Informatique
Graphique

Professeur:
Victor Ostromoukhov

Un programme mystérieux

```
%!PS
/Times-Roman findfont 150 scalefont setfont
180 100 moveto
60 rotate
(Bi) show
1 0 0 setrgbcolor (en) show
0 1 0 setrgbcolor (ve) show
0 0 1 setrgbcolor (nue) show
showpage
```

- Quel langage de programmation?
- Que fait ce programme? ESSAYER

Un programme mystérieux

```
%!PS
/Times-Roman findfont 150 scalefont setfont
180 100 moveto
60 rotate
(Bi) show
1 0 0 setrgbcolor (en) show
0 1 0 setrgbcolor (ve) show
0 0 1 setrgbcolor (nue) show
showpage
```

- Quel langage de programmation?
- Que fait ce programme? ESSAYER

Bienvenue

Page 3

Traitement de l'information et automatisation

Théorie

-3000 **Binaire** (Ying et Yang chinois)

-1750 **Code d'HAMMOURABI**

Le roi de Babylone (Mésopotamie), nommé HAMMOURABI, a fait graver cette stèle. Celle-ci composée d'un ensemble de sentences royales sous la forme:
SI {personne} ET {action} ALORS {sentence}

- Oldest Known Building Code of Law
- The Avalon Project: Code Of Hammurabi

-330 **Logique**

Elle est définie par le philosophe grec ARISTOTE.

820 **Travaux du mathématicien arabe AL KHOWARIZMI**

1000 **Zéro**

Inventé en Inde et rapporté en Occident par les invasions arabes, le zéro trouvera un ardent défenseur en la personne de Gerbert d'AURILLAC qui tentera de l'imposer lorsqu'il deviendra le pape Sylvestre II. Mais, ce n'est que vers le XIV^{ème} siècle, que le monde occidental l'acceptera définitivement.

1614 **Logarithmes** par NEPER

Grace aux travaux de L'Ecossais NEPER, la multiplication et la division peuvent être ramenées à deux opérations très simples: l'addition et la soustraction.

Traitement de l'information et automatisation

Théorie (Cont.)

1697 **Introduction du binaire en Europe** par G LEIBNITZ

Passionné par la *Dyadique*, LEIBNITZ fut conforté dans ses idées lorsqu'il apprit que le binaire avait été inventé par les Chinois plusieurs millénaires auparavant. Il exposera devant l'Académie des Sciences de Paris ses idées qui seront publiées dans "Explication de l'arithmétique binaire avec des remarques sur son utilité et sur le sens qu'elle donne des anciennes figures chinoises de Fou-Hi".

1840 **Principe des machines à calculer** par A LOVELACE

Pour elle, une machine à calculer doit comporter:

- Un dispositif permettant d'introduire les données numériques (cartes perforées, roues dentées...)
- Une mémoire pour conserver les valeurs numériques entrées
- Une unité de commande grâce à laquelle l'utilisateur va indiquer à la machine les tâches à effectuer
- Un "moulin" chargé d'effectuer les calculs
- Un dispositif permettant de prendre connaissance des résultats (imprimante...)

Ces principes seront, un siècle plus tard, à la base des premiers ordinateurs

1843 **Théorie de la programmation** par A LOVELACE

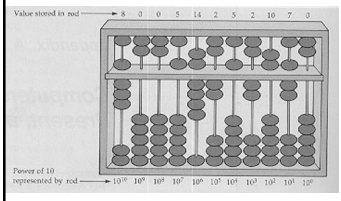
Ici, Ada définit le principe d'itérations successives dans l'exécution d'une opération. En l'honneur du mathématicien arabe AL KHOWARIZMI, elle appelle "algorithme" le processus logique permettant l'exécution d'un programme.

1854 **Théorie de la logique binaire** de G BOOLE

Dans "Les lois de la pensée", il explique que l'on peut coder les démarches de la pensée à l'aide de système n'ayant que deux états: ZERO-UN; OUI-NON; VRAI-FAUX...

Traitement de l'information et automatisation

Matériel



- 10000 **Boules et jetons d'argile**

Les Sumériens ont été les premiers à utiliser un système permettant de traiter l'information (transactions, mémorisation, contrôle...)

-3000 **Boulier**

Composé de perles évoluant sur des tiges, c'est déjà un appareil de calcul qui permet d'avoir la représentation du nombre (mémoire) et d'effectuer une opération sur ce nombre (calcul).
Principe du boulier
Soroban Museum
Le boulier chinois

100 **Abaque Romain**

C'est la version "Occidentale" du boulier. Son fonctionnement est sensiblement le même.

1200 **Quipu Péruvien**

1615 **Bâtons de NEPER (NAPIER)** Musée

En bois, en os ou en ivoire, ils se composent de 10 cases divisées par une diagonale; dans le triangle inférieur droit figure le chiffre des unités du produit et dans le supérieur gauche, le chiffre des dizaines. Ils seront utilisés pendant plus de 200 ans.
John Napier and Napier's Bones

Traitement de l'information et automatisation

Matériel (Cont.)

1623 **Machine à calculer** de W. SCHICKARD Musée

Composée de 6 cylindres *Népériens* , de réglettes coulissantes et de 6 disques opérateurs, cette machine était capable d'effectuer les reports de retenues dans un sens (addition) ou dans l'autre (soustraction). Détruite en 1624, elle ne sera reconstruite qu'en 1960 d'après les plans originaux.

1641 **La Pascaline**: Machine à calculer de B PASCAL Musée1 Musée2 Musée3 Musée4

Cette machine, qu'il a construite afin d'aider son père, est la première qui a réellement fonctionné. Elle servira de référence pour les machines futures.

1694 **Machine à calculer** de LEIBNITZ Musée

Cette machine, inspirée par la Pascaline, n'aura pas le succès de celle-ci. Cependant, elle comportait nombre d'innovations mécaniques comme le tambour à dents inégales, permettant de multiplier un nombre par rotations répétées de la manivelle principale.

1728 **Métier à tisser** de FALCON

C'est le premier à utiliser un "programme" sur plaquettes de bois perforées.

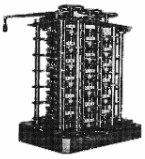
1806 **Métier à tisser à cartes perforées** de Joseph-Marie JACQUARD

Reprenant l'idée de FALCON, il met au point des métiers à tisser automatisés commandés par des cartes perforées et il en équipe les ateliers lyonnais. Les *canuts* (tisserands Lyonnais) craignant que ces machines ne prennent leur place, en détruiront la plupart. Cependant, elles finiront par connaître un très grand succès (10 000 en service en 1812).

Traitement de l'information et automatisation

Matériel (Cont.)

1820 **Machine à différences** de C BABBAGE Musée1 Musée2



Cette machine utilise le système décimal: la manivelle fait tourner les roues par dixièmes de tour. En raison de sa complexité (25 000 pièces) et de son coût, aucune des deux versions de cette machine ne sera completement finie par BABBAGE bien que le principe du calculateur soit parfaitement exact comme l'ont prouvé des chercheurs Britanniques qui en ont réalisé un exemplaire en 1991.

1822 **Arithmomètre** de Th. de COLMAR Musée

Cette machine, modeste sur le plan de la capacité de calcul (3 chiffres à l'inscripteur et 6 au totalisateur) fut néanmoins la première d'une longue série. Environ 1500 exemplaires furent commercialisés entre 1823 et 1878. La capacité augmenta progressivement (un exemplaire de 30 chiffres fut même réalisé) et des licences de productions furent concédées à l'étranger.

1833 **Machine analytique** de C BABBAGE

Alors que son projet de machine à différences s'enlisait, BABBAGE, aidé de A LOVELACE, conçut une machine composée d'un *moulin* (unité de calcul), d'un *magasin* (mémoire), et d'un *dispositif de contrôle*. Le tout utilisé à l'aide de cartes *opérations* , de cartes *de variables* et de cartes *nombres*. Grâce à cette machine, qui sera partiellement construite par son fils, BABBAGE est considéré comme l'authentique grand-père des ordinateurs modernes.

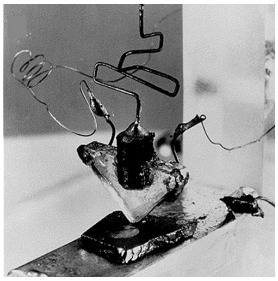
➔ The Analytical Engine Table of Contents

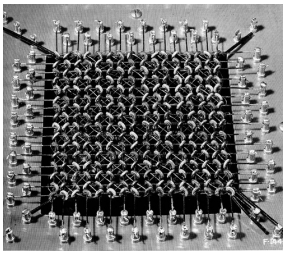
1867 **Machine à écrire**

Inventée par deux américains et commercialisée par la firme Remington, la machine à écrire mettra beaucoup de temps à se développer. La disposition des lettres (QWERTY...) vient du fait que les premières machines ne suivaient pas la cadence de certaines secrétaires; on a donc placé les lettres les plus utilisées sous les doigts les plus faibles.

L'ère de l'électronique

Matériel – composants (cont.)





1947 **Le transistor bipolaire à jonction** par J BARDEEN, W BRATTAIN et W SCHOCKLEY Musée

Il est constitué d'une très fine couche P entre deux couches N (ou bien l'inverse). Lorsque l'on fait circuler un faible courant entre une couche P et une couche N, un flux d'électrons entraîne une conduction entre les deux couches de même nature: c'est l'effet transistor. Ce composant est à l'origine d'une révolution dans l'électronique, en effet la faible énergie nécessaire pour le faire fonctionner, ainsi que sa petite taille rendent très vite les tubes obsolètes.

1954 **Transistor au silicium**

Beaucoup moins cher, plus facile à produire et à utiliser (mais hélas ayant une vitesse de conduction moins élevée) que le germanium, le silicium va devenir le symbole d'une nouvelle ère.

1959 **Transistor à effet de champ**

Plus proche de la triode que ne l'est le transistor bipolaire, celui-ci est composé d'une électrode appelée *grille* qui module la conductance entre une zone dite *source* et une autre dite *drain*.

1959 **Circuit intégré** par Jack KILBY

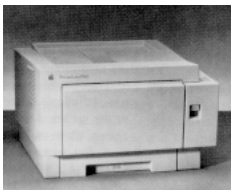
Le principe consiste à fabriquer dans un même bloc de semi-conducteur (une ***puce***) plusieurs composants (résistances, condensateurs, transistors). Cette idée sera reprise quelques mois plus tard par Robert NOYCE qui intégrant la technologie *planar* mettra au point des procédés toujours utilisés aujourd'hui.

1960 **Transistors Planar** par Jean HOERNI

C'est un transistor plat fabriqué à l'aide de gaz dopant positivement, négativement ou bien transformant le silicium en silice (oxyde de silicium) qui est un isolant.

L'ère de l'électronique

Matériel – interfaces



1951 **Écran** (oscilloscope)

L'oscilloscope, inventé un demi-siècle plus tôt sert d'abord à la vérification de l'état des bascules. Il deviendra par la suite une véritable interface, commandée par des circuits spécifiques, permettant l'affichage du texte et du graphisme.

1955 **Crayon optique**

En mesurant le temps écoulé entre le début du balayage du faisceau d'électrons et le moment où celui-ci "rencontre" la cellule du crayon optique, il est facile d'obtenir l'emplacement où pointe le crayon. Ce système sera très largement utilisé avant d'être détrôné par la souris.

1963 **Souris** par D ENGELBART Musée

Cette drôle de bestiole, composée le plus souvent d'une boule, de deux ou trois axes et d'un ou plusieurs boutons a tout d'abord été totalement rejetée avant de devenir un accessoire courant, au milieu des années 80. Le suivi des déplacements se fait à l'écran grâce à un curseur (en général une petite flèche).

1971 **Imprimante matricielle**

Grâce à l'utilisation de fines aiguilles qui viennent taper sur ruban encreur, il est maintenant possible d'imprimer des graphiques ou des caractères quelconques. On dit aussi de ces imprimantes qu'elles sont à aiguilles, ou à impacts.

1973 **Icônes**

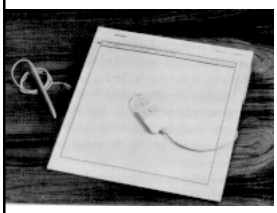
Suivant la même philosophie que la souris, l'icône est un petit dessin sur lequel on peut *cliquer* en amenant le pointeur de la souris dessus et en appuyant sur un des boutons. Ce système a fait la gloire du Macintosh.

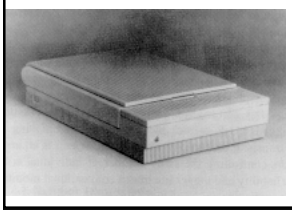
1975 **Imprimante laser**

Reprenant le principe du photocopieur, à savoir un tambour électrostatique sur lequel se fixe du toner, ce type d'imprimante permet d'obtenir une excellente qualité avec un prix de revient faible.

L'ère de l'électronique

Mémoire vive, supports de masse et réseaux





1928 **Carte perforée à 80 colonnes** d'IBM

Ce système qui deviendra un standard est la généralisation de la carte perforée qui est à l'origine de la compagnie.

1948 **Mémoires à tubes Williams**

Développée par F. C. Williams, ce type de mémoire utilise les charges résiduelles laissées sur l'écran d'un tube cathodique après qu'il ait été frappé par le faisceau d'électron.

1949 **Bande magnétique**

1954 **Mémoires vives à tores de ferrite**

Pendant une petite vingtaine d'année, ce principe de mémoire sera le plus utilisé avant d'être remplacé par la mémoire à semi-conducteurs.

1955 **Tambour magnétique**

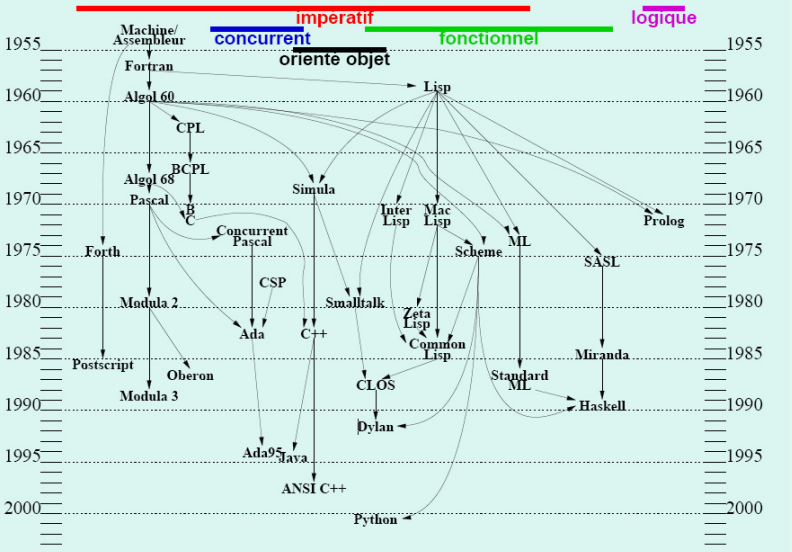
1956 **Disque magnétique** Musée

Aussi appelé *disque dur*, ce type de support deviendra incontournable lorsqu'il prendra sa forme actuelle en 1974: Le disque Winchester.

1968 **Mémoires à semi-conducteurs**

1972 **Disques souples**

Généalogie



Université de Montréal

Page 12

Syntaxe et sémantique

- Un langage (naturel ou de programmation) est défini par sa **syntaxe** et sa **sémantique**
- **Syntaxe:** apparence textuelle
- **Sémantique:** sens attaché au texte
- En Anglais:
  - **To be or not to be?** $\Rightarrow$ Être ou ne pas être?
  - **To to be be or not?** syntaxe incorrecte

Syntaxe des expressions

- Il existe 3 notations pour les expressions, qui se distinguent par la position de l'opérateur (p.e. +) par rapport aux opérandes
- **Infixe:** $\langle expression \rangle\ \langle op \rangle\ \langle expression \rangle$
  - Ex. en C: `1+2` calcule $1+2$
- **Préfixe:** $\langle op \rangle\ \langle expression \rangle\ \langle expression \rangle$
  - Ex. en Scheme: `(+ 1 2)` calcule $1+2$
  - Ex. en C: `pow(2,5)` calcule 2^5
- **Postfixe:** $\langle expression \rangle\ \langle expression \rangle\ \langle op \rangle$
  - Ex. en Postscript: `1 2 add` calcule $1+2$

Syntaxe des expressions

C	Scheme	Postscript
`2*(7-4)`	`(* 2 (- 7 4))` **parenthèses requises**	`2 7 4 sub mul` **pas de parenthèses**

- **En C:** les parenthèses sont requises seulement lorsque la priorité des opérateurs le demande

`2-7*4` = `2-(7*4)` = `((2)-((7)*(4)))`

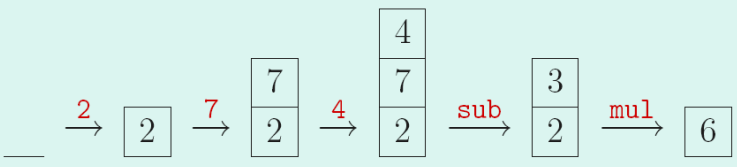
Un peu d'arithmétique

- **En Scheme:** chaque paire de parenthèses exprime une opération

($\langle op \rangle$ $\langle argument_1 \rangle$ $\langle argument_2 \rangle$ ...)

(* 2 (- 7 4))

- **En Postscript:** le calcul se fait avec une **pile**



Un peu d'arithmétique

C	Scheme ESSAYER	Postscript
`-5`	`-5` ou `(- 5)`	`-5` ou `5 neg`
`7.5 / 2`	`(/ 7.5 2)`	`7.5 2 div`
`7 / 2`	`(quotient 7 2)`	`7 2 idiv`
`7 % 2`	`(modulo 7 2)`	`7 2 mod`
`sqrt(2)`	`(sqrt 2)`	`2 sqrt`
`pow(10,5)`	`(expt 10 5)`	

- Scheme fait des calculs exacts lorsque c'est possible
- Attention! En C et Postscript les entiers sont limités (souvent de -2^{31} à $2^{31} - 1$)

Page 17

Variables

- "Pour contrôler une chose il faut pouvoir la nommer"

C	Scheme ESSAYER	Postscript
`int rayon = 10;`	`(define rayon 10)`	`/rayon 10 def`
`float pi = 3.1416;`	`(define pi 3.1416)`	`/pi 3.1416 def`
`2 * pi * rayon`	`(* 2 pi rayon)`	`2 pi mul rayon mul`

- En utilisant des noms symboliques le programme devient plus clair et plus facilement modifiable

Page 18

Abstraction

- **En C:**

```
void etoiles() { printf("***"); }

etoiles();
printf("Notes de l'examen");
etoiles();
```

- **En Postscript:**

```
/etoiles { (***) show } def

etoiles
(Notes de l'examen) show
etoiles
```

- Concepts de: **définition de fonction** et **appel de fonction**

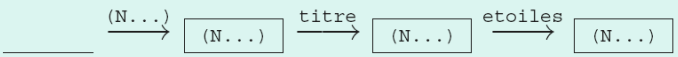
Page 19

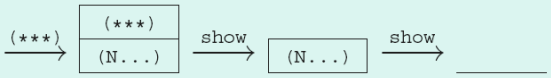
Paramètres

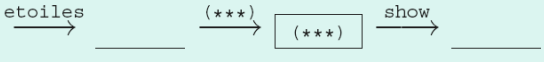
- **En Postscript:**

```
/etoiles { (***) show } def
/titre { etoiles show etoiles } def

(Notes de l'examen) titre
(Liste d'adresses) titre
```

- Exécution:







Page 20

Traitements conditionnels

- Le résultat Booléen d'un test permet à la forme conditionnelle de choisir un de deux traitements
- **En C:**

```
if (x<0)
  printf("x est négatif");
else
  printf("x est zéro ou positif");
```

- **En Scheme:**

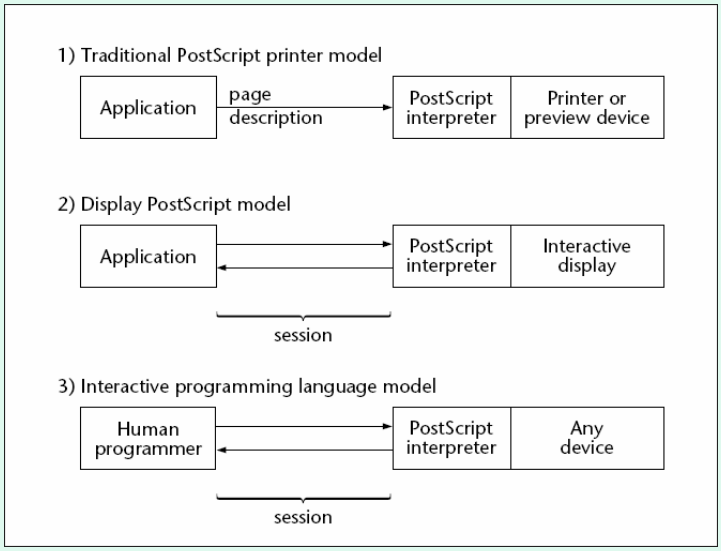
```
(if (< x 0)
    (display "x est négatif")
    (display "x est zéro ou positif"))
```

- **En Postscript:**

```
x 0 lt
  { (x est négatif) show }
  { (x est zéro ou positif) show } ifelse
```

Page 21

Printer vs. Display



Page 22

POSTSCRIPT Imaging Model

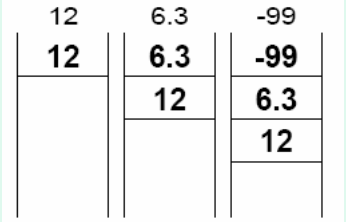
- **Current Page:** the *current page* is the "ideal page" on which POSTSCRIPT draws.
- **Current Path:** The *current path* is a set of connected and disconnected points, lines, and curves, etc.
- **Clipping Path:** The *current clipping path* is the boundary of the area that may be drawn upon.

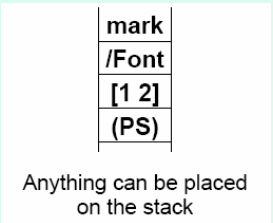
THE POSTSCRIPT STACK

- A stack is a piece of memory set aside for data which is to be immediately used by POSTSCRIPT. This memory area is organized in such a way that the last item put in is the first item available to be removed.
- This type of data structure is referred to as a *last in, first out* or *LIFO* stack.

Putting Numbers on the Stack

- 12 6.3 -99
- 1. Push the number *12* onto the stack
- 2. Place *6.3* on the stack, pushing *12* to the next position down.
- 3. Put *-99* onto the stack, pushing the first two numbers down one place.

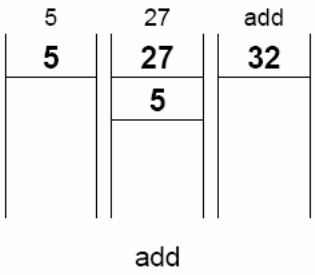




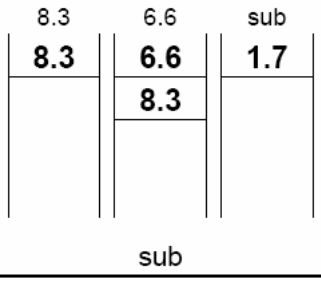
Page 25

ARITHMETIC

- **add** and **sub**
- 5 27 add



- 8.3 6.6 sub



Page 26

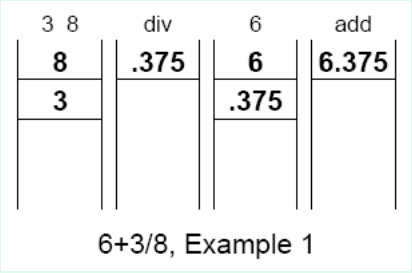
Other Arithmetic Operators

- **div** Divide the second number on the stack by the top number on the stack. For example, 13 8 **div** => 1.625
- **idiv** Divide the second number on the stack by the top number on the stack; only the integral part of the quotient is retained.

  25 3 **idiv** => 8
- **mod** 12 10 **mod** => 2
- **mul** 6 8 **mul** => 48
- **neg** -27 **neg** => 27

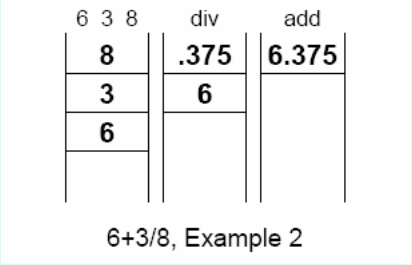
More-Complex Arithmetic

- 6 + (3/8)
- 3 8 div 6 add



6+3/8, Example 1

- 6 3 8 div add



6+3/8, Example 2

Stack Operators

- **clear** Removes all items from the stack.
  6 8 12 **clear** => —
- **dup** Duplicates the top item on the stack.
  6 **dup** => 6 6
- **pop** Remove the the top element from the stack.
  17 8 **pop** => 17
- **roll** Roll stack contents. Take two numbers from the stack. The top number tells POSTSCRIPT how many times and in which direction to rotate the stack; the second number is how many items are to be rotated.
  7 8 9 3 1 **roll** => 9 7 8
  7 8 9 3 -1 **roll** => 8 9 7

Stack Operators

- **==**

The **==** operator removes the top item from the stack and echos it over a communications channel

- **pstack**

This operator prints the contents of the entire stack. Unlike the **==** operator, **pstack** does not remove any of the stack's contents.

OPERATOR SUMMARY

Stack Operators

clear $ob_1 \ldots ob_i \Rightarrow$ —
Remove all stack contents

dup $ob \Rightarrow ob\ ob$
Duplicate top of stack

exch $ob_1\ ob_2 \Rightarrow ob_2\ ob_1$
Reverse order of top two objects on stack

pop $ob_1\ ob_2 \Rightarrow ob_1$
Remove top of stack

roll $ob_{n-1} \ldots ob_0\ n\ j \Rightarrow ob_{(j-1) \bmod n} \ldots ob_0\ ob_{n-1} \ldots ob_{j \bmod n}$
Rotate *n* elements *j* times

Math Operators

add $n_1\ n_2 \Rightarrow n_1+n_2$
Add two numbers

div $n_1\ n_2 \Rightarrow n_1+n_2$
Divide two numbers

idiv $n_1\ n_2 \Rightarrow \text{int}(n_1+n_2)$
Integer divide

mod $n_1\ n_2 \Rightarrow (n_1 \text{ MOD } n_2)$
Modulus

mul $n_1\ n_2 \Rightarrow n_1 \times n_2$
Multiply two numbers

sub $n_1\ n_2 \Rightarrow n_1 - n_2$
Subtract two numbers

Interactive Operators

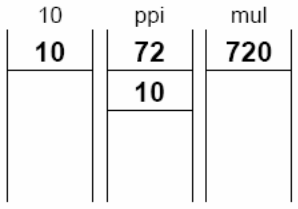
== $ob \Rightarrow$ —
Destructively display top of stack

pstack $ob_1 \ldots ob_i \Rightarrow ob_1 \ldots ob_i$
Display stack contents

DEFINING VARIABLES AND PROCEDURES

/ppi 72 def

10 ppi mul



Using a Variable

1. Push *10* on the stack,
2. Search the dictionary stack for the key *ppi* and put its value,*72*, on the stack,
3. Multiply the top two stack items together, leaving theirproduct on the stack.

```
% ----- Define box procedure ---
/box
{ 72 0 rlineto
0 72 rlineto
-72 0 rlineto
Overlapping Boxes
closepath } def

% --------- Begin Program -----------
newpath % First box
252 324 moveto box
0 setgray fill
newpath % Second box
270 360 moveto box
.4 setgray fill
newpath % Third box
288 396 moveto box
.8 setgray fill
showpage
```

Page 33

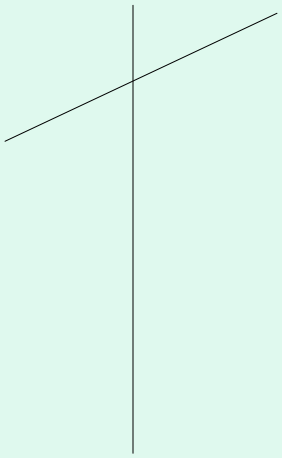
DRAWING LINES

```
newpath
144 72 moveto
144 432 lineto
stroke
showpage
```

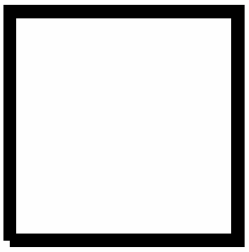
Page 34

DRAWING LINES

```
newpath
72 360 moveto
144 72 rlineto
144 432 moveto
0 -216 rlineto
stroke
showpage
```



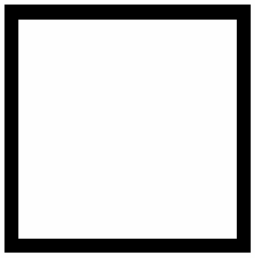
A Box



A Box

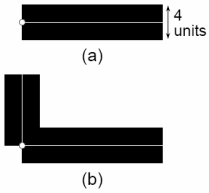
```
newpath
  270 360 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  –72 0 rlineto
  4 setlinewidth
stroke showpage
```

A Better Box



A Better Box

```
newpath
  270 360 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  closepath
  4 setlinewidth
stroke showpage
```



Page 37

FILLED SHAPES

A Filled Box

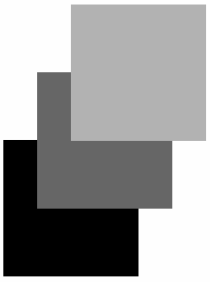
```
newpath
  270 360 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  closepath
  fill
showpage
```

A Gray Box

```
newpath
  270 360 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  closepath
  .5 setgray
  fill

showpage
```

Page 38

FILLED SHAPES



Overlapping Boxes

```
newpath          %Begin black box
  252 324 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  closepath
  fill

newpath          %Begin gray box
  270 360 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  closepath
  .4 setgray
  fill

newpath          %Begin lighter box
288 396 moveto
  0 72 rlineto
  72 0 rlineto
  0 –72 rlineto
  closepath
  .8 setgray

  fill
showpage         %Send to printer
```


OPERATOR SUMMARY

Path Construction Operators

closepath — ⇒ —
Closes the current path with a straight line to the last *moveto* point

lineto x y ⇒ —
Continue the path with line to *(x,y)*

moveto x y ⇒ —
Set the current point to *(x,y)*

newpath — ⇒ —
Clear the current path

rlineto x y ⇒ —
Relative *lineto* (currentpoint + *(x,y)*)

rmoveto x y ⇒ —
Relative *moveto*

Painting Operators

fill — ⇒ —
Fill current path with the current color

setgray n ⇒ —
Set the current color

setlinewidth n ⇒ —
Set the current line width

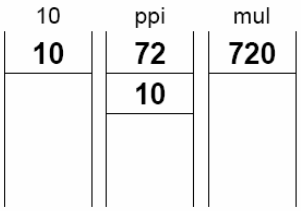
stroke — ⇒ —
Paint the current path with the current color and line width

Output Operators

showpage — ⇒ —
Transfer the current page to the output device

DEFINING VARIABLES AND PROCEDURES

/ppi 72 def
10 ppi mul



Using a Variable

1. Push *10* on the stack,
2. Search the dictionary stack for the key *ppi* and put its value, *72*, on the stack,
3. Multiply the top two stack items together, leaving their product on the stack.

DEFINING VARIABLES AND PROCEDURES

```
% ----- Define box procedure ---
/box
{ 72 0 rlineto
0 72 rlineto
-72 0 rlineto
Overlapping Boxes
closepath } def

% --------- Begin Program -----------
newpath % First box
252 324 moveto box
0 setgray fill
newpath % Second box
270 360 moveto box
.4 setgray fill
newpath % Third box
288 396 moveto box
.8 setgray fill
showpage
```


Overlapping Boxes

OPERATOR SUMMARY

Dictionary Operators

def key value ⇒ —
Associate *key* with *value* in the current dictionary

Using POSTSCRIPT Fonts

1. Find the information describing the font. This information is kept in a *font dictionary*, which contains the information necessary to produce a particular font, including the outline description of each character.
2. Scale the font to the size needed. The size is specified by the minimum vertical separation necessary between lines of text.
3. Establish the scaled font as the *current font*, in which all text is to be printed.

typography

```
/Times-Roman findfont
15 scalefont

setfont
72 200 moveto
(typography) show
showpage
```

Point Sizes

Gorilla

Gorilla

Gorilla

Gorilla

```
/showGorilla      % stack: x y ---

 { moveto (Gorilla) show }def
/Times-Roman findfont  6 scalefont  setfont
72 300 showGorilla
/Times-Roman findfont  10 scalefont  setfont
72 275  showGorilla
/Times-Roman findfont  15 scalefont  setfont
72 250  showGorilla
/Times-Roman findfont  20 scalefont  setfont
72 225 showGorilla

showpage
```


Typefaces

Typefaces
Typefaces
Typefaces
Typefaces

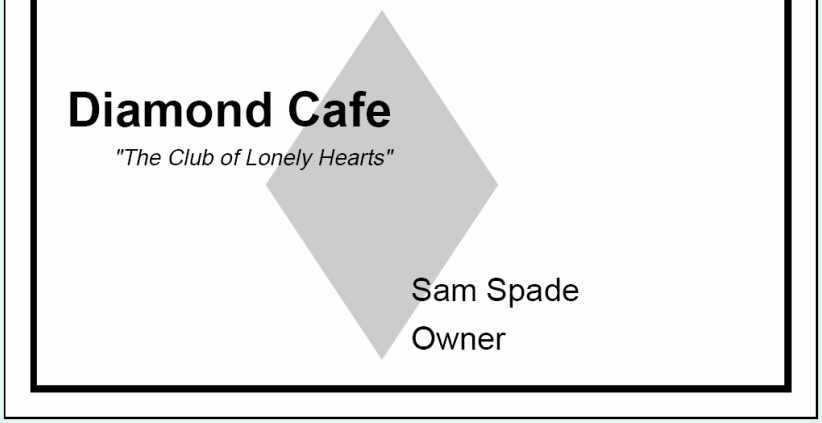
Typefaces
Typefaces
Typefaces
Typefaces

Typefaces
Typefaces
Typefaces
Typefaces

Τψπεφαχεσ

```
%-------- Define Procedures ------------
/vpos  720 def   % vertical position variable
/word  (Typefaces) def              % string variable

/choosefont       % Stack: typeface-name
 { findfont 15 scalefont setfont} def

/newline
 {/vpos vpos 15 sub def             %decrease vpos
  72 vpos moveto } def              %go to that line

/printword          %stk: typeface-name
 { choosefont     %set font
   word show      %show "typefaces"
   newline } def  %go to next line
%------------ Begin Program -----------------

72 vpos moveto %vpos starts as 720
/Times-Roman    printword
/Times-Bold     printword
/Times-Italic   printword
/Times-BoldItalic  printword
newline
/Helvetica    printword
/Helvetica-Bold  printword
/Helvetica-Oblique  printword
/Helvetica-BoldOblique  printword
newline
/Courier    printword
/Courier-Bold  printword
/Courier-Oblique  printword
/Courier-BoldOblique  printword
newline
/Symbol    printword
showpage
```

Graphics and Text (The Blue Book page 42)

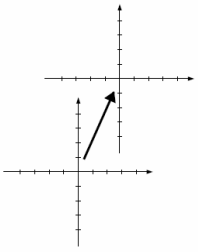


OPERATOR SUMMARY

Character and Font Operators

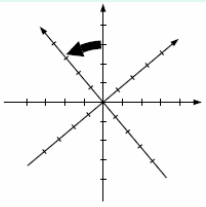
findfont	key ⇒ fdict Return dictionary for named font
scalefont	fdict n ⇒ fdict Return new scaled font dictionary
setfont	fdict ⇒ — Set current font
show	str ⇒ — Print *str* on the current page
stringwidth	str ⇒ x y Return width of *str*

COORDINATE SYSTEMS

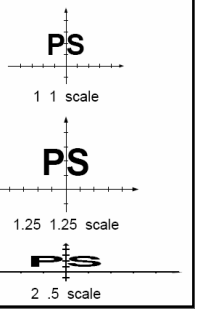


Translation

100 200 translate

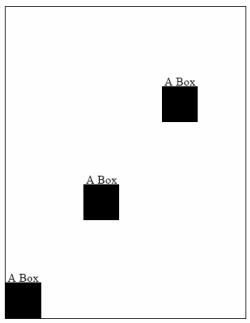


Rotation



Page 49

Translation

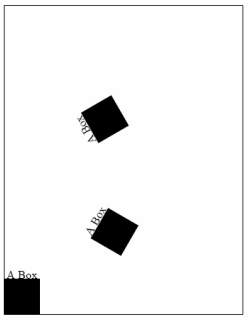


Translated Squares

```
/Times-Roman findfont 30 scalefont setfont

/square          %procedure to draw a
 { newpath       %  filled square
   0 0 moveto
   90 0 lineto   %define a square path
   90 90 lineto
   0 90 lineto
   closepath  fill  %fill it
   6 92 moveto   % & label it
   (A Box) show  } def

square           %do a square
200 250 translate                %move coord. sys.
square           %do another square
200 250 translate                %and move again
square           %do a third square

showpage
```

Page 50

Rotation

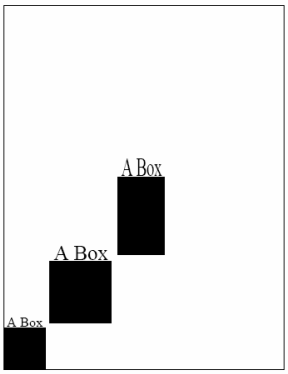


Rotated Squares

```
/Times-Roman findfont 30 scalefont setfont

/square          %procedure from
 { newpath       % previous program
   0 0 moveto
   90 0 lineto
   90 90 lineto
   0 90 lineto
   closepath  fill
   6 92 moveto   %Label the box
   (A Box) show  } def

square           %do a square
300 150 translate                %move coord. sys.
60 rotate        %and rotate it
square           %do it again...
300 150 translate
60 rotate
square           %do a third square

showpage
```

Page 51

Scaling

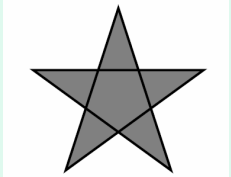


Scaled Squares

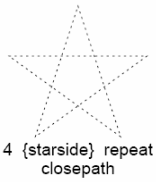
```
/Times-Roman findfont 30 scalefont setfont

/square          %procedure to draw a
 { newpath       % filled square
   0 0 moveto
   90 0 lineto
   90 90 lineto
   0 90 lineto
   closepath  fill
   6 92 moveto   %Label the box
   (A Box) show  } def

square           %do a square
100 100 translate
1.5 1.5 scale
square
100 100 translate
.75 1.25 scale   %non-uniform scaling
square

showpage
```

Page 52

```
/starside
{ 72 0 lineto %add line to path
currentpoint translate %move origin
-144 rotate } def %rotate coord. sys.

/star %stack: x y
{ moveto
currentpoint translate
4 {starside} repeat
closepath
gsave
.5 setgray fill
grestore
stroke } def

200 200 star
showpage
```



Star



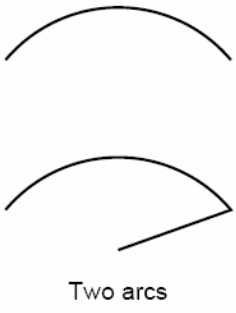
4 {starside} repeat
closepath

fill

grestore

stroke

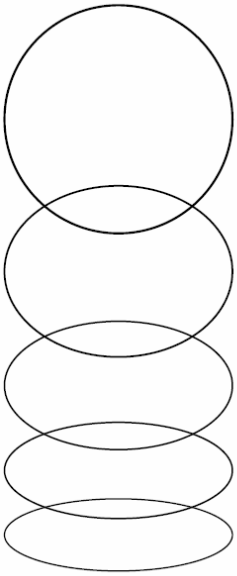
CURVES



Two arcs

```
newpath
300 400 54 40 140 arc stroke

newpath
300 365 moveto
340 345 54 40 140 arc stroke

showpage
```

Circles and Ellipses

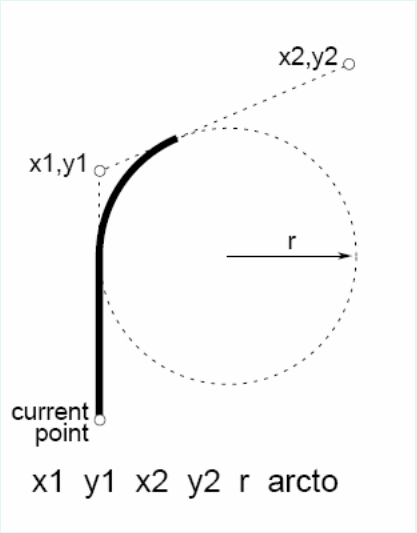


```
/doACircle
 { 0 0 54 0 360 arc stroke } def

/doAnEllipse
 { 1 .75 scale
   doACircle
   stroke } def

300 500 translate  doACircle

4 {0 –72 translate
   doAnEllipse} repeat

showpage
```

Ellipses

Page 55

Rounding Corners



Page 56

Example


```
% ---------- Define Procedures ------------

/Helvetica-Bold findfont  27 scalefont setfont

/fourpops
 { 4 {pop} repeat } def

/background                     %Black background
 { 0 18 moveto                  % with rounded corners
   0 72  108 72  18 arcto fourpops
   108 72  108 0  18 arcto fourpops
   108 0  0 0  18 arcto fourpops
   0 0  0 72  18  arcto fourpops
   fill } def
/moon
 { .6 setgray                   % set gray level
   81 45 18 0 360 arc fill      % draw a circle
   } def                        % end of definition

/omaha
 { 1 setgray
   0 -1 moveto
   1 2 scale                    % double y-scale
   (OMAHA) stringwidth pop      % width of word
   108 exch sub 2 div           % calc. indentation
   0 rmoveto                    % indent
   (OMAHA) show } def           % & print
% ---------- Begin Program ----------------
255 465 translate

background
moon
omaha

showpage
```


OPERATOR SUMMARY

Control Operators

repeat n proc ⇒ —
Execute *proc* *n* times

Coordinate Systems Operators

rotate angle ⇒ —
Rotate user space *angle* degrees counterclockwise about origin

scale x y ⇒ —
Scale user space by *x* horizontally and *y* vertically

translate x y ⇒ —
Move origin of user space to *(x,y)*

Graphics State Operators

grestore — ⇒ —
Restore graphics state from matching *gsave*

gsave — ⇒ —
Save current graphics state

Path Construction Operators

arc $x\ y\ r\ ang_1\ ang_2 \Rightarrow —$
Add counterclockwise arc to current path

arcn $x\ y\ r\ ang_1\ ang_2 \Rightarrow —$
Add clockwise arc to current path

arcto $x_1\ y_1\ x_2\ y_2\ r \Rightarrow xt_1\ yt_1\ xt_2\ yt_2$
Build tangent arc

currentpoint — ⇒ x y
return coordinates of current point

CONDITIONAL EXECUTION

- Comparisons

• **eq** =	• **ne** ≠
• **gt** >	• **lt** <
• **ge** ≥	• **le** ≤

- The **if** Operator

```
/chkforendofline
{ currentpoint pop      %get x-position
612 gt                  %greater than 612?
{0 -12 translate 0 0 moveto} if
} def
```

This procedure obtains the position of the current point and throws away the *y* coordinate. It then compares the remaining *x* coordinate to see if it is beyond the right edge of the current page.

Page 59

CONDITIONAL EXECUTION

- The **ifelse** Operator

bool {op1} {op2} ifelse

```
% ------- Variables & Procedures ---------
/scalefactor 1 def

/counter 0 def
/DecreaseScale
 { scalefactor .2 sub
   /scalefactor exch def } def

/IncreaseCounter
 { /counter counter 1 add def } def

/trappath          %construct a trapezoid
 { 0 0 moveto  90 0 rlineto
   -20 45 rlineto  -50 0 rlineto
   closepath } def

/doATrap
 { gsave
     1 scalefactor scale          %scale vert. axis
     trappath                     %construct path
     counter 2 mod                %is counter even?
     0 eq {.5} {0} ifelse         %choose grey or black
     setgray  fill
   grestore } def                 %restore scale, etc.
% ---------- Begin Program ----------
250 350 translate

5
{IncreaseCounter
 doATrap
 DecreaseScale
 0 20 translate } repeat

showpage
```

Page 60

CONDITIONAL EXECUTION

- The **for** Operator


```
/Times-Italic findfont  30 scalefont setfont

/printZip

 { 0 0 moveto  (Zip) show} def
320  400  translate

.95  –.05  0        % start  incr.  end
{setgray  printZip –1 .5 translate } for

1 setgray printZip

showpage
```

Page 61

OPERATOR SUMMARY

Control Operators

exit — ⇒ —
Exit innermost **for**, **loop**, or **repeat**

for j k l proc ⇒ —
For *i* = *j* to *l* step *k* do *proc*

if bool proc ⇒ —
If *bool* is *true*, then do *proc*

ifelse bool $proc_t$ $proc_f$ ⇒ —
If *bool* is *true* then do $proc_t$, else do $proc_f$

loop proc ⇒ —
Repeat *proc* forever

String and Conversion Operators

string n ⇒ str
Create string of length *n*

cvs ob str ⇒ str
Convert to string

Relational Operators

eq ob_1 ob_2 ⇒ bool
Test for equality

ne ob_1 ob_2 ⇒ bool
Test for inequality

gt n/str_1 n/str_2 ⇒ bool
Test for greater than

ge n/str_1 n/str_2 ⇒ bool
Test for greater than or equal to

lt n/str_1 n/str_2 ⇒ bool
Test for less than

le n/str_1 n/str_2 ⇒ bool
Test for less than or equal to

Page 62

Array Operators

Array Operators

[— ⇒ mark
Start array construction

] mark $ob_0 \ldots ob_i$ ⇒ array
End array construction

aload ary ⇒ $ob_0 \ldots ob_{n-1}$ ary
Get all elements of an array

array n ⇒ ary
Create array of size *n*

astore $ob_0 \ldots ob_{n-1}$ ary ⇒ ary
Put elements from stack into array

Polymorphic Operators

forall ary/dict/str proc ⇒ —
For each element do *proc*

get ary/dict/str index/key ⇒ value
Get value of *index/key* in object

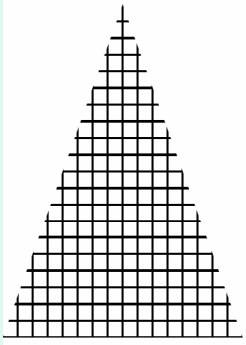
length dict/str/ary ⇒ n
Length of object

put ary/dict/str index/key value ⇒ —
Put *value* into object at index/key

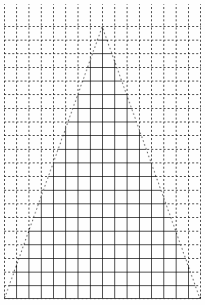
Stack Operators

mark — ⇒ mark
Push mark onto stack (same as [)

CLIPPING PATH



```
% ---- Procedures ----
/trianglepath
 { newpath
   0 0 moveto
   144 0 lineto
   72 200 lineto

   closepath } def

 /verticals
  { newpath
    0 9 144
    { 0 moveto
      0 216 rlineto } for
    stroke } def
 /horizontals
  { newpath
    0  10  200
    { 0 exch moveto
      144 0 rlineto } for
    stroke } def
 % ---- Begin Program ---
 230 300 translate
 trianglepath  clip                %set clipping path
 verticals          %Do grid
 horizontals

 showpage
```



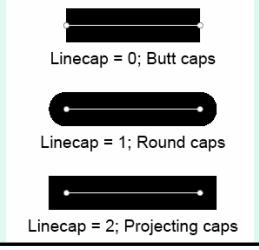
Only the part of the grid that falls within the triangular clipping path reaches the current page.

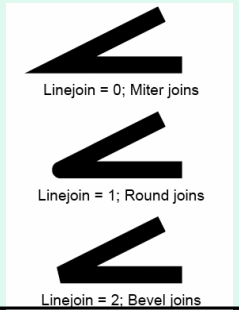
CLIPPING PATH


```
% -------- Procedures ----------
/Times-BoldItalic findfont
     27 scalefont setfont

/rays
 { 0 1.5 179
   {  gsave
        rotate
        0 0 moveto  108 0 lineto
        stroke
      grestore
   } for
 } def
% -------- Begin Program ---------
300 400 translate

.25 setlinewidth

newpath
0 0 moveto
(StarLines) true
charpath clip
newpath
54 –15 translate
rays

showpage
```


LINE-DRAWING DETAILS

- **setlinecap** Determines the appearance of line segment ends.
- **setlinejoin** Determines the method by which different line segments are joined.
- **setdash** Determines the pattern for dashed lines.





OPERATOR SUMMARY

Graphics State Operators

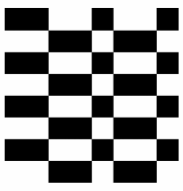
clip	— ⇒ — Set clipping boundary to current path
setdash	ary n ⇒ — Set dash array
setlinecap	0/1/2 ⇒ — Set shape of stroked line ends
setlinejoin	0/1/2 ⇒ — Set shape of stroked line joins
setmiterlimit	num ⇒ — Set maximum miter ratio

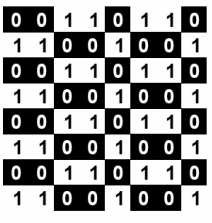
Page 67

image

- A Binary Image

300 400 translate %Move image to middle of page
72 72 scale %Make image one inch on a side
8 8 1 [8 0 0 8 0 0] {<c936>} image
showpage

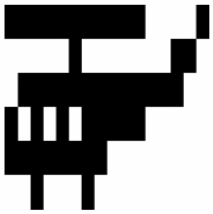




8 8 1 [8 0 0 8 0 0] {<c936>} image

Page 68

image



```
/Helicopter
 <dd ff 00 ff 54 1f 80 03 fb f9 00 1e> def

300 400 translate
72 72 scale

16 6 1 [16 0 0 6 0 0] {Helicopter} image

showpage
```

Page 69

OPERATOR SUMMARY

Graphics Output Operators

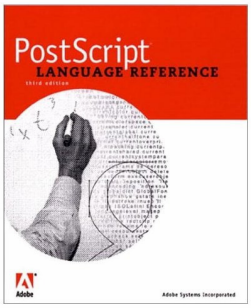
image scanlen #lines b/p [transform] {proc} ⇒ —
Render image onto current page

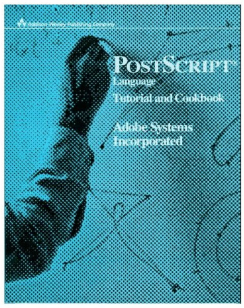
Polymorphic Operators

putinterval obj_1 i obj_2 ⇒ —
Copy obj_2 into obj_1 starting at *i*

Page 70

Références





- On-Line:

http://mdwconsulting.mdwserver.net/postscript/postscript-operators/

Page 71

TP1 (The Blue Book page 42)

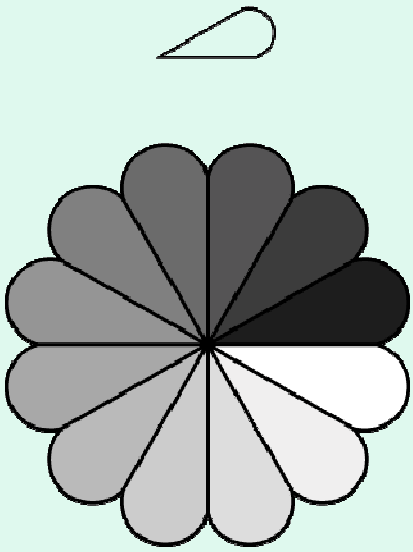
Diamond Cafe

"The Club of Lonely Hearts"

Sam Spade
Owner

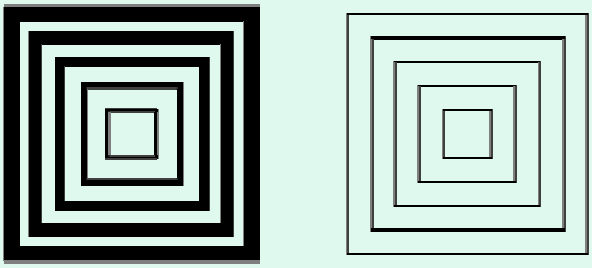
Page 72

TP2 (The Blue Book page 133)



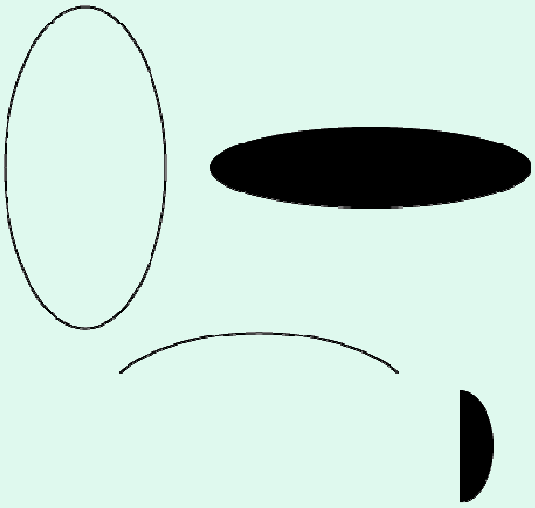
Page 73

TP3 (The Blue Book page 135)



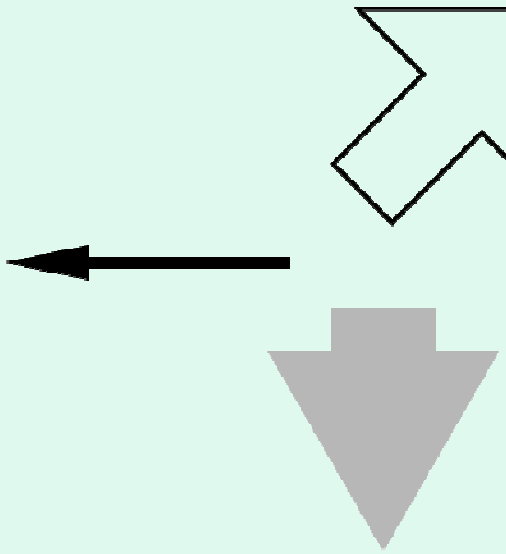
Page 74

TP4 (The Blue Book page 137)



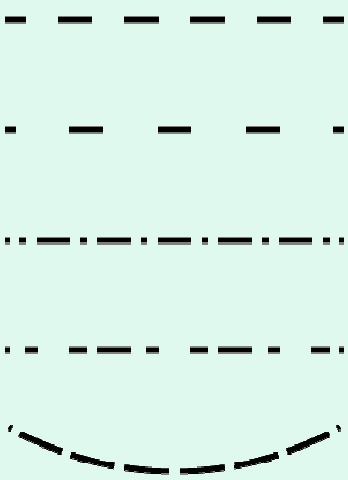
Page 75

TP35 (The Blue Book page 141)



Page 76

TP6 (The Blue Book page 145)



Page 77

TP7 (The Blue Book page 165)

TEXT POSITIONED VERTICALLY

SHOULD BE CENTERED ON

A COMMON CENTER LINE.

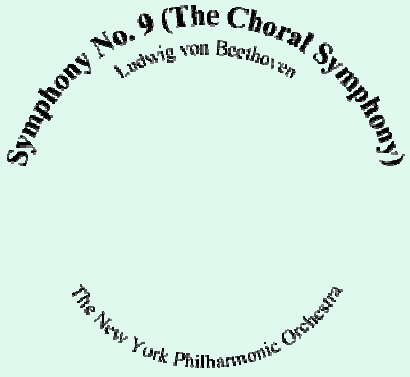
VERTICAL TEXT IN CAPITAL

LETTERS HAS MORE EVEN

spacing than lower case letters.

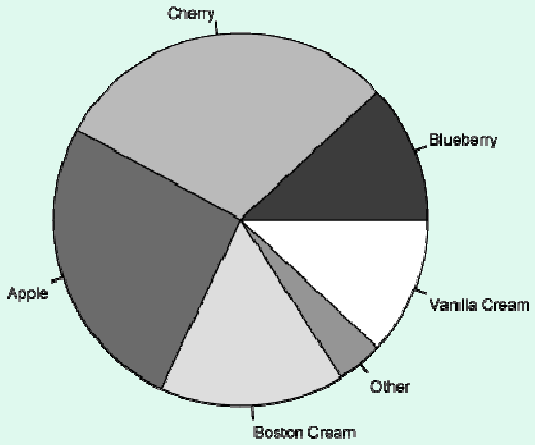
Page 78

TP8 (The Blue Book page 167)



Page 79

TP9 (The Blue Book page 187)



January Pie Sales

Page 80