Javascript nella scuola

Un’introduzione attuale e divertente
alla logica di programmazione

Bozza dei primi due capitoli

Giulio e Pia

16 novembre 2001

Indice

0.1 Introduzione 1

1 La tua prima pagina web **3**
1.1 Scrittura e visualizzazione del primo file di testo 3
1.2 Trasformiamolo il primo file in documento HTML 6
1.3 Arricchiamo il testo con quelche effetto grafico 7
1.3.1 Fondo, colori, dimensioni e allineamento del testo 7
Colore del fondo pagina 7
Il tag `<center>` 8
Intestazione `<h`*n*`>` 8
Opzione “size” del tag `<font>` 10
1.3.2 Controllo del colore mediante l’opzione *color* di `<font>` 10
1.3.3 Sottolineato, corsivo, grassetto e disallineamenti 10
1.3.4 Paragrafi 10
1.4 Mettiamo tutto insieme 11
1.5 Cosa sarebbe il web senza immagini? 11
1.6 Liste ordinate e non ordinate 13
1.7 Pronti per la navigazione? 13
1.8 Caratteri speciali 15
1.9 Commenti fra il codice HTML 16
1.10 Farsi navigare 16
1.11 Esercizi 16

2 Cominciamo a programmare **17**
2.1 Schema logico delle operazioni da eseguire 17
2.2 Primo programma in Javascript 19
2.3 Ciclo di istruzioni (loop) mediante “`do`–`while`” 19
2.4 Il trucco del giovane Gauss 22
2.5 `while` e `do`–`while` 23
2.6 Uso delle parentesi graffe in `while`, `do`–`while` e altri costrutti 23
2.7 `i++` o `++i`? 25
2.8 `for` 25
2.9 Le quattro operazioni aritmetiche 25
2.10 L’operatore aritmetico “modulo” 26
2.11 Altre operazioni matematiche (“oggetto” Math) 26
2.12 Controllo del flusso mediante `if` 27
2.13 Salta giro (“`continue`”) e uscita di emergenza (”`break`”) 31
2.14 `switch` – il commutatore 31

2.15 Funzioni 32
2.16 Controllo del risultato 35
2.17 Esercizi 35

0.1 Introduzione

Si sentono spesso amici e conoscenti che vorrebbero, per loro stessi o per i loro figli, "imparare ad usare il computer". Come è noto, le possibilita' offerte dai computer sono talmente tante e variegate che l'espressione "imparare ad usare il computer" può risultare vaga. Ci si può leggere, scrivere e far di conto. Lo si può usare per consultare materiale didattico. Può essere uno strumento di comunicazione alternativo alla posta convenzionale, ai circoli culturali o ai muretti. Oppure ci si può semplicemente giocare e sentire musica. Tutte applicazioni legittime, l'importante è chiarirsi l'uso che se ne vuol fare e programmare un investimento appropriato all'uso futuro, senza acquistare, tanto per fare un esempio, l'ultimo Pentium IV se si intende fare prevalentemente videoscrittura.

Molto probabilmente, l'uso meno frequente dei milioni di computer domestici è quello per il quale i computer sono nati, ovvero la programmazione. Qualcuno potrebbe obiettare che imparare a programmare non serve, perchè oggigiorno è possibile acquistare programmi belli e fatti per tutte le nostre esigenze e qualcuno scriverà quelli per le nostre esigenze future. A nostro giudizio questo punto di vista è un po' miope. Sarebbe come dire che bisognerebbe smettere di insegnare tutto quello che la maggior parte delle persone non utilizzerà nella vita pratica, come tutta la matematica e la geometria che si insegna dopo la scuola elementare, la letteratura, la filosofia e addiruttura, e forse soprattutto, lo scrivere, data la bassa percentuale di persone che scrive quoditianamente qualcosa che va oltre la lista della spesa o una cartolina dal mare. Insomma, a nostro giudizio, la programmazione è un fatto culturale, come sapere cos'è una radice quadrata o il teorema di Pitagora, e che quindi dovrebbe entrare di diritto nelle scuole insieme ad una non ben definita "alfabetizzazione informatica" che troppo spesso si riduce a cliccare su qualche icona o a stampare brani e immagini di una enciclopedia elettronica, spacciando queste ultime operazioni per una "ricerca".

Purtroppo, capiamo che, come gran parte delle cose formative, anche la programmazione richiede un certo impegno. Bisogna imparare un nuovo linguaggio, più astratto e formale di quelli umani (anche se fortunatamente più semplice di questi ultimi), padroneggiare qualche applicazione matematica, finanziaria o procedurare sul quale applicare il linguaggio di programmazione e concentrarsi sulla scrittura del programma. Poi c'è il problema di quale sia il linguaggio più appropriato da imparare: Basic, Cobol, Fortran, Pascal, C, C++ o Java? tanto per fare dei nomi. Ciascun programma ha i suoi vantaggi di semplicità, potenza o semplicemente di tradizione, come il Fortran, con il quale gli autori hanno avuto maggior confidenza ma che considerano oggigiorno decisamente obsoleto.

La scelta su Javascript è stata inizialmente casuale, nel senso che non è stata fatta nessuna vera analisi di comparazione fra i vari linguaggi. Anzi, questo testo è nato esattamente nel modo opposto di come si potrebbe pensare. Ci siamo imbattuti per caso in Javascript e ne siamo restati talmente positivamente impressionati che abbiamo pensato che forse esso poteva essere adatto ad avvicinare alla programmazione sia ragazzi che chiunque altro fosse interessato ad entrare un po' più da vicino nell'affascinante mondo dell'informatica.

A nostro giudizio, i punti di forza di Javascript per il nostro intento sono i seguenti:

- Il linguaggio contiene tutti gli elementi tipici di un linguaggio di programmazione, quale il controllo di flusso, i cicli e i sottoprogrammi (funzioni).

- Javascript ha molte somiglianze con linguaggi e programmi di scripting moderni, come C, C++, Java e Perl.

- Il linguaggio è interpretato e tutti coloro che hanno un browser come Internet Explorer o Netscape, ovvero tutti coloro che oggi possiedono un PC, hanno già gratis l'interprete.
- Javascript è eseguito all'interno di una pagina web e quindi il suo apprendimento richiede un minimo di padronanza del linguaggio HTML. Quindi come sottoprodotto dell'apprendimento della programmazione si impara anche a scrivere pagine web in un modo didatticamente, e spesso anche praticamente, piu' valido di editor che nascondono l'HTML.

Un difetto di Javascript è che non permette l'accesso a file. Questa manchevolezza del linguaggio è voluta, e dovuta al fatto che, tipicamete, il codice viene scaricato da un *server* remoto ed eseguire sul *client*, ovvero il computer dove gira il browser. Sarebbe quindi molto pericoloso se una pagina web cominciasse a scrivere sul disco dell'utente che l'ha scaricata. Però non riteniamo che questo sia un grande difetto, sia per l'uso sia didattico che per quello di sussidio alla scrittura di pagine web dinamiche che di questo linguaggio vogliamo fare.

Capitolo 1

La tua prima pagina web

Poiché Javascript è un linguaggio interpretato dal *browser,* tipicamente Internet Explorer o Netscape, è necessario imparare come costruire una semplice pagina web all'interno della quale inserire successivamente il codice di Javascript.

1.1 Scrittura e visualizzazione del primo file di testo

Per procedere con un certo ordine, cominciamo a crearci una cartella in cui porre i file che elaboreremo. Sebbene si supponga un minimo di familiarità con le operazioni di base sul PC, vediamo come creare la cartella e quindi creare un semplice file di testo accessibile anche dal browser.

Creazione della cartella C:\MioWeb

- Doppio-click su Risorse del Computer: → si apre una finestra che mostra alcune icone, fra le quali quella del disco rigido (tipicamente indicato con (C:)).
- Doppio-click sull'icona del disco rigido (C:): → si apre una nuova finestra che mostra altre icone relative a cartelle e file. In alto sulla finestra compare una *barra di controllo* con File, Modifica, Visualizza, etc.
- Click su File, quindi su Nuovo e infine su Cartella (vedi Fig. 1.1): in fondo alla finestra compare una nuova cartella con il nome provvisorio *evidenziato* Nuova Cartella. Digitare direttamente sulla tastiera MioWeb, senza la necessità di posizionarsi con il mouse sulla casella che contiene il nome provvisorio: appena si comincerà a digitare il nuovo nome, quello provvisorio si dissolverà. Digitato il nome, terminare con Invia (ovvero Enter, a seconda delle tastiere)
  Se per caso ci fossero dei problemi e non fosse possibile inserire il nome corretto, cliccare sulla casella del nome con il pulsante destro, quindi scegliere Rinomina e correggere il nome.
- Lasciare aperta la finestra di (C:), ci servirà fra poco. (Se la si dovesse chiudere, non c'è nessun problema: sarà sufficiente ripercorerre il cammino fatto per arrivarci.)

Creazione del file ciao **in** C:\MioWeb **con** Blocco Note

- Click su START in basso a sinistra dello schermo, click su programmi, quindi su Accessori e indine su Blocco Note: → si aprirà una finestra vuota intitolata Senza Nome – Blocco Note.

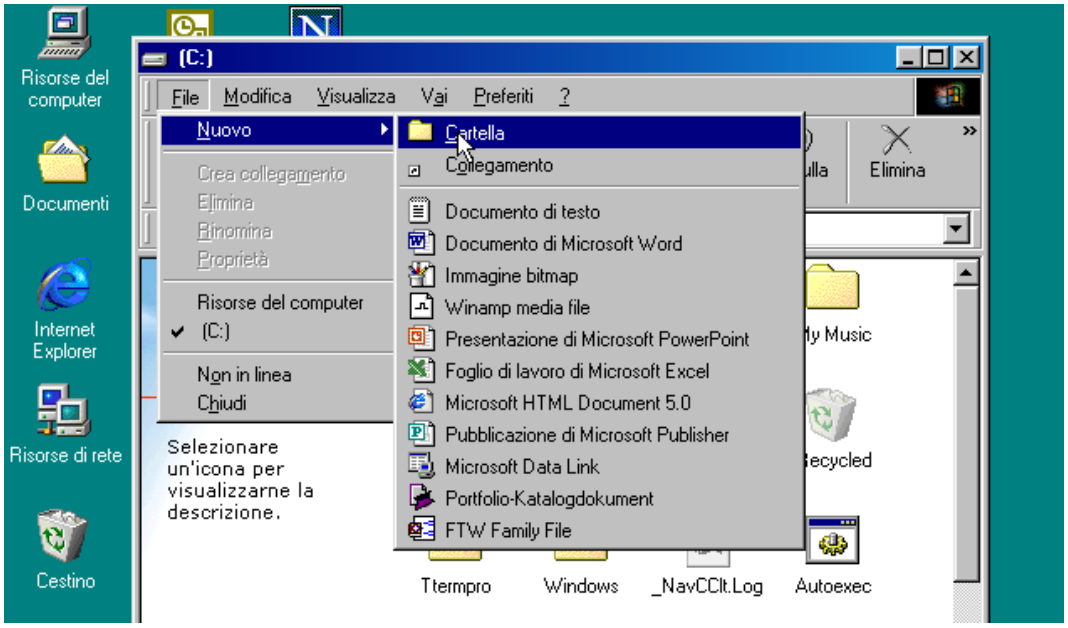

Figura 1.1: Creazione della cartella 'MioWeb'.

- Nella finestra scrivere qualcosa di banale, ad esempio

  ```
  Ciao a tutti!
  Questo è il primo testo che provo
  a visualizzare con il browser.
  ```

  Quindi bisogna salvare il contenuto del Blocco Note sul file ciao.html della cartella C:\MioWeb.

- Click su file della finestra del blocco Note, quindi su Salva con Nome. Si apre una finestra che permette di scegliere la cartella nella quale inserire il nome. Scegliere il percorso Risorse del computer → (C:) → MioWeb. Quindi scrivere ciao.html nella casella Nome del file e digitare Invia, oppure cliccare su Salva (Fig. 1.2).

Visualizzazione mediante Internet Explorer (o Netscape)

- Tornare sulla finertra che mostra la cartella (C:), all'interno della quale era stata creata la cartella MioWeb e doppio-cliccare sull'icona di quest'ultima. Si apre una nuova finestra che mostra il file ciao. L'icona mostra una e, ad indicare che il file è riconosciuto di pertinenza di Internet Explorer (o una N nel caso in cui il browser preferito sia Netscape).
- Doppio-click sul file ciao e attendere qualche secondo. Viene attivato il browser, il quale mostra il contenuto del file (Fig. 1.3)

Certo, come prima pagina non è molto eccitante, ma per cominciare non è male. Vedremo fra un po' come fare delle cose più carine. Per ora è istruttivo sapere che per fare una semplice pagina web con contenuto puramente testuale, c'è bisogno soltanto di conoscere come scrivere un file testo.

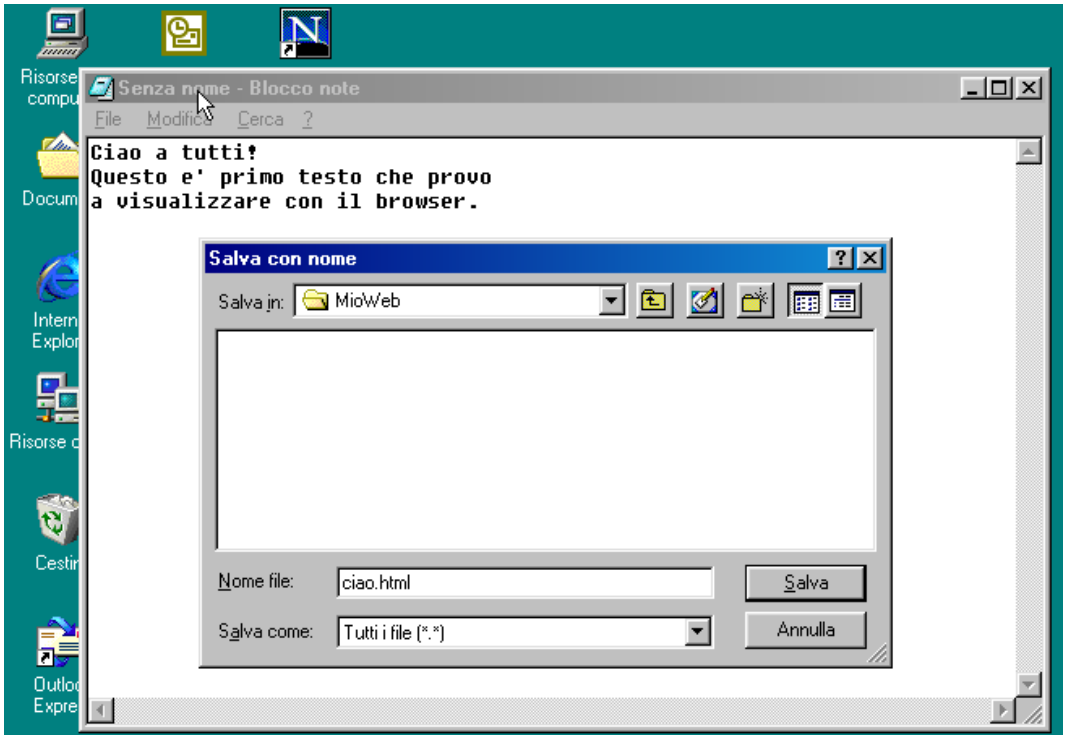

Figura 1.2: Creazione del file ciao.html.

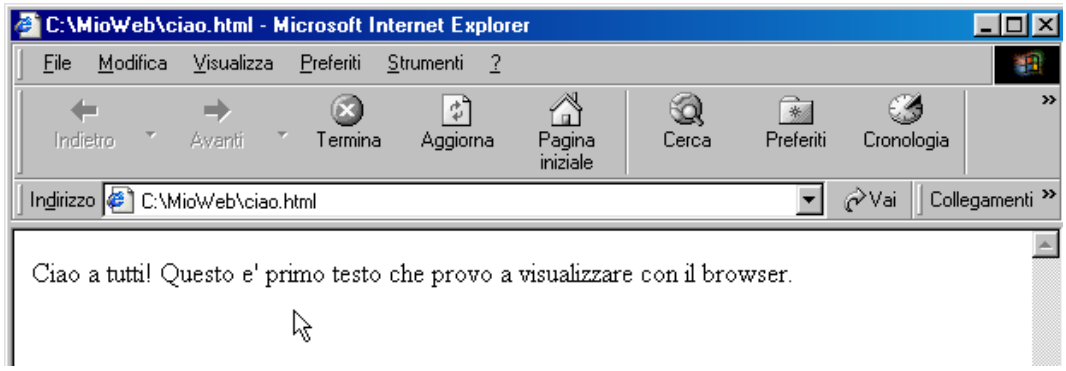

Figura 1.3: Visualizzazione con browser del primo testo.

Correzione e nuova visualizzazione del file Prima di andare avanti, è opportuno acquisire una certa padronanza con le procedure di modifica del file e di visualizzazione con il browser. Per esercizio, è conveniente cominciare completamente da capo, chiudendo tutte le finestre aperte (spegnere e riaccendere il computer è decisamente esagerato, anche se è comprensibile che questa operazione estrema dia un senso di sicurezza sul controllo del sistema...).

Modifica

- Selezionare il Blocco Note come illustrato sopra. Ora scegliere l'opzione Apri del menu File.
- Scegliere la cartella MioWeb. Soprendentemente essa risulterà vuota. Non c'è da preoccuparsi: è dovuto al fatto che normalmente ("di default") il Blocco Note cerca soltanto i file con l'estensione .txt, ad esempio ciccio.txt. Bisogna dirgli di cercare tutti i file, cliccando sulla freccia verso il basso a destra della casella Tipo file, sulla quale si trova scritto "documenti di testo". Scegliendo quindi "Tutti i file (*.*)", apparirà il nostro file ciao.html, sul quale si potrà doppio-cliccare per selezionarlo.
- Apportare le modifiche.
- Salvare con l'opzione Salva di File. Nel caso si ritenga che valga la pena di salvare sia la vecchia versione che la nuova, occorrerà "salvare con nome", come visto precedentemente.

Visualizzazione

- Il modo più semplice è, procedendo come descritto sopra, di aprire l'icona Risorse del computer, poi (C:) e infine Mioweb e doppio-cliccare sul file da visualizzare.
- È anche possibile indicare direttamente al browser il file da aprire, scrivendo nella finestrella dell'indirizzo della pagina web che si è abituati a riempire l'indirizzo del file, preceduto da file:, ad indicare che si tratta di un file all'interno del computer e non di un indirizzo http. Nel nostro caso scriveremo file:C:\MioWeb\ciao.html.

1.2 Trasformiamolo il primo file in documento HTML

Per capire la differenza sostanziale fra il "file-tto" ciao.html e una pagina web vera e propria, prova a navigare con il browser su una pagina web di tuo gradimento e a vedere cosa si nasconde dietro la bella pagina che vedi. Questo si può fare cliccando su visualizza della barra di controllo di Internet Explorer e quindi su HTML (su Netscape in inglese View seguito da Page source). Viene visualizzato qualcosa di incomprensibile, che per alcuni siti può essere raccapricciante. Ebbene, quelle non sono altro che le istruzioni che il browser esegue per mostrarvi la pagina web bella ordinata, a colori e con figure. Per esempio, vedrete che tutti questi testi cominciano con `<html>` (o `<HTML>` , maiuscole e minuscole sono irrilevanti in questo contesto, mentre saranno importanti in Javascript). Poi ci si incontra `<head>`, seguito dopo qualche riga da `</head>`. Il testo finisce con `</html>`, preceduto da `</body>` (questo fa venire il sospetto che da qualche parte dopo `</head>` debba esserci un `<body...>`). Queste istruzioni sono chiamate *tag*. Applichiamole al nostro semplice caso, trasformando il file `ciao.html` già esistente nel seguente:

```
<html>
<head>
<title>
Mia prima pagina web
</title>
</head>
<body>
Ciao a tutti!
Questo è il primo testo che provo
a scrivere in html.
</body>
</html>
```

e salviamolo con il nome ciao1.html.

Quando lo visualizziamo con il browser il risultato è inizialmente scoraggiante. Abbiamo fatto più lavoro per inserire i tag e la pagina risultante è decisamente più bruttina di quella che abbiamo creato con il file precedente: il browser non riconosce gli accapo! In realtà, questa è una proprietà del linguaggio HTML, che sembra inizialmente bizzarra, ma risulta comoda quando il codice HTML diventa complicato. In tale caso, possono servire spesso molte linee di testo per scrivere quello che il browser dovrà presentare su una sola riga. Quindi è opportuno distinguere gli accapo del Blocco Note, che sono per comodità di chi scrive, con gli accapo del prodotto finale come dovrà essere visualizzato dal browser.

Il tag per dire al browser di andare a capo è `<br>`. Se lo scriviamo ad ogni punto in cui vogliamo spezzare il testo, otteniamo il risultato desiderato (il file `ciao2.html` viene lasciato come semplice esercizio).

A questo punto, dopo l'introduzione dei tag `<html>`, `<head>`, `<body>` e `<br>` (il tag `<title>` è opzionale per ora) ci siamo ricondotti a quanto eravamo in grado di visualizzare senza alcun tag. Mostriamo ora come, con un paio di altri tag e di qualche opzione si riesce a fare qualcosa che va decisamente molto più in là del semplice file di testo iniziale ciao.html (che avremmo fatto meglio a chiamare ciao.txt! Infatti abbiamo utilizzato l'estensione .html soltanto come trucco per far partire il browser automaticamente).

1.3 Arricchiamo il testo con quelche effetto grafico

1.3.1 Fondo, colori, dimensioni e allineamento del testo

Colore del fondo pagina

La prima finezza che possiamo apportare è quella di dare un colore di fondo alla pagina web. È sufficiente che al posto di `<body>` si metta `<body bgcolor=colore>`, ove "colore" sta per uno dei tanti colori riconosciuti (vedi tabella 1.3.1 per una lista parziale), oppure un numero di sei cifre in notazione esadecimale (i numeri in tale notazione sono preceduti dal simbolo #). Ciascuna coppia di tale numero indica l'intensità di ciascun colore fondamentale. Quindi la scala di intensità di ogni colore va da `#00` a `#FF`, ovvero da 0 a 255 in notazione decimale. Ne segue che `#000000` sta ad indicare l'assenza di colori (nero), `#FFFFFF` tutti i colori all'intensità massima (bianco). Se le tre componenti sono uguali (ad esempio `#808080` o `#A2A2A2`) si hanno i toni di grigio. Il numero totale di combinazioni di colore è pari a $256 \times 256 \times 256 = 16777216$.

Tabella 1.1: Alcuni dei colori riconosciuti da html.

colore	valore RGB	colore	valore RGB	colore	valore RGB
white	`#FFFFFF`	black	`#000000`	gray	`#808080`
red	`#FF0000`	lime	`#00FF00`	blue	`#0000FF`
cyan	`#00FFFF`	magenta	`#FF00FF`	yellow	`#FFFF00`
salmon	`#FA072`	green	`#008000`	brown	`#A52A2A`
orange	`#FFA500`	purple	`#800080`	beige	`#F5f5DC`
darkred	`#8B0000`	darkgreen	`#006400`	darkblue	`#00008B`
silver	`#C0C0C0`	orangered	`#FF4500`	gold	`#FFD700`
violet	`#EE82EE`	aquamarine	`#7FFFD4`	pink	`#FFC0CB`
ivory	`#FFFFF0`	yellowgreen	`#9ACD32`	royalbue	`#4169E1`
olive	`#808000`	midnightblue	`#191970`	lightblue	`#ADD8E6`

Scegliamo ad esempio “lightblue”. E’ anche possibile definire il colore del testo aggiungendo l’opzione `text=colore` (si noto l’assenza di virgole fra una opzione e l’altra). Si provi ad esempio con

```
<body bgcolor=lightblue text=red>.
```

Ovviamente avremmo potuto scrivere

```
<body bgcolor=#ADD8E6 text=#FF0000>.
```

Il tag `<body>` ha molte altre *opzioni* alle quali non siamo interessati per il momento.

Il tag `<center>`

Potremmo essere interessati a mettere il saluto “Ciao a tutti!” al centro della pagina. Per fare questo è sufficiente inserire la scritta fra il tag `<center>` e la sua chiusura `</center>`. Il messaggio risulterà al centro della pagina web, indipendentemente dalla sua larghezza. Provare per credere.

Intestazione `<h`*n*`>`

Potremmo desiderare che il saluto compaia in caratteri più grandi del testo normale. Questo effetto può essere ottenuto in diversi modi.

Se la scritta viene racchiusa fra `<h`*n*`>` e `</h`*n*`>`, con *n* compreso fra 1 e 6, i caratteri hanno delle dimensioni che decrescono all’aumentare di *n*. `<h1>Salve a tutti!</h1>` produce quindi la scritta più grande. Inoltre la scritta viene visualizzata su una sola linea, senza la necessità di mettere dei tag `<br>` prima e dopo.

Se si vuole centrare il messaggio si può usare il tag `<center>` incontrato precedentemente. Oppure si può utilizzare l’opzione “`align=center`” all’interno del tag `<h`*n*`>`. Questa opzione permette di allineare l’intestazione anche a sinistra (“`left`”, valore di default) o a destra (“`right`”). Si provi a trasformare il file ciao2.html nel seguente ciao3.html e lo si visualizzi con il browser (vedi Fig. 1.4). Approfittiamo per applicare le due opzioni di `<body>` che abbiamo incontrato nei paragrafi precedenti:

```
<html>
<head>
```

Figura 1.4: Alcuni esempi di tag <h1>, <h2> . . . <h6>, con opzione `align` uguale a `left`, `center` e `right` .

```
<title>
Prove tag Hn
</title>
</head>
<body bgcolor=lightblue text=red>
<h1>Ciao a tutti!</h1>
<h2 align=center>Ciao a tutti!</h2>
<h3 align=right>Ciao a tutti!</h3>
<h4 align=left>Ciao a tutti!</h4>
<h5 align=center>Ciao a tutti!</h5>
<h6 align=right>Ciao a tutti!</h6>
</body>
</html>
```

Opzione "size" del tag `<font>`

Come abbiamo visto, il tag `<hn>` permette di controllare le dimensioni dei caratteri, ma il suo uso è limitato a testo che si vuole usare come intestazione (infatti la "h" sta a ricordare "header", intestazione). Per cambiare le dimensioni delle parole o dei singoli caratteri in modo più flessibile si può utilizare l'opzione "size=valore" del tag `<font>` usata per controllare anche il colore e il tipo di carattere, come vedremo fra poco. Il "valore" può essere un numero compreso fra 1 e 7, per dare delle dimensioni assolute, oppure può essere del tipo *+n* o *-n* per definire delle dimensioni relativamente a quelle correnti.

1.3.2 Controllo del colore mediante l'opzione *color* di `<font>`

Il tag `<font>` permette di scegliere anche altre opzioni. Una delle più interessanti, e la sola altra che vedremo qui, è usata di scegliere il colore. Basta scrivere `<font color=colore>`, ove "colore" è uno dei colori validi (vedi tabella ...), e tutto il testo seguente cambia colore. Con `</font>` si ripristina il colore di default. Se si vuole cambiare contemporaneamente sia colore che dimensione del testo basta scrivere è sufficiente mettere nel tag entrambe le opzioni. Ad esempio, se si vuole scrivere la parola "evviva" in rosso e di una dimensione leggeremnte maggiore del testo principale, basta scrivere `<font size=+1 color=red> evviva </font>`. Come si può intuire, con l'uso di questo tag è possibile visualizzare ogni carattere con un proprio colore, come ad esempio:

```
<font color=red>A
<font color=orangered>R
<font color=orange>C
<font color=yellow>O
<font color=yellowgreen>B
<font color=green>A
<font color=aquamarine>L
<font color=lightblue>E
<font color=blue>N
<font color=darkblue>O,/font>.
```

Prova a vedere cosa succede! Si noti come in questo esempio i tag `<font>` non sono stati terminati dai corrispondenti `</font>`, con eccezione dell'ultimo. In questo caso questo semplificazione è lecita in quanto i tag agivano sulla stessa opzione `color` e quindi la nuova opzione sostituisce quella precedente. Con la chiusura finale `</font>` il colore dei caratteri si riporta a quello iniziale ("default").

1.3.3 Sottolineato, corsivo, grassetto e disallineamenti

Altri tag per possono servire per evidenziare una parola o una parte del testo sono il *corsivo*, il **grassetto** e la sottolineatura. I tag rispettivi sono `<i>`, `<b>` e `<u>`, terminati rispettivamente – c'è ancora bisogno di dirlo? – da `</i>`, `</b>` e `</u>`. Mediante i tag `<SUB>` e `<SUP>` è possibile scrivere del testo $_{\text{abbassato}}$ o $^{\text{sollevato}}$ rispetto alla linea corrente.

1.3.4 Paragrafi

A volte può essere interessante allineare alcune righe (un *paragrafo*) a sinistra, al centro o a destra. Utilizzando opportunamente `<br>` e `<center>` riusciamo nei due primi due,

ma non nell'allineamento a destra. Fortunatamente il tag `<p>` permette di eseguire queste operazioni in modo molto semplice, mediante l'opzione `align` che può valere `left`, `center` o `right`. Non c'è bisogno di chiudere il tag con `</p>` se segue immediatamente dopo un altro `<p>` che apre un nuovo paragrafo. Se l'opzione `align` è omessa, si assume sia `left`.

1.4 Mettiamo tutto insieme

A questo punto possiamo rendere più vivace il file iniziale `ciao`, facendolo diventare anche un sommario delle cose imparate in questo capitolo.

```
<html>
<head>
<title>
Mia prima pagina web
</title>
</head>
<body bgcolor=lightblue>
<font color=red>
<h1 align=center>Ciao a tutti!</h1> </font>
Ho già imparato a cambiare il <font color=red>colore</font>
e le <font size=+2>dimensioni</font> dei caratteri.<BR>
<center> So centrare una scritta, </center>
So scrivere in <i> corsivo</i> e in <b>grassetto</b> o
<u>sottolineare</u> una parola.  <br>
E, ovviamente, queste modifiche ai caratteri possono
essere usate insieme:
<center>
<font color=darkblue size=+2><b><i><u>
Così , per intederci</u></i></b></font>
</center>
<p align=right>E questo è un esempio <br>
di testo allineato a destra
A volte può essere utile disallineare <sup>in alto</sup>
o <sub>in basso</sub> dei acratteri, come ad esempio in
f<sub>0</sub> = 2.5 10<sup>-4</sup> V. <br>
Vediamo cosa mi attende la prossima lezione...
</body>
</html>
```

Questo file è stato salvato come `ciaociao.html` e dà luogo alla pagina web mostrata in Fig. 1.5

1.5 Cosa sarebbe il web senza immagini?

Nell'immaginario collettivo il mondo del web è strettamente legato a pagine multicolore Vediamo quindi come inserire un file contenente un'immagine, sia essa una foto o una

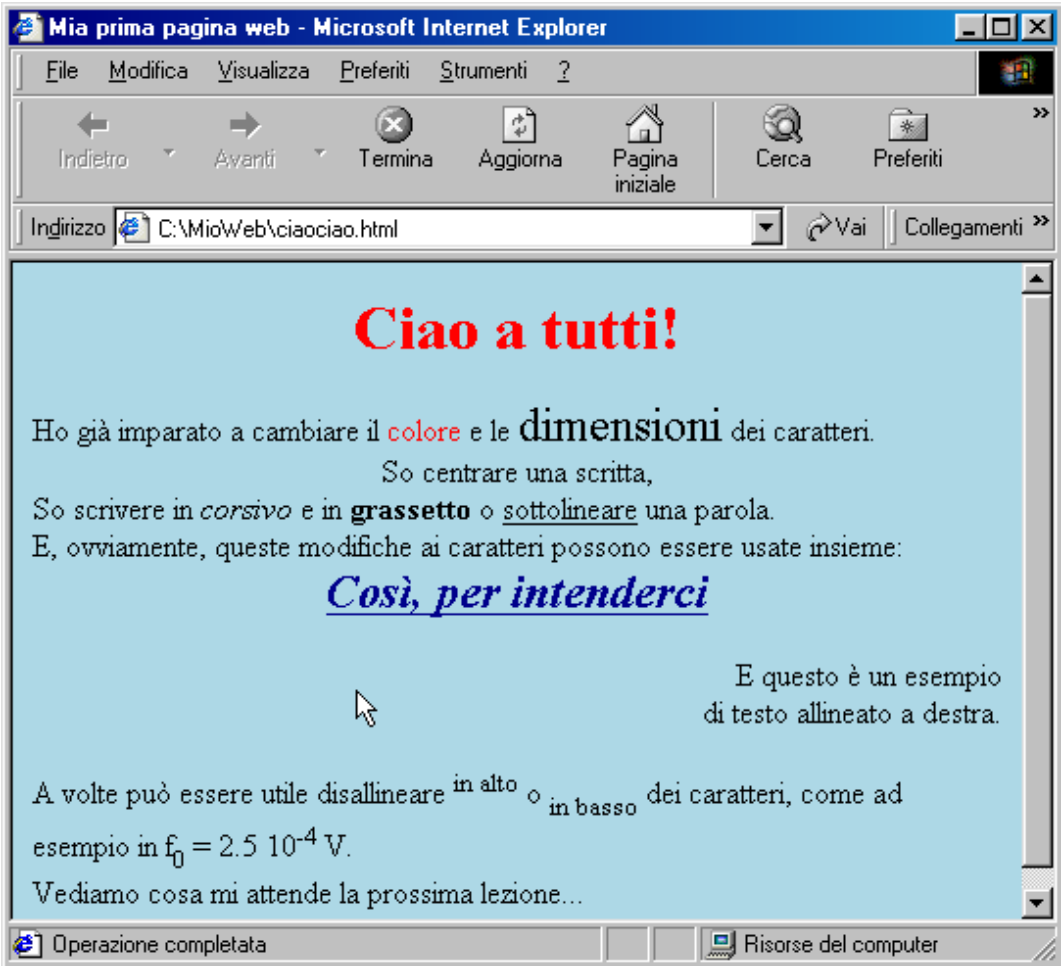

Figura 1.5: Pagina web riassuntiva dei primi tag incontrati.

elaborazione grafica. Per questo passo è opportuno avere una qualche immagine elettronica, sia essa in formato *jpeg*, *tiff*, *png* o altro. Chi non ha un'immagine da inserire se ne può scaricare una da qualche sito web, oppure le può trovare frugando nel computer.[1] Chi non è interessato a immagini, può tranquillamente andare oltre, essendo questo paragrafo puramente accessorio rispetto allo scopo primario di questo testo.

Il tag per inserire immagini è `<img>`, con opzioni che permettono di indicare il file contenente l'immagine (`src=`) e la larghezza (`width`), in *pixel*, con la quale l'immagine deve essere visualizzata nella pagina web. Il tag `<img>` non necessita del `</img>` di chiusura. Ad esempio, se file dell'immagine è *immagine.jpg* ed esso è situato stessa cartella del file html nel quale si vuole inserire l'immagine, basterà scrivere `<img src=immagine.jpg>` e l'immagine verrà visualizzata in quel punto, come se fosse un normale carattere (sebbene molto più grande, naturalmente). Con `<img src=immagine.jpg width=200>` l'immagine sarà visualizzata nella pagina web occupando 200 pixel di larghezza. L'altezza

[1] Prova a fare un "*Trova*" dei file "*.jpg", "*.png", etc, e sarai sorpreso di quante immagini sono nascoste nel computer e di cui non nospettavi neppure l'esistenza. Alcune sono di sistema, molte altre sono temporanee ("*cache*") salvate dal browser per risparmiarsi tempo in connessioni future.

verrà ricalcolata automaticamente per conservare le proporzioni originali dell'immagine. In alternativa si può anche definire l'altezza con *height*, ma bisogna prestare attenzione a utilizzare questa opzione in alternativa a *width*, perché, se i due valori non sono nel rapporto giusto, si deforma l'immagine.

Se l'immagine si trova invece in un'altra cartella bisogna dare l'indirizzo completo, il quale può essere *relativo* o *assoluto*. Ad esempio se l'immagine sta C:\immagini bisogna scrivere

```
<img src=C:\immagini\immagine.jpg>.
```

Se è in C:\MioWeb\immagini è sufficiente scrivere

```
<img src=immagini\immagine.jpg>.
```

L'immagine da caricare può anche trovarsi anche nello spazio web di un altro computer, e quindi scrivere

```
<img src=http://www.qualchesito.com/immagine.jpg>.
```

1.6 Liste ordinate e non ordinate

Capita spesso di dover fare un elenco di persone, immagini, collegamenti o altro. In HTML si possono utilizzare delle liste ordinate (tag `<ol>`) o non ordinate (`<ol>`). Ogni elemento di una lista viene preceduto dal tag `<li>`, il quale non ha bisogno di chiusura. Ne vedremo un esempio nel prossimo paragrafo.

1.7 Pronti per la navigazione?

Con le istruzioni dei paragrafi precedenti e con un po' di fantasia puoi cominciare a scrivere delle pagine graficamente accattivanti, ma che non sono ancora delle vere e proprie pagine web, in quanto mancano di uno degli ingredienti fondamentale per la navigazione: i collegamenti ipertestuali, detti anche e semplicemente *link*. Sono quelli che ti permettono di navigare per il mondo telematico e saltare da un sito all'altro del pianeta a colpi di *click*.

Il tag che crea un *iperlink* è `<a>`, la cui opzione più importante è `href` che definisce l'indirizzo di un sito web, o più semplicemente la locazione di un file nel tuo computer. Cominciamo a fare un piccolo esempio di navigazione in "acque territoriali", tanto per mostrare che si può imparare molto senza essere necessariamente connessi in rete, o senza aver attivato un vero *server* sul proprio computer. Immaginiamo di voler costruire una semplice pagina web la quale faccia da indice alle pagine web sviluppate finore (*ciao.html*, *ciao1.html*, *arcobaleno.html*, etc.). Approfittiamo per fare un esempio di liste non ordinate. Prepariamo quindi il seguente file *indice.html*

```
<html>
<head>
<title> Indice esempi web </title>
</head>
<body>
<h1 align=center>Prove di pagine web</h1>
<ul>
<li> <a href=ciao.txt>ciao.txt</a>
```

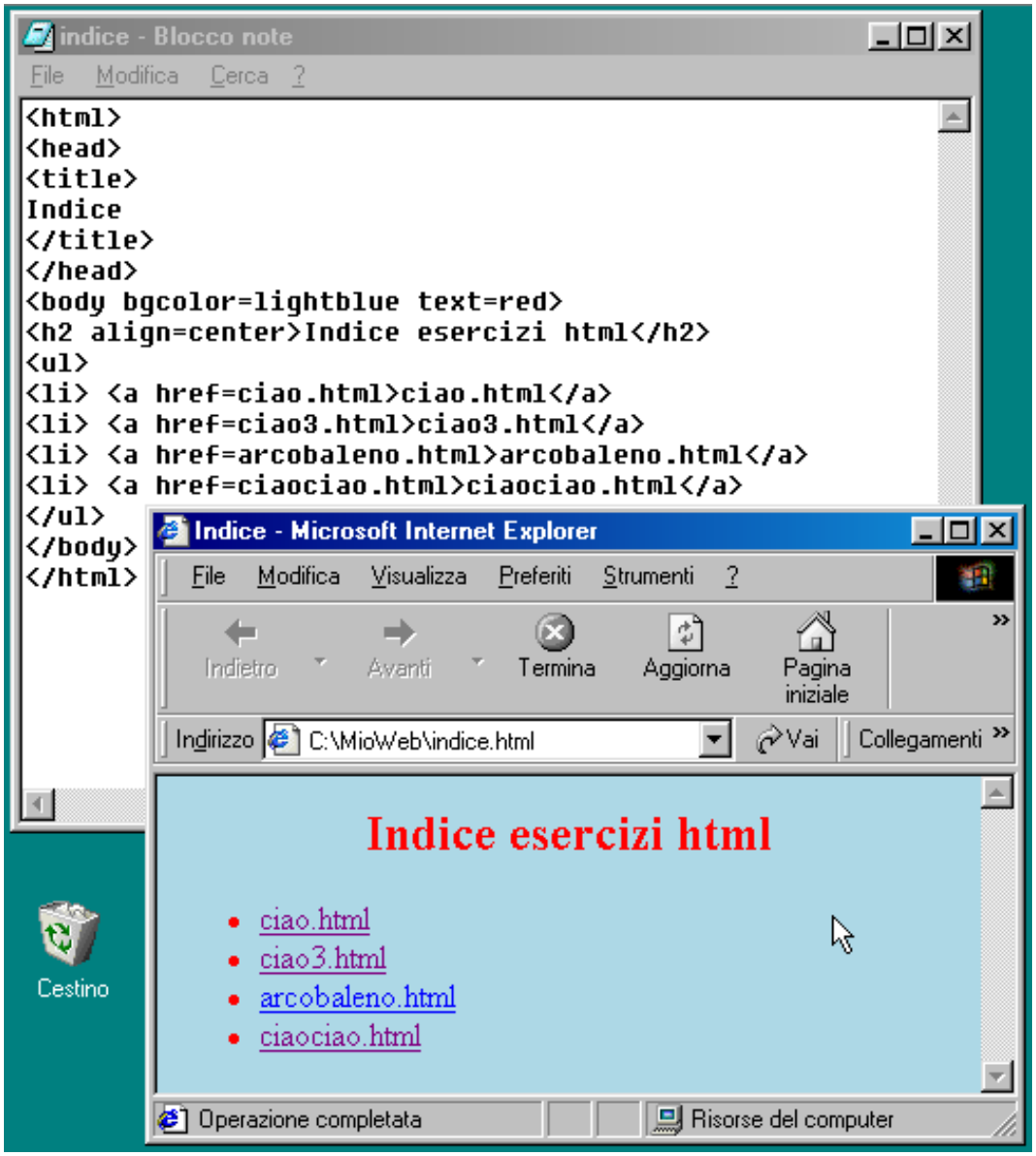

Figura 1.6: Uso del tag `<ul>` e dei link a file per creare un indice degli esercizi in HTML

```
<li> <a href=ciao.html>ciao.html</a>
<li> ...
<li> <a href=ciaociao.html>
       ciaociao.html
       </a> (riepilogo)
<ul>
</body>
</html>
```

Quanto compreso fra il tag di apertura `<a href=...>` e quello di chiusura `</a>` è "clickabile" ed viene evidenziato in modo diverso dal resto del testo. La parte clickabile può essere anche una immagine. Il link a *ciaociao.html* è stato volutamente scritto su più righe per mostrare meglio quali sono le tre parti che entrano in gioco nel link e per ricordare che gli accapo o spazi (oltre il primo) sono irrilevanti per il browser. Il risultato è mostrato in Fig. 1.6.

Inserire link relativi a pagine web su internet è altrettanto facile. Basta mettere in *href* l'indirizzo URL completo, ad esempio `http://www.qualchedominio.com`.

Da notare inoltre che il file linkato può assere anche una immagine (sia sul computer locale che su un sito remoto). Questa osservazione suggerisce di evitare di *caricare* tante immagini nella pagina web mediante il tag `<img>`. Come è noto a tutti, è un po' antipatico stare molto tempo in attesa che il computer abbia terminato di caricare immagini dalla rete. Quindi, a parte qualche piccola immagine che fa parte integrante della pagina web, come ad esempio un logo, è preferibile fare un link all'immagine. Nel caso visto precedentemente, il codice potrebbe essere

```
<a href=immagine.jpg> Vuoi vedere questa
bella foto al mare?  </a>
```

L'opzione *href* permette anche di definire un indirizzo di posta elettronica al quale mandare un mail. Al posto di `http://...` bisogna scrivere `mailto:`, immediatamente seguito dall'indirizzo di posta elettronica. Cliccando sul link, si apre una finestra pronta ad essere riempita con un messaggio. Ad esempio, se il tuo indirizzo email è `paolo@casa.mia.it`, basterà scrivere nel file html

```
<a href=mailto:paolo@casa.mia.it>scrivimi</a>
```

1.8 Caratteri speciali

Abbiamo visto negli esempi precedenti che soltanto il primo spazio che divide una parola dall'altra è riconosciuto dal browser. Come fare allora a inserire più spazi? È possibile istruire il browser mediante la speciale sequenza di caratteri * * che ha il significato di "nonbreaking space". Lo spesso risultato si ottiene con * *. La prima sequenza è mnemonica, mentre la seconda indica che si tratta del carattere nr. 160 dei circa 250 caratteri ISO.

La tabella 1.2 mostra alcuni dei più comuni caratteri. Si noti come anche i caratteri corrispondenti alle vocali accentate sono considerati speciali, nonostante essi compaiono nelle normali tastiere (italiane). Infatti, se si vuole scrivere una pagina web leggibile da tutti i browser si consigliato di non far uso dei caratteri accentati della tastiera.

Facciamo un esempio. Immaginiamo di voler visualizzare un testo del tipo

Il risultato è: 58°C $\pm$3°C per $C_1 = 2.3 \times 10^{-1}\,\mu$F

Il codice HTML corrispondente sarà

```
Il risultato &egrave;:  58 &deg;C &plusmn;3 &deg;C
per C<sub>1</sub> = 2.3&times;10<sup>-1</sup> &micro;F
```

Prova!

Tabella 1.2: **Valore numerico e mnemonico di alcuni caratteri ISO.**

Valore ISO	Nome	Simbolo	Valore ISO	Nome	Simbolo
#		#	*&*	*&*	&
<	*<*	<	*>*	*>*	>
{		{	*}*		}
~		~	*™*		TM
* *	* *	"spazio"	*©*	*©*	©
®	*®*	®	*°*	*°*	° (grado)
±	*±*	±	*µ*	*µ*	μ
È	*È*	È	*×*	*×*	×
à	*à*	à	*á*	*á*	á
è	*è*	è	*é*	*é*	é
ì	*ì*	ì	*ò*	*ò*	ò

1.9 Commenti fra il codice HTML

Come tuuti i linguaggi di programmazione, anche HTML permette di inserire dei commenti. Questi commenti non sono mostrati dal browser e servono come appunti interni di chi ha scritto il programma o per non far eseguire una parte di programma durante delle prove. Tutto quello che è compreso fra <!-- e --> viene semplicemente ignorato. Vedremo degli esempi del prossimo capitolo

1.10 Farsi navigare

[Come pubblicare la propria pagina web.]

1.11 Esercizi

Capitolo 2

Cominciamo a programmare

È luogo comune che "i computer sono stupidi ma veloci". Essi, almeno per ora, non fanno altro che eseguire molto rapidamente, anche molti milioni di volte al secondo, operazioni elementari assegnate loro dal programmatore. Sono quindi in grado di eseguire operazioni complesse che noi riusciamo ad impostare concettualmente ma che difficilmente riusciremmo a risolvere in pratica, almeno nel modo in cui sono esse state impostate. Però, siccome essi sono stupidi, dobbiamo dire loro esattamente quello che devono fare. Quando si sente dire che "il computer ha sbagliato", a parte casi rarissimi di guasti elettronici, ha sbagliato chi lo ha istruito. Quindi "lui" ha eseguito coscenziosamente . . . un'altra cosa.

2.1 Schema logico delle operazioni da eseguire

Cominciamo con esempio banale. Vogliamo sommare tutti i numeri interi da 1 a 1000. In principio è un problema che qualsiasi bambino può risolvere, in principio. . . Solo che ci vuole un bel po' di tempo, e non poca concentrazione. Solo se il bambino si chiama Gauss, si inventa un sistema ingegnoso per risolvere il problema evitando le mille somme. Come è noto, molto lavoro di matematici (Gauss diventò poi un grande fisico e matematico) è stato dedicato a trovare formule compatte, magari approssimate, per risolvere questo tipo di problemi. Sulla formula per risolvere questo problema ritorneremo fra poco. Per ora vediamo come istruire il computer ad eseguire "bovinamente" questo conto.

Lo schema concettuale delle operazioni che il computer deve svolgere (*diagramma di flusso*) è illustrato nella figura 2.1. s è la *variabile* che rappresenta la somma, mentre i rappresenta ciascuno dei mille addendi. Nelle prime due istruzioni si inizializzano s a zero e i a uno. Quindi (istruzione nr. 3) viene sommato i a s, che diventa 1, e i viene incrementato di 1 (istruzione nr. 4), diventando 2. È quindi sufficiente ripetere per mille volte questo *ciclo* e il gioco è fatto. Dobbiamo soltanto stare attenti a mettere un controllo sul valore di i. Finché non eccede 1000 dobbiamo ripetere le istruzioni 3 e 4. Appena esso diventa maggiore di 1000 bisogna interrompere il ciclo.

Si noti il significato del simbolo "=". Non significa che "s è uguale" a s+i (questo sarebbe vero, dal punto di vista matematico, soltanto se i valesse zero!). Nelle istruzioni dei calcolatori la notazione "$s = s + 1$" significa che il risultato dell'operazione a destra viene posto nella variabile che si trova a sinistra del segno $=$.

Con il nome di *variabile* intendiamo entità che possono assumere valori diversi, "variabili", nel corso del programma. Ad esempio, i acquisterà in questo programma tutti i valori

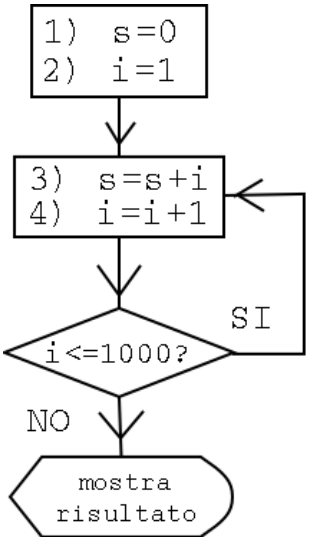

Figura 2.1: Diagramma di flusso del programma che somma i primi mille numeri interi.

interi compresi da 1 a 1001. La variabile s vale inizialmente 0, poi 1, quindi 3, 6, 10 e così via.

Ora che sappiamo cosa vogliamo che il computer faccia, glielo dobbiamo dire. Purtroppo i computer "parlano" un linguaggio molto lontano dal nostro, e ci serve un traduttore che trasformi le nostre istruzioni in un qualcosa da lui comprensibile. Questo è il ruolo degli *interpreti* e dei *compilatori*. La differenza fra queste due categorie di traduttori consiste nel fatto che i compilatori traducono l'intero codice da eseguire, mentre gli interpreti lo fanno una istruzione alla volta, con riduzione delle prestazioni. Ma dal punto concettuale non c'è differenza sostanziale.

Nel corso degli anni sono stati sviluppati molti linguaggi di programmazione (quelli intellegibili agli umani), un po' per essere ottimizzati a particolari applicazioni, un po' per tener conto dello sviluppo tecnologico che superava vecchie limitazioni sui dispositivo di lettura e scrittura che si riflettevano sul linguaggio stesso. Ad esempio, il FORTRAN, storico linguaggio per applicazioni scientifiche, ha una struttura legata alle vecchie schede perforate, in uso fino agli anni '70.

Data una particolare CPU, essa capisce un solo linguaggio linguaggio (si parla delle *istruzioni* del processore). Quindi esisteranno virtualmente tanti tipi compilatori e interpreti, uno per ogni linguaggio di programmazione e tipo di CPU (diciamo "virtualmente" in quanto chiaramente non esistono compilatori di linguaggi moderni in CPU obsolete; così pure alcuni compilatori possono essere adattato, con opportuni parametri, a diverse CPU; inoltre ci possono essere più ditte che producono compilatori). Comunque, una volta che abbiamo scritto un programma in un certo linguaggio di programmazione, esso può essere compilato per diversi computer.

Il bello di Javascript è che non bisogna preoccuparsi di procurarsi un compilatore o un interprete, in quanto ne sono già provvisti i browser, i quali riconoscono, mediante opportuni tag, le istruzioni all'interno di un file HTML.

2.2 Primo programma in Javascript

Scriviamo ora il programmino che esegue le operazioni descritte nel diagramma di flusso di figura 2.1. Javascript è un linguaggio interpretato che viene inserito nel codice html, con opportuni tag che avvertono il browser di trattarlo opportunemente. Tutto quello che è compreso fra il tag `<script>`, con opzione `language=''javascript''`, e la sua chiusura `</script>` viene interpretato cone codice Javascript, ad esempio

```
<html>
<head>
<title> Primo programma in Javascript</title>
</head>
<body>
<script language=''javascript''>
var n = 0;
var i = 1;
n = n + i;
i = i + 1;
</script>
</body>
```

Queste istruzioni corrispondono, come intuibile, alle prime quattro operazioni di figura 2.1. Se mettiamo queste istruzioni in un file con estensione html e lo apriamo con il browser, come abbiamo visto più volte nel capitolo precedente, scopriamo che ... non succede niente, ovvero avremo una una pagina completamente vuota, ma con il titolo giusto. Infatti abbiamo dato al computer le istruzioni per fare i conti, ma abbiamo dimenticato quelle per dirgli di mostrarci i risultati. Lo facciamo con l'istruzione `document.writeln( )`, usata per ora in forma molto rudimentale. Quindi, le istruzioni fra `<script ...>` e `</script>` diventano:

```
var n = 0;
document.writeln(s);
var i = 1;
document.writeln(i);
s = s + i;
document.writeln(s);
i = i + 1;
document.writeln(i);
```

Se ora apriamo il file con il browser otteniamo sullo schermo la seguenza 0, 1, 1, 2. Per avere un risultato più intellegibile dovremo imparare ad usare bene il "comando" di scrittura `document.writeln(i)`. Per ora ci possiamo accontenare. Cerchiamo prima di ottenere il risultato che ci interessa, poi vedremo come visualizzarlo meglio.

2.3 Ciclo di istruzioni (loop) mediante "`do–while`"

Il modo più semplice[1] per implementare il ciclo di istruzioni 3 e 4 di figura 2.1 è quello di usare delle istruzioni che corrispondono all'esatta traduzione inglese del seguente comando

[1] La sequenza di istruzioni che segue letteralmente il diagramma di flusso di figura 2.1 è diverso da quello descritto nel seguito. Bisognerebbe definire una *label* ("etichetta") in corrispondenza della istruzione 3 e usare

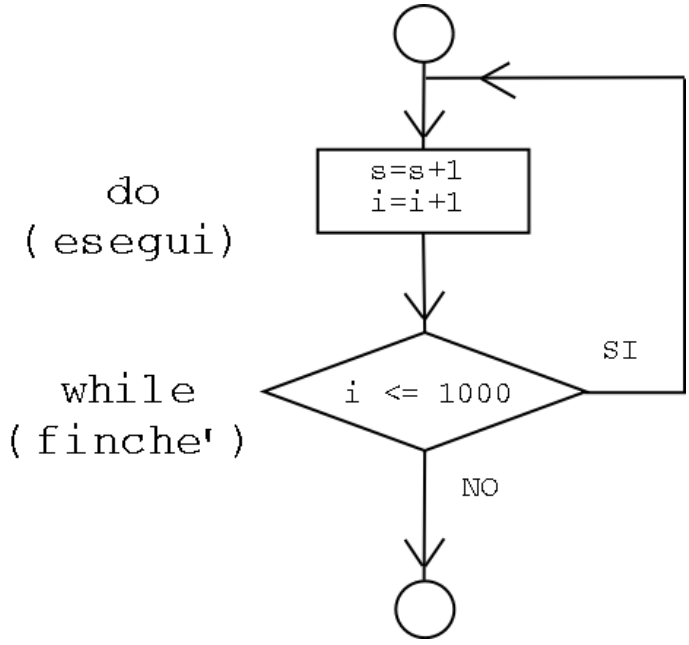

Figura 2.2: Riscrittura del diagramma di Fig. 2.4 mediante il costrutto `do–while`.

in italiano "**fai** quanto richiesto dalle istruzioni 3 e 4 **finché** i è minore o uguale a 1000". Il diagramma di flusso è mostrato in Fig. 2.2. In codice:

```
do {
     s = s + i;
     i = i + 1;
}
while (i <= 1000);
```

Quando provate a scrivere questo programma, vi accorgerete che sulla tastiera italiana mancano le parentesi graffe, usate spessissimo in tutti i moderni linguaggi di programmazione. Il fatto è che chi vende computer pensa (giustamente) più a chi li usa (a parte giochi e attività fatta a soli colpi di click...) come macchine da scrivere che chi ci fa programmazione (chi fa intensamente attività di programmazione si procura tipicamente una tastiera internazionale). Comunque, c'è un espediente con il quale è possibile digitare all'interno del blocco Note le parentesi graffe tenendo premuto il tasto Alt e digitando sul "tastierino" una sequenza di tre cifre corrispondenti al valore ISO del carattere (vedi tabella 1.2): con Alt-123 si ottiene "{", mentre con Alt-125 si ottiene "}". Si faccia attenzione al fatto che il carattere speciale appare soltanto quando si rilascia il tasto Alt.

A questo punto possiamo mettere tutte le componenti insieme, diamo il limite superiore come parametro (assegnato alla variabile *n*), aggiungiamo dei commenti, un messaggio esplicativo e otteniamo il primo programma utile in Javascript. Approfittiamo anche per vedere come è possibile scrivere in modo più compatto "`s = s + i`" e "`i = i + 1`"

un *goto* ("vai a") all'etichetta se la condizione $i \leq 1000$ è soddisfatta. Ma oggigiorno l'uso del *goto* è deprecato, al punto che alcuni linguaggio moderni non lo prendono più in considerazione.

```
<html>
<!-- qui comincia l'header -->
<head>
<title> Primo programma in Javascript</title>
</head>
<!-- qui comincia il body -->
<body>
<h2 align=center> Primo programma in Javascript </h2>
Le istruzioni Javascript eseguono pedantemente <br>
la somma dei primi 1000 numeri interi.  <br>
Il risultato è:<br> <b>
<!-- qui comincia il codice Javascript -->
<script language=''javascript''>
var s = 0;   // inizializziamo le variabili
var i = 1;
var n = 1000;
do {
    s += i;  // equivalente a 's = s + i'
    i++;   // equivalente a 'i = i + 1'
} while (i <= n);
document.writeln(s);
</b>
</script>
</body>
</html>
```

Ecco cosa ci mostra il browser:

2.4 Il trucco del giovane Gauss

Il codice Javascript fa senz'altro il "suo mestiere" ma, come abbiamo detto, non nel modo più intelligente, se per "intelligente" si intende "rapido" e "efficiente" (ma nella pratica della programmazione a volte può essere più intelligente risolvere rapidamente un problema che impiegare settimane a cercare la soluzione più appagante dal punto di vista teorico...). Se ci chiedessero di sommare $17 + 17 + 17 + \cdots$, diciamo 1000 volte, a nessuno verrebbe in mente di effettuare l'operazione alla lettera. Sappiamo infatti che il risultato è equivalente a quello che si ottiene dalla moltiplicazione 1000, il cui risultato non ha bisogno di tanti conti.

Si potrebbe pensare a ridurre la somma dei primi n numeri in un prodotto di tanti termini uguali, se riuscissimo a trovare il numero m tale che

$$1 + 2 + 3 + \cdots + n = n \times m\,.$$

Ebbene, tale numero esiste e non è altro che alla media aritmetica dei numeri che compaiono nella somma:

$$m = \frac{1 + 2 + 3 + \cdots + n}{n}\,.$$

Il problema sembra essersi complicato, perché ora, apparentemente, invece di una sola somma di n termini, ambiamo anche effettuare una divisione e una moltiplicazione. Il realtà, non c'è bisogno di tanti conti per ottenere m, in quanto esso è il "numero centrale" fra 1 e n, ovvero $m = (1 + n)/2$. In conclusione, abbiamo ottenuto

$$1 + 2 + 3 + \cdots + n = n \times \frac{n+1}{2}\,.$$

Prova con qualche n piccolo e vedrai. La storia dice che un bambino di nome Friedrich Gauss ricavò da solo questa formula alle elementari, quando il maestro aveva assegnato alla scolaresca di calcolare la somma di un centinaio di numeri[2]. Non vi stupirete quindi di sapere che l'effige di Gauss compare sulle banconote di 10 marchi, in riconoscimento della Germania per altri e ben più importi contributi contributi che Gauss apportò da grande alla matematica e alla fisica.

Con questo trucco, il codice Javascript si riduce a

```
<script language=''javascript''>
var s;
var n = 1000;
s = n*(n+1)/2
document.writeln(s);
</script>
```

o, in forma ancora più concisa, in

[2]Un altro modo di arrivare a questo risultato, forse quello originale seguito dal piccolo Gauss, è pensare a tutte le coppie di numeri $\{1, n\}$, $\{2, n-1\}$, $\{3, n-2\}$, etc. Se n è pari, abbiamo $n/2$ coppie e i numeri di ciascuna di essa danno somma $n + 1$. Se invece il numero è dispari, abbiamo $(n-1)/2$ coppie, più il numero centrale $(n+1)/2$ che rimane spaiato. Si può dimostrare facilmente, o vedere con qualche esempio, che, in entrambi i casi, la formula si riduce a $n \times (n+1)/2$.

```
<script language=''javascript''>
var n = 1000;
document.writeln(n*(n+1)/2);
</script>
```

2.5 while e do–while

Il costrutto do–while che abbiamo appena illustrato esegue il primo controllo dopo che è stata fatta la prima operazione. In alcune applicazioni si ha interesse a fare il controllo prima di eseguire le istruzioni. Javascript permette di entrambe le possibilità. Nella seconda compare soltanto while, essendo il "do" sottinteso:

```
while( ''condizione è vera''){
    ...
    istruzioni ...
    ...
}
```

Le figure 2.3 e 2.4 mostrano i diagrammi di flusso. Ovviamente il problema della somma dei 1000 numeri poteva essere risolto anche con il solo *while*. Ovviamente bisogna codificare le inizializzazioni e cambiare la condizione:

```
var s = 0;
var i = 0;   // attenzione al cambiamento!
var n = 1000;
while( i < 1000 ){   // condizione cambiata
    i++;
    s += i;  // si noti l'ordine invertito
}
```

2.6 Uso delle parentesi graffe in while, do–while e altri costrutti

Per essere precisi, l'esatta sintassi di while e do–while è

```
while( ''condizione è vera'')
    istruzione
```

e

```
do
    istruzione
while( ''condizione è vera'')
```

Le parentesi graffe servono soltanto quanto ci sono almeno due istruzioni da eseguire. Dal punto di vista logico è come se si trattasse di una sola istruzione *composta*. Questo vale anche per i costrutti for e if

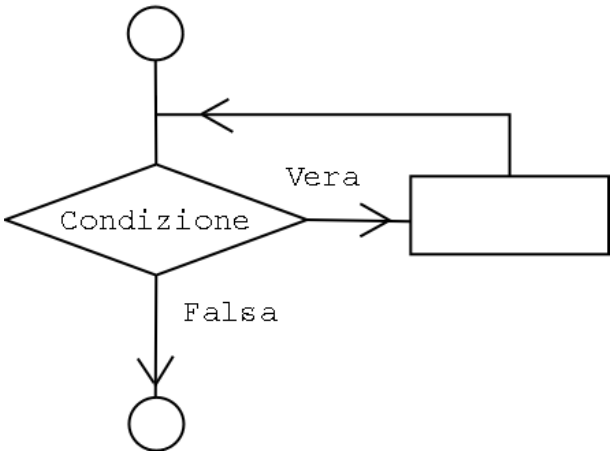

Figura 2.3: Diagramma di flusso della struttura `while`.

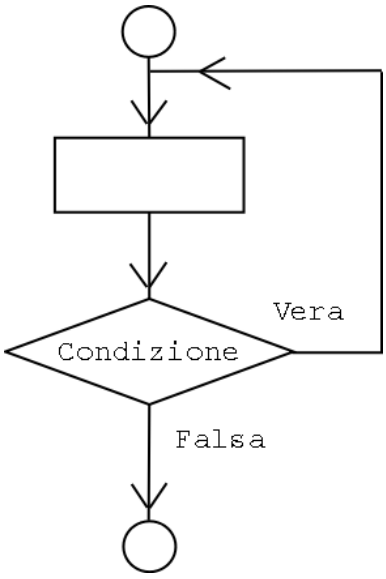

Figura 2.4: Diagramma di flusso della struttura di controllo `do–while`.

2.7 `i++` o `++i`?

Abbiamo già incontrato l'istruzione `i++` e ne abbiamo apprezzato la sua praticità. In realtà esiste la possibilità di mettere il doppio '+' prima della variabile, ad esempio `++i` (in questi esempi usiamo la variabile `i`, ma chiaramente si può anche scrivere `giorno++`, etc.). Nel caso questa istruzione sia isolata, come nel caso visto sopra, le due varianti danno lo stesso risultato. Quando invece sono a destra del segno di uguale, c'è una sottile differenza.

`i++`**:** La variabile viene prima usata e poi incrementata. Quindi

```
s = 3;
i = 0;
s += i++;
```

dà come risultato $s = 3$ e $i = 1$.

`++i`**:** La variabile viene prima incrementata e poi usata. Quindi

```
s = 3;
i = 0;
s += ++i;
```

dà come risultato $s = 4$ e $i = 1$.

2.8 `for`

I due costrutti con `while` che abbiamo illustrato sono molto potenti, in quanto permettono di gestire situazione molto complesse, comprese quelle in cui la variabile sulla quale si fa il controllo assume valori non ordinati. Ad esempio, si può continuare a leggere ("svuotare") un dispositivo finché non si incontri un carattere speciale che indichi errore o fine delle informazione ("end of file"). Quando invece si conosce bene la regola di avanzamento della variabile, è preferibile utilizzare il `for`, che segue la seguente sintassi:

```
for( ''inizializzazione''; ''test''; ''incremento'' )
   istruzione
}
```

Si ricorda che l'istruzione può essere anche composta, ovvero costituita da tante istruzioni poste all'interno di parentesi graffe.

La somma dei primi 1000 interi può essere scritta allora nel seguente modo:

```
s = 0;
for ( i = 1; i <= 1000; i++ )
    s += i;
```

2.9 Le quattro operazioni aritmetiche

Negli esempi precedenti abbiamo visto come sia semplice impostare semplici operazioni aritmetiche, essendo gli operatori gli stessi che si incontrano nelle calcolatrici tascabili: `+`, `-`, `*` e `/`. Come si impara a scuola, e cose si è imparato usando calcolatrici tascabili,

bisogna fare attenzione alle priorità delle operazioni. Ad esempio, $2 + 4 * 5$ è uguale a 22 (e non a 30), in quanto viene prima eseguita la moltiplicazione e poi l'addizione. Come è anche noto, se si vuole cambiare la priorità delle operazioni o si è in dubbio sulle priorità standard, si possono usare le parentesi. L'esempio precedente può essere trasformato in $(2 + 4) * 5$, il cui risultato sarà 40.

Quanto detto per le *costanti* (i normali numeri) è anche vero per le variabili. Provare ad eseguire il codice seguente:

```
<script language=''javascript''>
var a = 2;
var b = 4;
var c = 5;
d = a + b * c
e = (a + b) * c
document.writeln(d);
document.writeln(e);
</script>
```

2.10 L'operatore aritmetico "modulo"

Un quinto operatore aritmetico, molto importante nella programmazione, è quello di *modulo*, indicato in matematica con mod e in molti linguaggi di programmazione con il simbolo %. Il risultato di a modulo b è il resto dell'operazione a/b, ad esempio 29 mod 5 = 4. Se il risultato dell'operazione è zero, vuol dire che a è un multiplo di b, ovvero b è divisore di a.

L'operatore di modulo ha la stessa priorità di moltiplicazioni e divisioni.

2.11 Altre operazioni matematiche ("oggetto" Math)

Ovviamente Javascript conosce le principali funzioni matematiche. Per esse non esistono simboli speciali, come $\sqrt{}$, ma *metodi* dell'*oggetto* `Math`. Per ora non entriamo in dettaglio del perché di questi nomi. Vediamo soltanto come funzionano, cominciando proprio dalla radice quadrata:

```
<script language=''javascript''>
var a = 49;
var b;
b = Math.sqrt(a);
document.writeln(b);
</script>
```

Detto alla buona, `Math` indica che si tratta di una operazione matematica applicata alla variabile (o costante) fra parentesi. `sqrt` indica che si tratta della radice quadrata (in inglese **sq**uare **r**oo**t**).

In Javascript si dice che `sqrt` è un *metodo* dell'*oggetto* `Math`. Fra oggetto e metodo c'è un punto. L'argomento degli oggetti sarà approfondito nel seguito e impareremo anche come costruirli. Per ora utilizziamo soltanto gli oggetti standard di Javascript come fossero delle "funzioni" (nel senso generico).

In genere dentro la parentesi c'è un solo *argomento*, ma esistono anche metodi che agiscono su più argomenti e altri che non hanno bisogno di argomenti sulle quali agire.

Ecco una lista di importanti metodi (indichiamo anche gli argomenti con i generici nomi x e y):

- `sqrt(x)`: radice quadrata;
- `abs(x)`: valore assoluto (il "numero senza il segno");
- `round(x)`: arrotonda all'intero più vicino;
- `ceil(x)`: se x non è intero viene arrotondato all'intero immediatamente successivo;
- `floor(x)`: se x non è intero viene arrotondato all'intero immediatamente precedente;
- `min(x, y)`: calcola il minimo fra x e y;
- `max(x, y)`: calcola il massimo fra x e y;
- `pow(x, y)`: calcola x^y;
- `exp(x)`: calcola e^x (ovviamente si ottiene lo stesso risultato se si usa `paw` con opportuni argomenti);
- `log(x)`: logaritmo naturale di x;
- `sin(x)`, `cos(x)`, `tan(x)`: omonime funzioni trigonometriche, con gli argomenti in radianti (attenzione!);
- `asin(x)`, `acos(x)`, `atan(x)`: omonime funzioni trigonometriche inverse (risultato in radianti);
- `random()`: restituisce un numero casuale che può assumere con pari probabilità ogni valore fra 0 e 1.

Oltre a questi metodi, l'oggetto `Math` è anche in grado di fornire delle costanti, le più interessanti delle quali sono pi greco (`Math.PI` = 3.14...), la costante e di Eulero (`Math.E` = 2.71...) e il suo logaritmo decimale (`Math.LOG10E` = 0.434...). Quest'ultimo permette, insieme a `Math.log(x)` di calcolare il logaritmo decimale di un numero, in quanto $\log_{10}(x)$ = `Math.LOG10E` $\cdot$ `Math.log(x)`.

2.12 Controllo del flusso mediante `if`

Uno dei casi che si verificano spesso nella programmazione, oltre a dover ripetere tante volte operazioni analoghe, è quello di fare diverse operazioni a seconda delle condizioni che si verificano in un certo istante. Ad esempio, se il mouse è posizionato in un certo punto dello schermo il computer esegue una certa operazione, se esso è clickato ne può eseguire un'altra, se è doppio cliccato un'altra ancora, e così via. Si dice che l'azione del computer dipende dalle variabili legate al cursore (posizione e stato dei tasti).

Facciamo un esempio di come far decidere al computer cosa fare. Riprendiamo l'esempio della somma dei primi n numeri. Supponiamo di voler sommare soltanto i numeri che

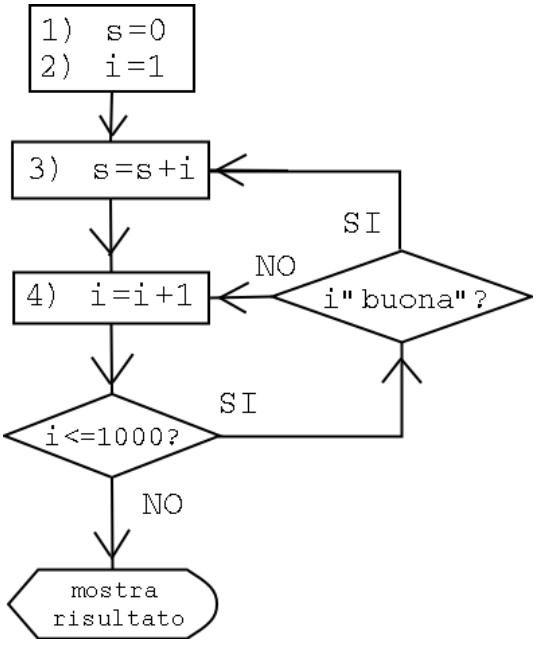

Figura 2.5: Diagramma di flusso del programma che somma i primi mille numeri interi che soddisfino una condizione speciale (numeri "buoni"), ad esempio siano divisibile per 5 e per 7 (vedi testo).

non siano divisibili né per 5 né per 7. Ora diventa un po' più complicato trovare una formula compatta. A meno che non si tratti di numeri molto grandi, per il quale il tempo di calcolo può diventare inaccettabilmente grande, o si voglia usare questa formula molte volte in programmi che devono girare velocemente, o non si abbiano particolari esigenze estetiche, il conto può essere effettuato modificando il codice del calcolo "bovino". Si ricorda che spesso non ha interesse la velocità del programma in sé, ma il tempo totale per sviluppare una procedura e applicarla. A meno di non avere particolari interessi matematici e/o esigenze culturali (e qualche forma di indipendenza economica...), non ha alcun senso spendere una settimana di lavoro per trovare una formula che permetta di fare in un millisecondo quello un algoritmo scritto in 20 minuti ci avrebbe impiegato 10 secondi, se questo era un conto "usa e getta".

Vediamo in figura 2.5 come si può cambiare il diagramma di flusso per soddisfare la condizione richiesta (indicata con "i buona" nella figura).

Come si è già capito le istruzioni somigliano molto ad un inglese molto telegrafico e con una sintassi rigorosa. Nel nostro caso si ha la semplice struttura

```
if (condizione)
istruzione
```

ove "istruzione" può anche una istruzione composta, come visto precedentemente.

La condizione "è divisibile per k" può essere scritta usando l'operazione di modulo e richiedendo che il risultato sia nullo:

```
i % k == 0
```

(si noti il doppio uguale, per non confondersi con il simbolo di assegnazione). La condizione complementare, "il numero i non è divisible per k", ovvero "i mod $k \neq 0$ si scrive in Javascript

```
i % k != 0
```

Siccome le due condizioni devono essere soddisfatte contemporaneamente dobbiamo unire le due condizioni con l'operatore logico AND, indicato con il simbolo "&&". Quindi nel nostro caso abbiamo il seguente `if`:

```
if ((i % 5 != 0) && (i %7 != 0))
  ''esegui somma''
```

La condizione congiunta è vera se entrambe sono vere. Inserendo questa condizione nel codice Javascript, inserendo 5 e 7 come parametri dell'algoritmo (variabili $d1$ e $d2$), abbiamo il seguente codice

```
<script language=''javascript''>
var s = 0;
var n = 1000;
var d1 = 5;
var d2 = 7;
for (i = 1; i <= 1000; i++)
  if ((i % d1 != 0 )&& (i % d2 != 0))
     s += i;
document.writeln(s);
</script>
```

in cui abbiamo utilizzato il `for` anziché il `do–while` e abbiamo cerscato di essere più sintetici del solito (per il risultato si veda Fig. 2.8). Si noti in particolare come, benché il `for` sia seguito da due linee di codice, non vendono usate le parentesi graffe. In realtà l'istruzione è una sola `if ...`, seppure su due linee. Anche se il *for* fosse stato preceduto da un `if` (ad esempio `if (n > 0)`), ugualmente non sarebbero servite le graffe. Comunque, in caso di dubbio, o per maggiore chiarezza, le graffe non fanno male.

La figura 2.6 mostra il diagramma di flusso dell'`if`. Questa istruzione può essere complementata da `else` ("altrimenti") per indicare le istruzioni alternative da eseguire se la condizione non è valida (vedi Fig. 2.7) Oppure si possono mettere più condizioni in cascata (con tanti `else if`), non necessariamente esclusive. Il programma esegue la prima condizione che si verifica. Quindi la struttura generale del controllo con `if` è la seguente:

```
if (condizione)
  istruzione 0
else if (condizione alternativa 1)
  istruzione 1
else if (condizione alternativa 2)
  istruzione 2
...
(altre condizioni)
...
else
  istruzione
```

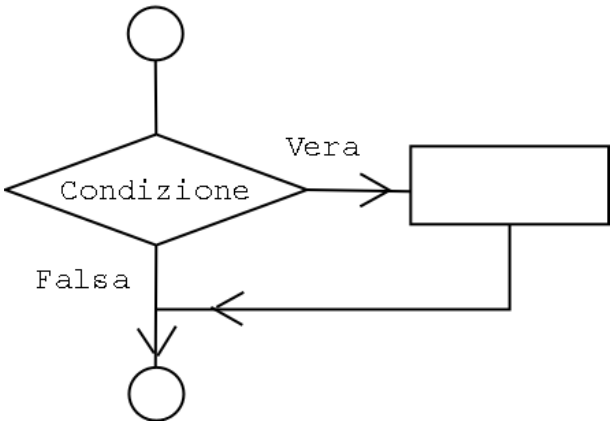

Figura 2.6: Diagramma di flusso della struttura di controllo `if`.

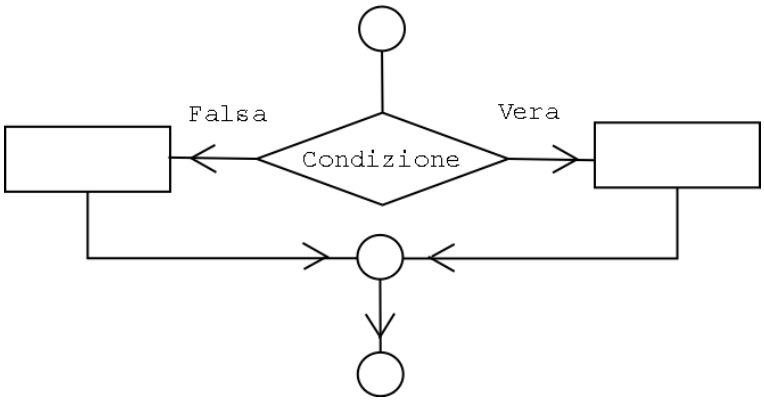

Figura 2.7: Diagramma di flusso della struttura di controllo `if–else`.

Ripetiamo ancora una volta che più istruzioni possono esswere raggruppate mediante parentesi graffe. Vedremo nel prossimo paragrafo un esempio di applicazione di `if`–`else`.

2.13 Salta giro (“`continue`”) e uscita di emergenza (”`break`”)

Come si poò facilmente immaginare, ci sono due situazioni spesso ricorrenti nei cicli `for`, `while` e `do`–`while`. Nella prima si può verificare all'interno del ciclo una condizione tale rendere inutile la continuazione dei cicli ciclo. Nella seconda non siamo più interessati a eseguire le istruzioni seguenti, ma vogliamo continuare con i cicli successivi. Le istruzioni che permettono dali scappatoie sono, rispettivamente, `break` e `continue`.

Facciamo un esempio, trasformando opportunamente il codice che somma i primi 1000 interi non divisibili né per 5 né per 7. Si capisce subito che l'`if` usato nel paragrafo precedente a tale scopo può essere sostituito da un *if* complementare nel quale, invece di eseguire le istruzioni, si passa al prossimo ciclo.:

```
if ( (i % 5 != 0) || (i % 7 != 0) ) continue;
```

Come uso del `break`, cambiamo lo scopo del programma. Invece di essere interessati alla somma, siamo interessati al numero tale per il quale la somma supera il valore di 100000.

```
<script language=''javascript''>
var s = 0;
var n = 1000;
var d1 = 5;
var d2 = 7;
for (i = 1; i <= 1000; i++) {
  if ((i % d1 = 0 ) || (i % d2 = 0))
     continue;
  s += i;
  if( s > 100000 ) break;
}
document.writeln(i);
document.writeln(s);
</script>
```

Nel primo if incontriamo per la prima volta l'operatore logico **OR**, scritto con il simbolo || nella maggior parte dei moderni linguaggi di programmazione. La condizione dell'`if` è vera se almeno una delle due sottocondizioni è verificata. L'esecuzione del programma è lasciata come esercizio. Il risultato è `s = 100128` per `i = 447`. Si noti come questi due valori vengono visualizzati uno di seguito all'altro, nonostante abbiamo due `document.writeln()` scritti su due righe diverse. Come si può immaginare, il problema è simile a quello incontrato per gli accapo dell'HTML e quindi avrà soluzione analoga. Ma per illustrarla dobbiamo capire un po' meglio cosa fa `document.writeln()`. Per ora ci possiamo accontentare.

2.14 `switch` – il commutatore

Per completezza mostriamo anche un costrutto in cui `break` è fondamentale e che può essere talvolta usato in alternativa a `if`–`else`.

```
switch (espressione) {
 case valore1:
     istruzione1
     break;
 case valore2:
     istruzione2
     break;
     break;
 ......
     ......
 default:
    altra istruzione
}
```

A seconda del valore dell'`espressione` è eseguito il `case` corrispondente. Se nessun `case` si verifica è eseguita l'istruzione di *default*. Ovviamente, l'istruzione può essere composta, ovvero, ripetiamo, costituita da più istruzioni unite da parentesi graffe. La differenza rispetto all'`if`–`else` è che i `case` di `switch` si riferiscono tutti alla stessa `espressione`, testata contro diversi possibili valori. È chiaro che è possibile scrivere un programma usando `if`–`else` che sia funzionalmente equivalente a quanto si può fare con `switch`:

```
if (espressione == valore1)
  istruzione1
else if (espressione == valore2)
  istruzione2
......
  ......
else
  altra istruzione
```

A cosa serve allora? Diciamo che il codice può essere più trasparente per talune applicazioni in quanto è chiaro a vista che si stanno eseguento delle scelte multiple legate al valore di un'unica espressione, mentre nel caso dell"`if`–`else` bisogna controllare gli `if` uno per uno.

Lo `switch` non verrà usato spesso nel seguito (o **pet niente**?)

2.15 Funzioni

Molto spesso succede di dover ripetere spesso in un programma una certa successione di istruzioni. Nasce quindi l'esigenza di dire al computer di ripetere quella sequenza, senza ripetere in ogni punto del programma la sequenza dei istruzioni. Tutti i linguaggi di programmazione permettono di scrivere e richiamare quelle che, a seconda dei linguaggi, vengono chiamate *procedure*, *sottoprogrammi* o *funzioni*.

In Javascript queste sequenze sono chiamate funzioni, vengono indicate con un nome e hanno tipicamente degli argomenti. Ad esempio, supponiamo che ci sia un qualche interesse, a parte quello didattico, di calcolare spesso la somma dei primi n numeri interi, con eccezione di quelli divisibili per due numeri che, come sopra, indichiamo con $d1$ e $d2$. La

funzione è ottenuta con piccole modifiche al codice precedente. Essenzialmente: aggiungiamo l'intestazione `function` con nome e argomenti dlla funzione; mettiamo l'istruzione *return* con la quale restituire il risultato al programma chiamante.

Per rendere la funzione più *robusta*, e come esercizio su `if`–`else`, mettiamo dei controlli sui parametri. Per fare questo, decidiamo che se essa è chiamata con n, $d1$ o $d2$ negativi o nulli, c'è qualcosa che non va. Se uno di questi casi si verifica, evitiamo di fare i conti e restituiamo semplicemente 0.

Le istruzioni per definire la funzione sono le seguenti

```
<script language=''javascript''>
function sumd(n, d1, d2) {
  var n, d1, d2, s;
  if ( (n > 0) && (d1 > 0) && (d2 > 0))
     for ( i = 1; i <= n; i++)
       if ((i %= d1 != 0 )&& (i % d2 != 0))
          s += i;
  else
     s = 0
  return s;
}
</script>
```

Le istruzioni per richiamare la funzione e che danno un risultato equivalente all'esempio del paragrafo precedente sono

```
<script language=''javascript''>
var risultato;
risultato = sumd(1000, 5, 7);
document.writeln(risultato);
</script>
```

Si noti come negli argomenti della chiamata non si dichiari esplicitamente quale sia il significato dei tre numeri 1000, 5 e 7. Quello che conta sono le posizione dei questi argomenti, che devono corrispondere a quelle della definizione della funzione: il primo argomento sta per n, il secondo per $d1$ e il terzo per $d2$. Quindi per usare correttamente le funzioni è importante conoscere l'ordine e il tipo dei parametri *in ingresso* e il significato della variabile restituita dalla funzione.

Una funzione non deve avere necessariamente degli argomenti, né essa deve restituire necessariamente un valore. Essa può semplicemente "svolgere delle funzioni" opportunamente definite dalle istruzioni stesse. Avremo modo di tornare su questi concetti nel seguito mediante esempi.

Come si può immaginare, ogni funzioni può richiamarne altre. In questo programma diventa ben strutturato e facilmente gestibile. Ad esempio quando diamo le istruzioni al computer "leggi il contenuto di un cdrom e travasane il contenuto nel masterizzatore", questa può essere vista come una funzione ad altro livello. Essa gestisce altre funzioni, ciascuna delle quali si occupa di una parte delle attività "subappaltando" altro "lavoro" ad altre funzioni. In principio, questi subappalti possono stratificarsi su molti livelli.

Si noti inoltre che il codice che chiama la funzione deve collocarsi all'interno del *body*, la funzione stessa può trovarsi anche all'interno dell'*header*.

La figura 2.8 mostra il codice completo e la visualizzazione del browsesr.

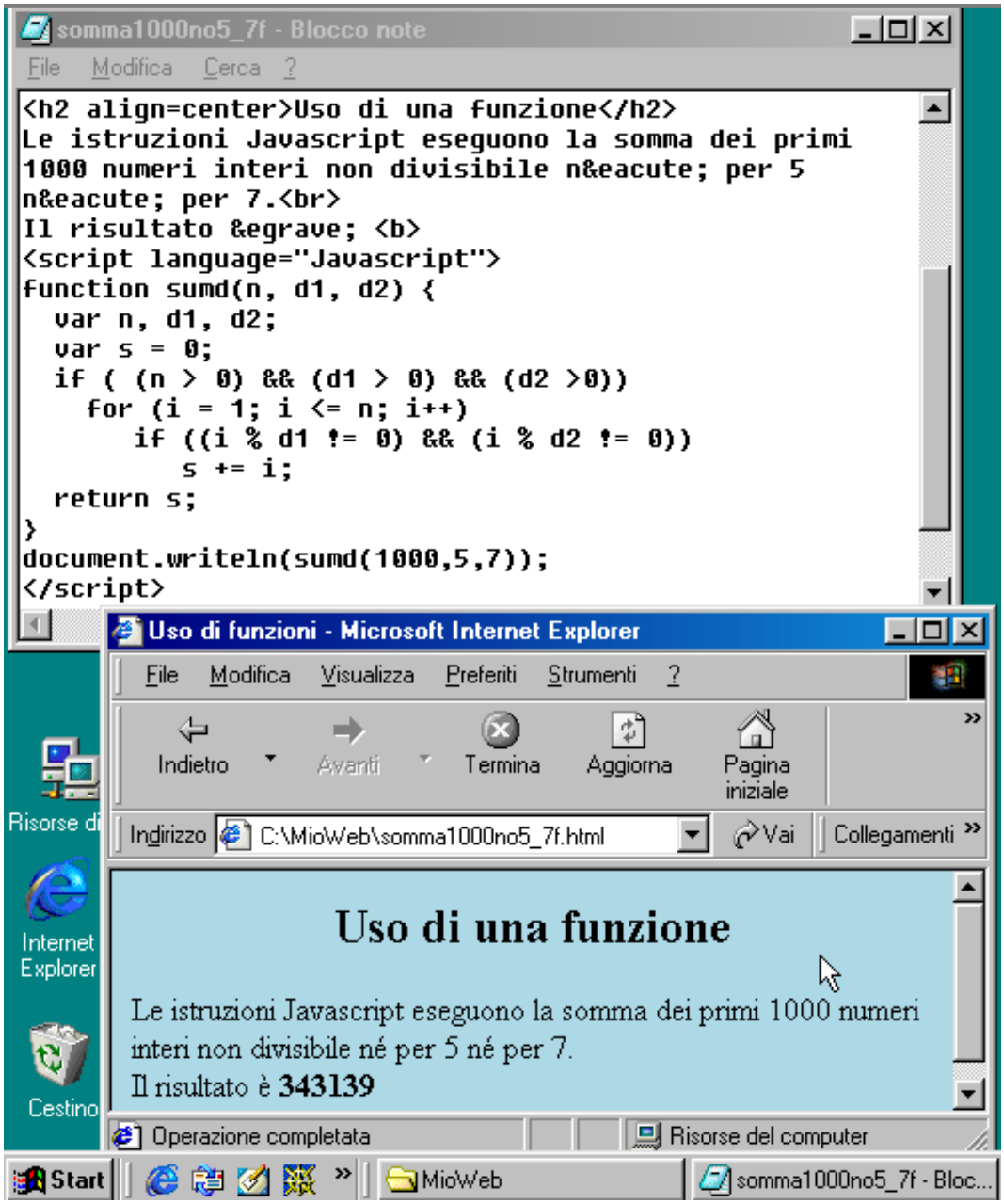

Figura 2.8: Variazione sul tema della somma dei mille interi, con uso di funzione.

2.16 Controllo del risultato

Terminiamo questo capitolo con dei commenti su come convincersi che i risultati dei programmi siano giusti. A rigor di logica, se tutte le istruzioni sono corrette anche il risultato è corretto. In pratica si possono verificare errori di battitura e di distrazione (diversi da quelli di battitura nel senso che si scambia una variabile per un'altra, ad esempio) e di cattiva interpretazione delle istruzioni.

Molti controlli sono immadiati: se si vuole che il fondo di una pagina sia blu e invece viene giallo, oppure se non ci sono degli accapo, o se non viene visualizzato niente, c'è chiaramente qualcosa che non va. Ma come facciamo ad essere sicuri che il risultato giusto dell'esempio precedente sia 343139? Nel caso della semplice somma dei primi mille interi avevamo un modo alternativo per valutare che il risultato dovesse essere $1000 \times 1001/2 = 500500$. Nei casi complicati, e la condizione di non divisibilità per 5 e per 7 rientra già a questa categoria, l'unico modo è di testare il programma in condizioni facilmente verificabili per altra via, oppure cercare di stimare almeno il risultato in modo approssimativo, o almeno come ordine di grandezza. Come esempio del primo metodo possiamo facilmente verificare che il risultato 33 per $n = 10$ è corretto, in quanto $(1+2+3+4+6+8+9)$. Come ordine di grandezza intendiamo il fatto che per $n = 1000$ ci aspettiamo qualcosa minore di 500500, ma non troppo (un risultato 12354 sarebbe senz'altro più che sospetto). Infine, come stima approssimativa, possiamo effettuare il seguente ragionamenento: rispetto alla semplice somma, mancano tutti i numeri divisibili per 5, che sono $1/5$ del totale, e tutti quelli divisibili per 7 ($1/7$ del totale). Quindi, ingenuamente dovremmo ridurre 500500 del 34.3%, ottenendo circa 329000. In realtà abbiamo contato due volte i numeri divisibili sia per 5 che per 7, che sono $1/35$ del totale. Quindi la percentuale da togliere è il 31.4% del totale, ottenendo 343200, in ottimo accordo con il risultato del programma. Quindi, tenendo conto che quest'ultimo è approssimato e che in caso di errore di programmazione ci saremmo aspettati risultati "qualsiasi", prendiamo senz'altro per buono 343139.

Per concludere questo argomento, vorrei invitare chi ci accinge a programmare a verificare sempre la ragionevolezza del risultato. Detto con una battuta, attribuita a vari fisici teorici, "non si dovrebbe nemmeno cominciare a fare un conto se non si sa più o meno quello che ci si aspetta".

2.17 Esercizi

(Le soluzioni si possono mettere su un sito web)

Glossario

Per ora collezioniamo le parole da descrivere

algoritmo ..

browser “sfogliatore” ..

client

compilatore ..

CPU ..

do

editor ..

hard disk (HD)

html ..

http ..

if

internet ..

interprete ..

link ..

loop

RAM

script .

server ..

tag .

web ..