Capitolo 48

PostScript: un linguaggio per la composizione finale

Benché il linguaggio PostScript sia nato per le stampanti, disponendo di Ghostscript potrebbe essere usato anche per scrivere direttamente. Lo scopo di questo capitolo è quello di introdurre all'uso diretto del linguaggio PostScript, in modo molto semplice, facendo riferimento prevalentemente al livello 1, eventualmente con qualche annotazione sul livello 2, mentre il livello 3 diventa troppo complesso e non più utile per un utilizzo diretto.

A ogni modo, gli esempi che si fanno sono stati verificati con Ghostscript e non con una vera stampante PostScript.

48.1 Impostazione generale

Un file PostScript è un file di testo, in cui le righe sono terminate indifferentemente con *<LF>* oppure con *<CR><LF>*, file che inizia con una riga simile al modello seguente:

```
%!PS-Adobe-livello_ps [EPSF-livello_eps]
```

Il livello è in pratica il numero di versione del linguaggio; per quanto riguarda il livello PostScript, si fa riferimento generalmente ai valori 1.0, 2.0 e 3.0. Il modello sintattico mostra la possibilità di aggiungere la stringa '**EPSF-*livello_eps***', con la quale si vuole specificare

che si tratta di un file PostScript incapsulato. In altri termini, un file PostScript normale inizia più o meno come nell'esempio seguente, dove si fa riferimento al livello 2:

```
%!PS-Adobe-2.0
…
```

In tal caso si intende lavorare su una serie di pagine; al contrario, se si sta realizzando una sola immagine nell'ambito di uno spazio determinato, si aggiunge la dichiarazione del tipo incapsulato:

```
%!PS-Adobe-2.0 EPSF-1.2
…
```

In generale, il simbolo di percentuale ('**%**') serve a introdurre dei commenti che non generano un risultato nella stampa; tuttavia, una sequenza di due simboli di percentuale ha un ruolo speciale per la dichiarazione di direttive importanti; inoltre, la stessa dichiarazione iniziale del tipo di file è preceduta da un simbolo percentuale. In generale, onde evitare equivoci, si indica un commento con un solo simbolo di percentuale seguito da almeno uno spazio:

```
% testo_commentato
```

Il commento può essere piazzato ovunque, tenendo presente che vale dal punto in cui appare, fino alla fine della riga.

Le direttive particolari che iniziano con due simboli di percentuale hanno la forma seguente:

```
%%direttiva [ : argomenti ]
```

In pratica, il nome delle direttive deve essere attaccato ai segni di percentuale; inoltre, se è prevista l'aggiunta di argomenti alla direttiva, dopo il nome della stessa appaiono due punti, seguiti da almeno uno spazio e dopo dagli argomenti previsti. Quello che segue è l'esempio di una struttura possibile per un file PostScript articolato su più pagine:

```
%!PS-Adobe-2.0
%%Creator: nome_del_redattore_del_file
%%DocumentPaperSizes: formato
%%EndComments
[%%BeginProlog
  prologo
%%EndProlog]
%%Page: numero_mostrato pagina_reale
istruzioni_ps
showpage
[%%Page: numero_mostrato pagina_reale
istruzioni_ps
showpage]…
%%Trailer
%%EOF
```

Il tutto dovrebbe essere abbastanza intuitivo: le prime istruzioni speciali, fino a '**%%EndComments**', descrivono il documento specificando in particolare le dimensioni della pagina; le istruzioni racchiuse tra '**%%BeginProlog**' e '**%%EndProlog**' possono servire per dichiarare delle funzioni utilizzate nel documento; le istruzioni relative a ogni singola pagina sono introdotte da '**%%Page: *m n***'; la visualizzazione della pagina, dopo la sua costruzione, si ottiene con l'istruzione '**showpage**'; il file termina con '**%%Trailer**' e '**%%EOF**'.

A questo punto, conviene vedere subito come si può articolare un file PostScript che non contiene pagine, ma una sola immagine:


```
%!PS-Adobe-2.0 EPSF-1.2
%%Creator: nome_del_redattore_del_file
%%BoundingBox: 0 0 larghezza_in_punti altezza_in_punti
%%EndComments
[%%BeginProlog
  prologo
%%EndProlog]
istruzioni_ps
showpage
%%Trailer
%%EOF
```

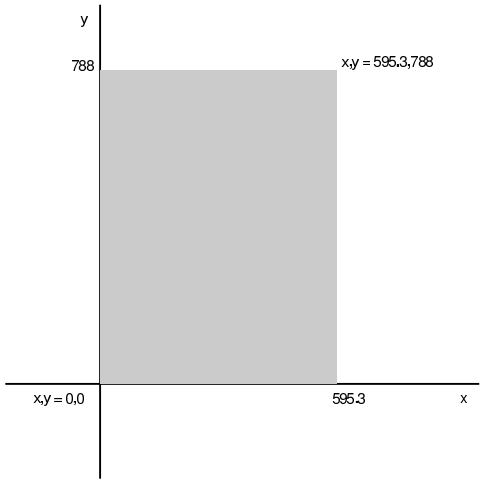
Si può osservare che la direttiva speciale '**%%BoundingBox**' va a sostituire '**%%DocumentPaperSizes**', allo scopo di indicare l'area in cui vanno rappresentate le istruzioni.

A ogni modo, la direttiva '**%%BoundingBox**' può essere usata anche per un file PostScript diviso in pagine, quando si vuole indicare un formato non standard, oppure se si vuole essere precisi.

48.1.1 Piano di lavoro e unità di misura

Il linguaggio PostScript è predisposto per fare riferimento a oggetti su un piano cartesiano ideale, in cui l'unità di misura normale è il punto tipografico, corrispondente secondo questo linguaggio a 1/72-esimo di pollice, pari a circa 0,35278 mm. In condizioni normali, la pagina è collocata sul piano cartesiano ideale come si vede nella figura 48.3, dove lo zero per ***x*** e ***y*** corrisponde esattamente con l'angolo inferiore sinistro; tuttavia, è consentita la definizione di pagine collocate in posizioni differenti, se questo può servire in qualche modo.

Figura 48.3. La pagina collocata su degli assi cartesiani ideali.



Gli oggetti grafici vengono disegnati sul piano cartesiano ideale; quello che risulta trovarsi sull'area della pagina può essere stampato.

48.1.2 Posizione corrente

Il PostScript può essere visto come un linguaggio per disegnare (tracciare curve, riempire delle aree e piazzare dei caratteri tipografici). Tutto questo avviene quasi sempre indicando delle coordinate, dove spesso la posizione di partenza ha importanza. Le coordinate iniziali si modificano con l'istruzione '**moveto**':

```
x y moveto
```

In pratica, si indicano due numeri, seguiti dalla parola chiave '**moveto**'. Il significato è molto semplice: il primo numero esprime la coordinata orizzontale (asse X), il secondo la coordinata verticale (asse Y). I valori si esprimono in punti tipografici.

48.1.3 Istruzioni speciali più importanti

Le istruzioni più importanti che iniziano con due segni di percentuale sono elencate brevemente nella tabella 48.4. Tuttavia, è il caso di aggiungere qualche piccola indicazione a proposito di alcune di queste.

Nessuna delle istruzioni che iniziano con due segni di percentuale è indispensabile; tuttavia alcune sono importanti. L'inserimento corretto di queste istruzioni rende il file PostScript più facile da gestire con gli strumenti comuni.

Tabella 48.4. Alcune istruzioni che iniziano con due segni di percentuale.

Istruzione	Descrizione
`%%Creator:` ***nome***	Il nome del programma che ha composto il file.
`%%DocumentPaperSizes:` ***formato***	Formato della carta.
`%%BoundingBox:` ***x_1 y_1 x_2 y_2***	Collocazione e dimensione della carta: da ***x_1***,***y_1*** a ***x_2***,***y_2***.
`%%Title:` ***titolo***	Titolo del documento.
`%%CreationDate:` ***data***	Data e ora di creazione del documento.
`%%Pages:` ***n***	Quantità di pagine contenuta.
`%%PageOrder: Ascend`|`Descend`	Ordine di apparizione delle pagine: ascendente o discendente.
`%%EndComments`	Fine dell'intestazione con le informazioni generali.
`%%BeginProlog`	Inizia un'area di definizione delle funzioni.
`%%EndProlog`	Termina l'area di definizione delle funzioni.
`%%BeginSetup`	Inizia un'area per l'inserimento di istruzioni di stampa.
`%%EndSetup`	Termina l'area delle istruzioni di stampa.
`%%Page:` ***x n***	Inizia la pagina ***n***-esima, rappresentata come ***x***.
`%%Trailer`	Conclude la serie delle pagine.
`%%EOF`	Conclude definitivamente il file.

L'istruzione '**`%%DocumentPaperSizes`**' serve intuitivamente per elencare le dimensioni possibili delle pagine. In generale si indica una sola parola chiave che esprime sinteticamente la dimensione della pagina, come si vede nella tabella 48.5. Probabilmente non conviene andare al di fuori di pochi standard; eventualmente è preferibile indicare le coordinate esatte attraverso l'istruzione '**`%%BoundingBox`**'.

Tabella 48.5. Formati di stampa comuni, indicabili come argomento dell'istruzione '**`%%DocumentPaperSizes`**'. Le dimensioni non sono necessariamente quelle reali, ma quelle conosciute dal linguaggio PostScript.

formato	larghezza punti	altezza punti	larghezza pollici	altezza pollici	larghezza centimetri	altezza centimetri
letter	612	792	8,50	11,00	21,59	27,94
legal	612	1008	8,50	14,00	21,59	35,56
a3	842	1190	11,6944	16,5278	29,7	42
a4	595	842	8,26389	11,6944	21	29,7
a5	421	595	5,84722	8,26389	14,85	21
b4	709	1002	9,84722	13,9167	25,0119	35,3483
b5	501	709	6,95833	9,84722	17,6742	25,0119

'**`%%BoundingBox`**' consente di indicare la posizione dell'angolo inferiore sinistro e di quello superiore destro della pagina. Di solito, le prime due coordinate che esprimono proprio la posizione dell'angolo inferiore sinistro, sono azzerate, a indicare che si parte dallo zero degli assi cartesiani ideali della superficie.

48.1.4 Aspetto delle istruzioni normali

Le istruzioni PostScript sembrano non avere inizio e fine, perché si possono collocare su una o più righe indifferentemente, senza alcun segno di separazione. Per esempio, si può scrivere:

```
newpath
 100   100 moveto
 100   431 lineto
 350   431 lineto
 350   100 lineto
closepath
```

Oppure, indifferentemente:

```
newpath 100 100 moveto 100 431 lineto 350 431 lineto ↩
↪350 100 lineto closepath
```

Naturalmente sono ammissibili tanti altri modi intermedi; comunque, è evidente che se si vuole scrivere del codice intelligibile occorre uno stile (come in tutti i linguaggi di programmazione).

La cosa che può apparire strana inizialmente è il fatto che i comandi che prevedono l'uso di argomenti, ricevono questi dati prima del nome del comando stesso. Per esempio, è già stata mostrata l'istruzione '**`moveto`**', la quale riceve l'indicazione delle coordinate prima del suo nome.

48.2 Linee e aree

La cosa più semplice che si può fare per cominciare a comprendere il linguaggio è quella di disegnare delle linee. In generale, il disegno avviene partendo dalle coordinate correnti, per cui questa indicazione non appare in modo esplicito, ma se necessario si definisce con uno spostamento attraverso l'istruzione '**`moveto`**'. Per le linee rette si possono usano le istruzioni '**`lineto`**' e '**`rlineto`**', dove la prima rappresenta un movimento con coordinate di destinazione assolute, mentre la seconda fa riferimento a coordinate di destinazione relative a quelle di partenza. Si osservino gli esempi seguenti, con cui si disegna lo stesso rettangolo largo 20 punti e alto 10 punti, a partire dalla coordinata ***x*** , ***y***=0 , 0:

```
  0    0 moveto
  0   10 lineto
 20   10 lineto
 20    0 lineto
  0    0 lineto
```

```
  0    0 moveto
  0   10 rlineto
 20    0 rlineto
  0  -10 rlineto
-20    0 rlineto
```

In pratica, l'istruzione '**`lineto`**' vuole l'indicazione del punto finale espresso come coordinata assoluta, mentre '**`rlineto`**' vuole una coordinata relativa alla posizione corrente.

Figura 48.10. Il rettangolo viene disegnato inserendo coordinate assolute, partendo dall'angolo inferiore sinistro e continuando con gli angoli successivi in senso orario.

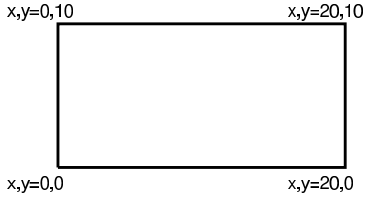
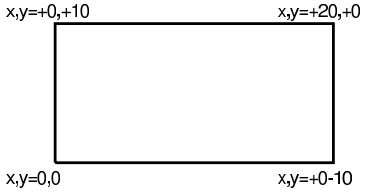


Figura 48.11. Il rettangolo viene disegnato inserendo coordinate relative.



Il disegno non viene tracciato se alla fine non si aggiunge un'istruzione '**stroke**', la quale non richiede argomenti. Dopo un'istruzione '**stroke**' viene perduto il riferimento alle coordinate correnti, per cui, se necessario, si deve ricominciare con un'istruzione '**moveto**'. Si osservi l'esempio seguente, in cui lo stesso rettangolo viene disegnato un segmento alla volta, riposizionando sempre le coordinate iniziali:

```
   0     0 moveto
   0    10 lineto stroke
   0    10 moveto
  20    10 lineto stroke
  20    10 moveto
  20     0 lineto stroke
  20     0 moveto
   0     0 lineto stroke
```

Naturalmente, se si preferisce questo modo di utilizzo dell'istruzione '**stroke**', si può anche cambiare un po' lo stile di scrittura:

```
   0     0 moveto    0    10 lineto stroke
   0    10 moveto   20    10 lineto stroke
  20    10 moveto   20     0 lineto stroke
  20     0 moveto    0     0 lineto stroke
```

Le linee, oltre alla collocazione, hanno due caratteristiche importanti: lo spessore e il colore. Lo spessore predefinito dovrebbe essere di un punto, mentre il colore predefinito è il nero. Si modifica lo spessore delle linee con l'istruzione '**setlinewidth**' e la colorazione (grigia) con l'istruzione '**setgray**'. Entrambi ricevono un solo argomento numerico: nel primo caso esprime lo spessore della linea e nel secondo rappresenta la luminosità, con un valore che va da zero a uno (zero rappresenta il nero e uno rappresenta il bianco).

Le istruzioni che alterano le caratteristiche delle linee, hanno effetto solo nel momento in cui appare l'istruzione '**stroke**'. In questo modo, si possono indicare le linee desiderate, quindi si possono cambiare le loro caratteristiche e infine si possono tracciare.

L'esempio seguente disegna lo stesso rettangolo già presentato, specificando un tratto di due punti tipografici di colore grigio (esattamente a metà tra il bianco e il nero). Si può osservare che le istruzioni '**setlinewidth**' e '**setgray**' sono state collocate subito prima dell'istruzione '**stroke**':

```
   0     0 moveto
   0    10 lineto
  20    10 lineto
  20     0 lineto
   0     0 lineto

         2 setlinewidth
         0 setgray
           stroke
```

Si possono tracciare delle linee per disegnare un poligono. L'esempio già visto rappresenta proprio un rettangolo, ma non è stato dichiarato esplicitamente il fatto che le linee devono congiungersi. Per farlo occorre dichiarare un percorso, con l'istruzione '**newpath**', che si conclude con '**closepath**'. Si osservi la variante seguente al disegno del rettangolo:


```
newpath
   0     0 moveto
   0    10 lineto
  20    10 lineto
  20     0 lineto
closepath

   2 setlinewidth
   0 setgray

stroke
```

In pratica, si tracciano le prime tre linee, mentre l'ultima viene indicata implicitamente con la richiesta di chiudere il percorso con l'istruzione '**closepath**'.

Il fatto di avere realizzato un poligono, consente di definire il colore di riempimento. In condizioni normali, quando non è stato fissato alcunché, il poligono è trasparente, mentre se si fissa un riempimento diventa opaco e il colore ricopre anche il bordo tracciato con le linee. Infatti, il bordo ha lo stesso colore fissato con l'istruzione '**setgray**', per cui tutto diventa dello stesso colore. Il colore di riempimento si definisce con l'istruzione '**fill**' e il colore usato è quello già fissato con l'istruzione '**setgray**'.

```
newpath
   0     0 moveto
   0    10 lineto
  20    10 lineto
  20     0 lineto
closepath

% 2   setlinewidth
   0.5 setgray
       fill
stroke
```

L'esempio mostra l'utilizzo dell'istruzione '**fill**' per colorare il rettangolo di grigio. Dal momento che lo spessore delle linee non serve più, l'istruzione relativa è stata commentata. Volendo mettere un bordo a questo rettangolo, occorre ridisegnarne sopra un altro trasparente, con il tratto desiderato:

```
newpath
   0     0 moveto
   0    10 lineto
  20    10 lineto
  20     0 lineto
closepath

   0.5 setgray
       fill
       stroke

newpath
   0     0 moveto
   0    10 lineto
  20    10 lineto
  20     0 lineto
closepath

   2   setlinewidth
   0   setgray
       stroke
```

Per terminare l'argomento sulle linee e sui poligoni, conviene mostrare un esempio completo che poi appare nella figura 48.19. Si osservi che il rettangolo viene disegnato da 1,1 a 21,11, utilizzando l'area da 0,0 a 22,12, la quale costituisce lo stretto indispensabile per dare lo spazio allo spessore della linea che è di due punti.

```
%!PS-Adobe-2.0 EPSF-1.2
%%Creator: Daniele Giacomini
%%BoundingBox: 0 0 22 12
%%EndComments

% Disegna un rettangolo

newpath
   1     1 moveto
   1    11 lineto
```

```
  21    11 lineto
  21     1 lineto
closepath

   0.5 setgray
       fill
       stroke

newpath
   1     1 moveto
   1    11 lineto
  21    11 lineto
  21     1 lineto
closepath

   2   setlinewidth
   0   setgray
       stroke

showpage

%%Trailer
%%EOF
```

Figura 48.19. Il rettangolo riempito e bordato. Il contorno è molto largo, ma è proporzionato rispetto al rettangolo che è alto solo 10 punti.

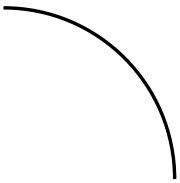
48.2.1 Archi e curve

Si può disegnare un arco o un cerchio completo con l'istruzione '**arc**'. In questo caso, non c'è bisogno di fare riferimento a una posizione corrente; anzi, è meglio eliminare tale informazione con un'istruzione '**stroke**' preventiva. L'istruzione '**arc**' richiede l'indicazione delle coordinate del centro del cerchio, la lunghezza del raggio, l'angolo di partenza e l'angolo di destinazione in direzione antioraria. L'esempio seguente disegna un arco con centro nella posizione ***x*** , ***y***=100 , 150, con raggio di 50 punti, da 0 a 90 gradi:

```
100 150 50 0 90 arc
```

In pratica, si tratta di quanto si vede nella figura 48.21.

Figura 48.21. Arco di cerchio disegnato da 0 a 90 gradi.



Se prima di disegnare il cerchio o l'arco di cerchio si tracciano altre linee, è conveniente chiudere i disegni precedenti con l'istruzione '**stroke**', per evitare di avere delle coordinate correnti attive nel momento in cui si usa l'istruzione '**arc**'. Diversamente, si otterrebbe una linea che collega le coordinate iniziali con il punto di partenza dell'arco disegnato.

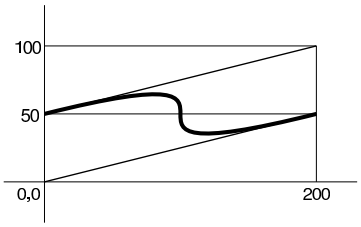
Per disegnare un cerchio completo, basta indicare l'intervallo di angoli da 0 a 360 gradi. Se si vuole riempire il cerchio, non è necessario utilizzare le istruzioni '**newpath**' e '**closepath**', perché si ottiene sempre un riempimento, anche quando il cerchio non è completo.

Il disegno di una curva è invece più complicato: si usa l'istruzione '**curveto**', ma oltre alle coordinate di destinazione, bisogna indicare la tangente del punto di inizio e del punto di destinazione. Prima di dare altre spiegazioni, conviene partire da un esempio visivo, come si vede nella figura 48.23. La curva da prendere in considerazione è rappresentata con un tratto più scuro. Si possono vedere due linee oblique, che partono rispettivamente dal punto di inizio e dal punto di arrivo della curva: si tratta delle tangenti che stabiliscono la

curvatura di partenza e di arrivo della linea disegnata. L'istruzione necessaria a disegnare questa curva è la seguente:

```
    0   50  moveto
  200  100    0    0  200   50  curveto
```

Figura 48.23. Esempio di una curva in cui sono evidenti le tangenti.



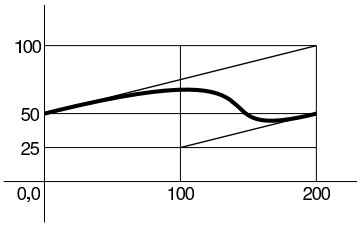
In pratica:

- le coordinate di partenza sono ***x*** , ***y***=0 , 50;
- la prima tangente è la linea che va da ***x*** , ***y***=0 , 50 (le coordinate di partenza) a ***x*** , ***y***=200 , 100;
- la seconda tangente è la linea che va da ***x*** , ***y***=200 , 50 (le coordinate di arrivo) a ***x*** , ***y***=0 , 0;
- le coordinate di arrivo sono ***x*** , ***y***=200 , 50.

Naturalmente, la lunghezza delle linee indicate come tangenti rende più o meno importante la curvatura relativa. Si osservi, nella figura 48.25, come si trasforma il disegno se si accorcia la linea tangente di arrivo come nell'esempio seguente:

```
    0   50  moveto
  200  100    100    25  200   50  curveto
```

Figura 48.25. Esempio di una curva in cui sono evidenti le tangenti.



Il modello sintattico per l'utilizzo dell'istruzione '**curveto**' è quindi il seguente:

```
x_tangente_inizio  y_tangente_inizio  x_tangente_fine  y_tangente_fine ↩
↪x_fine_curva  y_fine_curva  curveto
```

48.3 Salvare e recuperare le impostazioni grafiche

Le impostazioni grafiche, come quelle che si possono fissare con le istruzioni '**setlinewidth**' e '**setgray**', possono essere salvate e recuperate da una pila apposita. Si utilizza l'istruzione '**gsave**' per salvare l'impostazione corrente e '**grestore**' per recuperare le ultime impostazioni salvate. Si osservi l'esempio seguente:

```
1 setlinewidth
gsave
2 setlinewidth
gsave
4 setlinewidth
 50 120  moveto
200 120  lineto
stroke
grestore
 50 125  moveto
200 125  lineto
stroke
grestore
 50 130  moveto
200 130  lineto
stroke
```

Vengono accumulati tre spessori differenti per le linee, quindi si procede disegnando tre linee, dopo ognuna delle quali viene recuperata l'ultima impostazione grafica. In pratica, la linea da 50,120 a 200,120 viene disegnata con un tratto di quattro punti; la linea da 50,125 a 200,125 viene disegnata con un tratto di tre punti; la linea da 50,130 a 200,130 viene disegnata con un tratto di un punto di spessore.

Quando si disegna qualcosa, può essere opportuno racchiudere le modifiche alle caratteristiche entro una coppia '**gsave**'-'**grestore**', onde evitare di coinvolgere le impostazioni precedenti, che potrebbero riguardare il resto del file:

```
gsave
4 setlinewidth
 50 120  moveto
200 120  lineto
stroke
grestore
```

Nell'esempio che appare sopra, si inizia salvando le impostazioni e impostando uno spessore di quattro punti. Viene quindi indicata la linea e fissata con l'istruzione '**stroke**'. Al termine si recuperano le impostazioni grafiche. Nello stesso modo, si potrebbero salvare e modificare le impostazioni grafiche subito prima dell'istruzione '**stroke**':

```
 50 120  moveto
200 120  lineto
gsave
4 setlinewidth
stroke
grestore
```

Attraverso la tecnica del salvataggio e del recupero delle caratteristiche grafiche, è possibile disegnare un poligono bordato senza dover ripetere due volte il tratto del contorno. In pratica, il comando di riempimento viene dato entro un ambiente protetto da '**gsave**' e '**grestore**', come si vede nell'esempio seguente:

```
%!PS-Adobe-2.0 EPSF-1.2
%%Creator: Daniele Giacomini
%%BoundingBox: 0 0 22 12
%%EndComments

% Disegna un rettangolo

newpath
   1     1 moveto
   1    11 lineto
  21    11 lineto
  21     1 lineto
closepath

gsave
   0.5 setgray
       fill
grestore
   2   setlinewidth
   0   setgray
       stroke

showpage

%%Trailer
```



```
%%EOF
```

Il risultato è identico a quanto già visto nella figura 48.19.

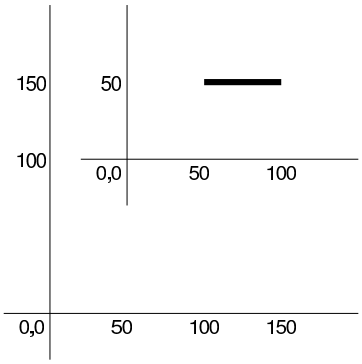
48.4 Spostamento del piano cartesiano e modifica della scala

È possibile spostare la posizione della superficie prima di dare altre istruzioni di scrittura di qualunque tipo. Si tratta in particolare delle istruzioni '**translate**' e di '**rotate**'. La prima di queste due sposta le coordinate 0 , 0 in una posizione nuova, prendendo come riferimento le coordinate precedenti; la seconda ruota gli assi attorno alle coordinate 0 , 0. Per esempio, con le istruzioni seguenti si ottiene una linea orizzontale dalla posizione 100 , 150 alla posizione 150 , 150:

```
 50 100 translate
 50  50 moveto
100  50 lineto
stroke
```

Infatti, l'istruzione '**translate**' sposta le coordinate 0 , 0 verso 50 , 100, secondo la collocazione precedente. La figura 48.31 mostra la collocazione iniziale e lo spostamento; la linea disegnata è quella che appare con tratto più scuro.

Figura 48.31. Esempio riferito a uno spostamento degli assi.

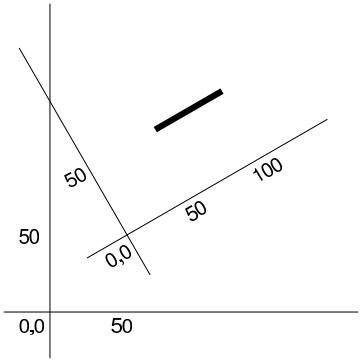


L'istruzione '**rotate**' fa ruotare il piano cartesiano sul centro delle coordinate 0 , 0. La figura 48.33 mostra cosa accade se si sposta lo zero nella posizione 0 , 0 e poi si ruota di 30 gradi, con le istruzioni seguenti:

```
   50   50  translate
   30  rotate

   50   50  moveto
  100   50  lineto
stroke
```

Figura 48.33. Esempio riferito a uno spostamento e a una rotazione degli assi.



Lo spostamento e la rotazione del piano sono informazioni che possono essere salvate e recuperate con le istruzioni '**gsave**' e '**grestore**'. Pertanto, prima di uno spostamento o di una rotazio-

ne, conviene salvare la situazione, per recuperarla quando questi cambiamenti non servono più.

```
gsave
   50   50  translate
   30  rotate

   50   50  moveto
  100   50  lineto
stroke
grestore
```

Oltre allo spostamento e alla rotazione del piano, si può modificare la scala, con l'istruzione '**scale**'. Gli argomenti dell'istruzione sono due valori che esprimono il rapporto nei confronti dell'asse X e nei confronti dell'asse Y. Dopo la modifica della scala, le coordinate 0 , 0 rimangono centrate sulla stessa posizione iniziale. L'esempio seguente serve a fare in modo che la scala dell'asse Y risulti schiacciata alla metà del valore precedente, mentre l'asse X non viene modificato:

```
1 0.5 scale
```

Anche le alterazioni della scala possono essere recuperate da un'istruzione '**grestore**'.

48.5 Ripetizione

Un gruppo di istruzioni, racchiuso tra parentesi graffe, può essere ripetuto più volte con l'istruzione '**repeat**':

```
n {istruzioni} repeat
```

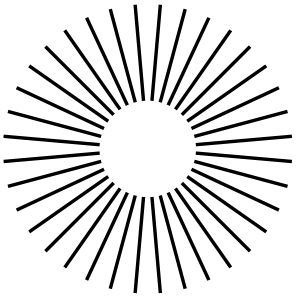
Evidentemente, perché ciò abbia un senso, è necessario che le istruzioni da ripetere creino ogni volta un cambiamento, come una rotazione o uno spostamento di assi. Per esempio,

```
36 {10 rotate 10 10 moveto 30 30 lineto} repeat
stroke
```

Disegna una stellina attorno alle coordinate 0 , 0, con un diametro di 80 punti. In questo caso, la rotazione riporta alla fine gli assi nella posizione di partenza, ma in generale conviene salvare e ripristinare la situazione:

```
gsave
36 {10 rotate 10 10 moveto 30 30 lineto} repeat
stroke
grestore
```

Figura 48.38. Esempio di un disegno ottenuto con la rotazione.



48.6 Testo

Per poter scrivere sulla superficie del foglio, è necessario selezionare il tipo di carattere e la dimensione prima di tutto. Questo si ottiene con alcune istruzioni che conviene usare assieme:

```
/carattere findfont dimensione scalefont setfont
```

Per esempio, per usare il carattere Helvetica alto 12 punti, si usa l'istruzione seguente:

```
/Helvetica findfont 12 scalefont setfont
```

La scrittura vera e propria avviene con l'istruzione '**show**', con la quale si colloca il testo a partire dalle coordinate correnti, per cui si fa precedere normalmente da un'istruzione '**moveto**':


```
(testo) show
```

Come si vede dal modello sintattico, il testo deve essere scritto tra parentesi tonde e collocato prima della parola chiave '**show**'.

Dovendo usare le parentesi tonde per delimitare il testo da visualizzare, deve esserci un modo per poter togliere a queste il valore sintattico normale, quando c'è la necessità di rappresentarle nel testo. In pratica si usano le sequenze '**\(**' e '**\)**' per rappresentarle; inoltre, dal momento che la barra obliqua inversa ha un significato speciale, per rappresentare questa si usa la sequenza '\\'. La tabella 48.40 riepiloga le sequenze speciali più importanti per il testo delimitato tra parentesi tonde.

Tabella 48.40. Alcune sequenze speciali per la scrittura di testo delimitato da parentesi tonde, con l'istruzione '**show**'.

Sequenza	Descrizione
\\	Barra obliqua inversa.
\(	Parentesi tonda aperta.
\)	Parentesi tonda chiusa.
nnn	Simbolo corrispondente al numero, in ottale, secondo la codifica attuale.

Se si seleziona un carattere non disponibile, viene utilizzato il Courier. La tabella 48.41 elenca i nomi standard dei tipi di carattere fondamentali che si possono utilizzare con il linguaggio PostScript.

Tabella 48.41. Nomi dei tipi di carattere comuni che possono essere utilizzati con il linguaggio PostScript.

Nome	Descrizione
Times Times-Roman	Times normale.
Times-Italic	Times corsivo.
Times-Bold	Times neretto.
Times-BoldItalic	Times neretto inclinato.
Helvetica	Helvetica normale.
Helvetica-Oblique	Helvetica inclinato.
Helvetica-Bold	Helvetica neretto.
Helvetica-BoldOblique	Helvetica neretto inclinato.
Courier	Courier normale.
Courier-Oblique	Courier inclinato.
Courier-Bold	Courier neretto.
Courier-BoldOblique	Courier neretto inclinato.
Symbol	Symbol.

Un'istruzione simile a '**show**', si occupa di visualizzare il testo, controllandone lo spostamento in corrispondenza di un simbolo particolare:

```
x y n (testo) widthshow
```

L'istruzione '**widthshow**' serve a fissare uno spostamento orizzontale (***x***) e uno spostamento verticale (***y***), subito dopo il simbolo corrispondente al numero ***n***. In pratica, se alla fine di ogni spazio si vuole aggiungere uno spazio orizzontale di due punti, si usa la forma seguente:

```
2 0 32 (testo) widthshow
```

Questa è anche la situazione tipica, in cui si vuole allargare lo spazio tra le parole, per adattare la riga scritta alla larghezza disponibile. Naturalmente, lo spazio normale può anche essere ridotto, se lo si desidera, utilizzando valori orizzontali negativi, come nell'esempio seguente:

```
-1 0 32 (testo) widthshow
```

Se si attribuisce un valore anche al secondo argomento numerico, si ottiene uno spostamento verticale, come nell'esempio seguente, in cui ogni parola viene alzata di due punti rispetto alla precedente:

```
0 2 32 (testo) widthshow
```

48.7 Salvataggio e recupero delle impostazioni della pagina nel complesso

Quando si scrive un documento composto da diverse pagine, diventa utile la possibilità di recuperare le impostazioni precedenti, prima di passare alla pagina successiva. È già stata presentata la coppia di istruzioni '**gsave**' e '**grestore**', specifica per le impostazioni grafiche. Per tutto, si può usare invece la coppia '**save**' e '**restore**'. Di solito si inizia una pagina con '**save**' e si conclude con '**restore**', in modo da garantire il recupero di tutto, senza dimenticare qualcosa.

```
%!PS-Adobe-2.0
%%Creator: nome_del_redattore_del_file
%%DocumentPaperSizes: formato
%%EndComments
[%%BeginProlog
  prologo
%%EndProlog]
%%Page: numero_mostrato pagina_reale
save
istruzioni_ps
showpage
restore
[%%Page: numero_mostrato pagina_reale
save
istruzioni_ps
showpage
restore]…
%%Trailer
%%EOF
```

Le istruzioni '**save**' e '**restore**' dovrebbero essere sempre annidate correttamente, nel senso che ogni istruzione '**restore**' va a recuperare l'ultima istruzione '**save**' che non sia già stata presa in considerazione da un altro '**restore**'. Se viene eseguito un '**restore**' che non risulta abbinato a un'istruzione '**save**', si genera un errore irreversibile.

Per ovviare all'inconveniente di dover seguire attentamente l'uso delle istruzioni '**save**' e '**restore**', si può attribuire un «nome» a un'istruzione '**save**', richiamando lo stesso nome nel momento del '**restore**':

```
/nome save def
…
…
…
nome restore
```

La cosa non è molto intuitiva, ma funziona così: il nome che viene dichiarato nel momento dell'uso dell'istruzione '**save**', viene posto davanti all'istruzione '**restore**'. Se nello spazio tra queste due istruzioni appaiono altre istruzioni '**save**', queste sono tutte annullate. Segue un esempio:


```
/Mia_Configurazione_Predefinita save def
…
…
…
Mia_Configurazione_Predefinita restore
```

Tabella 48.46. Tabella riassuntiva delle istruzioni più semplici del linguaggio PostScript.

Istruzione	Descrizione
x y moveto	Cambia le coordinate correnti senza disegnare.
x y rmoveto	Cambia le coordinate correnti in modo relativo.
x y lineto	Traccia una linea fino alle coordinate assolute indicate.
x y rlineto	Traccia una linea fino alle coordinate relative indicate.
x y m n o arc	Arco con centro in ***x*** , ***y***, raggio ***m***, da ***n*** a ***o*** gradi, in senso antiorario.
x y m n o arcn	Arco con centro in ***x*** , ***y***, raggio ***m***, da ***n*** a ***o*** gradi, in senso orario.
x y x y x y curveto	Curva indicando le coordinate di arrivo delle tangenti e infine della curva.
x y translate	Fa sì che ***x*** , ***y*** corrispondano alle nuove coordinate 0 , 0.
n rotate	Ruota di ***n*** il fondo, con centro sulle coordinate 0 , 0.
n {*istruzioni*} repeat	Ripete ***n*** volte le istruzioni tra parentesi graffe.
n setlinewidth	Spessore delle linee.
n setgray	Colorazione grigia: 0=nero; 1=bianco.
fill	Riempie i poligoni e le aree racchiuse entro le curve.
newpath	Inizia a disegnare un oggetto nuovo.
closepath	Unisce l'ultimo punto disegnato con il punto di partenza.
stroke	Fissa le linee tracciate.
gsave	Accumula le impostazioni grafiche.
grestore	Recupera le impostazioni.
save	Accumula tutte le impostazioni della pagina.
restore	Recupera le impostazioni della pagina.
/*nome* save def	Accumula tutte le impostazioni dichiarando un nome.
nome restore	Recupera le impostazioni riferite a quel nome.
/*nome* findfont *dim* scalefont setfont	Seleziona il carattere e la dimensione indicata.
(*testo*) show	Scrive il testo indicato usando il carattere già stabilito.
x y n (*testo*) widthshow	Scrive il testo indicato con uno spostamento ***x*** , ***y*** alla fine del simbolo ***n***.
x 0 32 (*testo*) widthshow	Scrive il testo indicato con l'aggiunta di uno spazio di ***x*** tra le parole.

48.8 La pila

Per poter scrivere codice PostScript un po' più complesso, diventa necessario l'utilizzo di istruzioni che realizzano delle espressioni, fino ad arrivare alla costruzione di funzioni (procedure) che possono essere richiamate successivamente. Purtroppo, le espressioni realizzate con questo linguaggio, diventano un po' complicate da leggere. Infatti, queste funzioni ricevono i loro argomenti prelevandoli da una pila (*stack*) ed emettono risultati inserendoli nella stessa pila.

```
dato… funzione
```

Osservando il modello, le informazioni che non sono riconducibili a nomi di funzione, vengono inserite in questa pila, che poi la prima funzione inizia a leggere. Si osservi l'istruzione seguente:

```
1 2 3 4 5 6 7 8 9 moveto lineto
```

I valori da uno a nove, vengono inseriti così come sono nella pila, poi ogni funzione preleva dalla pila la quantità di argomenti che la riguarda. In questo caso, '**moveto**' preleva gli ultimi due valori a essere stati inseriti, precisamente la coppia otto e nove, spostando le coordinate correnti in 8 , 9; successivamente è il turno di '**lineto**', la quale preleva altri due valori, precisamente il sei e il sette, tracciando una linea fino al punto 6 , 7. Pertanto, tutto è come se fosse scritto nel modo seguente:

```
8 9 moveto 6 7 lineto
```

Tuttavia, rimangono ancora altri valori nella pila, per altre funzioni successive, ammesso che vogliano usarli, perché se si inseriscono altri valori, questi ultimi vengono poi estratti per primi.

Dato questo meccanismo, diventano importanti alcune funzioni che consentono di intervenire su questa pila: '**clear**' svuota completamente la pila; '**pop**' preleva ed elimina l'ultimo valore inserito. A fianco di '**pop**' si potrebbe immaginare la presenza di una funzione con il nome *push*, ma in questo caso non serve, perché l'azione di inserimento nella pila avviene in modo implicito.

Sono un po' più difficili da comprendere le funzioni '**exch**' e '**roll**'. La prima scambia l'ordine degli ultimi due valori inseriti nella pila; la seconda esegue uno scorrimento, verso sinistra o verso destra di una certa quantità di questi valori:

```
n_elementi_da_scorrere scorrimento roll
```

Per esempio, se nella pila ci fossero i valori 1, 2, 3, 4, 5, 6 e 7, in questo ordine, per cui il primo a essere prelevato sarebbe il numero 7, l'istruzione '**3 2 roll**' trasformerebbe questa sequenza in 1, 2, 3, 4, 6, 7 e 5; al contrario, l'istruzione '**3 -2 roll**' trasformerebbe questa sequenza in 1, 2, 3, 4, 7, 5 e 6. In pratica, il primo valore indica quanti elementi prendere in considerazione, a partire dall'ultimo, mentre il secondo indica quante volte scorrere e in quale direzione.

Figura 48.49. Esempio di funzionamento dell'istruzione '**roll**', con uno scorrimento verso destra.

```
pila iniziale: 1 2 3 4 5 6 7

    3 2 roll          7 5 6  (primo scorrimento verso destra)
                      6 7 5  (secondo scorrimento verso destra)
pila finale:   1 2 3 4 6 7 5
```

Figura 48.50. Esempio di funzionamento dell'istruzione '**roll**', con uno scorrimento verso sinistra.

```
pila iniziale: 1 2 3 4 5 6 7

    3 -2 roll         6 7 5  (primo scorrimento verso sinistra)
                      7 5 6  (secondo scorrimento verso sinistra)
pila finale:   1 2 3 4 7 5 6
```

Quando una funzione restituisce un valore, lo fa inserendolo implicitamente nella pila. In questo modo, l'assegnamento a una variabile, così come si è abituati nei linguaggi di programmazione comuni, non c'è. Al massimo si definisce una funzione che restituisce un valore, inserendolo nella pila.

Anche le funzioni '**exch**' e '**roll**' prelevano dei valori dalla pila e poi ne inseriscono degli altri nella stessa. In pratica, '**exch**' preleva due valori, li scambia e li inserisce nella pila; '**roll**' preleva due valori, li interpreta, quindi preleva un gruppo di altri valori, li fa scorrere in un verso o nell'altro, quindi li inserisce nuovamente nella pila.

A questo punto si può cominciare a comprendere che i dati inseriti nella pila, quando ciò non avviene per mezzo di una funzione che restituisce qualcosa, devono avere una rappresentazione formale. Può

trattarsi di: valori numerici, che si scrivono come sono, utilizzando il punto per separare la parte decimale; stringhe, che sono delimitate da parentesi tonde e possono contenere delle sequenze di escape; espressioni, che sono delimitate tra parentesi graffe (si ricordi il caso della funzione '**repeat**'). I valori logici, *Vero* e *Falso*, non hanno una rappresentazione particolare e si indicano espressamente solo attraverso le funzioni '**true**' e '**false**'.

Tabella 48.51. Rappresentazione dei dati e gestione della pila.

Istruzione	Descrizione
intero[. *decimale*]	Inserisce il valore numerico nella pila.
(*stringa*)	Inserisce la stringa nella pila.
{ *espressione* }	Inserisce le istruzioni nella pila.
`clear`	Svuota la pila.
oggetto `pop`	Preleva dalla pila l'ultimo valore inserito.
oggetto_1 *oggetto_2* `exch`	Scambia gli ultimi due valori nella pila.
m *n* `roll`	Fa scorrere gli ultimi *m* elementi della pila di *n* posizioni verso destra.
m `-`*n* `roll`	Fa scorrere gli ultimi *m* elementi della pila di *n* posizioni verso sinistra.
oggetto `dup`	Preleva l'ultimo valore e ne inserisce due copie nella pila.

48.9 Funzioni comuni

Alcune funzioni operano su valori numerici, restituendo un risultato che, secondo la logica del linguaggio PostScript, viene inserito nella pila. Per esempio, la funzione '**add**' riceve due valori restituendo la somma di questi:

```
10 20 add 40 moveto
```

In questo caso, vengono sommati i valori 10 e 20, inserendo nella pila il valore 30. Così, si ottiene lo spostamento nelle coordinate 30 , 40, attraverso la funzione '**moveto**'.

I valori logici, come accennato, si indicano attraverso le funzioni '**true**' e '**false**', che si limitano rispettivamente a inserire nella pila il valore corrispondente. Possono generare risultati logici anche alcune funzioni di confronto e i valori logici possono essere rielaborati attraverso funzioni booleane. Infine, in base a un valore logico è possibile eseguire o meno un gruppo di espressioni. Si osservino gli esempi seguenti.

`10 20 lt`	La funzione '**lt**' confronta due valori e restituisce *Vero* se il primo (secondo la lettura umana) è minore. In questo caso, viene restituito *Vero*.
`10 20 lt 45 34 gt and`	La funzione '**and**' restituisce *Vero* se riceve due valori *Vero* simultaneamente. In questo caso, la funzione '**lt**' inserisce il valore *Vero* nella pila e anche la funzione '**gt**' inserisce un altro valore *Vero*, dal momento che il confrontando i valori 45 e 34 si vede che il primo è maggiore del secondo.
`10 20 lt {45 50 moveto} if`	La funzione '**if**' preleva un valore logico e un gruppo di istruzioni. Se il valore logico è *Vero* viene eseguito il raggruppamento di istruzioni. In questo caso, dato che la funzione '**lt**' inserisce il valore *Vero* nella pila, così può essere eseguito lo spostamento nelle coordinate 45 , 50.

`10 20 lt {45 50 moveto} ↩ ↪{50 45 moveto} ifelse`	La funzione '**`ifelse`**' preleva un valore logico e due gruppi di istruzioni. Se il valore logico è *Vero* viene eseguito il primo gruppo di istruzioni, altrimenti viene eseguito il secondo. In questo caso, dato che la funzione '**`lt`**' inserisce il valore *Vero* nella pila, così viene eseguito lo spostamento nelle coordinate 45 , 50.

Queste funzioni vengono descritte brevemente nella tabella 48.54.

Tabella 48.54. Espressioni matematiche, logiche e condizionali.

Istruzione	Descrizione
n `neg`	Inverte il segno del valore.
m ***n*** `add`	Somma i due valori.
m ***n*** `sub`	Sottrae ***n*** da ***m***.
m ***n*** `mul`	Moltiplica i valori.
m ***n*** `div`	Divide ***m*** per ***n***.
m ***n*** `mod`	Il resto della divisione intera di ***m*** per ***n***.
n `round`	Arrotonda ***n***.
n `abs`	Calcola il valore assoluto di ***n***.
n `sin`	Calcola il seno di ***n***.
n `cos`	Calcola il coseno di ***n***.
m ***n*** `min`	Restituisce il minimo tra due valori.
m ***n*** `max`	Restituisce il massimo tra due valori.
`true`	*Vero*.
`false`	*Falso*.
m ***n*** `gt`	*Vero* se ***m*** è maggiore di ***n***.
m ***n*** `ge`	*Vero* se ***m*** è maggiore o uguale a ***n***.
m ***n*** `lt`	*Vero* se ***m*** è minore di ***n***.
m ***n*** `le`	*Vero* se ***m*** è minore o uguale a ***n***.
m ***n*** `eq`	*Vero* se i valori sono uguali.
m ***n*** `ne`	*Vero* se i valori sono diversi.
bool `{`***istruzioni***`} if`	Esegue le istruzioni se il valore logico è *Vero*.
bool `{`***istr_1***`} {`***istr_2***`} ifelse`	Esegue il primo o il secondo gruppo di istruzioni in base al valore logico.

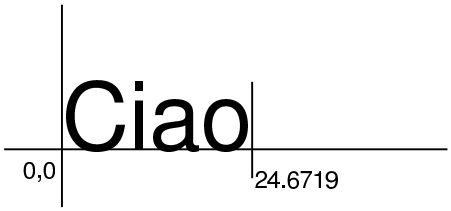
48.10 Operazioni sulle stringhe

A causa della struttura del linguaggio, la gestione delle stringhe non è affatto intuitiva: bisogna tradurre tutto nell'ottica della pila. Tanto per cominciare, la cosa più semplice che si può fare con una stringa è misurarne la lunghezza con l'aiuto della funzione '**`stringwidth`**'. Per la precisione, si tratta di determinare la posizione finale di una stringa collocata a partire dalle coordinate 0 , 0:

```
stringa stringwidth
```

Se si osserva la figura 48.55, si può vedere la stringa composta dalla parola «Ciao», scritta con il carattere Helvetica, avente un corpo di 12 punti. Come si vede, la sua lunghezza è di 24,672 punti.

Figura 48.55. Lunghezza di una stringa.



Quando c'è la necessità di convertire un valore in una stringa, si pone il problema dell'allocazione di memoria per la stringa stessa. Per esempio, la funzione '**`cvs`**' converte un valore in stringa, ma per farlo deve avere già una stringa da prelevare dalla pila:

```
valore stringa cvs
```

Volendo convertire il valore 23,45 in stringa, bisogna preparare prima una stringa di almeno cinque caratteri:

```
23.45 (     ) cvs
```

Per allocare una stringa, composta da caratteri <*NUL*>, ovvero 000_8, si può usare la funzione '**`string`**', che richiede l'indicazione della quantità di caratteri. Pertanto, la stessa cosa avrebbe potuto essere scritta nel modo seguente:

```
23.45 5 string cvs
```

Naturalmente, la funzione '**`cvs`**' si può usare per visualizzare la stringa generata, per esempio nel modo seguente:

```
10 10 moveto
23.45
5 string cvs
show
```

Si osservi che con '**`cvs`**', anche se si alloca una stringa più grande del necessario, questa viene ridotta alla dimensione richiesta dalla conversione.

Tabella 48.59. Espressioni relative a stringhe.

Istruzione	Descrizione
n `string`	Alloca una stringa di ***n*** caratteri <*NUL*>.
stringa `stringwidth`	Inserisce le coordinate finali della stringa nella pila.
valore ***stringa*** `cvs`	Restituisce una stringa corrispondente al valore.
valore ***n*** `string cvs`	Restituisce una stringa corrispondente al valore, con un massimo di ***n*** caratteri.

48.11 Funzioni

Una funzione si definisce attraverso la sintassi seguente:

```
/nome {istruzioni} def
```

In pratica, si vuole fare in modo che usando il nome indicato, si faccia riferimento automaticamente al gruppo di istruzioni contenuto tra parentesi graffe. Si noti che, eccezionalmente, se si tratta di definire una costante o se si vuole ridefinire il nome di un'altra funzione, non sono necessarie le parentesi graffe.

Come per qualunque altra funzione normale, anche le funzioni definite in questo modo ricevono gli argomenti delle chiamate dalla pila. Per esempio, la funzione '**`quadrilatero`**' che si potrebbe dichiarare nel modo seguente, va usata mettendo davanti, ordinatamente gli argomenti per le varie funzioni utilizzate:

```
/quadrilatero { newpath moveto lineto lineto lineto ↩
↪closepath stroke } def
```

Per esempio, volendo disegnare un quadrato con gli angoli nelle coordinate 0 , 0, 0 , 10, 10 , 10 e 10 , 0, si deve usare la funzione '**`quadrilatero`**' nel modo seguente:

```
10 0 10 10 0 10 0 0 quadrilatero
```

È importante osservare che la prima coppia di coordinate è quella presa in considerazione dall'ultima funzione '**lineto**' contenuta nel raggruppamento di '**quadrilatero**'.

Così come si definisce una funzione, si può attribuire a un nome un valore costante. In questi casi eccezionali, è consentita l'eliminazione delle parentesi graffe:

```
/nome costante def
```

Con la definizione di costanti, si può stabilire una volta per tutte il valore di qualcosa, come nell'esempio seguente:

```
/Margine_Sinistro 80 def
/Margine_Destro 80 def
/Margine_Superiore 100 def
/Margine_Inferiore 100 def
```

48.11.1 Variabili

Nel linguaggio PostScript non è prevista la gestione di variabili: tutto viene elaborato attraverso la pila. Tuttavia, esiste un trucco per ottenere qualcosa che assomigli a delle variabili; si tratta di sfruttare opportunamente la definizione di funzioni. È già stato visto l'assegnamento di un valore costante a un nome:

```
/nome costante def
```

Oppure:

```
/nome { costante } def
```

Se si vuole attribuire a una funzione un valore diverso, occorre un trucco, che si può schematizzare come segue:

```
/nome_1 { /nome_2 exch def } def
```

Si tratta di una funzione che ne dichiara un'altra, ma si osservi con attenzione: la parola chiave '**exch**' non è racchiusa tra parentesi graffe e non può esserlo, se si vuole che il meccanismo funzioni.

Per assegnare un valore alla funzione ***nome_2***, si utilizza una chiamata alla funzione ***nome_1***:

```
n nome_1
```

Per leggere il valore, si fa riferimento alla funzione ***nome_2***, come nell'esempio seguente in cui si utilizza questo dato come coordinata Y per uno spostamento:

```
n nome_1
m nome_2 moveto
```

48.12 Dizionari

La dichiarazione delle funzioni può essere inserita in un dizionario, da richiamare quando serve e da sostituire eventualmente con altre di un altro dizionario, quando la situazione lo richiede. In pratica, la definizione di dizionari di funzioni consente di fare riferimento a gruppi di funzioni solo nell'ambito di un certo contesto, ripristinando un utilizzo differente delle stesse subito dopo.

```
/dizionario n dict def
dizionario begin
    dichiarazione_di_funzione
    …
    …
end
```

Il modello sintattico mostra in che modo procedere alla dichiarazione di un dizionario. Si può osservare che prima della parola chiave '**dict**' occorre indicare un numero, allo scopo di definire una quantità di memoria da allocare per il dizionario stesso. Per abilitare l'uso delle funzioni dichiarate nel dizionario, si deve dichiarare espressamente:

```
dizionario begin
    istruzione
    …
    …
end
```

Eventualmente, la dichiarazione di utilizzo di un dizionario si può annidare; quando si raggiunge la parola chiave '**end**', termina il campo di azione dell'ultimo dizionario ancora aperto.

In generale, potrebbe essere conveniente inserire la dichiarazione dei dizionari nell'ambito delle istruzioni speciali '**%%BeginProlog**' e '**%%EndProlog**'. L'esempio seguente mostra un estratto di un file PostScript ipotetico, in cui si dichiara un dizionario (molto breve) e lo si utilizza immediatamente nell'ambito della pagina (i puntini di sospensione indicano una parte mancante del file che non viene mostrata).

```
%!PS-Adobe-2.0
%%DocumentPaperSizes: a4
%%EndComments
%%BeginProlog
/Mio_Dizionario 50 dict def
Mio_Dizionario begin
    /quadrilatero
    {
        newpath moveto lineto lineto lineto closepath stroke
    } def
    /Margine_Sinistro 80 def
    /Margine_Destro 80 def
    /Margine_Superiore 100 def
    /Margine_Inferiore 100 def
end     % Fine della dichiarazione del dizionario
        % «Mio_Dizionario»
%%EndProlog
Mio_Dizionario begin
%%Page: 1 1
…
…
…
showpage
%%Page: 2 2
…
…
…
showpage
end     % Fine del campo di azione del dizionario
        % «Mio_Dizionario»
%%Trailer
%%EOF
```

48.13 Caratteri da stampa

Il linguaggio PostScript, nelle sue prime versioni, prevede che il file contenente le istruzioni sia di tipo ASCII, dove è ammissibile usare gli 8 bit per esteso. Tuttavia, l'insieme di caratteri a disposizione non è il solito ISO 8859-1, ma qualcosa di diverso che può essere visto nella tabella 48.64.

Tabella 48.64. Insieme di caratteri normale del linguaggio PostScript.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220																	\237
\240		¡	¢	£	⁄	¥	ƒ	§	¤	'	"	«	‹	›	fi	fl	\257
\260		–	†	‡	·		¶	•	‚	„	”	»	…	‰		¿	\277
\300		`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\317
\320	—																\337
\340		Æ		ª					Ł	Ø	Œ	º					\357
\360		æ				ı			ł	ø	œ	ß					\377

Per fare riferimento a un simbolo in forma numerica, basta indicare il numero ottale corrispondente, di tre cifre, preceduto dalla barra obliqua inversa. Per facilitare la lettura, la tabella mostra questo modo di fare riferimento ai caratteri, all'inizio e alla fine di ogni riga.

Per poter usare la codifica corrispondente allo standard ISO 8859-1, è necessario dare delle istruzioni particolari. Volendo sbirciare nel codice PostScript generato da vari sistemi di composizione, si può osservare che ognuno utilizza un proprio modo, più o meno sofisticato. Probabilmente, la tecnica più semplice è quella che si può leggere nei sorgenti di Ghostscript, dove vengono dichiarate due funzioni apposite nel modo seguente:

```
/font-to-iso-latin-1
{
    dup length dict begin {1 index /FID ne {def} {pop pop}
    ifelse} forall
    /Encoding ISOLatin1Encoding def currentdict end
    dup /FontName get 80 string cvs (-ISOLatin1)
    concatstrings cvn
    exch definefont
} def

/find-latin-font
{
  findfont font-to-iso-latin-1
} def
```

In questo modo, quando si va a dichiarare l'uso di un certo tipo di carattere da stampa, invece dell'istruzione seguente, si usa piuttosto quella successiva:

```
/carattere findfont dimensione scalefont setfont
```

Forma modificata per la selezione della codifica ISO 8859-1:

```
/carattere find-latin-font dimensione scalefont setfont
```

48.14 Aspetto dei caratteri da stampa comuni

Il linguaggio PostScript mette a disposizione pochi tipi di carattere da stampa; tuttavia, è disponibile l'interprete Ghostscript, su moltissime piattaforme, che mette a disposizione un numero discreto di questi tipi. La tabella 48.76 elenca brevemente i nomi di questi tipi di carattere, mentre in altre tabelle successive viene mostrato l'aspetto di alcuni di questi, ordinando i simboli secondo l'insieme ISO 8859-1.

Tabella 48.66. Carattere da stampa AvantGarde.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	(	\	)	^	_	\137
\140	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.67. Carattere da stampa Bookman.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	·	¨		°	¸		″	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.68. Carattere da stampa Courier.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	·	¨		°	¸		″	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.69. Carattere da stampa Helvetica.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	’	(	)	*	+	,	–	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.70. Carattere da stampa NewCenturySchlbk.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	’	(	)	*	+	,	–	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.71. Carattere da stampa Palatino.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	’	(	)	*	+	,	–	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.72. Carattere da stampa Times.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	’	(	)	*	+	,	–	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.73. Carattere da stampa ZapfChancery.

	*															
\000																\017
\020																\037
\040		!	"	#	$	%	&	’	(	)	*	+	,	–	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	\117
\120	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	\137
\140	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	\157
\160	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~		\177
\200																	\217
\220	ı	`	´	ˆ	˜	¯	˘	˙	¨		°	¸		˝	˛	ˇ	\237
\240		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯	\257
\260	°	±	2	3	´	µ	¶	·	¸	1	º	»	¼	½	¾	¿	\277
\300	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï	\317
\320	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß	\337
\340	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï	\357
\360	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ	\377

Tabella 48.74. Carattere da stampa ZapfDingbats (solo con l’insieme di caratteri normale).

	*															
\000																\017
\020																\037
\040		✁	✂	✃	✄	☎	✆	✇	✈	✉	☛	☞	✌	✍	✎	✏	\057
\160	✐	✑	✒	✓	✔	✕	✖	✗	✘	✙	✚	✛	✜	✝	✞	✟	\077
\100	✠	✡	✢	✣	✤	✥	✦	✧	★	✩	✪	✫	✬	✭	✮	✯	\117
\120	✰	✱	✲	✳	✴	✵	✶	✷	✸	✹	✺	✻	✼	✽	✾	✿	\137
\140	❀	❁	❂	❃	❄	❅	❆	❇	❈	❉	❊	❋	●	❍	■	❑	\157
\160	❒	❑	❒	▲	▼	◆	❖	◗	|	❙	❚	❛	❜	❝	❞		\177
\200																	\217
\220																	\237
\240		❡	❢	❣	❤	❥	❦	❧	♣	♦	♥	♠	①	②	③	④	\257
\260	⑤	⑥	⑦	⑧	⑨	⑩	❶	❷	❸	❹	❺	❻	❼	❽	❾	❿	\277
\300	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	❶	❷	❸	❹	❺	❻	\317
\320	❼	❽	❾	❿	➔	→	↔	↕	➘	➙	➚	➛	➜	→	➝	➞	\337
\340	➟	➠	➢	➢	➤	➥	➦	➧	➨	➩	➪	➫	➬	➭	➮	➯	\357
\360		➱	➲	➳	➴	➵	➶	➷	➸	➹	➺	➻	➼	➽	➾		\377

Tabella 48.75. Carattere da stampa Symbol (solo con l'insieme di caratteri normale).

\000																\017
\020																\037
\040		!	∀	#	∃	%	&	∍	(	)	*	+	,	−	.	/	\057
\160	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	\077
\100	≅	Α	Β	Χ	Δ	Ε	Φ	Γ	Η	Ι	ϑ	Κ	Λ	Μ	Ν	Ο	\117
\120	Π	Θ	Ρ	Σ	Τ	Υ	ς	Ω	Ξ	Ψ	Ζ	[	∴	]	⊥	_	\137
\140	‾	α	β	χ	δ	ε	φ	γ	η	ι	φ	κ	λ	μ	ν	ο	\157
\160	π	θ	ρ	σ	τ	υ	ϖ	ω	ξ	ψ	ζ	{	|	}	~		\177
\200																	\217
\220																	\237
\240	∈	ϒ	′	≤	⁄	∞	ƒ	♣	♦	♥	♠	↔	←	↑	→	↓	\257
\260	°	±	″	≥	×	∝	∂	•	÷	≠	≡	≈	…	|	—	↵	\277
\300	ℵ	ℑ	ℜ	℘	⊗	⊕	∅	∩	∪	⊃	⊇	⊄	⊂	⊆	∈	∉	\317
\320	∠	∇	®	©	™	∏	√	·	¬	∧	∨	⇔	⇐	⇑	⇒	⇓	\337
\340	◊	〈	®	©	™	∑	⎛	⎜	⎝	⎡	⎢	⎣	⎧	⎨	⎩	⎪	\357
\360		〉	∫	⌠	⎮	⌡	⎞	⎟	⎠	⎤	⎥	⎦	⎫	⎬	⎭		\377

Tabella 48.76. Nomi dei tipi di carattere da stampa che solitamente sono disponibili con Ghostscript.

Nome	Descrizione
AvantGarde-Book	AvantGarde normale.
AvantGarde-BookOblique	AvantGarde inclinato.
AvantGarde-Demi	AvantGarde neretto.
AvantGarde-DemiOblique	AvantGarde neretto obliquo.
Bookman-Demi	Bookman neretto.
Bookman-DemiItalic	Bookman neretto corsivo.
Bookman-Light	Bookman normale.
Bookman-LightItalic	Bookman corsivo.
Courier	Courier normale.
Courier-Bold	Courier neretto.
Courier-BoldOblique	Courier neretto inclinato.
Courier-Oblique	Courier inclinato.
Helvetica	Helvetica normale.
Helvetica-Bold	Helvetica neretto.
Helvetica-BoldOblique	Helvetica neretto inclinato.
Helvetica-Narrow	Helvetica stretto.
Helvetica-Narrow-Bold	Helvetica stretto neretto.
Helvetica-Narrow-BoldOblique	Helvetica stretto neretto inclinato.
Helvetica-Narrow-Oblique	Helvetica stretto inclinato.
Helvetica-Oblique	Helvetica inclinato.
NewCenturySchlbk-BoldItalic	NewCenturySchlbk neretto corsivo.
NewCenturySchlbk-Bold	NewCenturySchlbk neretto.
NewCenturySchlbk-Italic	NewCenturySchlbk corsivo.
NewCenturySchlbk-Roman	NewCenturySchlbk normale.
Palatino-Bold	Palatino neretto.
Palatino-BoldItalic	Palatino neretto corsivo.
Palatino-Italic	Palatino corsivo.
Palatino-Roman	Palatino normale.
Times-Bold	Times neretto.
Times-BoldItalic	Times neretto inclinato.
Times-Italic	Times corsivo.
Times-Roman	Times normale.
ZapfChancery-MediumItalic	ZapfChancery-MediumItalic medio corsivo.
ZapfDingbats	ZapfDingbats (solo con l'insieme di caratteri normale).
Symbol	Symbol (solo con l'insieme di caratteri normale).

48.15 Distorsione e spostamento dei caratteri da stampa

Fino a questo punto è stato visto come selezionare un carattere da stampa con l'istruzione seguente o con quella successiva, nel caso sia disponibile la funzione relativa:

```
/carattere findfont dimensione scalefont setfont
```

```
/carattere find-latin-font dimensione scalefont setfont
```

In alternativa, è possibile sostituire l'indicazione della dimensione con qualcosa di più articolato, secondo uno dei due schemi seguenti:

```
/carattere findfont [dim_x rot_x rot_y dim_y sp_x sp_y] ↩
↪makefont setfont
```

```
/carattere find-latin-font [dim_x rot_x rot_y dim_y sp_x sp_y] ↩
↪makefont setfont
```

Tra parentesi quadre appaiono dei valori che, se non utilizzati, vanno lasciati azzerati. Per comprenderne il significato, conviene partire dalla situazione normale. Supponendo di volere dichiarare un carattere da stampa di tipo Helvetica, normale, di 12 punti, si può usare la definizione solita:

```
/Helvetica findfont 12 scalefont setfont
```

Oppure, in modo più complesso, quella seguente:

```
/Helvetica findfont [12 0 0 12 0 0] makefont setfont
```

In pratica, ***dim_x*** e ***dim_y*** servono per definire la dimensione orizzontale e verticale del carattere. In condizioni normali, la dimensione è la stessa; diversamente, con una dimensione orizzontale più grande di quella verticale si ottiene un carattere più largo, mentre con una dimensione verticale maggiore di quella orizzontale, si ottiene un carattere alto (o sottile). Si osservi l'esempio seguente, di un carattere Helvetica allargato e successivamente alzato. Il risultato, in rapporto, si può vedere nella figura 48.81.

```
/Helvetica findfont [30 0 0 12 0 0] makefont setfont
```

```
/Helvetica findfont [12 0 0 30 0 0] makefont setfont
```

Figura 48.81. Distorsione orizzontale e verticale del carattere da stampa.

Helvetica allargato

Helvetica alzato (stretto)

I valori corrispondenti a ***rot_x*** e ***rot_y*** permettono di inclinare l'asse X o l'asse Y del testo. In pratica, modificando ***rot_x*** si cambia l'andamento orizzontale del testo secondo l'angolo indicato. Per esempio un angolo di 10 gradi fa sì che la riga scritta sia inclinata verso l'alto (ovvero di 10 gradi in senso antiorario), mentre un angolo di -10 gradi genera una riga inclinata verso il basso. La figura 48.84 mostra il risultato che si può ottenere con le due distorsioni seguenti:

```
/Helvetica findfont [12 5 0 12 0 0] makefont setfont
```

```
/Helvetica findfont [12 -5 0 12 0 0] makefont setfont
```

Figura 48.84. Distorsione dell'inclinazione orizzontale.

Scrittura ascendente

Scrittura discendente

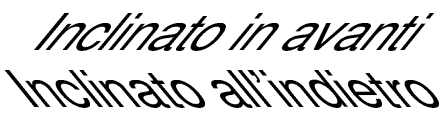
Intervenendo invece nel valore ***rot_y***, si cambia l'inclinazione del carattere, senza cambiare l'andamento della riga. In pratica, valori

positivi dell'angolazione generano un'inclinazione in avanti, simile a un corsivo, mentre valori negativi generano un'inclinazione all'indietro. La figura 48.87 mostra il risultato che si può ottenere con le due distorsioni seguenti:

```
/Helvetica findfont [12 0 10 12 0 0] makefont setfont
```

```
/Helvetica findfont [12 0 -10 12 0 0] makefont setfont
```

Figura 48.87. Distorsione dell'inclinazione verticale.



I valori corrispondenti a ***sp_x*** e ***sp_y*** rappresentano uno spostamento orizzontale e verticale, in punti, senza applicare delle distorsioni vere e proprie.

L'andamento del testo, che normalmente si svolge da sinistra a destra, può essere invertito, assegnando un valore negativo per il primo valore, ovvero per ***dim_x***. La figura 48.89 mostra la comparazione tra un testo scritto in modo normale e un altro che invece si sviluppa verso sinistra, con un comando simile a quello seguente:

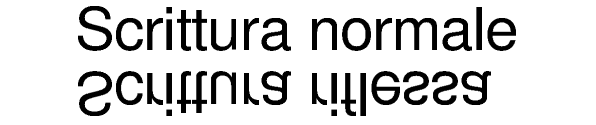
```
/Helvetica findfont [-12 0 0 12 0 0] makefont setfont
```

Figura 48.89. Scrittura normale e scrittura da destra a sinistra.

Così come si può invertire l'andamento del testo in orizzontale, si può invertire in modo verticale, ottenendo una sorta di riflessione. La figure 48.91 mette a confronto un testo scritto in modo normale e un altro modificato con l'istruzione seguente:

```
/Helvetica findfont [12 0 0 -12 0 0] makefont setfont
```

Figura 48.91. Scrittura ruotata sull'asse orizzontale.



48.16 Esempi di funzioni

In questa sezione si raccolgono alcuni esempi di funzioni PostScript che possono essere utili a vario titolo. Naturalmente, trattandosi di un linguaggio specifico per la stampa, non vengono proposti esempi di programmazione standard.

48.16.1 Unità di misura

L'unità di misura utilizzata è sempre il punto tipografico, che in questo ambito corrisponde a 1/72-esimo di pollice, ovvero a 0,3527777 mm (per converso, un millimetro è pari a 2,834646 punti). Volendo usare unità di misura più comuni, si possono realizzare alcune funzioni molto semplici che si limitano a moltiplicare il valore per una costante, in modo da ottenere come risultato l'equivalente in punti:

```
/cm { 28.346456 mul } def  % <n> cm % converte 'cm' in punti
/mm { 2.8346456 mul } def  % <n> mm % converte 'mm' in punti
/pt { 1 mul } def          % <n> pt % non converte alcunché
/in { 72 mul } def         % <n> in % converte 'in' in punti
```

In questo modo, al posto di inserire un valore puro e semplice, basta aggiungere subito dopo la sigla dell'unità di misura, che in realtà è una funzione. Per esempio:

```
3 cm 18 cm moveto
```

Evidentemente, la funzione '**pt**' è inutile, ma può servire per mantenere coerenza con il resto, nel momento in cui si utilizzi sistematicamente questo meccanismo per indicare le coordinate o le distanze.

48.16.2 Funzioni diagnostiche

Pur non trattandosi di un linguaggio di programmazione normale, quando si cerca di realizzare qualcosa di particolare, può essere comoda la possibilità di mostrare un valore da qualche parte, per verificare il contenuto di una data informazione.

```
/diag_display_number % <n> <x> <y> diag_display_number
{
    gsave
    /Helvetica findfont 10 scalefont setfont
    moveto
    100 string cvs show
    grestore
} def
```

La funzione appena mostrata, serve per ottenere la conversione di un numero in stringa, che poi viene visualizzato nelle coordinate previste. Andrebbe usata nel modo seguente, dove ***x*** , ***y*** sono le coordinate a partire dalle quali mostrare il valore:

```
n x y diag_display_number
```

Dal momento che questa funzione preleva il valore dalla pila, potrebbe essere conveniente la duplicazione di tale valore prima di utilizzarlo:

```
n dup x y diag_display_number
```

Volendo completare il problema con una funzione equivalente per la visualizzazione delle stringhe, basta la variante seguente:

```
/diag_display_string % <stringa> <x> <y> diag_display_string
{
    gsave
    /Helvetica findfont 10 scalefont setfont
    moveto
    show
    grestore
} def
```

48.16.3 Gestione di stringhe

Quando si disegnano delle figure o dei grafici, può essere comodo disporre di qualche funzione che faciliti la collocazione di didascalie o di annotazioni di qualunque tipo. Qui viene proposto un sistema molto semplice, con cui è possibile piazzare delle stringhe allineate correttamente a sinistra, a destra o al centro, date le coordinate di riferimento. Si parte con la definizione di alcune «variabili», che servono per fissare i punti di riferimento delle stringhe:

```
/set_line_t   { /line_t    exch def } def
/set_line_l   { /line_l    exch def } def
/set_line_r   { /line_r    exch def } def
/set_line_c   { /line_c    exch def } def
/set_line_h   { /line_h    exch def } def
```

La funzione '**line_t**' viene usata per conoscere la posizione verticale (Y) della stringa da collocare; la funzione '**line_l**' serve per fornire la posizione iniziale (X) di una stringa da allineare a sinistra; '**line_r**' fornisce la posizione finale (X) di una stringa da allineare a destra; '**line_c**' fornisce la posizione centrale (X) di una stringa da centrare; infine, '**line_h**' serve per conoscere la distanza tra le righe.

Per usare le funzioni che vengono presentate nel seguito, devono essere impostati inizialmente i valori per le variabili appena descritte. Per esempio, sapendo che si vuole scrivere un testo allineato a sinistra a partire dalla coordinata 100 , 50, con una distanza tra le righe di 14 punti, basta impostare i valori nel modo seguente:

```
 50 set_line_t
100 set_line_l
 14 set_line_h
```

Dal momento che non si devono centrare le righe e nemmeno allineare a destra, le altre variabili non servono, altrimenti occorrerebbe impostare '**line_c**' o '**line_r**', al posto di '**line_l**'.

```
/show_line_left % <stringa> show_line_left
{
    % Si posiziona all'inizio della riga.
    line_l line_t moveto

    % Mostra la riga.
    show

    % Sottrae alla posizione Y l'altezza della riga.
    line_t line_h sub set_line_t
} def
```

Si osservi la funzione '**show_line_left**': si usano i valori restituiti dalle funzioni '**line_l**' e '**line_t**' per impostare le coordinate iniziali, quindi si visualizza la stringa. Subito dopo si provvede a ridurre il valore della variabile corrispondente alla funzione '**line_t**' del valore restituito da '**line_h**' (la distanza tra le righe), in modo da poter continuare con la visualizzazione di altre stringhe, con lo stesso allineamento sinistro, subito sotto quella appena inserita.

```
/show_line_right % <stringa> show_line_right
{
    % Calcola la lunghezza della riga.
    dup % fa una copia della stringa
    stringwidth % restituisce le coordinate finali X e Y
    pop % elimina la coordinata Y

    % Sottrae dalla posizione destra la lunghezza della
    % riga, cambiando poi il segno.
    line_r sub  neg

    % Si posiziona correttamente.
    line_t moveto

    % Mostra la riga e va all'inizio di una riga nuova.
    show

    % Sottrae alla posizione Y l'altezza della riga.
    line_t line_h sub set_line_t
} def
```

La funzione '**show_line_right**' è molto simile, con la differenza che occorre fare qualche calcolo per individuare la posizione orizzontale di inizio, sapendo la posizione finale ottenuta dalla funzione '**line_r**'. Per questo, viene fatta una copia della stringa, che quindi viene misurata con la funzione '**setlinewidth**'. Da questa misurazione si espelle l'informazione verticale, che risulta inutile, sottraendo poi a questa il valore restituito da '**line_r**'. Quello che si ottiene è la distanza dalla posizione destra finale, con segno invertito, pertanto si inverte nuovamente con la funzione '**neg**'. Disponendo della coordinata X, si aggiunge la coordinata Y, data da '**line_t**', spostando la posizione corrente, per poi mostrare la stringa e infine preparare nuovamente la posizione verticale per una nuova riga.

```
/show_line_center
{
    % Calcola la lunghezza della riga.
    dup % fa una copia della stringa
    stringwidth % restituisce le coordinate finali X e Y
    pop % elimina la coordinata Y
    2 div % divide la lunghezza a metà

    % Sottrae dal centro la metà della riga, cambiando poi
    % il segno.
    line_c sub  neg

    % Si posiziona correttamente.
    line_t moveto

    % Mostra la riga e va all'inizio di una riga nuova.
    show

    % Sottrae alla posizione Y l'altezza della riga.
    line_t line_h sub set_line_t
} def
```

La funzione '**show_line_center**' centra la stringa in riferimento alla posizione '**line_c**'. I calcoli sono simili a quelli per l'allineamento a destra, con la differenza che la distanza dal centro è pari alla metà della lunghezza della stringa. Il resto è equivalente.

L'esempio seguente mostra come scrivere qualcosa con queste funzioni; la figura 48.102 mostra il risultato che si ottiene osservando il riquadro che va da 0 , 0 a 150 , 100:

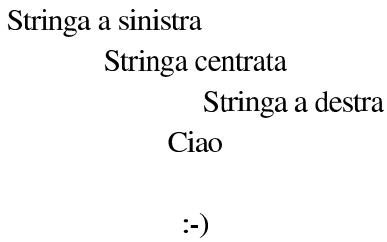
```
/Times-Roman findfont 10 scalefont setfont

 90 set_line_t
 10 set_line_l
140 set_line_r
 75 set_line_c
 14 set_line_h

(Stringa a sinistra) show_line_left
(Stringa centrata) show_line_center
(Stringa a destra) show_line_right
(Ciao) show_line_center
() show_line_center
(:-\)) show_line_center

showpage
```

Figura 48.102. Esempio nell'uso delle funzioni sulle stringhe.



Naturalmente, si possono predisporre anche delle abbreviazioni a queste funzioni:

```
/L { show_line_left } def
/R { show_line_right } def
/C { show_line_center } def
```

In questo modo, l'indicazione delle stringhe può essere ridotto alla forma seguente:

```
(Stringa a sinistra) L
(Stringa centrata) C
(Stringa a destra) R
(Ciao) C
() C
(:-\)) C
```

48.17 Approfondimento: modifica sistematica dei contenuti

Un problema abbastanza frequente negli uffici è la creazione di un'immagine da sovrapporre a tutte le stampate, come un marchio, o come una fincatura. Per esempio, al posto di far stampare della carta intestata, si vuole fare in modo che la stampante stampi automaticamente l'intestazione, senza fare la cosa in due passaggi. Questa sezione cerca di mostrare un metodo pratico per risolvere il problema.

48.17.1 Modifica del file PostScript

Per aggiungere qualcosa a una stampa si deve intervenire nel file PostScript prima che questo raggiunga la coda di stampa. Inizialmente si può fare qualche esperimento per scoprire in che punto conviene inserire le proprie modifiche. Tuttavia, se il contenuto del file PostScript è compresso, è praticamente impossibile insinuarsi nel suo codice. Nel caso fosse necessario espandere il codice PostScript, si può tentare di agire con il comando seguente:

```
$ gs -q -dNOPAUSE -dBATCH -sDEVICE=pswrite ↩
↪    -sOutputFile=output_file_ps ↩
↪    -c save pop -f input_file_ps [Invio]
```

Dove ***input_file_ps*** è il nome del file originale da espandere, mentre ***output_file_ps*** è il nome di quello che si vuole ottenere con la trasformazione.

Osservando un file PostScript, generato da un programma comune per la stampa, eventualmente dopo il procedimento di espansione appena mostrato, si dovrebbe individuare facilmente la fine di una pagina cercando l'istruzione '**showpage**':

```
PDFVars/TermAll get exec end end
userdict /pgsave get restore
showpage
%%PageTrailer
%%EndPage
```

Normalmente, subito prima di questa istruzione appare anche l'istruzione '**restore**' che serve a ripristinare le condizioni normali. Se effettivamente si recuperano tutte le impostazioni predefinite del linguaggio PostScript si può inserire subito prima dell'istruzione '**showpage**' qualcosa che si vuole sovrascrivere sulla pagina; per esempio un codice simile a quello seguente:

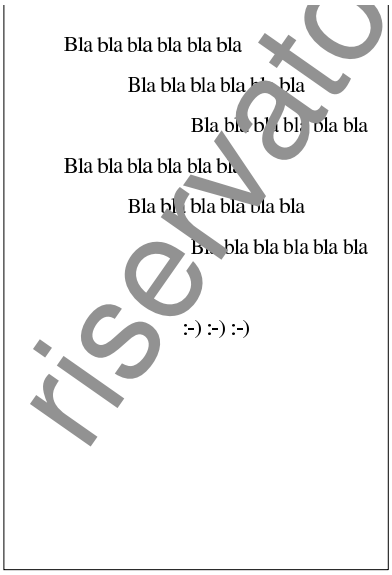
```
PDFVars/TermAll get exec end end
userdict /pgsave get restore
save
/cm { 28.346456 mul } def
   0 cm   0   cm moveto
  21 cm  29.7 cm lineto
   0 cm  29.7 cm moveto
  21 cm   0   cm lineto
stroke
restore
showpage
%%PageTrailer
%%EndPage
```

In questo caso si immagina di avere a disposizione una pagina A4, sulla quale si sovrascrive una «X» con due linee tra gli angoli opposti. Nel prossimo esempio, si intende sovrapporre la scritta «riservato» che appare in modo obliquo sulla pagina:

```
PDFVars/TermAll get exec end end
userdict /pgsave get restore
save
/cm { 28.346456 mul } def
    55  rotate
    7 cm  0 cm moveto
    /Helvetica findfont 7 cm scalefont setfont
    (riservato)
    0.5 setgray
    show
stroke
restore
showpage
%%PageTrailer
%%EndPage
```

La figura successiva mostra una pagina con un contenuto senza significato, sulla quale appare la scritta sovrapposta:

Figura 48.108. Esempio di una pagina con la sovrapposizione della scritta «riservato» in primo piano.

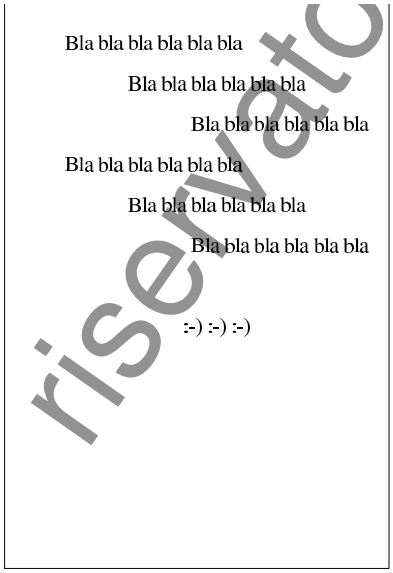


Se invece quello che si vuole è semplicemente di intervenire sullo sfondo della pagina, al posto di inserire il codice immediatamente prima dell'istruzione '**showpage**' occorre farlo all'inizio della pagina, precisamente subito dopo l'istruzione speciale '**%%Page:**', per esempio così:

```
%%Page: 1 10
save
/cm { 28.346456 mul } def
    55 rotate
    7 cm   0   cm moveto
    /Helvetica findfont 7 cm scalefont setfont
    (riservato)
    0.5 setgray
    show
stroke
restore
585 600 bop 1319 39 a Ii(Appunti)37 b(di)d(inf)-5
b(or)t(m)m(atica)35 b(liber)m(a)105 b Ic(2003.01.01)3042 243
y Ii(V)-13 b(olume)71
```

Nella figura successiva si mostra nuovamente l'esempio di una scritta obliqua che questa volta viene posta sullo sfondo, senza disturbare così la lettura del documento.

Figura 48.110. Esempio di una pagina con la scritta «riservato» sullo sfondo.



Eventualmente, per evitare sorprese, prima di modificare un file PostScript lo si può rielaborare attraverso '**ps2ps**'.

48.17.2 Due programmi Perl che fanno tutto da soli

Per risolvere il problema della modifica di file PostScript si può realizzare un piccolo programma Perl che legge un file contenente le aggiunte, filtrando un file PostScript:

```
ps-overwrite.pl file < file_ps_originale > file_ps_sovrascritto
```

```
ps-background.pl file < file_ps_originale > file_ps_sovrascritto
```

Per esempio, se il file '`aggiunte`' contiene il codice PostScript da aggiungere, i due comandi seguenti servirebbero per generare il file '`prova-modificato.ps`' dal file '`prova.ps`'; il primo per aggiungere il codice in primo piano, il secondo per aggiungerlo sullo sfondo:

```
$ cat prova.ps | ps-overwrite.pl aggiunte ↩
↪> prova-modificato.ps [Invio]

$ cat prova.ps | ps-background.pl aggiunte ↩
↪> prova-modificato.ps [Invio]
```

Ecco il primo programma:

```
#!/usr/bin/perl
local ($external_postscript_file) = $ARGV[0];
local ($external_postscript_code) = "";
local ($line)                     = "";

# Carica dal file esterno il codice da aggiungere.
open (PSCODE, "< $external_postscript_file");
while ($line = <PSCODE>)
  {
    $external_postscript_code = $external_postscript_code . $line;
  }
close (PSCODE);

# Legge lo standard input e cerca l'istruzione "showpage" per
# inserire il codice aggiunto.
while ($line = <STDIN>)
  {
    if ($line =~ m/^\%/)
      {
        # Si tratta di un commento o di un'istruzione speciale.
        print STDOUT ($line);
      }
    elsif ($line =~ m/^(.*)\bshowpage\b(.*)$/)
      {
        # Si aggiunge il codice.
        print STDOUT ($1);
        print STDOUT ("\n");
        print STDOUT ("save");
        print STDOUT ("\n");
        print STDOUT ("$external_postscript_code");
        print STDOUT ("\n");
        print STDOUT ("restore");
        print STDOUT ("\n");
        print STDOUT ("showpage");
        print STDOUT ("\n");
        print STDOUT ($2);
        print STDOUT ("\n");
      }
    else
      {
        print STDOUT ($line);
      }
  }
```

Ecco il secondo programma:

```
#!/usr/bin/perl
local ($external_postscript_file) = $ARGV[0];
local ($external_postscript_code) = "";
local ($line)                     = "";

# Carica dal file esterno il codice da aggiungere.
open (PSCODE, "< $external_postscript_file");
while ($line = <PSCODE>)
  {
    $external_postscript_code = $external_postscript_code . $line;
  }
close (PSCODE);

# Legge lo standard input e cerca l'istruzione speciale "%%Page:" per
# inserire il codice aggiunto.
while ($line = <STDIN>)
  {
    if ($line =~ m/^\%\%Page\:\s/)
      {
        # Si aggiunge il codice a partire dalla riga successiva.
        print STDOUT ($line);
        print STDOUT ("save");
        print STDOUT ("\n");
        print STDOUT ("$external_postscript_code");
        print STDOUT ("\n");
        print STDOUT ("restore");
        print STDOUT ("\n");
      }
    else
      {
        print STDOUT ($line);
      }
  }
```

Si può osservare che il codice aggiunto viene contornato automaticamente dalle istruzioni '**save**' e '**restore**' per maggior sicurezza.

48.17.3 Utilizzo pratico di questi programmi

Per fare in modo che tutte le stampe vengano alterate con qualche tipo di informazione, si deve intervenire nella gestione delle code di stampa. Se si dispone di un sistema di stampa BSD o compatibile, si può agire nel file '`/etc/printcap`', aggiungendo delle code differenti per ogni tipo di modifica che si vuole apportare; a ognuna di queste si deve abbinare un filtro differente.

```
lp|HP Laserjet:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/log/lp-acct:\
        :lf=/var/log/lp-errs:\
        :if=/etc/magicfilter/ljet4-filter:\
        :pl#66:\
        :pw#80:\
        :pc#150:\
        :mx#0:\
        :sh:
```

L'esempio che si vede sopra fa riferimento a una direttiva della coda di stampa predefinita, a cui si associa il filtro '`/etc/magicfilter/ljet4-filter`'. Si tratta precisamente di un programma scritto per Magicfilter (sezione 27.5), il quale contiene in particolare le direttive seguenti:

```
# PostScript
0       %!      filter  /usr/bin/gs  -q -dSAFER
-dNOPAUSE -r600 -sDEVICE=ljet4 -sOutputFile=- -
0       \004%!  filter  /usr/bin/gs  -q -dSAFER
-dNOPAUSE -r600 -sDEVICE=ljet4 -sOutputFile=- -
```

A questo punto si potrebbero scrivere una serie di file contenenti codice PostScript da aggiungere alle stampe, a seconda delle esigenze. Si suppone di collocare questi file nella directory '`/etc/ps-modifica/`'. Si osservi il listato successivo che si riferisce ad altre due code di stampa speciali:

```
logo|HP Laserjet con sovrascrittura del logo aziendale:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/log/lp-acct:\
        :lf=/var/log/lp-errs:\
        :if=/etc/magicfilter/ljet4-logo-filter:\
        :pl#66:\
        :pw#80:\
        :pc#150:\
        :mx#0:\
        :sh:

fatt|HP Laserjet con fincatura delle fatture:\
        :lp=/dev/lp0:\
        :sd=/var/spool/lpd/lp:\
        :af=/var/log/lp-acct:\
        :lf=/var/log/lp-errs:\
        :if=/etc/magicfilter/ljet4-fattura-filter:\
        :pl#66:\
        :pw#80:\
        :pc#150:\
        :mx#0:\
        :sh:
```

A questo punto si prende il file '`/etc/magicfilter/ljet4-filter`' e se ne fanno due copie: rispettivamente '`/etc/magicfilter/ljet4-logo-filter`' e '`/etc/magicfilter/ljet4-fattura-filter`'. All'interno di questi file si devono modificare le direttive riferite al filtro PostScript; in questo caso si mostra l'uso dello script '**`ps-overwrite`**', per la sovrapposizione in primo piano, ma l'uso dell'altro script è identico:

```
# PostScript
0       %!      filter  /usr/bin/ps-overwrite
/etc/ps-modifica/logo | /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
0       \004%!  filter  /usr/bin/ps-overwrite
/etc/ps-modifica/logo | /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
```

```
# PostScript
0       %!      filter  /usr/bin/ps-overwrite
/etc/ps-modifica/fattura
| /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
0       \004%!  filter  /usr/bin/ps-overwrite
/etc/ps-modifica/fattura
| /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
```

Se si vuole aggiungere anche un passaggio con '**`ps2ps`**':

```
# PostScript
0       %!      filter  /usr/bin/ps2ps - -
| /usr/bin/ps-overwrite /etc/ps-modifica/logo
| /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
0       \004%!  filter  /usr/bin/ps2ps - -
| /usr/bin/ps-overwrite /etc/ps-modifica/logo
| /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
```

```
# PostScript
0       %!      filter  /usr/bin/ps2ps - -
| /usr/bin/ps-overwrite /etc/ps-modifica/fattura
| /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
0       \004%!  filter  /usr/bin/ps2ps - -
| /usr/bin/ps-overwrite /etc/ps-modifica/fattura
| /usr/bin/gs  -q -dSAFER -dNOPAUSE
-r600 -sDEVICE=ljet4 -sOutputFile=- -
```

Naturalmente rimangono da preparare i file '`/etc/ps-modifica/logo`' e '`/etc/ps-modifica/fattura`' con il codice PostScript che si preferisce.

Alla fine, è sufficiente selezionare una coda o l'altra per aggiungere il logo o la fincatura di una fattura:

```
$ lpr -Plogo ... [ Invio ]

$ lpr -Pfatt ... [ Invio ]
```

48.18 Riferimenti

- *Cappella Archive*, http://www.cappella.demon.co.uk/index.html
- David Byram-Wigfield, *Practical PostScript*, http://www.cappella.demon.co.uk/bookpdfs/pracpost.pdf
- David Byram-Wigfield, *Making an electronic book*, http://www.cappella.demon.co.uk/tinypdfs/06eb06ook.pdf
- Paul Bourke, *PostScript Tutorial*, http://local.wasp.uwa.edu.au/~pbourke/dataformats/postscript/
- *First Guide to PostScript*, http://www.tailrecursive.org/postscript/postscript.html
- *Portable Document Format Reference Manual*, 1999, *http://www.hamburg.baw.de/docs/pdfspec.pdf* (non più disponibile)