Università degli Studi di Brescia
Facoltà di Economia
Esercizi per gli studenti del corso di Linguaggi di Programmazione

A. Bugatti - alessandro.bugatti@istruzione.it

Indice

Introduzione

Questo eserciziario è suddiviso in capitoli che seguono l'andamento delle lezioni del corso di Linguaggi di Programmazione. Ogni capitolo affronta un argomento (HTML, CSS, ecc.) proponendo degli esercizi risolti e per ogni esercizio viene spiegato qual è il problema che si intende risolvere, la logica d'impostazione della soluzione ed eventuali passaggi operativi qualora ve ne fossero. Inoltre per ogni esercizio è segnato qual è il file contenente il codice risolutivo.

Nelle varie appendici vengono spiegati operativamente come debbono essere utilizzati gli strumenti software allegati a questo eserciziario. Tutti gli strumenti software sono di libero utilizzo e, dove possibile, si è cercato di utilizzare applicativi multi-piattaforma.

Capitolo 1

Il linguaggio HTML

In questo primo capitolo verrà esaminata la costruzione di un sito statico per la ditta di arredamento Muffa Biluffa. Il sito nella sua versione finale avrà un menù di navigazione e una serie di pagine per esplorare il catalogo di mobili.

1.1 Pagina vuota con titolo

- PROBLEMA: si vuole creare una pagina HTML composta solo da un titolo centrato con la scritta Muffa Biluffa.
- SOLUZIONE: la prima cosa che apparirà nella pagina HTML sarà il DOCTYPE, che in questo esempio e in tutti i successivi sarà di tipo HTML 4.01 Transitional. Si è scelto questo DOCTYPE in quanto è attualmente il più diffuso, anche se verrà poi superato dalle moderne versione di XHTML. Inoltre in questo modo la validazione risulta più semplice da raggiungere (ognuno di questi esercizi è stato validato con HTML Tidy[1] e, fatta salva qualche piccola modifica, dovrebbe risultare anche codice XHTML valido). Per inserirlo facilmente in tutte le pagine HTML o lo si copia e incolla in ogni pagina oppure, se si utilizza Scite come editor, basta schiacciare la combinazione di tasti CTRL + '<' e nel menù che appare scegliere HTML (Struttura completa).
  Nell'HTML vero e proprio possiamo distinguere, oltre ai tag di apertura e chiusura *html*, le due sezioni *head* e *body*. Nella sezione *head* si trovano alcuni tag aggiunti automaticamente dal validatore Tidy, mentre è invece importante inserire il tag *meta* indicante la codifica utilizzata tramite:

```
<meta http-equiv="content-type"
content= "text/html charset=ISO-8859-1">
```

  Nella sezione *body* invece è sufficiente inserire il tag *h1* per inserire il titolo (ogni browser si preoccuperà poi di come rendere il testo all'interno

[1] HTML Tidy è un'utilità open source per la validazione e la "ripulitura" di file HTML e XML sviluppato da David Raggett all'interno del W3C.

di questo tag, che si ricorda non è relativo alla presentazione ma solo alla struttura).

- FILE: 001/index.html

1.2 Pagina vuota con titolo centrato

- PROBLEMA: modificare la pagina precedente in modo che la scritta Muffa Biluffa appaia centrata.
- SOLUZIONE: basta aggiungere l'attributo *align* al tag *h1* e dargli il valore *center*. Questa soluzione in realtà fa uso di una proprietà che influisce sulla presentazione e quindi è attualmente classificata come *deprecated*, quindi non dovrebbe essere usata (anche se tutti i browser la interpretano correttamente). Al suo posto, come vedremo nel prossimo capitolo, verranno usate le proprietà CSS. L'utilizzo di proprietà *deprecated* sarà presente anche nei prossimi esercizi, sempre con lo scopo di mostrare come ottenere certi effetti utilizzando solo codice HTML, considerato anche che moltissimo codice "reale" fa ancora uso di queste proprietà.
  Un'altra possibilità è quella di usare il tag *center*, anch'esso deprecato.
- FILE: 001/index001.html, 001/index002.html

1.3 Immagini ai lati del titolo

- PROBLEMA: si vogliono aggiungere delle immagini ad entrambi i lati del titolo per rendere l'intestazione più accattivante.
- SOLUZIONE: come prima cosa bisogna inserire l'immagine e per far questo si fa uso del tag *img*.

```
<img src="img/logo.jpg" alt="Logo aziendale"
border="0">
```

In questo tag vanno valorizzati l'attributo *src* con il nome del file da visualizzare, l'attributo *alt* con il testo che verrà mostrato nel caso non sia visualizzata l'immagine e l'attributo *border* (deprecato) per aggiungere eventualmente una cornice. Per quanto riguarda l'attributo *src* è sempre una buona idea usare dei percorsi relativi (nell'esempio img/logo.jpg), in modo che se il sito venisse spostato in una nuova posizione sul server non nascerebbero problemi con i riferimenti alle immagini. Se ad esempio fosse stato inserito

```
src="www.miosito.com/img/logo.gif"
```

il cambiamento del nome di dominio o lo spostamento del sito in una sottocartella del web server avrebbe reso irraggiungibile l'immagine. Un'altra

buona regola è quella di rispettare il case del nome del file, in modo da non aver problemi se il sito fosse ospitato in hosting su un server con file-system case-sensitive[2] (tutti i S.O. Unix-like). L'attributo *alt* anche se non indispensabile è buona norma, per questioni di accessibilità, valorizzarlo sempre, in modo che ad esempio uno *screen reader*[3] sia in grado di rendere il contenuto dell'immagine ad un utente con deficit visivi oppure che i browser testuali possano rendere il più possibile correttamente la pagina. Il problema però non è ancora risolto: l'elemento *h1*, che è un elemento di blocco, causa lo scorrimento del titolo nello spazio sottostante (vedi index003.html), non permettendo di raggiungere l'obiettivo cercato. Per far questo utilizzando solo HTML dobbiamo ricorrere ad una tabella di una riga e tre colonne senza margini visibili: in questo modo possiamo posizionare la prima immagine nella prima casella, il testo del titolo nella seconda e la stessa immagine di nuovo nella terza casella (se una stessa immagine compare più volte in una pagina il browser la richiede al server una volta sola). Questo approccio viola nuovamente la regola di separazione tra la struttura e la presentazione, perchè si è utilizzato un tag strutturale (*table*) per ottenere un effetto di presentazione grafico (l'affiancamento delle immagini al testo). Vedremo successivamente come ottenere lo stesso effetto utilizzando i CSS.

- FILE: 001/index003.html, 001/index004.html

1.4 Menù orizzontale

- PROBLEMA: si vuole inserire un menù orizzontale con le seguenti voci: Chi siamo, Dove siamo, I nostri articoli.
- SOLUZIONE: qui le soluzioni sarebbero molte, quella utilizzata è molto semplice e utilizza i tag *big* (deprecato) per evidenziare il testo e il tag *span* che viene utilizzato nel caso si voglia differenziare il comportamento delle voci di menù aggiungendo degli stili. Da notare che è stata usata l'entità * * (Non-Breaking SPace) per inserire uno spazio. In questo caso particolare si poteva anche usare uno spazio "normale", ma si potrebbero trovare dei codici HTML che contengono una serie di queste entità per formattare il testo, ad esempio distanziandolo dal bordo sinistro. Generalmente questi codici sono creati da editor WYSIWYG[4] e l'utilizzo di * * è sconsigliato perché effetti analoghi possono essere ottenuti con i CSS.
- FILE: 001/index005.html

[2] In tali sistemi operativi logo.gif, Logo.gif e logo.GIF sono tutti file differenti

[3] Uno screen reader è un software che cerca di identificare e interpretare ciò che è mostrato a video, generalmente producendo come output del parlato, ma anche codice braille.

[4] WYSIWYG è l'acronimo di What YouSee Is What You Get e si riferisce a quegli editor che mostrano all'utente il risultato finale: nel caso di editor HTML un esempio potrebbe essere FrontPage o DreamWeaver

1.5 Inserire dei link ipertestuali

- PROBLEMA: si vogliono inserire dei link alle pagine Chi siamo, Dove siamo, I nostri articoli.
- SOLUZIONE: si utilizza il tag *a* (anchor) per linkare la pagina a degli altri documenti (lo stesso tag può essere usato anche per fare dei riferimenti interni al documento, ad esempio nel caso di un indice e di diverse sezioni). L'attributo da valorizzare è *href*, che deve contenere il nome del file linkato. Anche in questo caso, come per le immagini, è consigliabile usare dei riferimenti relativi e non assoluti e rispettare il case dei nomi dei file.
- FILE: 001/index006.html, 001/index007.html, 001/index008.html

1.6 Inserire un paragrafo

- PROBLEMA: si vuole inserire del testo all'interno di una pagina.
- SOLUZIONE: sebbene il testo possa essere inserito anche esternamente a qualsiasi tag, si consiglia di inserirlo all'interno di un tag *p* (paragraph), in modo che il browser inserisca dello spazio fra paragrafi diversi e anche per poter successivamente applicare degli stili. Si può notare come il testo inserito serva solo a dar l'impressione di un contenuto per avere visivamente l'idea di come verrebbe resa la pagina. Molti strumenti per la creazione di pagine web creano testi con una serie di varianti di Lorem Ipsum[5].
- FILE: 001/index006.html

1.7 Inserire un'immagine in un paragrafo

- PROBLEMA: si vuole inserire una o più immagini all'interno di un paragrafo.
- SOLUZIONE: nella prima soluzione l'immagine viene inserita esternamente al paragrafo, che essendo un elemento di blocco fa scorrere l'immagine sotto di esso. Se invece viene inserita nel paragrafo il testo le scorre intorno in modi che possono essere diversi. Nel caso più semplice l'immagine verrà posizionata a sinistra e la prima riga del testo si troverà alla sua destra allineata all'estremo inferiore dell'immagine. Solitamente questo effetto non è quello desiderato ed è più probabile che si desideri che il testo (e non solo la prima riga) scorra su tutto un lato dell'immagine . Per far questo è possibile utilizzare l'attributo *align* (deprecato) e settarlo a *right*

[5] Questo testo è usato fin dal 1500 nell'industria tipografica perché, pur non avendo un significato, da l'impressione di essere testo reale e quindi è utile per valutare visivamente la resa di una pagina. Il testo originale è comunque preso da un trattato di Cicerone a cui sono state spostate le parole.

(l'immagine sarà a destra e il testo a sinistra) o a *left* (l'immagine sarà a sinistra e il testo a destra). Esistono anche altri valori ma sono meno utilizzati.

- FILE: 001/index007.html, 001/index007a.html, 001/index007b.html, 001/index007c.html

1.8 Aggiungere una lista non ordinata

- PROBLEMA: si vuole aggiungere una lista non ordinata per ognuna delle tipologie di mobili (camera da letto, bagno, sala, cucina), in cui ogni voce faccia riferimento a una pagina diversa .
- SOLUZIONE: in questo caso il tag da utilizzare è *ul* (unordered list) per iniziare la lista e per ogni voce della lista il tag *li* (list item). Ognuna delle voci della lista poi conterrà al suo interno un link alla pagina corretta.
- FILE: 001/index008.html

1.9 Aggiungere un footer alle pagine

- PROBLEMA: si vuole aggiungere una sezione a fondo pagina (solitamente denominata footer) che contiene informazioni pertinenti al sito in generale e non alla pagina che la contiene in particolare
- SOLUZIONE: non essendo questa una parte che necessita di tag particolari ma può essere fatta in qualunque modo risulti appropriato, sono stati inserite due informazioni solitamente utili per l'utente: i contatti fisici tradizionali dell'azienda (numero di telefono, fax, indirizzo, ...) e un link per inviare una mail all'amministratore del sito. Per creare un link che permetta di aprire il client di posta con un messaggio preindirizzato è sufficiente nell'attributo *href* del tag *a* inserire il prefisso *mailto:* seguito dall'indirizzo di posta elettronica. Inoltre è stata inserita anche un'immagine che indica che la pagina è HTML 4.01 valido tramite il tag

```
<a href="http://validator.w3.org/check?uri=referer">
<img src="img/valid-html401.png"
    alt="Valid HTML 4.01 Transitional">
</a>
```

  Chiaramente questo non garantisce che la pagina sia veramente valida, ma permette all'utente, cliccando sull'immagine, di validare la pagina utilizzando il validatore del W3C[6].
- FILE: 001/index008.html

[6] Il W3C è un consorzio internazionale che si occupa di sviluppare specifiche, linee guide, software e strumenti vari per portare il WEB al suo pieno potenziale. Il validatore che si trova all'indirizzo validator.w3c.org è un software che analizza una pagina HTML e ne indica la validità o gli eventuali errori presenti.

1.10 Creazione di tabelle

- PROBLEMA: utilizzare delle tabelle per mostrare delle coppie di dati di tipo attributo-valore
- SOLUZIONE: in questo caso viene utilizzata una tabella per uno scopo legittimo, in quanto le coppie di dati attributo-valore hanno un tipo di struttura che ben si presta a essere rappresentata in una tabella, nella cui prima colonna vengono inseriti i nomi degli attributi (codice, tipologia,...) e nella seconda i loro rispettivi valori. Dopo aver utilizzato il tag *table* per iniziare la tabella, ogni riga successiva viene compresa all'interno dei tag *tr* (table row) e ogni casella appartenente ad una riga nel tag *td* (table data). Si può quindi vedere come una tabella sia organizzata per righe e questo comporta che il numero di colonne non necessariamente coincide tra una riga e quelle adiacenti (dipende da quanti elementi *td* contiene ogni riga). In questo caso il risultato è che le righe con un numero di colonne inferiori a quello massimo avranno soltanto le prime celle, mentre al posto delle successive non si avrà niente. Se si volesse invece allargare una cella in modo che comprenda più colonne bisognerebbe utilizzare l'attributo *colspan* del tag *td*.
  Da notare infine come nel codice HTML "vecchio" le tabelle erano spesso usate per questioni di layout (come nell'esercizio 1.3), considerando anche il buon supporto dei browser alle tabelle e quindi la possibilità di ottenere risultati uguali su browser diversi.
- FILE: 001/camera.html, 001/letto.html, 001/comodino.html, 001/armadio.html

Capitolo 2

I fogli di stile - CSS

In questo capitolo verranno ripresi gli esercizi del capitolo precedente e verranno reimplementati utilizzando i fogli di stile per la resa grafica, lasciando inalterato il contenuto. Si vedrà come evitare di usare gli attributi HTML deprecati ottenendo gli stessi risultati grafici e anzi potendone ottenere di completamente nuovi.

2.1 Applicare uno stile locale

- PROBLEMA: creare il titolo della pagina di colore rosso, centrato e piuttosto grande.
- SOLUZIONE: utilizziamo come prima modalità uno stile locale, cioè una dichiarazione che viene inserita all'interno dei tag di cui si vuole modificare l'aspetto grafico. Per fare questo si ricorre all'attributo *style* e come valore si inserisce una stringa con i vari attributi e i rispettivi valori. In questo caso:

```
<h1 style="color: red; font-family:Verdana, Helvetica;
font-size=24px; text-align:center;">
```

  dove il significato degli attributi è palese. Da notare due aspetti:

  - la famiglia dei font (*font-family*) può essere un qualunque nome di font (anche più di uno come in questo caso): il browser sceglierà il primo font a disposizione che trova e se non lo trova lo sostituirà con uno di default. Non essendo attualmente possibile sapere a priori quali font sono supportati da tutti i browser su ogni sistema operativo una possibilità è quella di indicare al posto di un font specifico (o insieme a un font specifico) una famiglia generica di font, sarà poi il browser a prendersi cura di utilizzare un font con le giuste caratteristiche. La famiglie di font sono: *serif* (proporzionati e con le grazie),

sans-serif (proporzionati e senza grazie), *monospace* (non proporzionati), *cursive* (emulano la calligrafia umana) e *fantasy* (nessuno dei precedenti)

- uno stile locale si applica solo al blocco contenuto all'interno del tag in cui è inserito, in questo caso *h1*. Se all'interno dello stesso file dovesse apparire un altro blocco di tipo *h1* a questo non verrà applicato nessuno stile.

- FILE: 002/index.html

2.2 Applicare un foglio di stile interno

- PROBLEMA: ottenere lo stesso risultato dell'esercizio precedente applicando un foglio di stile interno.

- SOLUZIONE: stavolta la dichiarazione di stile viene inserita all'interno della sezione *head* del file HTML e, come vedremo, è del tutto analoga a quella di un foglio di stile esterno. Il tag da utilizzare è ancora *style*, indicando come attributo il tipo *type="text/css"*. Rispetto all'esercizio precedente la differenza è solo nel fatto che adesso a tutti gli elementi h1 contenuti nel file si applicheranno le regole CSS.

- FILE: 002/index001.html

2.3 Applicare un foglio di stile esterno

- PROBLEMA: ottenere lo stesso risultato dell'esercizio precedente applicando un foglio di stile esterno.

- SOLUZIONE: anche in questo caso la dichiarazione di stile viene inserita all'interno della sezione *head* del file HTML e viene utilizzato il tag *link*

```
<link rel="stylesheet" type="text/css" href="001.css" />
```

  Tutti gli esempi che seguiranno verranno realizzati utilizzando fogli di stile esterni.

- FILE: 002/index002.html, 002/001.css

2.4 Realizzare il titolo con il logo

- PROBLEMA: ottenere lo stesso risultato dell'esercizio 1.3 applicando un foglio di stile.

- SOLUZIONE: in questo caso viene utilizzata la proprietà *float*, che toglie un elemento dal flusso normale di visualizzazione per farlo scorrere a destra o sinistra di altri elementi. In questo caso le due immagini sono state inserite all'interno di tag *div*, per renderle degli elementi di blocco. Il comportamento normale in questa situazione farebbe si che le immagini e il contenuto del tag *h1* vengano posizionati uno sotto l'altro essendo tre elementi di blocco. Se però viene dato il valore *left* alla proprietà *float* della prima immagine e al tag *h1*, si ottiene che risultano allineati perché la prima immagine permette ai blocchi successivi di scorrergli a destra (essendo lei posizionata a sinistra con il floating valorizzato a *left*), e il tag *h1* permette all'ultima immagine di scorrergli a destra.
- FILE: 002/index003.html, 002/002.css

2.5 Aggiungere un menù orizzontale

- PROBLEMA: aggiungere un menù orizzontale come nell'esercizio 1.4.
- SOLUZIONE: la prima soluzione serve per ottenere un risultato simile a quello dell'esercizio 1.4. Il menù nel file HTML viene inserito all'interno di un *div* di classe *menu*, in modo da poter aggiungere un colore di sfondo al menù. Ogni voce del menù è sempre un *div* di classe *menu_item*. Nel file 003.css vengono settate alcune proprietà delle voci di menù per aggiungere dei bordi, un colore di sfondo e dei margini. Alcuni browser (tra cui Firefox) permettono di selezionare fogli di stile diversi se una pagina HTML ne ha associato più di uno. Per associare più di uno stile a una pagina occorre aggiungere un tag link per ogni stile alternativo in questo modo

```
<link rel="alternate stylesheet" type="text/css" href="004.css
    "   title="alternativo">
```

  In questo esempio è stato aggiunto il foglio di stile *004.css* e il browser permetterà di visualizzarlo scegliendo in un apposito menù (in Firefox si trova Visualizza -> Stile pagina...). In questo stile rispetto al precedente si è cercato di rendere il menù come nella navigazione a schede, aggiungendo il pseudo-selettore *hover* per dare l'effetto di cambio del colore al passaggio del mouse e aggiungendo uno stile al link attivo (*activelink*) in modo da differenziarlo dagli altri.
- FILE: 002/index004.html, 002/003.css, 002/004.css

2.6 Inserire un'immagine e posizionarla in punti diversi

- PROBLEMA: aggiungere un'immagine all'interno di un paragrafo e modificarne la posizione con i fogli di stile.

- SOLUZIONE: anche in questo caso sono stati usati due fogli di stile per ottenere stili diversi. All'interno di *005.css* si è semplicemente aggiunta la proprietà *float* con il valore *left*, in modo che l'immagine si trovi sulla sinistra rispetto al testo (se si modifica il valore a right si otterrà l'effetto contrario). In *006.css* invece si è anche modificata la dimensione dell'immagine portando la sua larghezza a 200 pixel (l'altezza si adatterà di conseguenza) e aggiungendo del padding per evitare che il testo rimanga incollato all'immagine.

- FILE: 002/index005.html, 002/005.css, 002/006.css

2.7 Modificare i parametri del testo

- PROBLEMA: modificare le proprietà di un testo, come l'allineamento, il font usato, il peso, lo stile.

- SOLUZIONE: con lo stile *007.css* si vede come la pagina assuma una presentazione piuttosto sgradevole. Per migliorarla si possono modificare alcune proprietà dei font ottenendo un risultato migliore con il foglio *008.css*. In questo foglio è stato cambiato il font dei link ad Arial, il footer è stato centrato, la scritta introduttiva è stata ingrandita e le scritte del footer sono state rese molto piccole, con l'indirizzo e-mail in italico.

- FILE: 002/index006.html, 002/007.css, 002/008.css

2.8 Modificare le caratteristiche di una tabella

- PROBLEMA: aggiungere proprietà alle celle di una tabella utilizzando selettori di attributo, classi e pseudo-classi dinamiche.

- SOLUZIONE: nel file *letto.html* si vuole che la tabella abbia la prima colonna delle caratteristiche con un font diverso e italico. Per far questo nel file HTML si fa in modo che ogni cella della prima colonna abbia un *id* associato (*id* che contiene il nome del campo) e utilizzando i selettori di attributi si può ottenere quanto desiderato. In particolare volendo che la cella con *id* uguale a *codice* sia in grassetto è sufficiente aggiungere un selettore di *id*.
  Nel file *comodino.html* si vogliono colorare in maniera alternata le celle della prima colonna. Di nuovo nel file *html* si aggiungerà ad ogni cella la classe *pari* o *dispari*, avendo cura di alternarle e poi nel file *011.css* si applicheranno colori di sfondo diversi alle due classi.
  Infine in *armadio.html* si vuole che scorrendo con il mouse sulle celle di destra queste vengano evidenziate cambiando il colore di sfondo: per far questo è sufficiente utilizzare lo pseudo selettore hover nel file *012.css*.

- FILE: 002/letto.html, 002/armadio.html, 002/comodino.html, 002/010.css, 002/011.css, 002/012.css

Capitolo 3

XML

In questo capitolo verranno proposti dei semplici esercizi per imparare a creare dei file XML conformi a un DTD e a uno Schema. Inoltre verrà proposto un esercizio per vedere come manipolare un file XML tramite il linguaggio di programmazione Java.

3.1 Creare un file XML

- PROBLEMA: creare un file XML che permetta di memorizzare informazioni relative a una rubrica telefonica. Ogni persona nella rubrica deve essere caratterizzata da un nome, un cognome, un indirizzo e un numero di telefono.
- SOLUZIONE: per creare il file è sufficiente utilizzare un qualunque editor di testo (o un editor specializzato), facendo eventualmente attenzione all'encoding utilizzato. La prima riga deve necessariamente contenere la dichiarazione di quale versione XML si sta utilizzando e dell'encoding che nel nostro caso sarà

```
<?xml version="1.0" encoding="UTF-8"?>
```

  Successivamente inizia il file XML vero e proprio con eventuali commenti, compresi tra i simboli <!- - e - ->. La root del documento in questo caso sarà l'elemento *rubrica*, che essendo l'elemento root deve essere presente una volta soltanto e come ogni elemento XML deve avere un tag di chiusura. Successivamente verranno inseriti degli elementi *persona*, ognuno contenente gli elementi *nome*, *cognome*, *indirizzo* e *telefono*, con i rispettivi tag di chiusura.
- FILE: 003/rubrica.xml

3.2 Creare un file DTD

- PROBLEMA: creare un file DTD (Document Type Definition) che serva per validare il file XML dell'esercizio precedente.
- SOLUZIONE: utilizzando nuovamente un editor di testo come prima cosa indichiamo come deve essere fatto l'elemento radice, che in questo caso è *rubrica* che può contenere una o più *persone*

```
<!ELEMENT rubrica (persona+)>
```

  come indicato dal segno +. Ogni persona poi contiene esattamente un nome, un cognome, un indirizzo e un numero di telefono. Ognuno di questi elementi contiene dati che possono essere analizzati dal parser XML e che vengono indicati con *PCDATA*.
- FILE: 003/rubrica.dtd, 003/rubrica1.xml

3.3 Creare uno Schema XML

- PROBLEMA: creare uno Schema XML che serva per validare il file XML dell'esercizio 3.1.
- SOLUZIONE: utilizzando nuovamente un editor di testo dobbiamo come prima cosa inserire la dichiarazione della versione XML, perché uno Schema XML è esso stesso un file XML valido. Successivamente si indica a che schema si fa riferimento tramite la dichiarazione

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

  dove si può vedere che lo schema al quale si fa riferimento è quello del W3C presente all'URL http://www.w3.org/2001/XMLSchema. La sintassi degli schemi è più ricca di quella dei DTD e permette di specificare meglio le relazioni tra i vari elementi: come si può notare la *rubrica* è un tipo complesso formato da una sequenza di zero o più persone, dove la *persona* è a sua volta un tipo complesso formato da una sequenza di *nome*, *cognome*, *indirizzo* e *telefono*, ognuno dei quali è di tipo stringa e compare esattamente una volta.
- FILE: 003/rubrica.xsd, 003/rubrica2.xml

3.4 Manipolare un file XML con Java

- PROBLEMA: creare un programma Java che carichi il file XML dell'esercizio 3.1 e permetta di gestire i dati contenuti come una semplice rubrica: dovrà quindi essere possibile effettuare una ricerca per cognome, aggiungere un nuovo nominativo ed eliminare un nominativo esistente.

- SOLUZIONE: come prima cosa bisognerà includere i package che contengono le varie classi per il caricamento, la manipolazione e il salvataggio di file XML. Nel main, dopo aver controllato che la chiamata del programma sia fatta passando il nome del file della rubrica, bisogna effettuare una serie di passaggi per caricare il documento: viene istanziato un nuovo oggetto di tipo *DocumentBuilderFactory*, dal quale si ottiene un *DocumentBuilder* che è l'oggetto che si occupa di fare il parsing del file e il risultato viene inserito in un oggetto di tipo *Document* che sarà quello su cui verranno effettuate le successive operazioni. Si può notare come viene anche fatta una gestione degli errori essenziale, in cui vengono rilevate le eccezioni che potrebbero generarsi in questa fase e i messaggi di errore vengono stampati a video (siccome il programma è didattico la gestione degli errori è tenuta al minimo).
  Per ognuna delle operazioni (ricerca, inserimento e cancellazione) è stato fatto un metodo separato, che verrà brevemente commentato:

  - *find*: come prima cosa viene creata la lista dei nodi di tipo persona,

    ```
    NodeList personaLst = doc.getElementsByTagName("persona");
    ```

    all'interno della quale si farà la ricerca. Con un ciclo *for* si scorre tutta la lista per vedere se si trova il cognome cercato oppure se non è presente: nel primo caso si ritorna una stringa con tutti i dati del nominativo, altrimenti la stringa "Non trovato". All'interno del for ogni nodo viene recuperato tramite il metodo *item*, passando l'indice del nodo. Il nodo recuperato viene poi "castato" per farlo diventare di tipo elemento (poiché un nodo potrebbe essere anche di altri tipi) e, in maniera analoga a quanto fatto con persona, si estrae una lista di nodi *cognome* (in questo caso uno solo). Su quel nodo, applicando i metodi opportuni, viene fatto il confronto con la stringa cercata e nel caso dia risultato positivo si costruisce il valore di ritorno.

    ```
    Node personaNode = personaLst.item(i);
    Element personaElement = (Element) personaNode;
    NodeList cognomeLst = personaElement.getElementsByTagName(
        "cognome");
    Element cognomeElement = (Element) cognomeLst.item(0);
    if(s.equals(((Node)cognomeElement.getChildNodes().item(0))
        .getNodeValue()))
    {
    //Costruzione del valore di ritorno
    String information = s + "␣";
    ```

  - *insert*: come prima istruzione viene recuperato il nodo radice, quello che contiene il valore *rubrica*. Per ogni elemento (*nome*, *cognome*, *indirizzo*, *telefono*) che si vuole inserire, viene creato un nodo e vengono immessi i valori passati alla funzione come nel codice seguente

    ```
    Element nome = doc.createElement("nome");
    nome.insertBefore(doc.createTextNode(n), nome.getFirstChild());
    ```

Come si può vedere viene creato un elemento *nome* e, tramite il metodo *insertBefore*, viene inserito il valore (in questo caso *n* che è un parametro passato alla funzione *insert*) all'interno dell'elemento. Dopo aver creato i quattro elementi *nome*, *cognome*, *indirizzo* e *telefono*, questi vengono inseriti all'interno di un nodo *persona* precedentemente creato e infine questo nodo viene inserito all'interno del nodo radice.

- *delete*: questo metodo funziona in maniera simile al metodo *find*: dopo aver cercato se il nodo da cancellare è presente, questo viene eliminato utilizzando il metodo *removeChild* applicato al nodo *persona*.
- *writeXmlFile*: questo metodo implementa la scrittura di un documento DOM su di un file e viene fatto seguendo dei passaggi standard.

```
try {
    // prepara il documento DOM per la scrittura
    Source source = new DOMSource(doc);
    // Prepara il file di output
    File file = new File(filename);
    Result result = new StreamResult(file);
    // Scrive il documento DOM sul file utilizzando un
        Transformer
    Transformer xformer = TransformerFactory.newInstance().
        newTransformer();
    xformer.transform(source, result);
} catch (TransformerConfigurationException e) {}
catch (TransformerException e) {}
```

In questo caso non vengono gestiti gli errori ma solo catturati.

- FILE: 003/rubrica.java, 003/test.xml

Capitolo 4

Programmazione lato client con Javascript

In questo capitolo verrà utilizzato il linguaggio Javascript per creare dei semplici programmi lato client attraverso la costruzione di classi e la programmazione orientata agli oggetti: in particolare si cercherà di tenere separata quella che è la logica applicativa da ciò che è l'interfaccia verso l'utente. Come ultimo esercizio si vedrà invece come attraverso Javascript sia possibile caricare un file XML e visualizzarlo nel browser.

4.1 Convertitore di valuta

- PROBLEMA: creare un semplice convertitore di valuta che trasformi una quantità da lire in euro e viceversa.
- SOLUZIONE: in questo e negli esercizi seguenti cercheremo di separare la logica applicativa dall'interfaccia, quindi avremo sempre due file, uno con estensione .js che conterrà la classe che implementa la logica applicativa e un altro con estensione .html che realizza l'interfaccia (avremo solitamente anche un terzo file .css per gestire la grafica, del quale però non ci occuperemo). La classe che permetterà di gestire la conversione verrà chiamata Convertitore e conterrà solo tre metodi:
  - il costruttore: si occupa di settare il fattore di conversione LIRE_EURO e la "direzione" del cambio, da euro verso lire o viceversa.
  - il metodo *convert*: fa la conversione vera e propria, tenendo conto di quella che è la direzione del cambio.
  - il metodo *toggle*: inverte la direzione del cambio.

  Il file *convertitore.html* contiene la semplice interfaccia, composta da due text box, una per inserire il valore da convertire e l'altra che mostra il

risultato, un bottone per effettuare la conversione e due radio button per scegliere che tipo di conversione effettuare. Questo file include il file precedente tramite la direttiva

```
<script src="convertitore.js" type="text/javascript">
</script>
```

all'interno della sezione *head* e in risposta all'evento *onload* della pagina crea un oggetto di tipo Convertitore, che verrà utilizzato per fare le conversioni.
In questo esercizio inoltre si è anche fatto un file separato *interfacciaConvertitore.js*, che contiene due funzioni: *changeLabels* che scambia le etichette dei text box in relazione al tipo di conversione da effettuare e *check* che effettua un minimo controllo sui valori in input.

- FILE: 004/convertitore.html, 004/convertitore.js, 004/interfacciaConvertitore.js, 004/convertitore.css

4.2 Gioco del tris

- PROBLEMA: creare un programma che permetta a due persone di giocare a tris e sia in grado di segnalare il termine del gioco.
- SOLUZIONE: la classe principale verrà chiamata Tris e come proprio stato conterrà una matrice $3 X 3$ che rappresenta il campo di gioco, un attributo *giocatore* per tenere memoria di chi deve fare la mossa corrente e un attributo *mossa* per sapere quante mosse sono state già fatte. La matrice viene inizializzata con tutti 0 per indicare che non è stata fatta nessuna mossa: per indicare una mossa del primo giocatore verrà usato il numero 1 e per indicare una mossa del secondo il numero 2.
  I metodi implementati saranno:
  - il costruttore: setta il campo di gioco, il turno del giocatore (1) e il numero di mosse (0).
  - il metodo *move*: inserisce la mossa di un giocatore in una certa casella rappresentata dalle coordinate x e y. Non c'è necessità di fare un controllo sui loro valori perché l'interfaccia non consente di inserirli scorrettamente, nel caso la casella scelta contenga già un valore non viene fatto niente. Inoltre viene modificato il turno e incrementato il numero di mosse.
  - il metodo *check*: controlla lo stato della partita. Ritorna 1 se ha vinto il giocatore 1, 2 se ha vinto il giocatore 2, 0 se è patta e 3 se non è ancora finita.
  - il metodo *turn*: ritorna il giocatore che deve fare la mossa corrente (1 o 2) e serve all'interfaccia per sapere che simbolo disegnare (croce o pallino).

  - il metodo *reset*: riazzera la situazione della partita e serve per fare più partite con lo stesso oggetto.

  Il file *tris.html* in questo caso non contiene quasi nulla oltre all'inclusione dei file Javascript e al metodo *onload* per chiamare la funzione che creerà l'interfaccia: l'unico elemento contenuto è un tag *div* che servirà come "segnaposto" per le funzioni di interfaccia che inseriranno il codice HTML vero e proprio.
  All'interno del file *interfacciaTris.js* viene creato l'oggetto di tipo Tris che gestirà la logica del gioco e sono contenute le funzioni per la gestione dell'interfaccia: *inizializzaGioco* che predispone la tabella di gioco e azzera lo stato dell'oggetto Tris, *makeTable* che costruisce il codice HTML della tabella e per ogni casella inserisce il gestore degli eventi e *clickCasella* che viene chiamata ogni volta che viene fatto un click su una casella e gestisce la visualizzazione dello stato del gioco appoggiandosi sulla classe Tris. Si è scelto di utilizzare delle immagini per rendere il gioco più accattivante da un punto di vista grafico, ma si sarebbero potute utilizzare delle lettere.

- FILE: 004/tris.html, 004/tris.js, 004/interfacciaTris.js, 004/tris.css

4.3 Calcolatrice

- PROBLEMA: creare un programma che implementi una calcolatrice tradizionale con in tasti numerici, le quattro operazioni, il cambio di segno, la radice quadrata e l'elevamento a potenza.

- SOLUZIONE: la classe principale verrà chiamata Calcolatrice e come proprio stato conterrà una variabile *ultimoTasto* contenente il valore dell'ultimo tasto premuto, *primoPunto* che indica se è già presente un punto decimale oppure no, *ultimoOperatore* che contiene il tipo dell'ultimo operatore inserito (binario, unario o uguale), *operatore* che contiene l'ultimo operatore inserito, *op1* e *op2* che contengono gli operandi su cui verrà eseguita l'operazione.
  I metodi implementati saranno:

  - il costruttore: chiama il metodo *clickReset*.
  - il metodo *clickReset*: setta gli attributi ai valori iniziali corretti e mostra 0 nel display.
  - il metodo *clickCifra*: gestisce l'inserimento di una cifra o del punto decimale. Controlla solamente per evitare di inserire più punti decimali nello stesso numero.
  - il metodo *clickOperatoreUnario*: gestisce l'inserimento di un operatore unario (il cambio di segno o la radice quadrata), calcolando il risultato (tramite il metodo *eseguiOperazioneUnaria*) e mostrandolo sul display.

  - il metodo *clickOperatoreBinario*: gestisce l'inserimento di un operatore binario (le quattro operazioni e l'elevamento a potenza) verificando se c'è un'operazione in corso, nel qual caso calcola e mostra il risultato, oppure se è stato premuto dopo l'immissione del primo operando, nel qual caso non mostra niente a video ma setta gli attributi opportunamente.
  - il metodo *clickOperatoreUguale*: gestisce l'inserimento del segno uguale, in modo che se dopo aver calcolato il risultato di un'operazione venisse premuto nuovamente il segno uguale, l'operazione verrebbe ripetuta.
  - i metodi *isUltimoTasto**: sono una famiglia di metodi di utilità che controllano cos'è l'ultimo tasto premuto (una cifra, un operatore, l'uguale).
  - i metodi **Display*: servono a leggere/scrivere i valori nel display della calcolatrice.
  - i metodi *eseguiOperazione**: eseguono le operazioni vere e proprie.

  Il file *calcolatrice.html* contiene, oltre alla creazione dell'oggetto Calcolatrice nell'evento *onload*, l'interfaccia della calcolatrice realizzata tramite una tabella e l'associazione della pressione dei tasti al metodo corretto da chiamare.

- FILE: 004/calcolatrice.html, 004/calcolatrice.js, 004/calcolatrice.css

4.4 Caricare file XML con Javascript

- PROBLEMA: creare un programma che carichi all'interno di una pagina HTML il contenuto di un file XML.

- SOLUZIONE: in questo esercizio il programma Javascript è stato scritto internamente al file HTML e le funzioni sono solamente due. *LoadXML* si preoccupa di caricare il file XML, l'unica cosa da notare è che siccome Internet Explorer ha un comportamento diverso per quanto riguarda il parser XML, deve essere fatto un controllo per effettuare la chiamata corretta all'oggetto che istanzia il parser

```
//Codice per caricare il file con IE
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async = false;

//Codice per caricare il file con Firefox, Opera, etc.
xmlDoc = document.implementation.createDocument("","",null);
```

  L'altra funzione è la *createTable* che si occupa di leggere i dati dal file XML e produrre il codice HTML che li inserisce in una tabella (ovviamente è possibile scegliere qualsiasi tipo di formattazione per mostrarli in output).

Utilizzando l'oggetto ottenuto con loadXML si recuperano i nodi di tipo persona e con un ciclo *for* si scorrono tutti: per ognuno si cercano poi i valori degli attributi da mostrare a video. Quest'ultimo passo è stato fatto utilizzando due modalità diverse (anche se ce ne potrebbero essere altre).

```
//nome estratto con il primo modo
text+="<tr><td>" + element[i].getElementsByTagName("nome").
    item(0).firstChild.nodeValue + "<\/td>";
//cognome estratto con il secondo modo
text+="<td>" + element[i].childNodes[3].firstChild.nodeValue +
     "<\/td>";
```

Nel primo modo si richiedono gli elementi con il tag *nome* (in questo caso è presente un elemento soltanto), viene restituita una *collection* di cui noi prendiamo l'*item* di posto 0 e stampiamo il valore del nodo *firstChild* (o *lastChild* poiché in questo caso coincidono). Nel secondo modo andiamo al nodo al quarto posto di persona (non al secondo posto come sembrerebbe più logico essendo il cognome il secondo attributo di persona, poiché anche gli "a capo" vengono visti come nodi) e da lì stampiamo il valore del nodo *firstChild*.

- FILE: 004/rubrica.html, 004/rubrica.xml

Capitolo 5

Fondamenti di PHP

In questo capitolo verranno proposti degli esercizi per familiarizzare con il linguaggio di scripting lato server PHP. Ogni esempio presenterà come interagire con gli utenti attraverso le form utilizzando i vari campi di input. Questi esercizi separano la parte di inserimento dei dati da quella che li elabora all'interno di due file distinti, uno HTML e uno PHP, distinzione valida da un punto di vista didattico che verrà però successivamente superata per favorire un approccio più realistico. In tutti gli esercizi verrà trascurata la parte grafica, che risulterà molto scarna, per concentrarsi solo sulla parte di elaborazione e interazione client-server.

Verrà inoltre per il momento trascurata la parte relativa alla sicurezza, che nelle applicazioni reali riveste invece un ruolo di primo piano, in quanto queste applicazioni sono pubbliche e possono essere "attaccate" sia da utenti che commettono errori in buona fede sia da soggetti che lo fanno con intenzioni dolose.

5.1 Casella di testo

- PROBLEMA: creare un programma che faccia inserire all'utente il proprio nome e cognome e costruisca una pagina in risposta con i dati inviati.
- SOLUZIONE: in questo esercizio, come i tutti i successivi, viene utilizzato un campo di input (in questo caso una casella di testo) per spedire dei dati al server, il quale li elabora e fornisce una pagina in risposta. Per spedire dei dati si utilizza il tag HTML *form*, che può contenere input di vario tipo.

```
<form action="saluti.php" method="post">
```

  Due sono gli attributi essenziali: l'attributo *action* che indica a quale pagina sul server deve essere indirizzata la richiesta e l'attributo *method*, che indica quale tra i metodi *POST* e *GET* del protocollo *http* deve essere

utilizzato per spedire i dati al server. In questo esercizio la pagina che elaborerà i dati sarà *saluti.php* e il metodo di spedizione sarà il metodo *POST*. All'interno del form sono state inserite due caselle di testo, una per il nome e una per il cognome.

```
<input name="nome" type="text">
```

Anche in questo caso bisogna valorizzare almeno due attributi: l'attributo *name* che dà un nome alla casella di testo e che servirà al codice della pagina *saluti.php* per riferirsi ai dati inviati e l'attributo *type*, in questo caso con valore *text*, che indica al browser che dovrà creare una casella di testo. Sempre all'interno del form deve essere inserito un pulsante di tipo *submit*, che quando premuto farà si che il browser invii la richiesta e visualizzi la risposta.

```
<input type="submit" value="Invia i dati">
```

L'attributo *value* contiene la stringa che il browser inserirà nel bottone, al posto di quella di default (che ad esempio per Firefox localizzato in italiano è "Invia richiesta").
La pagina *saluti.php* contiene una parte statica formata da codice HTML e una dinamica con il codice PHP.

```
<?php
    echo "<p>Ciao " . $_POST["nome"] . " " . $_POST["cognome"]
        . "</p>";
?>
```

Come si può notare lo script PHP viene rinchiuso all'interno dei tag *<?php* e *?>* per indicare all'interprete PHP che quella è la parte da elaborare (il resto viene semplicemente copiato verso l'output). Il codice in questione utilizza il costrutto *echo* per mandare in output una stringa, in questo caso formata dal nome e dal cognome preceduti da Ciao. Se ad esempio l'utente avesse inserito nella pagina HTML i valori Mario e Rossi l'output sarebbe "Ciao Mario Rossi". Per recuperare i valori spediti dal client in PHP è sufficiente far riferimento al super array[1] globale $_POST, inserendo come indice il nome del campo di cui si desidera recuperare il valore. Nel caso i dati fossero stati spediti con il metodo GET il super array da utilizzare sarebbe stato $_GET. Per concatenare le stringhe in PHP viene utilizzato l'operatore punto (.).

- FILE: 005/saluti.html, 005/saluti.php

[1] Da notare che in PHP gli array si comportano sia da array classici, quindi con un indice numerico per individuare un elemento, sia, laddove possibile, come mappe, in cui ad un'etichetta di tipo stringa è associato un valore, come nel caso visto.

5.2 Menù a discesa

- PROBLEMA: creare un programma che implementi una semplice calcolatrice con le quattro operazioni.
- SOLUZIONE: per realizzare l'interfaccia si è scelto di utilizzare due caselle di testo per l'input degli operandi e un menù a discesa per permettere all'utente di poter scegliere l'operazione. Il menù a discesa in HTML viene reso in questo modo:

```
<select name="operazione">
   <option value="+">+</option>
   <option value="-">-</option>
   <option value="*">*</option>
   <option value="/">/</option>
</select>
```

  L'attributo *name* del tag *select* individua il nome che verrà utilizzato dal codice lato server, come già visto in precedenza per le caselle di testo. Per ognuna delle opzioni del menù viene utilizzato il tag *option*: l'attributo *value* indica il valore che verrà inviato al server nel caso sia selezionata quella voce, la stringa contenuta nel tag quello che verrà visualizzato a video. Anche se in questo esempio le due cose coincidono, in generale possono assumere valori diversi.
  La pagina *operazioni.php* che riceverà i valori non farà altro che effettuare uno *switch* sul tipo di operazione e dopo averla svolta mostrerà il risultato nella pagina di risposta.
- FILE: 005/operazioni.html, 005/operazioni.php

5.3 Check box semplice

- PROBLEMA: creare un programma che richieda all'utente di indicare se conosce o meno il linguaggio di programmazione PHP.
- SOLUZIONE: essendo la richiesta di tipo vero o falso si può utilizzare un checkbox :

```
<input name="scelta" type="checkbox">
```

  In questo caso l'attributo *type* è valorizzato a *checkbox* e il browser lo renderà con il tipico quadratino che può essere spuntato (l'esatta resa dipenderà dal browser e dal sistema operativo utilizzato).
  La pagina *checkbox.php* utilizzerà un meccanismo leggermente diverso per controllare il valore inviato: tramite la funzione *isSet*[2] si verificherà se

[2] In PHP i nomi delle funzioni sono case-insensitive, quindi isset, isSet e ISSET sono la stessa funzione. I nomi delle variabili invece sono case-sensitive, quindi $i e $I sono variabili diverse.

una variabile esiste oppure no[3] e nel caso di un checkbox la variabile verrà valorizzata qualora il campo sia stato spuntato, altrimenti no.

- FILE: 005/checkbox.html, 005/checkbox.php

5.4 Check box multipli

- PROBLEMA: creare un programma simile al precedente che richieda all'utente di indicare se conosce o meno alcuni linguaggi di programmazione.
- SOLUZIONE: come per l'esempio precedente vengono utilizzati dei checkbox. In questo caso viene utilizzato un array di checkbox con nome *scelta* e per farlo si utilizza il seguente codice:

```
PHP<input name="scelta[]" type="checkbox"><br>
C++<input name="scelta[]" type="checkbox"><br>
Java<input name="scelta[]" type="checkbox"><br>
```

La pagina *checkboxes.php* guarderà se i vari elementi dell'array *scelta* sono stati valorizzati con un meccanismo del tutto simile all'esempio precedente

```
if (isset($_POST["scelta"][0]))
   echo "<p>PHP - Si</p>\n";
else
   echo "<p>PHP - No</p>\n";
```

- FILE: 005/checkboxes.html, 005/checkboxes.php

5.5 Textarea

- PROBLEMA: creare un programma che permetta all'utente di inserire una serie di informazioni libere di una certa lunghezza riguardo al suo curriculum come programmatore.
- SOLUZIONE: il problema potrebbe essere risolto utilizzando una serie di caselle di testo, ma da un punto di vista dell'usabilità non sono appropriate poiché dovendo inserire molto testo la parte che non può essere contenuta scorre e risulta non più visibile. Un controllo più adatto è invece il *textarea*, che non è altro che una casella di testo che può contenere testo su più linee.

```
<textarea name="competenze" cols="50" rows="5">
Linguaggi conosciuti:

Strumenti di sviluppo utilizzati:
```

[3] Essendo il PHP un linguaggio interpretato non necessita di una dichiarazione di variabile prima dell'utilizzo, quindi una variabile è settata quando viene valorizzata per la prima volta, altrimenti non esiste.

```
Conoscenza database:

</textarea>
```

Due attributi utili sono *cols* e *rows*, per indicare rispettivamente il numero di colonne e il numero di righe che conterrà il *textarea*. Può essere utile anche la possibilità che offre di inserire del testo precompilato al suo interno, per fornire delle guide alla compilazione: a differenza del normale codice HTML nel quale spazi aggiuntivi e ritorni a capo vengono ignorati, il testo contenuto all'interno di un *textarea* viene reso dal browser così com'è, allo stesso modo del tag HTML *pre* (preformatted). La pagina *textarea.php* non farà altro che mostrare a video il testo inserito, che viene inserito tra i tag *pre* in modo da avere la stessa formattazione che ha utilizzato l'utente.

- FILE: 005/textarea.html, 005/textarea.php

5.6 Campi nascosti

- PROBLEMA: creare un programma che permetta all'utente di inserire il suo nome, poi in una pagina successiva il suo linguaggio di programmazione preferito e infine in una terza pagina faccia un riassunto dei dati inseriti nelle due precedenti.

- SOLUZIONE: il problema proposto è banale e di scarsa utilità, ma appartiene ad una famiglia di problemi che si incontrano molto spesso: navigare attraverso varie pagine mantenendo memoria dello stato della navigazione. Siccome *HTTP* è un protocollo stateless (senza memoria) non possiamo appoggiarci su di esso per raggiungere tale obbiettivo, ma dobbiamo usare mezzi differenti. Esistono diversi modi di raggiungere questo risultato e i campi nascosti sono uno di questi. La prima pagina (*hidden.html*) non fa altro che richiedere il nome dell'utente con una casella di testo e passarlo alla pagina *hidden.php*. L'unica cosa diversa rispetto agli esercizi precedenti è che viene usato il metodo *GET* e guardando la barra degli indirizzi del browser si può notare come il dato passi attraverso l'URL.

```
http://localhost/ldp/005/hidden.php?nome=Alessandro
```

A questo punto per passare il valore alla pagina successiva esso viene "nascosto" alla vista dell'utente e inserito all'interno del codice HTML generato dalla pagina *hidden.php* tramite la seguente istruzione:

```
<input type="hidden" name="nome"
value="<?php echo $_GET["nome"];?>">
```

Ovviamente lo scopo del campo nascosto non è quello di mantenere segreta un'informazione (basta leggere il codice HTML per vederne il valore), ma solamente di passare dei valori non inseriti dall'utente ma ad esempio passati da pagine precedenti. Quindi dal punto di vista della pagina

hidden2.php non ci sarà nessuna differenza tra i dati inseriti nella pagina precedente dall'utente e quelli provenienti dal campo nascosto.

- FILE: 005/hidden.html, 005/hidden.php, 005/hidden2.php

5.7 Radio button

- PROBLEMA: creare un programma che faccia scegliere all'utente una sola risposta possibile tra quattro differenti possibilità.

- SOLUZIONE: il *radio button* è un controllo pensato per risolvere il problema di un'unica scelta possibile tra diverse alternative. Ogni casella di input (solitamente rappresentata con un pallino) nel codice HTML ha lo stesso *name* delle altre in modo da essere mutualmente esclusive: da ciò deriva che è possibile mettere diversi gruppi di *radio button* nella stessa pagina semplicemente utilizzando nomi diversi. Inoltre è una buona idea quella di inserire sempre la possibilità "Non lo so" o "Nessuna risposta" o qualcosa di simile, in quanto non essendo possibile deselezionare un gruppo di radio button deve essere possibile per l'utente poter scegliere di non rispondere. Per far si che ci sia un'opzione selezionata quando viene caricata la pagina basta inserire l'attributo *checked* nel radio che si intende preselezionare.

```
<input name="risposta" type="radio" value="Javascript">
    Javascript<br>
<input name="risposta" type="radio" value="PHP">PHP<br>
<input name="risposta" type="radio" value="VBscript"> VBscript
    <br>
<input name="risposta" type="radio" value="Non lo so" checked=
    "checked"> Non lo so<br>
```

La pagina *radio.php* riceverà il valore selezionato allo stesso modo delle caselle di testo e poi potrà trattarlo come desidera.

- FILE: 005/radio.html, 005/radio.php

5.8 Password

- PROBLEMA: creare un programma che permetta all'utente di inserire una password.

- SOLUZIONE: il campo password è esattamente la stessa cosa di una casella di testo con l'unica differenza di mostrare dei pallini (o asterischi o altro) al posto delle lettere digitate dall'utente. Da un punto di vista della sicurezza questo impedisce che un curioso alle proprie spalle veda la password che stiamo scrivendo, ma questa è l'unica cosa che fa. Il dato inserito passerà comunque in chiaro sulla rete, a disposizione di chiunque abbia accesso ai pacchetti. Nell'esempio è stato volutamente utilizzato il metodo GET

per mostrare che la password viene visualizzata in chiaro nella barra degli indirizzi.

```
http://localhost/ldp/005/password.php?pw=pippo
```

Il metodo POST elimina questo inconveniente ma ancora fa transitare la password in chiaro sulla rete: per ottenere la sicurezza che la password non possa essere letta bisogna utilizzare un canale cifrato.

- FILE: 005/password.html, 005/password.php

5.9 Cookies

- PROBLEMA: creare un programma che permetta di mantenere memoria di alcune scelte effettuate dall'utente.
- SOLUZIONE: i *cookies* sono un meccanismo per memorizzare delle informazioni sul client che saranno rispedite al server ad ogni interazione successiva, in modo da permettere al server di utilizzare queste informazioni per compiti che richiedano memoria di quanto fatto. Nell'esercizio si crea una pagina dove all'utente vengono chieste informazioni sul font che preferisce, in modo ad esempio da personalizzare la sua esperienza di navigazione ad ogni visita successiva allo stesso sito.
  La pagina *cookies.php* non fa altro che mostrare i campi di scelta, utilizzando degli array PHP e il costrutto *foreach* per popolarli.

```
$font = array("arial", "helvetica", "sans-serif", "courier");
...
foreach ($font as $var)
   echo "<option>$var</option>\n";
```

Come si vede il *foreach* non è altro che una versione più comoda del *for* per navigare attraverso un array: all'interno delle parentesi la prima variabile è l'array da esplorare e dopo la parola chiave *as* viene messo un nome di variabile che ad ogni giro del ciclo verrà valorizzato con l'elemento corrente dell'array.
Invece *cookies1.php* è la pagina che si occupa eventualmente di settare i cookie o se sono già settati di impostare i font di conseguenza. Come prima cosa si controlla tramite la funzione *empty* se sono stati spediti dei dati dalla pagina precedente: se è così questi vengono settati come cookie e verranno scritti sul computer client (eventualmente sovrascritti se già presenti), altrimenti non vengono spediti.

```
$settings = false;
if (!empty($_POST["name"]))
{
   $name_sel = $_POST["name"];
   setcookie("name", $name_sel, time()+3600);
   $font_sel = $_POST["font"];
```

```
    setcookie("font", $font_sel, time()+3600);
    $size_sel = $_POST["size"];
    setcookie("size", $size_sel, time()+3600);
    $settings = true;
}
```

L'istruzione *setcookie* deve essere eseguita prima di qualsiasi output, poiché essendo i cookie spediti nella sezione header HTTP, se fosse stata già spedita una parte della sezione data (il codice HTML), non sarebbe più possibile per il server spedire i cookie. Questa funzione ha tre parametri: il primo è il nome con cui il cookie verrà memorizzato sul client, il secondo è il valore che conterrà e il terzo il tempo di vita del cookie in Unix time (secondi a partire dalla mezzanotte del primo gennaio 1970). Se quest'ultimo parametro non è presente il cookie verrà eliminato alla chiusura del browser.
Successivamente la pagina ha tre percorsi di esecuzione:

- se i cookie sono appena stati settati avvisa l'utente che dalla prossima volta le preferenze verranno riconosciute e realizzate

```
if ($settings)
{
    echo "Dalla prossima visita i tuoi cookie permetteranno
         al sito di conoscere le tue preferenze";
    echo "</body></html>";
    exit(0);
}
```

- se i cookie sono scaduti o non sono stati settati (cioè la pagina cookies1.php è stata chiamata direttamente senza passare da cookies.php) avvisa l'utente di questo fatto

```
else if (!isset($_COOKIE["name"]))
    echo "<p>I cookie sono scaduti o non sono stati settati
        </p>";
```

- se la pagina è chiamata dopo che i cookie sono stati settati informa l'utente e modifica il codice HTML in modo da adeguarsi alle preferenze dell'utente

```
else
{
  echo "<p>Ciao " . $_COOKIE["name"] . ", i tuoi cookies
      sono i seguenti:</p>\n";
    echo "<font face=\"" . $_COOKIE["font"];
    echo "\" size=\"" . $_COOKIE["size"] . "\">";
    echo "Tipo di font = " . $_COOKIE["font"];
    echo "<br>\n";
    echo "Dimensione del font =" . $_COOKIE["size"];
    echo "<br></font>";
}
```

- FILE: 005/cookies.php, 005/cookies1.php

5.10 Strutture di controllo

- PROBLEMA: creare un programma che pianifichi le tappe di un viaggio in macchina dati in ingresso la capacità del serbatoio in litri, il consumo in km/l e la lunghezza del viaggio, supponendo che le stazioni di rifornimento siano disponibili nel punto esatto in cui la macchina finisce il carburante.
- SOLUZIONE: come prima cosa è necessario calcolare quanti chilometri è in grado di fare la macchina con un pieno e successivamente calcolare il numero di soste. Con questo valore è possibile stampare la tabella di marcia tramite un ciclo *for* nel seguente modo:

```
for ($i=1;$i<=$soste;$i++)
{
   $d = $kmPieno*$i;
   echo "$i ° sosta al Km $d <br>\n";
}
```

  dove *$soste* sono il numero di soste e *$kmPieno* sono quanti kilometri può fare la macchina con un pieno.

- FILE: 005/rifornimento.html, 005/rifornimento.php

5.11 Unire input e risposta

- PROBLEMA: rifare il programma dell'esercizio 5.2 sulla calcolatrice in modo che sia contenuto in un unico file PHP che si occupi sia di ricevere l'input che di mostrare l'output.
- SOLUZIONE: perché si vuole raggiungere questo obbiettivo? Perché da un punto di vista della manutenzione avere due file separati che compongono parti dello stesso programma non è una cosa bella, mentre è molto meglio dover gestire un file solo. L'idea in questo esercizio è di controllare se arrivano dei dati dalla pagina stessa: in caso affermativo si svolge l'operazione e si mostra nuovamente la parte per immettere l'input, altrimenti (la pagina è stata chiamata per la prima volta) si mostra solo l'input. Per fare questo il primo controllo viene fatto in questo modo:

```
if (isset($_POST["primo"]))
{
   $num1=$_POST["primo"];
   $num2=$_POST["secondo"];
   switch($_POST["operazione"])
   {
      case "+" : $x = $num1 + $num2;
        break;
      case "-" : $x = $num1 - $num2;
        break;
      ...
   }
}
```

con *isset* si controlla se sono arrivati dei dati ed eventualmente si esegue l'operazione. Nella parte successiva di codice se è stato calcolato si mostra il valore di *$x*, in ogni caso si mostrano le caselle di input, inserendo se necessario nella prima casella il valore calcolato precedentemente. Quando si utilizza un solo file, sempre per motivi di migliore manutenzione, nell'attributo *action* non si inserisce il nome del file ma la variabile PHP *$_SERVER["PHP_SELF"]*: ci penserà l'interprete a sostituirla con il nome del file. Questo permette di non preoccuparsi di eventuali cambiamenti futuri del nome del file.

```
<form action="<?php echo $_SERVER["PHP_SELF"];?>"
    method="post">
```

- FILE: 005/operazioniMigliorato.php

Capitolo 6

PHP e MySQL

In questo capitolo si vedrà come utilizzare PHP per accedere ad un database server che nei nostri esercizi sarà MySQL. Ogni esercizio affronterà le tipiche interazioni con un database: recupero dei dati, creazione, modifica e cancellazione di record.

6.1 Connessione al database e recupero di informazioni

- PROBLEMA: creare un programma che interroghi un database per recuperare e mostrare al client una lista di studenti.
- SOLUZIONE: come prima cosa bisogna inserire i dati all'interno del database MySQL: questi dati si trovano nel file *prova.sql*. Questo può essere fatto sia attraverso il client testuale che attraverso phpMyAdmin: con quest'ultimo strumento, dopo aver creato un database di nome *prova*, lo si seleziona e si clicca sul tab *Importa*, si seleziona il file e si esegue. Se tutto è andato a buon fine nel database *prova* dovrebbero essere presenti le tabelle *studenti* e *voti*. La prima cosa da fare nel codice del programma sarà di aprire una connessione verso il database tramite le seguenti istruzioni

```
//stabilisce la connessione
mysql_connect("localhost","guest","guest");
//seleziona il database
mysql_select_db("prova");
```

mysql_connect riceve tre parametri: l'indirizzo del server su cui è in esecuzione MySQL (che può essere sia un nome di dominio che un indirizzo IP), il nome utente con cui connettersi al database e la password. *mysql_select_db* si limita a selezionare il database sul quale opereranno tutti i comandi dati successivamente. Perché la connessione funzioni è necessario creare preventivamente l'utente *guest*: per farlo sempre attraverso

phpMyAdmin si va alla schermata di partenza e si sceglie la voce *Privilegi*, successivamente si seleziona *Aggiungi nuovo utente*. Nella schermata che compare si inseriscono il nome utente e la password (guest, guest) e come host si inserisce localhost (in tal modo l'utente guest ha accesso al database solamente in locale), non si segna nessun privilegio globale e si esegue. A questo punto l'utente guest esiste ma non ha nessun privilegio, allora si seleziona la modifica dell'utente e si sceglie nella form *Privilegi specifici del database* il database *prova*, al quale poi si assegnano tutti i permessi.
Se la connessione va a buon fine il programma, nel caso debba effettuare una query, seguirà sempre più o meno questi passaggi

```
1  $query="SELECT * FROM Studenti ORDER BY Cognome";
2  $result = mysql_query ($query);
3  echo "<ul>";
4  while($row = mysql_fetch_array($result))
5  {
6      echo ("<li>" . $row["cognome"] . "- <i>" . $row["nome"] . "
           </i> -");
7      echo ("Data di nascita: " .$row["nascita"]."</li>\n");
8  }
9  echo("</ul>");
10 mysql_close();
```

Alla riga 1 viene creata la query, che in questo caso è statica ma che vedremo sarà più spesso dinamica. La riga 2 fa si che il testo della query venga fisicamente spedito al database e, nel caso sia tutto corretto, il risultato viene memorizzato all'interno della variabile *$result*. A questo punto bisogna mostrare l'output all'utente attraverso codice HTML e in questo caso è stato scelto un elenco puntato ma poteva andare bene qualsiasi altra cosa adatta (del testo con a capo, una tabella, ...). Per scorrere i risultati (è da tenere presente che il risultato di una query è sempre una tabella, anche se fosse di una sola riga e di una sola colonna) si usa tipicamente il costrutto di riga 4: il *while* scorre finché vengono visitate tutte le righe e per ogni "giro" il contenuto della riga corrente viene memorizzato nell'array *$row*. Questo array ha tanti elementi quanti sono le colonne che compongono il risultato e ogni cella può venire indirizzata utilizzando sia un indice numerico che il nome della colonna (è preferibile quest'ultima possibilità in quanto l'ordine delle colonne potrebbe variare a seconda di come viene formulata la query). Il corpo del *while* contiene le istruzioni di output che creano il codice HTML che renderà l'elenco nel modo desiderato. Infine alla riga 10 si chiude la connessione con il database.

- FILE: 006/listaStudenti.php, 006/prova.sql

6.2 File di configurazione per la connessione

- PROBLEMA: migliorare il programma dell'esercizio 6.1 introducendo l'utilizzo di un file di configurazione della connessione al database.

- SOLUZIONE: l'esercizio 6.1 può essere sostanzialmente migliorato modificando la modalità con cui si gestisce la connessione, recuperando i parametri da un file esterno piuttosto che inserendoli direttamente nel codice. Il motivo di questo accorgimento appare evidente quando si ha a che fare con siti reali con una dimensione dell'ordine delle centinaia di pagine PHP. Ci sono molti motivi per cui può capitare di dover cambiare qualcosa nei parametri, ad esempio la modifica dell'indirizzo IP del database server o la modifica della password per questioni di sicurezza. Nello scenario precedente sarebbe necessario dover modificare il codice di ogni singolo file per allinearli alla modifica, cosa che rappresenta chiaramente un incubo nell'amministrazione di un sito. La soluzione è appunto quella di scrivere i parametri in un file esterno e poi di includere questo file utilizzando l'istruzione *include*. Questo file, solitamente con estensione .inc.php[1], contiene i parametri di connessione con una modalità che ricorda le direttive #DEFINE del linguaggio C

```
<?php
define("NOME","guest");
define("PASSWORD","guest");
define("SERVER","localhost");
define("DATABASE","prova");
?>
```

Ad ogni nome simbolico corrisponde un valore (ad esempio a NOME corrisponde guest): il programma che include questo file farà poi riferimento ad ogni parametro utilizzando il nome simbolico, che durante l'interpretazione verrà sostituito con il suo valore.

```
//includo il file contenente i parametri di connessione
include "connessione.inc.php";
//stabilisce la connessione
mysql_connect(SERVER,NOME,PASSWORD);
//seleziona il database
mysql_select_db(DATABASE);
```

In questo modo un cambiamento di parametro dovrà essere fatto solo all'interno del file di configurazione e la modifica si rifletterà automaticamente in tutti i file del sito.

[1] La spiegazione di questa prassi risiede nel fatto che il suffisso .inc serve ai programmatori per indicare che si tratta di un file di configurazione, però utilizzato da solo potrebbe dare dei problemi di sicurezza, perché se il server HTTP non fosse configurato correttamente la richiesta di quel file produrrebbe come risposta il file in chiaro, quindi con i parametri di connessione a disposizione di tutti. Se invece si aggiunge anche l'estensione php, il server richiede all'interprete PHP di elaborare il file e la risposta è un file senza nessun contenuto, poiché l'istruzione *define* non produce nessun output.

- FILE: 006/listaStudentiMigliorato.php, 006/connessione.inc.php

6.3 Query parametriche

- PROBLEMA: creare un programma che consenta di recuperare un elenco di studenti dal database *prova* e di ordinare secondo diversi criteri (nome, cognome, data di nascita).
- SOLUZIONE: a differenza degli esercizi precedenti in questo caso l'utente inserirà un criterio di ordinamento e per far questo possiamo usare un menù a discesa con le tre alternative. Quando la pagina riceve questo dato compone la query aggiungendo alla clausola ORDER BY il campo su cui ordinare

```
//crea una query utilizzando i parametri passati con il metodo
    POST
$query="SELECT * FROM Studenti ORDER BY ";
if (isset($_POST["ordina"]))
   $query .= $_POST["ordina"];
else
   $query .= "nome";
```

Come si vede la prima parte della query è statica, cambia solo il parametro finale, che se è stato scelto viene inserito nella query, altrimenti viene messo il parametro di default. La stessa cosa può essere fatta utilizzando il metodo GET al posto del POST. In questo caso non cambia nulla per quanto riguarda la parte della costruzione della query, il cambiamento si ha nella creazione del sistema di scelta. L'utente vedrà tre link alle possibili scelte, ognuno dei quali avrà inserito come parametro il valore per l'ordinamento.

```
<p>Scegli il criterio di ordinamento</p>
<ul>
   <li><a href="<?php echo $_SERVER["PHP_SELF"]?>?ordina=nome"
       >
      Ordina per nome</a></li>
   <li><a href="<?php echo $_SERVER["PHP_SELF"]?>?ordina=
       cognome">
      Ordina per cognome</a></li>
   <li><a href="<?php echo $_SERVER["PHP_SELF"]?>?ordina=
       nascita">
      Ordina per data di nascita</a></li>
</ul>
```

- FILE: 006/listaStudentiOrdinataPost.php, 006/listaStudentiOrdinataGet.php

6.4 Gestione degli errori

- PROBLEMA: riscrivere l'esercizio 6.1 in modo da gestire i più comuni errori relativi all'accesso a un database.

- SOLUZIONE: siccome la gestione degli errori sarà presente in tutti i programmi che accedono ad un database vale la pena di scrivere le funzioni in un file separato. Il primo errore da controllare è se la connessione è andata a buon fine, cosa che potrebbe non avvenire per svariati motivi (il server non è disponibile, l'utente o la password sono scorretti, la rete non funziona, ...)

```
$link_id = mysql_connect(SERVER, NOME, PASSWORD);
if(!$link_id)
{
   $MYSQL_ERRNO = 0;
   $MYSQL_ERROR = "Connessione fallita a " . SERVER;
   return 0;
}
```

  Il valore di ritorno è un numero positivo che identifica la connessione nel caso che tutto sia andato bene, altrimenti è 0 (che viene interpretato come falso). Se la connessione è andata a buon fine potrebbe generarsi un errore con la chiamata a *mysql_select_db*. In questo caso, a differenza di prima, il server risponderà con un messaggio di errore, che potrà essere usato per debug. PHP ha due funzioni per leggere l'ultimo messaggio di errore inviato dal server, *mysql_errno* e *mysql_error*: la prima ritorna il numero di errore e la seconda una stringa descrittiva dell'errore. Entrambi questi controlli possono essere inseriti nella funzione *db_connect* contenuta nel file *common.php*, che si occuperà quindi di questi errori.
  Un altro genere di errore è quello che si può avere dopo aver inviato una query: anche in questo caso i motivi possono essere molteplici, ma il più comune è un errore di sintassi nello statement SQL. Per controllare questo errore è sufficiente testare il valore di ritorno di *mysql_query*: nel caso sia falso la query non è andata a buon fine e di nuovo potremo stampare il messaggio di errore

```
$result = mysql_query ($query);
if (!$result)
{
   echo "<p>Query fallita: " . $query ."</p>";
   echo "<p>" . sql_error() . "</p>";
   html_footer();
   exit();
}
```

  Può anche essere utile stampare il testo della query perché spesso è facile risalire all'errore semplicemente leggendola.

- FILE: 006/listaStudentiGestioneErrori.php, 006/common.inc.php

6.5 Inserimento di un record

- PROBLEMA: creare un programma che permetta all'utente di inserire un nuovo record all'interno del database *prova*.

- SOLUZIONE: l'inserimento non è molto diverso da quanto visto in precedenza per una query che recupera i dati. In questo caso si utilizzerà l'istruzione SQL INSERT INTO inserendo i parametri passati dall'utente

```
$nome=$_POST["nome"];
$cognome=$_POST["cognome"];
$nascita=$_POST["nascita"];
$query="INSERT INTO studenti (nome,cognome,nascita) VALUES ".
    "('$nome','$cognome','$nascita')";
```

  Anche in questo caso sarà necessario controllare se la query sia stata eseguita correttamente e può essere utile anche mostrare all'utente i dati appena inseriti, in modo che possa verificare di non aver fatto errori (un'altra possibilità potrebbe essere quella di utilizzare Javascript per mostrare una finestra con il riepilogo dei dati e chiedere di confermare l'invio degli stessi).

- FILE: 006/inserisciStudente.php

6.6 Gestione dell'input

- PROBLEMA: riscrivere il programma dell'esercizio 6.5 in modo da gestire gli errori di input più frequenti.

- SOLUZIONE: prendiamo in considerazione solamente due tipi di errori, uno di tipo "sintattico" e uno di tipo "semantico". Questo non vuol dire che esistano solo queste tipologie, anzi, lo scopo di questo esercizio è solo quello di chiarire come l'input che giunge dall'utente dovrebbe essere sempre controllato, poiché le applicazioni PHP girano per definizione in un ambiente ostile (la rete), dove l'utente può sia commettere errori in buona fede sia invece inserire intenzionalmente dell'input in grado di sovvertire il comportamento del programma, in modo da avere accesso a informazioni riservate, guadagnare privilegi, ecc.
  Il primo esempio di problema è di tipo "sintattico" ed è dovuto al fatto che alcuni caratteri che inserisce l'utente possono dare problemi quando vengono copiati all'interno del codice. Se l'utente dovesse inserire nel campo nome la stringa "D'amico" che contiene un apostrofo al suo interno, il valore verrà concatenato nella query SQL, portando a un errore perché quell'apostrofo verrà interpretato come la fine del valore nome e quindi rimarrà una virgoletta che non chiude, come si può vedere.

```
INSERT INTO studenti (nome,cognome,nascita) VALUES ('D'amico',
    'Orizio','2004-03-12')
```

  In realtà utilizzando gli strumenti di questo corso (Xampp versione 1.6.2) non si verifica questo problema perché PHP è configurato in modo da avere

la variabile di configurazione *magic_quote_gpc*[2] a *on*. Questa configurazione istruisce l'interprete PHP a inserire automaticamente il carattere di escape (backslash) in modo che i caratteri speciali (', ", \ e NULL) siano interpretati correttamente se inseriti in una stringa di codice. Quindi la query SQL precedente risulta così:

```
INSERT INTO studenti (nome,cognome,nascita) VALUES ('D\'amico'
    ,'Orizio','2004-03-12')
```

Questo però non elimina del tutto i problemi, in quanto nel codice precedente, nel mostrare a video i dati inseriti, la stringa del nome conterrebbe il backslash. Per ovviare a questo si può usare la funzione *stripslashes*, che elimina i backslash e riporta il testo alla sua forma originaria. Tutto bene se sappiamo come è configurato PHP sul server, ma a volte la stessa applicazione deve girare su decine o centinaia di server diversi, quindi l'unica possibilità realistica è far si che il programma verifichi la configurazione del server in run time (con la funzione *get_magic_quote_gpc*) e agisca di conseguenza. Un codice di questo tipo va però oltre agli scopi del presente esercizio e non verrà trattato.
Il secondo tipo di errore si può vedere con l'esempio della data: una data per essere valida deve rispettare una certa sintassi e anche essere semanticamente corretta (il 2007-02-31 pur essendo corretto sintatticamente non è una data valida). Questo controllo lo si può fare in alcuni modi:

- impedendo all'utente di sbagliare, utilizzando al posto di una casella di testo una serie di tre menù a tendina. Questo risolve il problema della sintassi, ma non quello della semantica
- con Javascript facendo un controllo lato client prima di far spedire i dati
- con PHP lato server

Qui si vedrà un modo per implementare la terza soluzione.

```
if (ereg ("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})",
    $nascita, $regs))
{
    if (checkdate($regs[2],$regs[3],$regs[1]))
        $valida = true;
    else
        $valida=false;
}
```

Come si può vedere nella condizione dell'*if* viene controllata la validità sintattica della data utilizzando la funzione *ereg* che si basa sulle espressioni regolari. Se è corretta viene scomposta nell'array *regs* e tramite la funzione *checkdate* viene controllata la validità semantica.

[2] Il significato di gpc è GET, POST, COOKIES, poiché questo comportamento viene applicato a tutte le variabili che vengono spedite al server attraverso questi tre canali.

- FILE: 006/inserisciStudenteMigliorato.php

6.7 Modifica e eliminazione di record

- PROBLEMA: scrivere un programma che permetta di modificare e/o cancellare i record degli studenti contenuti nel database prova.
- SOLUZIONE: la prima cosa da decidere in questo caso è come impostare l'interfaccia utente e per mantenerla semplice ma funzionale si può prendere spunto da quella di phpMyAdmin e utilizzare una tabella con l'elenco degli studenti e per ognuno un link per la modifica e uno per la cancellazione. Per creare questa pagina sarà sufficiente scandire i record degli studenti e per ognuno creare il link passando come parametro l'ID, come si può notare nel seguente codice.

```
echo "<thead><tr><th>Cognome</th><th>Nome</th><th>Data di
    nascita</th><th> </th><th> </th></tr></thead>";
while ($row = mysql_fetch_array($result))
{
   echo "<tr>";
   echo "<td>" . $row["cognome"] . "</td>";
   echo "<td>" . $row["nome"] . "</td>";
   echo "<td>" . $row["nascita"] . "</td>";
   echo "<td><a href=\"".$_SERVER["PHP_SELF"]."?action=modify&
       ID=" .$row["ID"]. "\">Modifica</a></td>";
   echo "<td><a href=\"".$_SERVER["PHP_SELF"]."?action=delete&
       ID=" .$row["ID"]. "\">Elimina</a></td></tr>\n";
}
```

La pagina modificaEliminaStudenti.php avrà poi tre percorsi di esecuzione:

  - la scelta dello studente da modificare che porterà alla pagina di modifica con i dati dello studente precaricati
  - la conferma delle modifiche allo studente che verranno in tal modo inserite nel database
  - l'eliminazione dello studente, che in questo caso per semplicità non richiederà nessuna conferma da parte dell'utente

Per la modifica dopo aver recuperato i dati corrispondenti all'ID passato come parametro verrà creata una form con i campi precompilati.

```
echo "<input type=\"hidden\" name=\"ID\" value=\"" . $row["ID"
    ] . "\">\n";
echo "<input type=\"hidden\" name=\"action\" value=\"
    confirmModify\">\n";
echo "<table>";
echo "<tr><td>Nome</td>";
echo "<td><input type=\"text\" name=\"nome\" value=\"" .$row["
    nome"]. "\"></td></tr>\n";
...
```

Da notare come vengano utilizzati i campi nascosti per mantenere memoria dell'ID del record che dovrà essere passato alla pagina successiva e dell'azione da intraprendere (*confirmModify*). Il codice che riceverà questi dati si comporterà allo stesso modo di quello dell'inserimento visto nell'esercizio 6.5. Per quanto riguarda l'eliminazione, non richiedendo nessuna conferma, si tratterà solamente di eliminare il record con l'ID passato come parametro.

- FILE: 006/modificaEliminaStudenti.php

Capitolo 7

PHP esempi avanzati

In questo capitolo verranno mostrati alcuni esempi di utilizzo di PHP per risolvere problemi complessi. Verrà trattato l'utilizzo delle sessioni per gestire uno schema di autenticazione che svincoli il programmatore dai dettagli implementativi, l'utilizzo di Ajax e HttpXMLRequest per aggiornare solo parti di una pagina e, sempre utilizzando HttpXMLRequest, verrà creata un'applicazione in grado di recuperare dati XML da una fonte esterna per utilizzarli al proprio interno.

7.1 Autenticazione tramite sessioni

- PROBLEMA: scrivere un programma che permetta ad un utente di autenticarsi in modo da poter avere accesso ad altre pagine del sito e successivamente di poter effettuare il logout.
- SOLUZIONE: il problema dell'autenticazione solitamente è strettamente connesso a quello dell'autorizzazione, ma in questo esercizio vedremo solo come risolvere il primo. Esistono diversi modi per affrontarlo, alcuni dei quali attraverso librerie specializzate (ad esempio Auth di Pear) che semplificano al massimo la scrittura di codice: qui si vedrà come il problema possa essere gestito attraverso l'uso delle sessioni PHP. Il problema dell'autenticazione presenta due aspetti distinti: il primo, quello dell'autenticazione vera e propria, consiste nel verificare che una persona sia ciò che dice di essere solitamente attraverso il meccanismo nome utente/password, andando a vedere se si trova una coppia corrispondente nel database. Il secondo ha a che vedere con il fatto di poter riconoscere l'utente autenticato in tutte le successive pagine del sito che visiterà, senza dover richiedere ogni volta l'autenticazione. Per questo entrano in gioco le sessioni che sono un meccanismo integrato in PHP che, nascondendo i dettagli al programmatore, simula la presenza di una sessione di navigazione (si ricorda che HTTP come protocollo non è in grado di garantire questo). Per far ciò vengono create due funzioni, *auth* e *logout*, inserite

nel file *common.inc.php*. La prima funzione riceve tre parametri, l'id della connessione al database, il nome utente e la password. Come prima cosa viene chiamata la funzione PHP *session_start*, che permette all'interprete di attivare i meccanismi per la gestione della sessione e che dovrà essere chiamata in ogni pagina in cui è necessario verificare l'autenticazione. Successivamente si controlla se è già in corso una sessione[1], nel qual caso significa che l'utente si è già precedentemente autenticato e la funzione ritorna *true*. Se così non fosse si richiede al database se è presente una riga con quel nome utente e password: in caso affermativo si salvano nelle variabili di sessione i dati d'interesse che potranno poi essere recuperati nella successiva navigazione (nell'esempio si salva solo l'ID e il nome dello studente), negli altri casi si ritorna *false*.

```
//Si recupera il record dell'utente
$row = mysql_fetch_array($result);
//Si salvano nell'array authdata le informazioni d'interesse
$authdata = array("ID"=>$row["ID"], "login"=>$login);
//Si copia $authdata nelle variabili di sessione
$_SESSION['authdata']=$authdata;
```

La funzione *logout* non fa altro che verificare se è già presente una sessione e in caso affermativo la distrugge con le chiamate alle funzioni *session_unset* e *session_destroy*.
A questo punto si può costruire una pagina con una form di inserimento della coppia nome utente/password, che verifica tramite *auth* se l'utente è autenticato: se l'autenticazione va a buon fine o è già autenticato non verrà più mostrata la form ma un messaggio di benvenuto e un link al logout e a un'altra pagina. Quest'ultima pagina serve solo a far vedere che una volta autenticato l'utente può essere riconosciuto in tutte le pagine successive.

- FILE: 007/autenticazione.php, 007/autenticazione2.php, 007/common.inc.php

7.2 Aggiornamento parziale di una pagina con Ajax

- PROBLEMA: scrivere un programma che chieda la registrazione di un utente facendogli inserire il comune di nascita scelto all'interno di tutti i comuni italiani.

- SOLUZIONE: questo problema apparentemente banale rappresenta l'esigenza di molte pagine web di avere una maggiore interattività e non dover soffrire del tempo dovuto al caricamento. La prima soluzione diretta che si potrebbe pensare di implementare è quella di recuperare tutti i comuni da

[1] La sessione viene identificata da una stringa casuale esadecimale di 32 caratteri generata dal server, la cui lunghezza garantisce l'univocità.

un database[2] e successivamente inserirli in un menù a tendina da mostrare all'utente tramite codice PHP nel seguente modo

```
$query="SELECT codiceIstat, comune FROM comuni ORDER BY comune
    ";
...
//inizio dell'elenco
echo "<select name=\"nascita\">";
while($row = mysql_fetch_array($result))
   echo "<option value=\"". $row["codiceIstat"] . "\">" . $row
       ["comune"] . "</option>\n";
```

Il problema è che il codice prodotto da questo script contiene un elenco di più di 8000 comuni e quindi il file HTML inviato al client pesa circa 350 kB, rallentando notevolmente la navigazione. La soluzione si ottiene utilizzando Ajax (Asynchronous JavaScript and XML) e la funzione HttpXmlRequest, che è una metodologia di programmazione web che consente di scambiare dati con il server senza la necessità di ricaricare tutta la pagina, ma modificando solamente la parte che interessa. Questo esempio viene quindi riprogettato inserendo due menù a discesa: il primo con un elenco di provincie e il secondo inizialmente vuoto, ma che viene popolato dopo che è stata scelta una provincia solo con i comuni che gli appartengono, permettendo alla dimensione della pagina di scendere a qualche kB. Come prima cosa vengono create delle funzioni di utilità in Javascript che gestiranno lo scambio dei dati. La funzione CreaOggetto si occupa di istanziare un oggetto HttpXmlRequest in maniera indipendente dal browser.

```
var richiesta;
var browser = navigator.appName;
if (browser == "Microsoft Internet Explorer")
{
   richiesta = new ActiveXObject("Microsoft.XMLHTTP");
}
else
{
   richiesta = new XMLHttpRequest();
}
return richiesta;
```

La funzione *selector* è quella che viene invocata quando il menù a discesa delle provincie modifica il suo valore e apre la comunicazione con una pagina del server (*processa.php*), recuperando i dati relativi solo alla provincia selezionata e, attraverso la funzione *gestisciContenuto*, inserendoli nel menù a discesa dei comuni.

```
http.open('get','processa.php?provincia=' + document.
    iscrizione.provincia.options[document.iscrizione.provincia
    .selectedIndex].value);
```

[2] L'elenco dei comuni si trova nel file *comuni.sql* e può essere caricato nel database prova come già visto per le tabelle degli studenti.

```
http.onreadystatechange = gestisciContenuto;
http.send(null);
```

Come si vede la funzione *gestisciContenuto* è legata all'handler dell'evento *onreadystatechange*, che si genera quando la comunicazione tra il client e il server cambia stato. Questa funzione non fa altro che controllare se lo stato è 4 (completato) e dopo aver letto la risposta del server la inserisce all'interno del menù denominato *comune*

```
if (http.readyState == 4)
{
   var response = http.responseText;
   document.getElementById('comune').innerHTML = response;
}
```

Processa.php ritorna i dati già in formato HTML e questo semplifica il codice di *gestisciContenuto*: da notare come l'oggetto XMLHttpRequest nonostante il suo nome possa essere utilizzato per passare dati in qualsiasi formato (come avviene in questo caso) e non solo XML.

- FILE: 007/registrazione.php, 007/registrazioneAjax.php, 007/processa.php, 007/ajax.js, 007/comuni.sql

7.3 Recupero di dati da una fonte esterna con Ajax

- PROBLEMA: scrivere un programma che recuperi dei dati azionari da un sito esterno in formato XML e li mostri a video
- SOLUZIONE: questo esercizio è molto simile a quello precedente, la differenza sta solo nel fatto che questa volta i dati ritornati sono in formato XML e quindi devono essere trattati per essere trasformati in HTML. Il vantaggio di questo approccio sta nel fatto che il *provider* dei dati, che in questo esempio potrebbe essere un'agenzia di borsa, si limita a fornire un file XML magari aderente a un certo Schema o DTD, sarà il *consumer* dei dati che dovrà preoccuparsi di elaborarlo per farne quello che ritiene opportuno. Le differenze nel codice sono minime: la funzione *gestisciContenuto* dovrà fare il parsing del file XML e navigare al suo interno per ottenere i vari elementi, attraverso il seguente codice

```
var response = http.responseXML.documentElement;
var lista = "";
var titoli = response.getElementsByTagName('valore');
for (var i=0; i < titoli.length; i++)
lista += "<tr>" + titoli[i].firstChild.data + "</tr>";
    document.getElementById('valori').innerHTML = lista;
```

Si può notare come il metodo per leggere il contenuto del file XML sia responseXML.documentElement e non come prima responseText. Il file

retrieving.php, che fa da provider dei dati, in questo caso crea dei dati casuali con l'avvertenza di formattarli in modo che siano XML valido.

- FILE: 007/azioni.php, 007/retrieving.php, 007/ajax2.js

Appendice A

Esercizi proposti

A.1 Linguaggio HTML

A.1.1 Esercizio 1

Creare una pagina HTML con un titolo grande centrato e lo sfondo celeste.

A.1.2 Esercizio 2

Aggiungere alla pagina dell'esercizio precedente del testo con alcune parole in grassetto e altre in italico.

A.1.3 Esercizio 3

Creare una pagina che contenga il codice di un programma in modo che sia preformattato e mantenga l'aspetto originale

A.1.4 Esercizio 4

Creare una pagina che contenga una serie di 25 figure 100x100 px disposte a formare una griglia 5 x 5.

A.1.5 Esercizio 5

Modificare la pagina dell'esercizio precedente in modo che ogni figura sia un link a una pagina dove l'immagine cliccata appaia in dimensione normale, supposto che questa sia 500 x 500.

A.1.6 Esercizio 6

Creare una pagina contenente una lista non ordinata di link ad altre pagine e una lista ordinata di link all'interno della pagina stessa.

A.1.7 Esercizio 7

Creare una pagina contenente una tabella di votazioni di un esame.

A.2 CSS

A.2.1 Esercizio 1

Scrivere un file HTML con il proprio nome e cognome con le lettere di colore rosso su sfondo bianco. Quando si passa con il mouse sul nome questo deve diventare azzurro e grassetto, quando si passa sul cognome questo deve diventare nero e italico.

A.2.2 Esercizio 2

Creare un file HTML con dei link che abbiano l'apparenza di bottoni, cliccando sui quali si viene rimandati ad altre pagine.

A.2.3 Esercizio 3

Utilizzando le classi creare un testo in cui alcune parole di classe "importante" siano evidenziate rispetto al testo e i paragrafi di classe "codice" siano resi con un font diverso.

A.2.4 Esercizio 4

Creare un file in cui le prime lettere dei paragrafi siano di dimensione due volte maggiore rispetto al resto e di colore blu.

A.2.5 Esercizio 5

Creare un file contenente un menù di tipo verticale a sinistra con link ad altre pagine.

A.2.6 Esercizio 6

Creare un file che contenga un'immagine usata come sfondo del testo.

A.3 Javascript

A.3.1 Esercizio 1

Modificare la classe Convertitore e l'interfaccia dell'esercizio 4.1 in modo che sia possibile scegliere tra almeno 10 conversioni diverse (supponendo i cambi fissi).

A.3.2 Esercizio 2

Prendendo spunto dalla classe Tris dell'esercizio 4.2 creare una classe per giocare a Forza 4.

A.3.3 Esercizio 3

Modificare la classe Tris dell'esercizio 4.2 in modo che sia anche possibile giocare contro il computer.

A.3.4 Esercizio 4

Creare una funzione Javascript che controlli il valore di una data inserita all'interno di un campo di testo nel formato AAAA-MM-GG e mostri un messaggio d'errore se la data non è corretta.

A.3.5 Esercizio 5

Scrivere uno script per controllare se un campo di testo contenente un indirizzo e-mail sia sintatticamente valido.

A.4 PHP

A.4.1 Esercizio 1

Creare un programma che produca lo stesso output di quello dell'esercizio 5.10. A differenza di quell'esercizio questa volta i distributori di benzina non si trovano esattamente dove finisce la benzina ma seguono questo schema: il primo distributore si trova a 50 km dalla partenza, il secondo a 30 km dal primo, il terzo a 50 km dal secondo, il quarto a 30 km dal terzo e così via, alternando sempre 50 e 30 km. Il numero di tappe deve essere il minore possibile (quindi non bisogna fermarsi a fare il pieno ad ogni distributore che si incontra ma soltanto quando si sa che non si ha sufficiente benzina per arrivare al prossimo), la macchina parte con il pieno e ogni volta che si ferma ad un distributore fa il pieno (tranne all'ultima sosta). Il programma deve anche indicare quanta benzina verrà messa nel serbatoio all'ultima sosta, considerando che la macchina deve arrivare alla fine del viaggio con il serbatoio vuoto. Se per caso non fosse possibile completare il viaggio l'applicazione dovrà comunicarlo all'utente.

A.4.2 Esercizio 2

Creare un programma che prendendo spunto dall'esercizio 5.11 aggiunga le funzioni presenti nell'esercizio 4.3 per ottenere un calcolatrice come quella implementata però tramite un linguaggio lato server (la parte grafica può essere riprogettata a piacimento).

A.4.3 Esercizio 3

Creare un programma che permetta di recuperare le informazioni su uno studente inserendo il suo cognome.

A.4.4 Esercizio 4

Creare un programma che permetta di inserire dei film all'interno di una base di dati opportunamente costruita, dove i film possono essere VHS o DVD e appartenere ad un genere. Di ogni film interessa il titolo, l'anno di produzione e il regista.

A.4.5 Esercizio 5

Supponendo che il database dell'esercizio precedente appartenga a un videonoleggio, ampliarlo aggiungendo la possibilità di memorizzare le informazioni sui soci e sui prestiti effettuati. Dovranno esserci delle pagine PHP per inserire un nuovo socio, per inserire un nuovo prestito, per visualizzare l'elenco dei film non ancora restituiti con i soci che li hanno presi in prestito e per mostrare una lista dei film più visti.

A.4.6 Esercizio 6

Creare un programma PHP e relativo database che permetta di inserire i dati delle elezioni. La pagina di inserimento dei dati dovrà mostrare un menù a discesa con l'elenco dei partiti, un menù a discesa con la lista delle sezioni e un campo di testo per inserire il numero di voti presi dal partito selezionato nella sezione selezionata. Dovrà esserci anche una pagina di riepilogo dei dati che mostri per ogni partito i voti presi in ogni sezione, il totale dei voti presi e la percentuale sul totale.

A.4.7 Esercizio 7

Creare un programma PHP per la gestione di un medagliere. La pagina di inserimento dei dati dovrà mostrare un menù a discesa con la lista dei paesi partecipanti e un menù con il tipo di medaglia (oro, argento e bronzo). Ogni volta che si selezioneranno un paese e una medaglia, una medaglia di quel tipo verrà aggiunta al paese selezionato. Deve essere presente anche una pagina di riepilogo del medagliere, con i paesi ordinati per quantità di medaglie con gli stessi criteri delle Olimpiadi.

A.4.8 Esercizio 8

Utilizzando quanto visto reimplementare il sito del capitolo 2 utilizzando esclusivamente pagine PHP. I dati sui mobili dovranno essere recuperati da un database. Per chi volesse utilizzarlo viene fornito un database con qualche dato inserito (*arredamento.sql*), altrimenti il database può essere creato ex-novo.

Appendice B

Strumenti di sviluppo

B.1 L'editor RScite

SciTE è un potente editor, studiato per il programmatore, ma che può essere utilizzato da chiunque, indipendentemente dalla sua professione. Ad un primo approccio, SciTE appare semplice e "scarno", con un'interfaccia nella quale trova posto solo un menù, poche icone e un'ampia area di lavoro.

Una delle peculiarità di SciTE è l'ampio utilizzo delle combinazioni di tasti, ed in genere, la predilezione nell'uso della tastiera. Questa caratteristica si pone lo scopo di consentire l'editazione del testo senza mai staccare la mani dalla tastiera, migliorando il livello di efficienza e produttività.

B.1.1 Configurazione e utilizzo

Alcuni file aggiuntivi devono essere copiati a mano nelle opportune cartelle: php.properties e css.api devono essere copiati nella cartella principale di RScite (la versione localizzata in italiano di Scite); tidy.exe deve essere sovrascritto a quello già presente nella cartella tools/tidy. Per poter vedere nel menù contestuale la scritta "Edit with Scite" seguire le istruzioni contenute all'interno del file wscitecm121.zip.

Dopo aver aperto un file l'editor capirà la sintassi in base all'estensione, colorando correttamente le parole chiave.

Per aver l'aiuto contestuale di PHP copiare il file php_manual_it.chm in C:\Programmi\RScite\PHP\HELP\ (dove PHP\HELP deve essere creata): a questo punto posizionando il cursore nei pressi di una funzione PHP e premendo F1 si otterrà l'apertura del manuale alla pagina della funzione.

Per inserire dei tag HTML premere CTRL + < e apparirà una finestra di dialogo dove scegliere i tag da inserire. Per modificare o inserire nuovi tag aprire il file htmlfr.ini nella cartella luasrc.

Per avere accesso a dei comandi aggiuntivi utili (ad esempio per produrre dei codici colore HTML) premere il tasto F12.

Per ottenere un elenco di autocompletamento delle parole chiave digitare le lettere iniziali e premere CTRL + SPACE: questo metodo funziona per i vari tipi di file con risultati diversi: nei file PHP completa le funzioni, nei file HTML i tag ecc.

Per ottenere gli attributi dei tag HTML dopo aver inserito il tag premere CTRL + - e per ottenere i valori degli attributi CSS aggiungere un parentesi tonda dopo l'attributo (che successivamente andrà tolta).

Per il manuale completo e per ulteriori informazioni si può far riferimento al sito ufficiale `http://www.scintilla.org/SciTE.html` e al sito della versione localizzata in italiano `http://rsoftware.altervista.org/index.php?mod=none_scite`. Quest'ultima versione contiene altri strumenti molto utili: un editor esadecimale (ICY editor), un comparatore di file (winMerge) e un programma per la validazione di file HTML in locale (Tidy) .

B.2 L'ambiente di sviluppo XAMPP

XAMPP è una distribuzione Apache facile da installare contenente MySQL, PHP e Perl. Contiene inoltre molti altri moduli tra cui un server SMTP, un FTP server e il supporto a OpenSSL.

B.2.1 Configurazione e utilizzo

La versione di XAMPP raccomandata è quella in formato zip autoscompattante e si consiglia di installarla in c: per evitare possibili problemi di permessi su alcune installazione di Windows. Dopo averla estratta dovrebbe già essere funzionante, comunque eseguire xampp-setup la prima volta per permettere l'eventuale settaggio dei percorsi se ce ne fosse bisogno.

Per avviare l'esecuzione dei vari server esistono sostanzialmente due modalità:

- tramite il programma xampp-control che fornisce un'interfaccia grafica per avviare o arrestare i vari servizi
- tramite i file batch start-xampp, stop-xampp, che vengono attivati con doppio click

Per controllare se tutto funziona dopo aver fatto partire xampp con una delle modalità sopra indicate aprire il browser e nella barra degli indirizzi 127.0.0.1. Se tutto funziona correttamente apparirà la pagina iniziale di xampp.

In caso di problemi si può usare il programma xampp-portcheck, che controlla se le porte usate da xampp (almeno la 80 per HTTP e la 3306 per MySQL) sono libere: problemi potrebbero essere dati dal fatto che altri programmi le stanno utilizzando (ad esempio IIS per la porta 80) oppure che un firewall locale ne impedisce l'apertura.

Per l'amministrazione dei database MySQL si può utilizzare la web application phpMyAdmin, raggiungibile nella schermata di partenza di xampp in un link nel menù di sinistra.

La cartella dove inserire i file HTML e PHP per farli leggere dal server Apache si chiama htdocs e si trova direttamente nella cartella xampp.

B.3 Il browser Firefox

Firefox è un web browser open-source del Mozilla Project che ha come scopo quello di fornire un'esperienza di navigazione il più possibile piacevole, di ridurre al minimo i problemi di sicurezza e di implementare gli standard web così come definiti dal W3C.

Firefox è stato realizzato in modo da permettere la creazione di plug-in esterni per aggiungere nuove funzionalità e quelli consigliati per questo corso sono due: WebDeveloper e Firebug.

WebDeveloper aggiunge una barra al menù dalla quale possono essere ispezionati tantissimi aspetti della pagina mostrata nel browser, che permettono di scoprire eventuali errori nella pagina, visualizzare informazioni utili, validare la pagina e molto altro.

Firebug verrà principalmente utilizzato per fare il debug di codice Javascript, avendo la possibilità di inserire breakpoints, permettere un'esecuzione passo-passo, visualizzare il valore delle variabili, ma permette di visualizzare anche altri aspetti come ad esempio i dati scambiati nella comunicazione tra client e server, comprese le intestazioni di protocollo.

B.4 TopStyle Lite

La versione Lite di questo prodotto è gratuita ed è un ottimo editor CSS. Aprendo un file CSS nel pannello principale verrà mostrato il testo del file, mentre nel pannello di destra sarà possibile aggiungere le proprietà desiderate, ognuna delle quali presenta anche i valori che può assumere. Nel pannello inferiore per ogni modifica effettuata sarà subito possibile vedere l'effetto che si ottiene, permettendo in tal modo la sperimentazione sui CSS fino ad ottenere il risultato desiderato.

B.5 XML Copy Editor

Questo programma è open source ed è un semplice editor di file XML. Oltre a poter editare i file XML e ottenerne il syntax highlighting, è possibile validare un file XML rispetto a uno Schema o a un DTD.