma equivale quindi a crearla. Nella terminologia informatica, la scrittura di una nuova variabile viene detta* dichiarazione*, quindi quando si crea una variabile è corretto dire che la si* dichiara*.*

Per il nome della variabile è possibile utilizzare lettere, numeri e il simbolo di underscore (_), ma il primo carattere non può essere un numero. Ecco alcuni esempi:

```
$anno;
$anno1;
$_anno;
```

Queste tre istruzioni (si noti il punto e virgola alla fine) creano tre variabili valide e vuote, cioè senza alcun valore memorizzato.

ATTENZIONE *È possibile dichiarare variabili utilizzando per il loro nome lettere maiuscole o minuscole, anche se è preferibile adottare la norma di impiegare sempre le minuscole. Una volta dichiarata, una variabile potrà essere utilizzata solo rispettando il case del suo nome. Per PHP* `$nome` *e* `$Nome` *sono due variabili diverse.*

NOTA *In PHP, una volta che una variabile è stata dichiarata è possibile inserirvi qualsiasi tipo di dato. Non è quindi necessario specificare a priori se una variabile dovrà contenere testo, numeri o altri tipi di valori. Questa esigenza esiste invece in altri linguaggi.*

L'associazione di un valore a una variabile avviene tramite il simbolo `=`. Per esempio, questa istruzione

```
$anno = 2009;
```

crea una variabile `$anno` e vi memorizza il valore numerico `2009`.

I tipi di dato che più comunemente vengono inseriti in una variabile sono numeri (interi o decimali), stringhe di testo e valori booleani.

Nel caso di numeri è necessario ricordare che PHP separa i decimali con il punto e non con la virgola, quindi l'istruzione

```
$pi_greco = 3.14;
```

memorizza correttamente nella variabile `$pi_greco` il valore `3.14`, mentre l'istruzione

```
$pi_greco = 3,14;
```

produce un errore.

Per memorizzare una stringa di testo è invece necessario utilizzare una coppia di apici (singoli o doppi) per delimitarla. Ecco un paio di esempi validi:

```
$testo = "questa è una stringa";
$testo = 'questa è una stringa';
```

Per le variabili che contengono stringhe sono necessarie due considerazioni.

- La coppia di apici usati come delimitatori deve essere dello stesso tipo (non è possibile cioè aprire una stringa con un apice doppio e chiuderla con un apice singolo e viceversa).
- Non è possibile utilizzare all'interno di una stringa racchiusa da doppi apici un apice doppio, né utilizzare all'interno di una stringa racchiusa da singoli apici un apice singolo; questa eventualità renderebbe impossibile all'interprete stabilire dove termina esattamente una stringa.

Pertanto le istruzioni che seguono non sono valide e produrrebbero un errore:

```
$testo = "questa stringa contiene un doppio apice (")
e quindi non è valida";
$testo = 'questa stringa contiene un apice singolo (')
e quindi non è valida';
```

Per utilizzare apici singoli e doppi all'interno di stringhe racchiuse dal medesimo tipo di apice è necessario usare un *carattere di escape* \ (*backslash*) anteponendolo all'apice in modo che esso venga riconosciuto come parte della stringa e non come delimitatore:

```
$testo = “questa stringa contiene un apice doppio (\”)
ma è valida grazie al backslash”;
$testo = ‘questa stringa contiene un apice singolo
(\‘) ma è valida grazie al backslash‘;
```

Per quanto riguarda i valori booleani essi – come già ricordato nel Capitolo 3 e nel Capitolo 4 – servono per esprimere le condizioni di vero e falso. In PHP queste condizioni vengono rese attraverso le parole `TRUE` e `FALSE`. Il loro uso è frequente per memorizzare il fatto che una condizione sia stata soddisfatta o meno, per esempio l’autenticazione a una casella di posta elettronica:

```
$login = TRUE; // autenticazione riuscita
$login = FALSE; // autenticazione fallita
```

NOTA *Nel caso sia necessario distruggere una variabile e il suo contenuto, è possibile utilizzare la funzione* `unset()`, *in questo modo:* `unset(variabile);`.

Variabili predefinite

PHP prevede un certo numero di variabili *predefinite*. Queste variabili possono essere sempre utilizzate all’interno di qualsiasi programma e il loro nome è pertanto riservato e non può essere attribuito a nessun’altra variabile.

RIFERIMENTI *Un elenco completo delle variabili predefinite è visibile all’indirizzo* `http://it.php.net/manual/ro/reserved.variables.php`.

Le variabili predefinite sono utilizzate per memorizzare automaticamente diversi tipi di dati tra cui le informazioni inserite nei form HTML (o nei parametri delle query string degli URL). In pratica nel momento in cui un utente compila un campo di un form e preme il pulsante di conferma, le informazioni in esso contenute vengono memorizzate in queste variabili e inviate alla pagina PHP specificata nell’attributo action dell’elemento `<form>` (si veda ancora una volta il Capitolo 3). Quindi possono essere recuperate e utilizzate dal programma PHP per elaborare una risposta.

Una di queste variabili è `$_REQUEST`. Per recuperare le informazioni inserite in questa variabile si usa la seguente sintassi:

```
$v = $_REQUEST[‘valore_name’];
```

attraverso la quale il contenuto dell’elemento del form il cui attributo `name` ha valore `valore_name` viene memorizzato nella variabile `$v`.

Ecco un semplice esempio. Si tratta di un file nominato `prova.php` che attraverso un form invia un valore a se stesso. Il suo output prima e dopo l’invio dei dati dall’utente è visibile nella Figura 5.3.

```
<?php
// Inizio codice PHP
$v = $_REQUEST[‘valore’];
if ($v != ‘’) {
echo “<h1>Mi hai mandato il valore: $v</h2>”;
}
?>
<!-- Inizio codice HTML -->
<form action=’prova.php’>
Digita qui il tuo valore:
<input name=’valore’ type=’text’ />
<input value=’Invia’ type=’submit’ />
```

```
</form>
```

Leggiamo e spieghiamo il funzionamento di questo codice.

La prima istruzione PHP crea una variabile `$v`, quindi utilizza la variabile predefinita `$_REQUEST` per memorizzare in `$v` il contenuto dell'elemento `<input>` con attributo `name` impostato a `valore`.

Quindi la seconda istruzione esegue un controllo su `$v` utilizzando la struttura di controllo if e l'operatore `!=` (diverso da). Se il contenuto di `$v` è diverso da una stringa vuota, il programma esegue le istruzioni contenute tra le parentesi graffe.

RIFERIMENTI *Torneremo tra poco sulle strutture di controllo e sugli operatori.*

Figura 5.3 Uno scambio di dati tra form HTML (sopra) e la risposta elaborata da un file PHP (sotto).

La prima volta che viene caricata la pagina prova.php l'utente non ha ancora digitato nulla e di conseguenza il programma ignora le istruzioni: la variabile `$v` esiste poiché è stata dichiarata, ma non ha contenuto.

Segue una sezione di codice HTML con un form: questa sezione è statica e viene pertanto sempre inserita nella pagina inviata in risposta al browser.

Quando l'utente compila il form e invia i dati alla pagina prova.php, viene effettuato un nuovo controllo su `$v`: ora questa variabile ha un valore e le istruzioni contenute nelle parentesi graffe vengono eseguite.

echo è un costrutto del linguaggio PHP il cui scopo è ***stampare*** qualcosa. In questo caso echo stampa il testo contenuto nella stringa delimitata da doppi apici e il valore memorizzato in `$v`.

NOTA *Tutti i linguaggi di programmazione prevedono un costrutto analogo a* `echo` *(lo stesso PHP prevede anche il costrutto* `print` *il cui funzionamento è simile a* `echo`*). Lo scopo di questi costrutti è scrivere qualcosa nel documento che viene elaborato dal programma. Poiché come ricordato PHP è principalmente un linguaggio per creare dinamicamente pagine web,* `echo` *viene normalmente utilizzato per scrivere le istruzioni HTML e i testi che faranno parte della pagina inviata in risposta al browser, il quale si occuperà poi di interpretarla mostrandola all'utente.*

ATTENZIONE *Quando si stampa una stringa con* echo *bisogna utilizzare gli stessi accorgimenti su apici singoli e doppi ricordati in precedenza.*

NOTA *Si noti come in questo modo sia possibile stampare stringhe al cui interno è presente una variabile, di cui non viene visualizzato nell'output il nome (*`$v`*), ma il contenuto (*`30000`*).*

La programmazione web lato server

Quello che abbiamo appena visto è l'essenza della *programmazione web lato server*: con questa espressione ci si riferisce in pratica a tutte quelle tecnologie che possono essere utilizzate su un server per creare una pagina web, o qualsiasi altro tipo di documento da distribuire via Web.

I linguaggi come PHP, in grado di interfacciarsi con database, file XML, ma anche librerie di immagini (e numerose altre fonti di dati), sono il fulcro di questa attività di programmazione, che rimane sempre invisibile all'utente (*lato server*, appunto), in quanto quest'ultimo riceverà sempre e solo il documento elaborato in risposta.

Una volta compreso come inviare dati al server e come recuperali nel codice PHP, non resta altro da fare che impratichirsi nella loro elaborazione imparando a gestire più dati contemporaneamente attraverso le funzioni che il linguaggio mette a disposizione, prevedendo condizioni e controlli differenti, per modulare la risposta in base a criteri diversi.

Tutto questo senza mai perdere di vista che lo scopo finale è rendere un servizio all'utente, il quale si aspetta sempre una risposta chiara, l'informazione giusta o, in assenza di questa, un'indicazione su come proseguire la sua ricerca, e questo sia che l'interlocutore sia Google o il sito del negozio sotto casa.

Un linguaggio di programmazione va quindi prima di tutto inteso come uno strumento, il cui scopo è automatizzare il più possibile una serie di operazioni al fine di creare e fornire informazione. Lo studio delle sue particolarità serve esclusivamente per comunicare attraverso le macchine nel miglior modo possibile.

La sintassi delle funzioni

In precedenza abbiamo descritto il ruolo delle funzioni, ricordando che le funzioni native possono essere sempre utilizzate in qualsiasi punto del programma e che una funzione può lavorare su una o più variabili.

Abbiamo anche già incontrato una funzione, `unset()`, il cui compito è distruggere una variabile e il suo contenuto.

NOTA *Un possibile utilizzo di questa funzione potrebbe essere quella di distruggere i dati immessi dell'utente per autenticarsi a un sito uno volta che ha deciso di effettuale il* logout*, al termine di una sessione di lavoro (per esempio dopo aver letto e risposto ai messaggi presenti nella casella di posta elettronica).*

La sintassi di questa funzione è semplice:

```
unset($variabile);
```

Questa istruzione semplicemente distrugge una variabile di cui non rimane più traccia, ma molte volte le funzioni sono utilizzate per ricavare una dato diverso che deve essere poi memorizzato in una nuova variabile. Ecco un esempio:

```
$v = strtoupper($stringa);
```

Questa istruzione utilizza la funzione `strtoupper` sulla variabile `$stringa` per trasformare tutto il testo in essa contenuto in maiuscolo e memorizzalo nella variabile `$v`.

Come forse appare già chiaro da questi due esempi, la sintassi delle funzioni PHP prevede che l'oggetto su cui la funzione opera sia racchiuso tra parentesi tonde. Nelle parentesi tonde è quindi contenuto l'***argomento*** (o gli argomenti) della funzione.

Come abbiamo detto, una funzione può lavorare su una o più variabili e pertanto avere più

argomenti. All'interno delle parentesi tonde i diversi argomenti devono poi essere separati da una virgola.

Gli argomenti delle funzioni non sono però sempre variabili: alcune funzioni prevedono argomenti come stringhe, numeri e valori booleani il cui scopo varia da funzione a funzione. Ecco un esempio di una funzione che opera con tre argomenti:

```
$v = str_replace("e'", "è", $stringa);
```

Questa istruzione utilizza la funzione `str_replace` sulla variabile `$stringa` per ricercare e sostituire la stringa e' con è, prima di memorizzarla nella variabile `$v`: in pratica svolge un piccolo controllo ortografico inerente la sintassi della "e accentata grave".

ATTENZIONE *Se l'argomento è una stringa, bisogna utilizzare gli stessi accorgimenti su apici singoli e doppi visti in precedenza.*

NOTA *Introduciamo qui anche include. Si tratta di una funzione che permette di richiamare e includere in un programma file esterni. Nello sviluppo web risulta particolarmente utile perché permette di dividere in più file le singole parti che compongono una pagina, richiamandole poi quando necessario. Possono essere richiamati file di diverso tipo, non solo* `.php`*. Per esempio, si può creare un file* `.html` *contenente l'header di un sito (cioè il contenuto del tag* `<head>`*, che normalmente è condiviso in tutte le pagine) e quindi inserire in ogni nuova pagina del sito l'istruzione include* `'header.html';` *(il valore di include è quindi l'URL del file da richiamare che può essere espresso – tra apici singoli o doppi – in maniera relativa o assoluta). I vantaggi saranno due: ogni pagina avrà meno righe di codice e quindi sarà più facilmente leggibile e, soprattutto, dovendo modificare l'header questa operazione potrà essere fatta una sola volta nel file* `header.html`*.*

Partendo da questi accorgimenti è possibile utilizzare tutte le funzioni messe a disposizione da PHP. Non è possibile qui elencarle e discuterle tutte, e forse non avrebbe neppure senso. Il loro studio e apprendimento deve passare da necessità pratiche: un programma va visto in termini di passaggi per arrivare a un risultato e ogni passaggio può essere risolto individuando la funzione o le funzioni che singolarmente, o combinate, svolgono proprio il compito prefisso.

Programmare vuol dire prima di tutto scomporre un problema e pensare come risolvere le sue parti. Ancora una volta ritorniamo quindi sull'importanza del concetto di *granularità*, qui non tanto dell'informazione (come era invece per XML, HTML e i database), ma del flusso di pensiero che conduce a nuova informazione, e quindi a nuova conoscenza.

Le funzioni sono solo gli strumenti per risolvere le parti, i "grani del problema".

Ma dove trovare e visionare questi strumenti?

Il sito ufficiale di PHP mette a disposizione un'ampia sezione di documentazione liberamente consultabile, nota come il Manuale. In esso sono elencate tutte le funzioni disponibili, il tipo di elaborazione, il risultato restituito e il numero e la sintassi degli argomenti supportati.

RIFERIMENTI *L'indirizzo è* `http://it.php.net/manual/en/funcref.php`*. Qui le funzioni sono raggruppate per macrocategorie. È preferibile cominciare a esplorare questa documentazione a partire dalle funzioni che operano sulle stringhe (voce Strings) più facilmente utilizzabili e comprensibili. Non bisogna poi dimenticare di provare a utilizzare il motore di ricerca interno, che permette tra l'altro di circoscrivere la ricerca alle sole funzioni, ricordando che spesso il nome di una funzione è indicatore del tipo di elaborazione svolta e quindi del risultato ottenibile.*

La sintassi degli operatori

Come abbiamo già ricordato, gli operatori sono dei simboli che permettono di svolgere delle operazioni sui dati.

PHP, come altri linguaggi, mette a disposizione diversi operatori; qui vedremo quelli di uso più comune. Alcuni li abbiamo già incontrati nel Capitolo 4, parlando della sintassi delle query SQL, e il

loro utilizzo è grossomodo simile. Ma procediamo con ordine.

Operatori aritmetici

Questi operatori permettono di eseguire calcoli all'interno di un'istruzione. Ai classici operatori + (operatore di somma), – (operatore di sottrazione), * (operatore di moltiplicazione) e / (operatore di divisione), vanno affiancati % (restituisce il resto di una divisione), ++ (operatore di incremento) e –– (operatore di decremento).

Questi operatori lavorano solo su valori numerici e la loro sintassi è semplice:

```
$risultato = $a + $b;
$risultato = $a 7;
$risultato = 2 * $a;
$risultato = $a / 3;
$risultato = $a % 7;
```

Ognuna di queste espressioni memorizza nella variabile $risultato il valore ottenuto dall'operazione utilizzata. Con l'utilizzo delle parentesi è anche possibile scrivere espressioni più complesse:

```
$a = 20;
$b = 10;
$risultato = ($a * 3) + ($b / 2);
echo $risultato; // stampa il valore 65
```

Gli operatori ++ e –– hanno invece una sintassi leggermente diversa: essi lavorano su un singolo valore che incrementano o decrementano di una unità. Inoltre, a seconda del fatto che siano posti prima o dopo il valore lo incrementano o decrementano prima o dopo che l'istruzione l'ha utilizzato. Qualche esempio chiarirà meglio il concetto:

```
$a = 10;
echo ++$a; // qui il valore di $a viene
           // incrementato di 1 (10+1)
           // e quindi stampato, l'output è 11
echo $a++; // qui il valore di $a viene stampato
           // l'output è 11, quindi $a viene
           // incrementato di 1 (11+1)
echo $a;   // qui l'output è 12
```

Nel primo caso il valore di $a viene prima incrementato e quindi utilizzato nell'espressione. Nel secondo caso il valore di $a viene prima utilizzato nell'espressione e quindi incrementato. Alla fine il valore di $a è stato comunque incrementato di due unità.

ATTENZIONE *L'utilità di questi operatori apparirà più chiara tra qualche pagina, affrontando la struttura di controllo FOR.*

Operatori di confronto

Questi operatori permettono di esprimere condizioni sulla base di uguaglianze e disuguaglianze tra valori. La tabella che segue li riassume.

Operatore	Significato
==	uguale
!=	non uguale
===	identico
!==	non identico

<	minore
>	maggiore
<=	minore o uguale
>=	maggiore o uguale

Questi operatori vengono utilizzati nelle strutture di controllo per specificare le condizioni che governano il flusso di esecuzione di un programma (se si avvera una condizione, per esempio se un valore è maggiore, minore o uguale a un altro, il programma esegue delle istruzioni differenti).

Particolare attenzione va però qui posta sugli operatori `==` e `!=`, e `===` e `!==`.

I primi due servono per valutare o meno l'eguaglianza di due dati.

NOTA *La sintassi dell'operatore di uguaglianza prevede due uguali in quanto un singolo `=`, come abbiamo visto, serve per attribuire un valore a una variabile.*

Ma uguaglianza non significa identità totale. Per esempio, PHP può interpretare uguale ma non identico il contenuto di queste due variabili:

```
$a = 1;
$b = '1';
```

Entrambe contengono il valore 1, ma in `$a` esso è un numero, mentre in `$b` una stringa. Il tipo di dato contenuto costituisce quindi la differenza tra le due variabili che pertanto sono uguali ma non identiche.

Operatori logici

Questi operatori permettono di esprimere condizioni articolate all'interno delle strutture di controllo. Gli operatori logici di cui si fa maggiore uso sono `AND` e `OR`.

L'operatore `AND` permette di specificare due (o più) condizioni che devono essere tutte soddisfatte e quindi vere affinché il programma esegua determinate istruzioni.

NOTA *Un semplice esempio di condizioni di tipo `AND` può essere l'autenticazione a un sito: solo se nome utente e password con cui ci si vuole autenticare sono contemporaneamente validi le condizioni sono soddisfatte e il login ha esito positivo. Se invece uno dei due parametri non è valido, l'autenticazione viene respinta.*

L'operatore `OR` permette di specificare due (o più) condizioni delle quali almeno una deve essere soddisfatta e quindi vera affinché il programma esegua determinate istruzioni.

NOTA *Nelle strutture di controllo gli operatori `AND` e `OR` sono utilizzati per legare insieme più espressioni costruite con gli operatori di confronto.*

Uguale e punto

Questi due operatori non sono accomunabili per utilizzo. Ognuno permette di svolgere un'operazione diversa.

Abbiamo già visto il simbolo uguale (=), il cui scopo è attribuire un valore a una variabile. In pratica = è un *operatore di attribuzione*.

Il punto (`.`) è invece un *operatore di concatenazione* e serve per concatenare due o più stringhe. Ecco un esempio:

```
$nome = 'Fabio';
$cognome = 'Brivio';
$utente = $nome . ' ' . $cognome;
```

In questo caso nella variabile `$utente` viene memorizzato un valore testuale frutto dell'unione di tre stringhe: il contenuto di `$nome`, una stringa composta solo da uno spazio e il contenuto di `$cognome`. Il valore di `$utente` è quindi `Fabio Brivio`.

La sintassi delle strutture di controllo

Lo scopo delle strutture di controllo è governare il comportamento di un programma in base al verificarsi di determinate condizioni.

PHP, come gli altri linguaggi, ne prevede diverse. Non le vedremo tutte ma solo quelle sufficienti a cominciare a comprendere che tipo di controllo è possibile effettuare sull'esecuzione di un programma.

IF, IF-ELSE, IF-ELSEIF-ELSE

I nomi di questi tre controlli sono abbastanza esplicativi e intuitivi.

IF permette di effettuare un singolo controllo che *se* positivo porterà all'esecuzione di determinate istruzioni.

La sua sintassi è la seguente:

```
IF (condizione) {
// istruzioni da eseguire
// se la condizione è soddisfatta
}
```

Le condizioni vanno sempre espresse tra parentesi tonde mentre le istruzioni da eseguire tra parentesi graffe.

NOTA *In tutte le strutture di controllo, dopo la parentesi graffa di chiusura (}) non è necessario inserire un punto virgola (;).*

Anche IF-ELSE permette di effettuare un singolo controllo: *se* positivo verranno eseguite delle istruzioni, ***altrimenti*** verranno eseguite altre istruzioni previste proprio nel caso di fallimento del controllo.

La sua sintassi è la seguente:

```
IF (condizione) {
// istruzioni da eseguire
// se la condizione è soddisfatta
} ELSE {
// istruzioni da eseguire
// se la condizione non è soddisfatta
}
```

NOTA *La differenza tra* `IF` *e* `IF-ELSE` *è quindi che il primo non prevede alternative, mentre il secondo ne prevede una.*

`IF-ELSEIF-ELSE` permette invece di effettuare più controlli per ognuno dei quali sono previste delle istruzioni da eseguire *se* positivo. Solo se nessuno di essi verrà superato il programma eseguirà le istruzioni introdotte dall'ultimo `ELSE`.

La sua sintassi è la seguente:

```
IF (condizione) {
// istruzioni da eseguire
// se la prima condizione è soddisfatta
} ELSEIF (condizione) {
// istruzioni da eseguire
// se la seconda condizione è soddisfatta
} ELSEIF (condizione) {
// istruzioni da eseguire
// se la terza condizione è soddisfatta
```

```
} ELSEIF (condizione) {
// istruzioni da eseguire
// se la condizione N è soddisfatta
} ELSE {
// istruzioni da eseguire
// se nessuna condizione è soddisfatta
}
```

Per quanto riguarda invece la sintassi delle condizioni, come abbiamo ricordato esse vanno espresse utilizzando gli operatori logici e di confronto. Sintatticamente non cambia nulla sia che la condizione sia introdotta da un `IF`, un `ELSEIF` o un `ELSE`. Ecco alcuni esempi:

```
($a == $b) {
// la condizione è soddisfatta quando
// il contenuto delle due variabili è uguale
}
(($a == $b) OR ($a > 0)) {
// la condizione è soddisfatta quando
// il contenuto delle due variabili è uguale
// o il valore di $a è maggiore di 0
}
```

NOTA *Si noti come in presenza di un operatore logico (qui* `OR`*) le due condizioni espresse vadano racchiuse singolarmente da una nuova coppia di parentesi tonde.*

```
(($a === $b) AND ($a < 100)) {
// la condizione è soddisfatta quando
// il contenuto delle due variabili è identico
// e il valore di $a è minore di 100
}
```

FOR

`IF-ELSEIF-ELSE` permettono di controllare se e quando eseguire una porzione di codice. Qualunque sia l'esito del controllo, determinate istruzioni vengono o non vengono eseguite solo una volta.

A volte è invece necessario controllare e specificare quante volte un'istruzione deve essere eseguita. `FOR` permette proprio questo: effettuare dei *cicli* su parti di codice, ripetendone l'esecuzione un numero precisato di volte.

NOTA *Un utilizzo frequente di cicli si ha quando PHP viene usato per scrivere un programma in grado di interrogare un database eseguendo una query di selezione su una tabella. Attraverso le funzioni del linguaggio dedicate ai database è prima possibile contare i risultati restituiti. Quindi attraverso un ciclo le informazioni presenti in ogni record restituito possono essere estratte e stampate, per esempio per riportare all'utente i risultati in formato HTML, magari strutturando le informazioni in una tabella con i marcatori* `<td>` *e* `<tr>`*. Questo vuol dire che ogni record può essere processato allo stesso modo, o meglio che le istruzioni di stampa e i marcatori* `<td>` *e* `<tr>` *da utilizzare per creare la pagina di risposta sono applicabili a tutti i risultati della query e perciò possono essere scritti una sola volta nel codice all'interno di un ciclo, che le ripeterà tante volte quanti sono i record restituiti.*

RIFERIMENTI *PHP permette di lavorare con diversi database. È possibile visionarli all'indirizzo* `http://it.php.net/manual/en/refs.database.php` *e da lì accedere alle funzioni dedicate a ognuno di essi.*

La sintassi di `FOR` è la seguente:

```
FOR (indice; controllo; azione) {
// istruzioni da eseguire
// se il controllo è soddisfatto
```

```
}
```

All'interno delle parentesi tonde introdotte da `FOR` sono presenti tre istruzioni (si noti il punto e virgola) che conducono all'esecuzione delle istruzioni contenute all'interno delle parentesi graffe.

Il ciclo viene ripetuto tutte le volte che queste tre istruzioni vengono eseguite (in pratica quando il programma ha terminato di eseguire l'istruzione di *azione*).

NOTA *Il punto e virgola non deve essere inserito dopo l'istruzione di azione.*

Il ciclo termina quando la seconda istruzione, l'unica a effettuare un controllo, non viene soddisfatta.

Normalmente l'istruzione di *indice* è semplicemente una variabile che contiene un valore numerico. Quindi l'istruzione di *controllo* verifica una condizione (espressa con uno degli operatori di confronto) sul valore della variabile. Se questa condizione viene soddisfatta, l'istruzione di *azione* cambia il valore della variabile (normalmente utilizzando gli operatori aritmetici di incremento o decremento). Da questo valore il ciclo ripartirà una volta eseguite le istruzioni contenute nelle parentesi graffe.

Nella Figura 5.4 viene mostrata la pagina generata dall'esecuzione del seguente codice:

```
$i = 0;
FOR ($i; $i < 10; $i++) {
echo "Il valore di indice è $i <br/>";
}
```

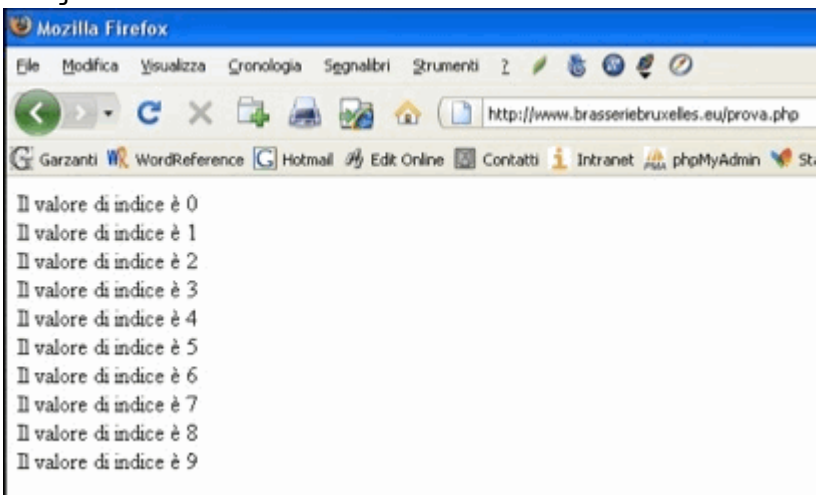


Figura 5.4 Risultato di un ciclo `FOR`.

Come si vede, prima del `FOR` viene creata la variabile `$i` con valore `0`. Quindi il ciclo per prima cosa fissa `$i` come indice, quindi ne controlla il valore (se minore di `10`) e se la condizione è soddisfatta incrementa di una unità `$i` ed esegue le istruzioni tra le graffe. Giunto al valore `9`, `$i` soddisfa ancora la condizione, viene di nuovo incrementato e le istruzioni tra le graffe eseguite. Quindi `$i` viene di nuovo testato, ma poiché ora ha valore 10 la condizione non è più soddisfatta e il ciclo si interrompe.

La sintassi degli array

Abbiamo già detto che un array è una struttura complessa nella quale possono essere inseriti più dati contemporaneamente associandoli a un indice.

La sintassi per dichiarare e quindi creare array è molto semplice:

```
$nomi = array();
```

Questa istruzione crea un array vuoto come nome `$nomi`.

NOTA *Per i nomi degli array valgono le stesse regole sintattiche viste per i nomi delle variabili.*

Una volta che l'array è stato creato può essere popolato di dati in maniera quasi analoga a una variabile, l'unica differenza è la presenza di una coppia di parentesi quadre (`[]`) di fianco al nome dell'array:

```
$nomi[] = "Fabio";
$nomi[] = "Paolo";
$nomi[] = "Ermanno";
```

Queste tre istruzioni inseriscono nell'array le stringhe `Fabio, Paolo, Ermanno`.

ATTENZIONE *In assenza delle parentesi quadre* `$nomi` *sarebbe stato interpretato come una variabile e quindi le tre stringhe sarebbero state scritte una sull'altra e solo l'ultima –* `Ermanno` *– sarebbe alla fine rimasta memorizzata.*

In assenza di indicazioni un nuovo dato viene inserito in un array assegnandogli il primo indice libero.

La struttura dell'array `$nomi` sarebbe a questo punto uguale a quella visibile nella Figura 5.5.

```
Array
(
    [0] => Fabio
    [1] => Paolo
    [2] => Ermanno
)
```

Figura 5.5 In PHP gli indici degli array partono dal numero 0.

È possibile aggiungere un nuovo elemento specificandone l'indice. Per farlo è sufficiente inserire il valore dell'indice tra le parentesi quadre:

```
$nomi[99] = "Giuseppina";
```

Questa istruzione inserisce nell'array `$nomi` la stringa `Giuseppina` associandole come indice il valore 99.

RIFERIMENTI *In realtà esistono anche altre sintassi che permettono di creare e popolare array più velocemente. Per motivi di sintesi qui non vengono affrontate, ma se si lavora con gli array vale la pena prenderne visione. Una visita all'indirizzo* http://it.php.net/manual/en/language.types.array.php *sarà di aiuto, mentre all'indirizzo* http://it.php.net/manual/en/ref.array.php *sono elencate le funzioni messe a disposizione per lavorare con gli array.*

Una volta inseriti in un array i dati possono essere recuperati attraverso il loro indice. La sintassi prevede in questo caso l'uso delle parentesi graffe, in questo modo:

```
echo $nomi{2};
```

Questa istruzione recupera il valore dell'array `$nomi` con indice 2 e lo stampa.

Tuttavia non sempre si conosce l'indice corrispondente al valore di interesse. Per questo è possibile scorrere tutto un array utilizzando il costrutto `FOREACH`.

`FOREACH` è per certi aspetti simile a `FOR`, dal quale però si differenzia in quanto il suo scopo è eseguire cicli su tutto il contenuto di un array. In pratica `FOREACH` permette di scandire uno dopo l'altro tutti gli elementi di un array.

Ecco la sua sintassi:

```
FOREACH (array AS indice => valore) {
// esegui queste istruzioni
}
```

Un ciclo di questo tipo si potrebbe parafrasare in questo modo: "*Per ogni* elemento dell'*array* specificato recupera l'*indice* e il relativo *valore*, quindi esegui queste istruzioni".

Nella Figura 5.6 viene mostrata la pagina generata dall'esecuzione del seguente codice:

```
FOREACH ($nomi AS $i => $v) {
echo "All'indice numero $i è presente $v <br>";
  IF ($v == 'Fabio') {
  echo "<b>Ciao Fabio! Ti ho trovato</b><br/>";
  }
}
```

Per ogni elemento dell'array `$nomi`, il ciclo recupera l'indice e il relativo valore e li memorizza in due variabili – `$i` e `$v` – che quindi passa alle istruzioni inserite nelle parentesi graffe. Per prima cosa viene stampata una stringa contenente i due valori. Poi viene eseguito un controllo specifico attraverso un `IF`: se la condizione è soddisfatta, viene stampata una seconda stringa. Questo esempio, pur nella sua semplicità, mostra come sia possibile individuare valori in un array senza conoscerne l'indice.

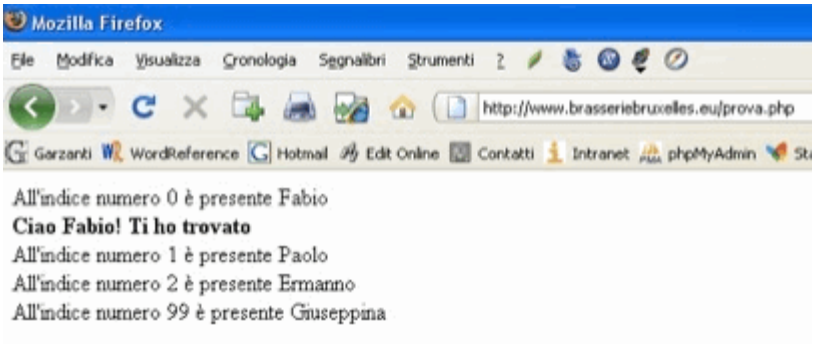


Figura 5.6 Il risultato di un ciclo FOREACH.

Il ciclo termina automaticamente quando anche l'ultimo elemento dell'array è stato scandito.

NOTA *Questo esempio mostra anche come cicli e controlli diversi possano essere utilizzati insieme all'interno di un programma per implementare algoritmi via via sempre più complessi.*

Per concludere

Qui termina questa piccola introduzione ai linguaggi di programmazione. Abbiamo visto come tutti permettano di esprimere attraverso righe di codice cosa le macchine dovranno fare in determinate condizioni e in che modo. L'unica differenza sarà la sintassi che potrà cambiare in base al linguaggio utilizzato, quindi i programmi saranno più complessi in relazione all'algoritmo implementato e alla capacità del programmatore.

Cominciare a programmare non è difficile e ottenere dei risultati è ancora più facile nell'ambito della programmazione web con PHP: in questi casi quello che il programma deve fare è semplicemente costruire una pagina HTML, la cui sintassi non è poi molto complicata. Con un po' di pratica e un po' di curiosità si può veramente fare molto.

Prima di concludere – veramente –, un'ultima regola, quella che i bravi programmatori conoscono e mettono in pratica: DRY, ovvero "Don't Repeat Yourself", non ti ripetere mai, non scrivere due volte lo stesso codice (ma se devi usarlo più volte, programma in un modo che se ne occupi la macchina).

Come non essere d'accordo? Siamo umanisti e conosciamo bene il valore della singolarità.

Conclusione

Il ruolo dell’umanista

Una convinzione percorre tutte le pagine di questo libro: il controllo dell’informazione – e quindi della conoscenza – passa dal controllo della struttura dell’informazione.

Da qui le macchine partono per svolgere tutte quelle operazioni che rendono un computer un oggetto fantastico e utile: contare, elaborare, presentare i dati.

Qui si torna quando abbiamo bisogno di un’informazione diversa.

Guardare ai file in termini di struttura del dato è la prima cosa che un’umanista dovrebbe imparare a fare.

Scoprire in che formato può essere salvato un file, cercare il formato meno pesante, meglio (o meno) carico di informazioni di stile e routine, meglio strutturato, quindi spostarsi da un programma all’altro, per elaborarlo o anche processarlo in base alle necessità. Questa è un’abilità preziosa che ha molto a che fare con la gestione della conoscenza. Ma molto spesso la si dimentica, o semplicemente la si ignora.

Credo che gli umanisti dovrebbero imparare a svilupparla, o quantomeno a intuirla.

Questo libro può essere solo l’inizio, ma forse per qualcuno può indicare una strada. Una strada costellata di pratica.

Perché nel mondo dell’informatica si impara facendo. L’informatica è uno strumento al servizio dell’uomo e come per ogni altro strumento il suo apprendimento passa dalla pratica.

Quindi leggete su Internet o nelle pagine di un libro, e cercate delle risposte ovunque pensate possano essere scritte, ma non dimenticatevi di praticare.

L’informatica è fatta di linguaggi e qui ne abbiamo solo introdotto alcuni.

Non dovete certo impararli tutti. Ma cominciate a vedere le lingue dell’informatica come strumenti. E non dimenticate che – come per ogni lingua – il primo passo per il suo apprendimento è lo studio della grammatica, per imparare a leggere e scrivere. Solo fatto questo si può passare a ragionare e creare pensieri complessi.

Quando potrete pensare nella lingua delle macchine allora potrete comandarle.

E ricordate: le macchine sono molto più semplici degli uomini.

Buon proseguimento.

Fabio Brivio

18 dicembre 2008, 11:23 pm

Appendice A

La lingua di Google

Il World Wide Web è un incredibile serbatoio di risorse e informazioni. Nel momento in cui vengono scritte queste pagine Wikipedia ricorda che, secondo uno studio del 2001, sarebbero 550 i bilioni di documenti presenti sul Web, mentre più di 100 milioni i siti web attivi. Queste stime vanno con buona ragione valutate per difetto.

Un numero simile, già di per sé impressionante, spaventa ancora di più se si considera che tutte queste informazioni non sono pubblicate in una maniera sistematica adatta ad agevolarne il recupero.

Spesso si fatica a trovare quello che si cerca anche conoscendo il sito, l'indirizzo dove cercare o sapendo quantomeno da dove partire: ma in caso contrario venire a capo di una qualsiasi necessità di ricerca diventa pressoché impossibile.

Il problema di cercare, e soprattutto trovare, si è manifestato presto, insieme alla vertiginosa e rapida crescita del Web: in un sistema così aperto e democratico, che permette di pubblicare documenti eterogenei potenzialmente a chiunque, il caos non è tardato ad arrivare.

Per risolvere questo problema sono state studiate diverse strategie, ma oggi il principale strumento per accedere con profitto ai contenuti del Web è costituito dai *motori di ricerca*.

I motori di ricerca

Un motore di ricerca si compone di due parti.

La prima ruota intorno a un software in grado di muoversi automaticamente tra pagine e siti web, leggendone i contenuti e registrandone i cambiamenti, per poi metterli in relazione e quindi archiviarli. Questo software è detto *crawler*, *spider* o *webbot* (da *web* e *robot*). Tra questi nomi il più diffuso è forse spider.

Questa parte è nascosta all'utente, che si può così concentrare sull'interrogazione del motore attraverso un'apposita interfaccia, o meglio un sito web che permette di immettere chiavi, o termini, di ricerca (in gergo *query*, "domanda" in inglese), per accedere poi alla consultazione dei risultati riportati da un algoritmo programmato per rilevare le pertinenze tra query e pagine web archiviate dallo spider.

Esistono diversi motori di ricerca. Tra i più famosi vi sono Yahoo! (`http://it.yahoo.com/`), Live Search di Microsoft (`http://www.live.com/`) e Google (`http://www.google.it/`).

Tra questi Google si è guadagnato nel corso degli anni una leadership incontrastata (basti pensare che nel mondo anglosassone *to google* ha assunto il significato di "cercare in Rete").

Google

Uno dei motivi alla base del successo di Google è senza dubbio la sua semplicità di utilizzo: basta digitare i termini da ricercare, la query, nell'apposito campo e premere il pulsante *Cerca con Google*. In un attimo viene visualizzata la pagina dei risultati composta di default dai primi dieci risultati che Google reputa più pertinenti.

La semplicità di Google può però trarre in inganno, mascherandone le complete potenzialità e limitandone l'utilizzo e lo sfruttamento.

Google è molto di più che una casella di testo per le ricerche. La quantità di informazioni che è in grado di gestire e servire, unita alle modalità attraverso le quali un utente può fruirne, fanno di Google un editore a tutti gli effetti: un editore flessibile in quanto in grado di adattarsi alle esigenze di ogni utente; sicuramente il più grande editore della Rete e forse anche del mondo intero.

NOTA *Quanto appena detto assume ancora più valore se si nota come le pagine dei risultati di Google assomiglino concettualmente alle pagine dei quotidiani cartacei: in entrambe gli spazi che contengono le informazioni sono spesso affiancati da box e riquadri pubblicitari (in Google etichettati come* Collegamenti sponsorizzati*).*

Inoltre Google è dotato di un proprio linguaggio e di alcune regole sintattiche utilizzabili per svolgere interrogazioni più precise e efficaci. Di questo parleremo nelle prossime pagine.

NOTA *Sebbene quanto segue sia relativo esclusivamente all'esperienza con Google, alcune note e regole sono applicabili anche ad altri motori di ricerca.*

Comprendere Google

Prima di proseguire, dobbiamo soffermarci un momento per comprendere i procedimenti e le logiche che regolano il funzionamento di Google e che di conseguenza ne hanno decretato il successo.

Come funziona Google? Quali sono i meccanismi che si nascondono dietro alla pagina dei risultati?

Il cuore di Google è rappresentato da PageRank, un software sviluppato per classificare le pagine web in ordine di importanza. PageRank crea in pratica i cosiddetti indici di popolarità delle pagine web e per farlo si basa su un concetto proprio del mondo universitario che gli studenti dell'Università di Stanford (California), Larry Page e Sergey Brin, gli ideatori e fondatori di Google, conoscevano bene: come una pubblicazione accademica acquista valore in relazione al numero di volte che viene citata, così una pagina web aumenta la sua popolarità in base al numero di volte che viene *linkata*, e poiché il Web è una immensa rete di link anzi, il diritto a *linkare* è l'essenza stessa del Web, la pura enumerazione dei link è fondamentale per valutare la notorietà di una pagina.

Tuttavia Google non si ferma qui. Viene preso in considerazione anche il "peso" di chi cita, vale a dire la notorietà della pagina che contiene un link, e questo sempre secondo la logica che una citazione a cura di un professore universitario di fama internazionale ha più valore di una a opera di uno studente laureando.

Inoltre costruendo i suoi indici Google prende in esame anche l'attinenza dei contenuti. Il link su una pagina che tratta di storia medievale verso un'altra pagina che tratta del periodo altomedievale in Italia viene valutato come più significativo.

Si capisce quindi come esista una relazione diretta tra la popolarità di una pagina e la sua posizione tra i risultati per una query specifica. Tuttavia questo da solo non è garanzia di successo. Le pagine, oltre a essere importanti, devono infatti essere correlate ai termini ricercati: solo così i risultati potranno essere pertinenti. Google effettua quindi accurate analisi testuali per recuperare tra i suoi indici pagine rilevanti rispetto ai criteri di ricerca dei dati.

I termini di una query vengono ricercati e conteggiati nella pagina, valutandone il peso semantico, che è maggiore per esempio se un termine è presente in un titolo piuttosto che in un paragrafo. La loro ricorrenza frequente è segno di pertinenza, ma anche i contenuti di pagine correlate a quella in

analisi vengono presi in considerazione, in modo da contestualizzare una pagina per stabilire se risponde o meno ai criteri di ricerca indicati prima di inserirla nell'elenco dei risultati.

SUGGERIMENTO *Una legge non scritta recita che i risultati più interessanti sono sempre contenuti nelle prime tre pagine dei risultati (quindi di norma tra i primi 30 risultati). Se questo può essere vero in alcuni casi, altre volte invece, sfogliando con un po' di pazienza anche le pagine successive, è possibile imbattersi proprio nell'informazione desiderata.*

Parlare la lingua di Google

Imparare a utilizzare con profitto Google, trovando in poco tempo quello che si sta cercando, vuol dire essenzialmente imparare a porre le domande – *query* – giuste, in modo da indirizzare nella maniera più precisa possibile l'algoritmo di ricerca. Per questo è importante conoscere almeno minimamente il linguaggio sviluppato da Google per ottimizzare le operazioni di ricerca.

Per prima cosa è utile sapere che Google non è *case-sensitive*, cioè non fa distinzione tra maiuscole e minuscole. Questo vuol dire che interrogando il motore con le query Apogeo, `APOGEO` o ancora apogeo, i risultati saranno i medesimi. Google non dà infatti importanza alla forma (il *case*, maiuscolo o minuscolo) con cui le query vengono scritte, mentre considera con attenzione la sintassi.

Una query può quindi essere composta da uno o più termini e questi termini possono essere messi in relazione tra loro utilizzando dei costrutti logici, gli *operatori di ricerca*, grazie ai quali è possibile creare catene di interrogazione, o *query complesse*.

ATTENZIONE *Google tiene in considerazione l'ordine in cui vengono digitati i termini della ricerca. Per esempio, la query* `apogeo libri` *non produce gli stessi risultati di* `libri apogeo`*. Al primo termine viene dato un peso maggiore.*

L'utilizzo di questi operatori permette di eseguire interrogazioni composte da diversi termini relazionati tra loro, in modo da restringere sensibilmente l'ambito della ricerca e quindi individuare nel minor tempo possibile l'informazione desiderata.

Ogni operatore di ricerca ha una specifica funzione e, quasi sempre, può essere utilizzato insieme ad altri operatori.

Nelle prossime pagine gli operatori di ricerca vengono presentati raggruppati per categorie. Per prima cosa introduciamo i cosiddetti *operatori di base*.

Operatori di base

Google utilizza e riconosce sette operatori di base: AND, OR, – (meno), + (più), "" (virgolette), ~ (tilde) e * (asterisco).

Operatore AND

Permette di definire un insieme di termini che devono essere presenti contemporaneamente su una pagina affinché essa sia inclusa tra i risultati di una ricerca. Ritenuto fondamentale da Google, questo operatore viene inserito automaticamente tra i termini di una query e quindi non è necessario digitarlo direttamente. Per esempio, la query `apogeo libri` per Google è identica ad `apogeo AND libri`: entrambe restituiscono tra i risultati pagine che contengono sia il termine "apogeo", sia "libri".

Operatore OR

Permette di definire un insieme di termini dei quali uno qualunque, indifferentemente, deve essere presente su una pagina affinché essa sia inserita tra i risultati di una ricerca. Va interposto tra i

termini da ricercare, come in `apogeo OR libri`. Questa query restituisce tra i risultati pagine dove compaiono o il termine “apogeo” o “libri”.

Operatore – (meno)

Permette di specificare cosa escludere dai risultati di una ricerca, scremando così il numero dei risultati. Va anteposto, senza lasciare spazio, al termine oggetto di esclusione, come in `apogeo –libri`: questa query restituisce solo pagine che contengono il termine “apogeo” escludendo quelle che contengono il termine “libri”.

Operatore + (più)

Per impostazione predefinita Google non riconosce lettere accentate o altri segni diacritici (à, è, é, ì, ò, ù, ü e così via), che vengono elaborati come normali lettere dell’alfabeto (quindi senza accenti, dieresi e così via). Inoltre vengono automaticamente ignorati i termini ritenuti da Google troppo comuni per essere discriminanti ai fini di una ricerca, per esempio articoli, preposizioni ma anche cifre. L’operatore permette di specificare al motore di ricercare un termine o una frase rispettando i segni diacritici e i termini comuni. Va anteposto, senza lasciare spazio, al termine oggetto di attenzione particolare, come in baviera +münchen: questa query restituisce pagine che contengono il termine “baviera” e il termine “münchen” scritto con la “u” con dieresi.

Operatore “” (virgolette)

Permette di definire una frase specifica che si desidera ricercare. I termini che compongono la frase vanno inseriti tra le virgolette (o doppi apici), come in `“il catalogo dei libri di Apogeo”`: questa query elenca pagine che contengono esattamente la frase “il catalogo dei libri di Apogeo”.

SUGGERIMENTO *In alcune situazioni una buona tecnica di ricerca consiste nell’interrogare Google digitando come query una parte della risposta che si sta cercando. Per esempio, volendo sapere la data di nascita di Alessandro Manzoni, si potrebbe pensare di utilizzare una query come* `Alessandro Manzoni nascita`*, ma risultati migliori si ottengono con* `“Alessandro Manzoni nasce”`*, vale a dire la frase che dovrebbe o potrebbe precedere, in un documento, la risposta ricercata. Google elencherà nei risultati solo pagine che contengono la frase tra virgolette, quindi con un’alta probabilità di contenere l’informazione desiderata (Figura A.1).*

Figura A.1 Utilizzo dell’operatore virgolette.

Operatore ~ (tilde)

Permette di allargare una ricerca anche ai sinonimi del termine a cui è associato. Va anteposto, senza lasciare spazio, al termine del quale si reputano interessanti anche i sinonimi ai fini della

ricerca, come in `como ~alberghi`: questa query restituisce pagine dove sono presenti il termine "como" ma anche "alberghi", "hotel" ed eventuali altri sinonimi noti a Google.

SUGGERIMENTO *Il simbolo tilde non è presente sulla tastiera del computer. Per digitarlo è necessario utilizzare la combinazione di tasti ALT+0126: tenendo premuto il tasto ALT, bisogna digitare la sequenza numerica* 0126*, quindi rilasciare ALT.*

Operatore * (asterisco)

Può essere utilizzato in sostituzione di uno dei termini da ricercare. Si tratta di una sorta di jolly, utile, per esempio, per la ricerca di frasi contenenti termini incerti o di cui non si ricorda l'esatta sintassi. Si colloca nella query al posto del termine mancante o incerto, come in `mi * di immenso`: questa query elenca pagine contenenti "mi" seguito da un termine e quindi da "di immenso".

Operatori che lavorano su URL

Questo genere di operatori consentono di compiere ricerche utilizzando indirizzi internet (URL). Alcuni possono essere utilizzati in combinazione con altri operatori, altri funzionano solo se utilizzati singolarmente.

Operatore site:

Permette di restringere l'ambito di una ricerca all'interno di un singolo sito. Si colloca nella query seguito, senza spazio, dall'indirizzo (URL) del sito, come in `libri site:www.apogeonline.com`: questa query elenca solo pagine di `www.apogeonline.com` che contengono il termine "libri".

Operatore cache:

Permette di recuperare la copia cache di una pagina web di cui si conosce l'URL. È seguito, senza spazio, dall'indirizzo per il quale si vuole eseguire la ricerca. Può essere utilizzato insieme ad altri termini, come in cache:`www.apogeonline.com` catalogo: questa query fornisce la copia cache di `www.apogeonline.com` nella quale è evidenziato il termine "catalogo" (Figura A.2).

Figura A.2 Una copia cache: nell'intestazione è presente la data a cui risale la copia; poco più sotto si noti il messaggio "Sono stati evidenziati i seguenti termini usati nella ricerca".

TERMINOLOGIA *La copia cache di una pagina può essere vista come un sua fotografia. Può succedere che tra i risultati elencati da Google siano presenti link a pagine che, per diversi motivi, non sono più accessibili. La copia cache permette di risolvere questa impasse, mostrando la pagina in questione "fotografata" al momento dell'ultima visita dello spider di Google.*

Operatore link:

Permette di visualizzare tutte le pagine che contengono un link a uno specifico URL. È seguito, senza spazio, dall'indirizzo per il quale si vuole eseguire la ricerca, come in `link:www.apogeonline.com`: questa query elenca pagine che contengono un link indirizzato a `www.apogeonline.com`.

Operatore related:

Permette di ricercare pagine ritenute simili a quella indicata. È seguito, senza spazio, dall'indirizzo per il quale si vuole eseguire la ricerca, come in `related:www.apogeonline.com`: questa query elenca siti relazionabili a `www.apogeonline.com`.

Operatore info:

Restituisce alcune informazioni su un sito. È seguito, senza spazio, dall'indirizzo per il quale si vuole eseguire la ricerca, come in `info:www.apogeonline.com`: questa query fornisce una breve descrizione del sito `www.apogeonline.com` insieme a un pratico accesso alle ricerche effettuabili con gli operatori `cache:`, `related:`, `link:` e `site:`.

ATTENZIONE *Gli operatori* `link:`, `related:` *e* `info:` *non possono essere utilizzati in combinazione o abbinamento con altri operatori.*

Operatori che lavorano su sezioni di una pagina

Questi operatori permettono di circoscrivere le ricerche ad alcuni elementi di una pagina web, come titoli, link o URL. Utilizzarli vuol dire chiedere a Google di ricercare dei termini esclusivamente nei testi contenuti negli elementi HTML <`title`> <`a`> o negli indirizzi internet (URL). Restringere la ricerca a questi elementi è utile perché, in una pagina web ben concepita, essi sono altamente indicativi del contenuto della pagina stessa: un termine qui presente può quindi essere considerato come indice di maggiore attinenza alle necessità di ricerca.

Gli operatori di questo tipo si dividono in due categorie: `IN` e `ALLIN`.

ATTENZIONE *Gli operatori* `IN` *possono essere utilizzati in query complesse con altri operatori, mentre gli operatori* `ALLIN` *funzionano in modo corretto solo se utilizzati singolarmente.*

Operatore inurl:

Permette di circoscrivere la ricerca ai soli indirizzi internet (URL). L'oggetto della ricerca segue, senza spazio, l'operatore, come in `inurl:apogeo`: questa query recupera documenti nel cui indirizzo Internet è presente il termine "apogeo".

Operatore allinurl:

Versione potenziata di `inurl:`. Permette di indicare due o più termini da ricercare. Il primo termine deve essere digitato dopo l'operatore senza lasciare spazio, che invece deve sempre essere inserito tra i termini successivi, come in `allinurl:apogeo catalogo`: questa query recupera documenti nel cui indirizzo sono presenti i termini "apogeo" e "catalogo".

Operatore intitle:

Permette di circoscrivere la ricerca ai soli titoli dei documenti web (tag <`title`>). L'oggetto della ricerca segue, senza spazio, l'operatore, come in `intitle:apogeo`: questa query recupera solo pagine web nel cui titolo è presente il termine "apogeo".

Operatore allintitle:

Versione potenziata di `intitle:`. Permette di indicare due o più termini da ricercare. Il primo termine deve essere digitato dopo l'operatore senza lasciare spazio, che invece deve sempre essere

inserito tra i termini che seguono, come in `allintitle:apogeo catalogo`: questa query recupera solo documenti nel cui titolo sono presenti i termini “apogeo” e “catalogo”.

Operatore inanchor:

Permette di circoscrivere la ricerca ai soli link dei documenti web (tag <a>). L’oggetto della ricerca segue, senza spazio, l’operatore, come in `inanchor:apogeo`: questa query recupera solo pagine web nei cui link è presente il termine “apogeo”.

Operatore allinanchor:

Versione potenziata di `inanchor:`. Permette di indicare due o più termini da ricercare. Il primo termine deve essere digitato dopo l’operatore senza lasciare spazio, che invece deve sempre essere inserito tra i termini che seguono, come in `allinanchor:apogeo catalogo`: questa query recupera solo pagine web nei cui link sono presenti i termini “apogeo” e “catalogo”.

Altri operatori

In conclusione ecco un elenco di altri operatori non accomunabili da nessuna caratteristica, ma utili per svolgere ricerche particolari.

Operatore filetype:

In Rete i documenti sono distribuiti in diversi formati e non solo come pagine web (HTML). L’operatore `filetype:` permette di restringere l’ambito di una ricerca a un formato specifico. Si colloca nelle query seguito, senza spazio, dall’estensione del formato, come in apogeo `filetype:pdf`: questa query recupera solo file PDF che contengono il termine “apogeo”.

NOTA *L’operatore* `filetype:` *funziona con diversi formati di file, tra cui* PDF, DOC, XLS, PPT, RTF, TXT, JPG, GIF *e* PNG.

Operatore define:

Permette di recuperare la definizione di un termine specifico. Il termine da definire segue, senza spazio, l’operatore, come in `define:francescano`: questa query fornisce tutte le definizioni del termine “francescano” recuperabili in Rete. La pagina dei risultati mostra una o più definizioni del termine accompagnate dal link alla pagina web da cui la definizione è stata estratta (Figura A.3).

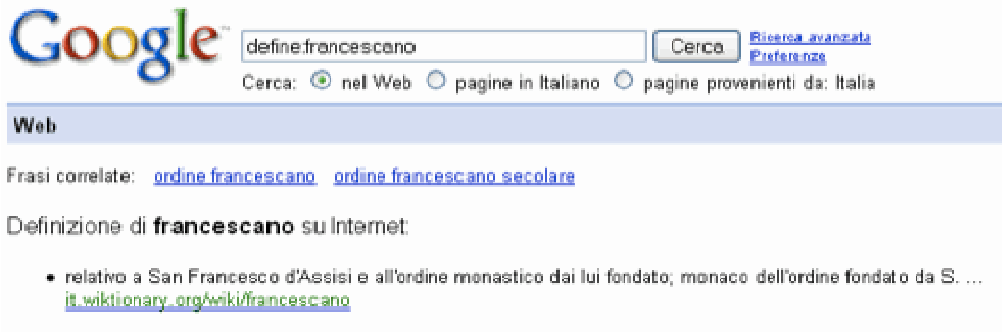


Figura A.3 La definizione del termine “francescano”.

Operatore in

Questo operatore si differenzia da tutti gli altri in quanto non è un operatore di ricerca in senso stretto, bensì di conversione. Lavora sulle unità di misura di valuta, volume, lunghezza, massa, potenza, informazione, elettricità, superficie, tempo e quantità. Nella query, a sinistra dell’operatore, va inserito il valore che si vuole convertire, mentre a destra va inserita la misura di riferimento. Per esempio, la query `1879 euro in US$` restituisce il valore in dollari USA di 1879 euro (Figura

A.4).

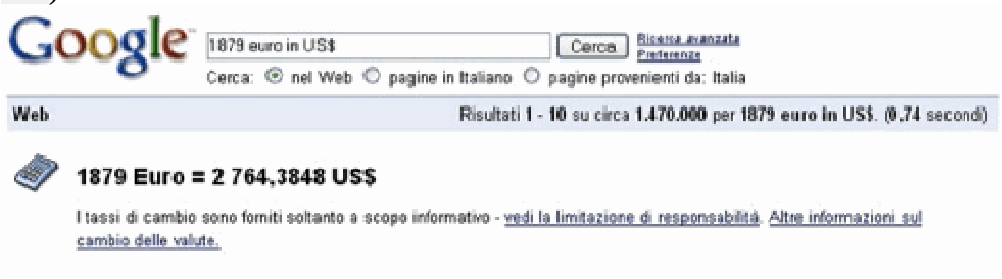


Figura A.4 Una conversione effettuata con l'operatore in.

Suggerimenti per la ricerca oltre Google

Prima di concludere ricordiamo un paio di casi in cui Google potrebbe non bastare.

Può capitare che Google ci indirizzi a pagine web molto lunghe e articolate, nelle quali i termini della query non sono subito individuabili. In questi casi, per localizzarli e quindi concentrare l'attenzione sul passaggio del documento che probabilmente contiene l'informazione desiderata, è possibile premere la combinazione di tasti `CTRL+F`: il browser aprirà una piccola finestra dove digitare nuovamente il termine desiderato, la cui prima occorrenza nel documento verrà evidenziata in tempo reale (Figura A.5).

Figura A.5 La finestra Trova nel browser Firefox permette di individuare velocemente termini in un documento, in questo esempio "mendicante".

SUGGERIMENTO *Per spostarsi su altre eventuali occorrenze dello stesso termine nella pagina bisogna premere il tasto F3.*

A volte può invece essere necessario contattare il webmaster o una persona responsabile del sito che si sta visitando. Molti siti prevedono una sezione *Contatti*, ma in assenza di questa si può fare riferimento ai servizi Whois.

Si tratta di servizi web che dato un URL forniscono diverse informazioni su di esso, tra cui il

cosiddetto *Registrant* (Registrante), cioè la persona o la società che ha registrato il dominio.

SUGGERIMENTO *Esistono diversi servizi Whois; una semplice ricerca con Google per il termine "whois" accompagnato dal dominio di primo livello a cui appartiene il sito di interesse (per esempio com, eu, it e così via) ne fornirà un lungo elenco.*

Spesso in questo modo è possibile recuperare informazioni molto dettagliate, tra cui indirizzi postali e contatti telefonici (Figura A.6). Si tratta di informazioni sensibili, che devono quindi essere utilizzate con cautela e nel rispetto delle leggi, in particolare quella sulla privacy.

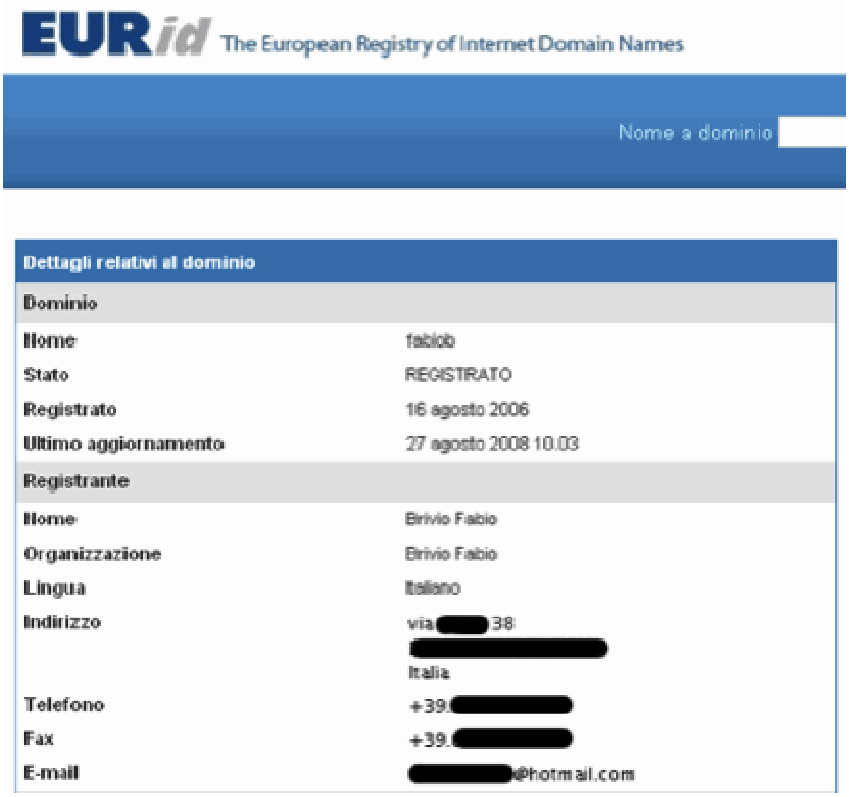


Figura A.6 Risultati restituiti dal Whois di EURid (specifico per i domini .eu) per il dominio di secondo livello fabiob.

Per concludere

In queste pagine abbiamo visto diversi operatori che possono essere utilizzati per creare query specifiche, adatte a svolgere interrogazioni mirate.

Una minima conoscenza di questi costrutti permette di sfruttare con maggiore profitto le potenzialità di un motore di ricerca come Google.

Tuttavia, come ricordato all'inizio di questa appendice, Google è un mondo complesso e articolato, che racchiude diversi servizi e strumenti di ricerca.

Avvicinandosi al suo utilizzo non deve quindi mancare una visita agli strumenti e servizi accessibili dal menu posizionato in alto a sinistra nella pagina principale (Figura A.7).

Figura A.7 Alla scoperta dei servizi di Google.

Basta veramente poco per scoprire che le informazioni servite da Google vanno ben oltre le normali pagine web e che con pochi clic è, per esempio, possibile passare dalla ricerca sul Web alla consultazione della copia digitale di un libro, alla ricerca e visualizzazione di video o ancora alla creazione e consultazione di un itinerario stradale personalizzato.

Indice