Pugo Pedagog presenterar:

PostScript (en uppLYSning)

GÖK-SKÖTARNA

LiTH

UppLYSning Postscript

Anders ”Pugo” Karlsson
<http://www.pugo.org/>
pugo@pugo.org

Upplägg

- Postscripts historia
- Språket Postscript
- Rita i Postscript
- Generering av Postscript
- Korrekta dokument

Postscript, hur det började

- Som mycket annat, Xerox Parc
(GUI, laserskrivare, ethernet, etc.)
- John Warnock utvecklar i början av 1980–talet
språket Interpress för att styra Xerox skrivare
- Xerox ger inget stöd, efter två år ger John och hans
chef Chuck Geschke upp och startar eget
- Adobe startas, döpt efter en liten vik bakom
Warnocks hus i Los Altos, Californien

Adobe och Postscript

- Först tänkte de bygga avancerade skrivare runt sitt system men de ändrar sig till att bygga system till andras skrivare.
- Postscript Level 1 tog ca 20 manår att utveckla och lanserades 1984.

Varför slog Postscript?

- Apple!
  - 1985 började Macintosh–försäljningen avta och Apple behövde en killer–app.
  - Steve Jobs gillade tekniken och investerade $2.5M i Adobe för att de skulle utveckla en Postscript–miljö för Apple LaserWriter
- Aldus PageMaker
- Desktop Publishing var fött!
  - LaserWriter, Postscript och PageMaker räddade Apple och gjorde Adobe och Aldus rika.

Postscript Level 2

- Efter succén med Postscript fortsatte utvecklingen med Postscript level 2:
  - Bättre VM, bättre minneshantering
  - Färgseparering, bilduppackning, komposit–fonter, bättre cachening av fonter och mönster
- Långsamt genomslag, nästan tio år

Postscript 3

- Adobe tycker det är coolare med ”Postscript 3” än ”Postscript Level 3”.
- Nya features:
  - Support för mer än 256 gråskalor per färg
  - Support för PDF
  - Bättre färgseparering
  - Hype–trams: ”Web–ready printing”

Varför Postscript?

- Innan Postscript:
  - Gytter av standarder
  - Inkompatibilitet!
  - Bitmappad halvtaskig grafik
- Postscript ger:
  - Flexibelt språk för att beskriva avancerade sidor
  - Välspridd standard
  - Portabilitet, device–oberoende!
  - Vektorbaserad grafik

Nackdelar

- Postscript–tolkar kräver intelligenta skrivare (De första LaserWriter–skrivarna med Postscript hade bättre CPU än de Mac:ar de servade).
- Kostar därför mer pengar än ”dumma” skrivare (exempelvis de korkade Windows–skrivare som finns på marknaden)
- Ett avancerat språk gör det svårt eller omöjligt att validera ett dokuments korrekthet utan att köra det och kontrollera manuellt.

Språket Postscript

- Stackbaserat
(operand stack, dictionary stack, execution stack)
- Interpreterande
- Operationer är postfixa, precis som en RPN–kalkylator
- Har en hel del likheter med Forth
- Körs i en virtuell maskin

Stacken, Postscripts hjärta

- Alla operationer är stackbaserade, funktioner hämtar sina argument på stacken och lägger tillbaka resultatet.
- Internt hanteras funktioner som exekveras på en egen stack (execution stack).
  - Viktigt: Håll reda på stacken! Försäkra att dina funktioner inte lämnar rester!
-
```
[pugo@pudas pugo]$ gs -dNODISPLAY
GNU Ghostscript 5.50 (2000-2-13)
Copyright (C) 1998 Aladdin Enterprises, Menlo Park, CA.  All rights reserved.
This software comes with NO WARRANTY: see the file COPYING for details.
GS>1 2 3
GS<3>
GS<3>stack
3
2
1
GS<3>
```

Räkna med stacken

- Räkneoperationer på stacken:
  - Add, sub, mul, div

```
GS<3>stack
3
2
1
GS<3>
GS<3>mul add
GS<1>stack
7
GS<1>
```

  - 15 * (7 – 3):

```
GS>15 7 3 sub mul =
60
GS>
```

Stackoperationer

- Några av operationerna för att hantera stacken:
  - `Clear` – töm stacken
  - `count` – räkna antalet saker på stacken och lägg summan på stacken
  - `dup` – duplicera översta elementet på stacken
  - `exch` – skifta plats på de två övesta objekten
  - `index` – kopiera ett värde nere i stacken och lägg överst
  - `pop` – kasta bort det översta objektet

Ghostscript

Literals – executables

- Interpretatorn skiljer mellan ”literals” och ”executables”.
- Literaler hanteras strikt som data och läggs på stacken.
- Executables exekveras.
  - Integers, flyttal och strängkonstanter är alltid literals.
  - Namn är literals om de börjar med ”/”.

Deklarera ord

- Variabler är värden som bundits till ord.
- Funktioner är kod som bundits till ord.

- Syntax:
  `/ord <objekt> def`
- Exempel:

```
GS>/tretton 13 def
tretton =
13
GS>/fjorton tretton 1 add def
GS>fjorton =
14
GS>
```

Dictionary

- En dictionary är en samling namn–värde–par inom ”<<” och ”>>”.
- Alla bundna ord lagras i en dictionary
- Alla öppna dictionarys lagras på en dictionary–stack som söks igenom när ord skall slås upp.
- Typsnitt finns i en font dictionary.
- Kan skapas med hjälp av funktioner, eller direkt.

Att använda en dictionary

- Funktioner för att skapa dictionary;

```
/extensiondict 3 dict def
extensiondict begin
   /a  (adam) def
   /b  (bertil) def
   /c  (caesar) def
end
```

- En dictionary kan skapas direkt:

```
GS>/x << /a (adam) /b (bertil) /c (caesar) >> def
GS>x type =
dicttype
```

- Sökning kan ske med ”known” och ”get”:

```
GS>x length =
3
GS>x /a known =
true
GS>x /a get
GS<1>stack
adam
```

Strängar

- Strängar skrivs vanligen som text inom (parentes)
- Går även att skriva som hexkod: `<414243>`
- Escape–koder:

```
\n     Newline
\r     Carriage return
\t     Horizontal TAB
\b     Backspace
\f     Form feed
\\     Backslash
\(     Left parenthesis
\)     Right parenthesis
\ddd   The character code ddd, where ddd is in octal.
```

- Kan skrivas ut med ”print”: `(foo\n) print`

Kodblock

- Kodblock skrivs genom att skriva kod mellan `{ måsvingar }`.
- Är egentligen en exekverbar array
- Exempel:

```
GS>/double { 2 mul } def
GS>42 double
GS<1>stack
84
GS<1>
```

- ```
  %!
  /fak  % <n> -- <n!>
  { dup 1 eq
      { } { dup 1 sub fak mul } ifelse
  } def

  5 fak

  GS<1>stack
  120
  ```
  ```

Bind

- För att snabba upp ett kodblock används med fördel funktionen `bind` på kodblocket. Det ersätter de ord som går att ersätta med deras betydelse.
- Detta innebär att man slipper göra en uppslagning av ordet och exekvering av detta varje gång det körs.
- Exempel:

```
/prtstr
 { dup stringwidth pop currentpoint pop    % get string width and current point
   add RM gt                               % sum > right margin?
   {newline} if                            % if so, next line
   show } bind def
```

Sanningsvärden

- Sanningsvärden:
  - `<x> <y> eq  –  equal, sant om x och y är lika`
  - `<x> <y> ne  –  not equal, sant om x och y är olika`
  - `<x> <y> gt  –  greater than, sant om x är större än y`
  - `<x> <y> ge  –  greater than or equal, sant om x är större än eller lika med y`
  - `<x> <y> lt  –  less than, sant om x är mindre än y`
  - `<x> <y> le  –  less than or equal, sant om x är mindre än eller lika med y`
- Operationera ’or’, ’and’, ’xor’ och ’not’ kan användas för att bygga ihop operationer:

```
GS>1 2 gt not =
true
```

```
GS>1 2 lt 5 5 eq and =
true
```

Styrfunktioner – if, ifelse

- Logiska val med ”if” och ”ifelse”
  - `<sanningsvärde> <kodblock> if`
  - `<sanningsvärde> <kodblock> <kodblock> ifelse`

- Exempel:

```
1 2 lt { (Basun\n) print } if
Basun


GS<1>1 2 lt { (foo\n) } { (bar\n) } ifelse print
foo

GS<1>2 1 lt { (foo\n) } { (bar\n) } ifelse print
bar
```

For–Loopar

- Syntax:
  `[startvärde] [inkrement] [stoppvärde] {funktion} for`
- Aktuellt värde läggs på stacken
  (kör ”pop” om det inte behövs!)
- Exempel:

```
GS<1>1 1 3 { = } for
1
2
3

GS<1>3 -1 1 { = } for
3
2
1
```

Forall–loopar

- För att iterera över en mängd används ”forall”:

```
<array>   {kodblock}  forall   % varje element
<sträng> {kodblock}  forall   % varje bokstav
<dict>    {kodblock}  forall   % varje värdepar
```

- Exempel:

```
[ (x) (y) (z) ] { print } forall
xyz

(basun) { = } forall
98
97
115
117
110
```

Andra loopar

- ”repeat” kan användas för att köra något upprepat:
  `<antal> {kodblock} repeat`
- ”loop” kör ett kodblock tills ”exit” exekveras:
  `{kodblock} loop`
- Exempel:

```
GS>10 { (x) print } repeat
xxxxxxxxxx

GS>10 { 1 sub dup 0 eq { exit } { dup = } ifelse } loop =
9
8
7
.
.
1
0
```

Exempel på loop

- En triangel:

```
%!PS
% print a triangle
1 1 10 {                                 % Loopa tio varv
   1 exch 1 exch { pop (*) print} for    % Skriv ut *
   (\n) print                            % Skriv radbrytning
} for
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Att börja rita

- I PostScript ritar man pather.
- För att rita en figur gör man:
  - Starta en path med ”newpath”
  - Skapa figuren med hjälp av linjer och bågar (behöver inte hänga samman)
  - Rita ut med ”stroke” och fyll med ”fill”

En enkel box

- En enkel box:

```
%!
/inch {72 mul} def       % Convert inches->points (1/72 inch)

newpath                  % Start a new path
1 inch 1 inch moveto     % an inch in from the lower left
2 inch 1 inch lineto     % bottom side
2 inch 2 inch lineto     % right side
1 inch 2 inch lineto     % top side
closepath                % Automatically add left side to close path
stroke                   % Draw the box on the paper
showpage                 % We're done... eject the page
```

Exempel

Relativa koordinater

- Inne i ett objekt använder man med fördel relativa koordinater:

```
%!
/inch {72 mul} def            % Convert inches->points (1/72 inch)

/box    %  <x> <y> box  --
{
   newpath
   moveto                     % This is very important!
   0 inch  1  inch  rlineto
   1 inch  0  inch  rlineto
   0 inch -1  inch  rlineto
   closepath
} def

0 1 inch 600 { dup box fill } for
```

- Observera att det är viktigt att moveto körs <u>inom</u> en path för att relativa koordinater skall fungera!

Exempel

Bågar och böjar

- En båge skapas med ”arc”
  `<x> <y> <r> <a1> <a2> arc`

```
%!

/inch {72 mul} def       % Convert inches->points (1/72 inch)

/myarc   %  <x> <y> myarc  --
{
   newpath
   30 -52 180 arc   % <x> <y> <r> <a1> <a2> arc
   stroke
} def

100 25 700 { dup 2 div exch myarc } for
```

Exempel

Bågar och linjer

- För att rita vidare på en path fortsätter man med linjer och det man vill:

```
%!

/inch {72 mul} def      % Convert inches->points (1/72 inch)

/myarc   %  <x> <y> myarc  --
{
   newpath
   30 -100 190 arc    % <x> <y> <r> <a1> <a2> arc
   10 10 lineto
   closepath
   stroke
} def

200 200 myarc
```

Exempel

Translationer

- För att rita avancerad grafik underlättar det ofta mycket att translatera koordinatsystemet tillfälligt till den punkt där grafiken skall ritas.
  - Förflyttningar sker med: `<x> <y> translate`
  - Rotationer sker med: `<z> rotate`
  - Skalningar sker med `<sx> <sy> scale`
- För att kunna återgå från translationen sparar man ”state” innan och återgår till den sparade efteråt:
  - State sparas med hjälp av `gsave`. Detta sparar state på en state–stack.
  - `grestore` plockar en state från stacken och återgår till den.

Exempel på translation

- I följande exempel translateras koordinatsystemet för att kunna nå ”mitten”:

```
%!PS-Adobe-3.0

/pacman
{
   gsave                              % Save state (1)
   0 0 0 setrgbcolor                  % Set color to black
   moveto currentpoint translate      % Move and translate
   0 0 100 30 -30 arc                 % Draw an arc
   0 0 lineto                         % Line to the centre of the arc
   gsave                              % Save state (2)
      200 200 0 setrgbcolor           % Set color to yellow
      fill                            % Fill out creation
   grestore                           % Restore state (2)
   stroke                             % Draw the outline
   grestore                           % Restore state (1)
} def

250 300 pacman
```

Exempel

Trigonometri

- Sinus och cosinus:

```
<grader> sin

<grader> cos
```

```
%!
/n 30 def
/step 360 n div def
/dot { newpath 1 0 360 arc closepath fill } def

300 400 translate      % Start a bit up right on the paper

0 step 180 { pop
    0 0.3 360  { /x exch def
                 x cos 250 mul x 2 mul sin 200 mul dot
               } for
    step rotate
} for



showpage
```

Exempel

Rekursion

- Rekursion är tillåtet (och användbart). Tänk dock på att det alltid är samma ”scope” – stacken!

```
/depth 0 def
/maxdepth 10 def
/down {/depth depth 1 add def} def
/up   {/depth depth 1 sub def} def

/DoLine
 {0 144 rlineto currentpoint
  stroke translate 0 0 moveto } def

/FractArrow
 {gsave 0.7 0.7 scale                % Save state and scale down
  10 setlinewidth                    % Set linewidth
  down DoLine                        % Down one level
  depth maxdepth le                  % Confirm that we should continue
  { 135 rotate FractArrow            % Rotate and recurse
    -270 rotate FractArrow } if      % Rotate and recurse
  up grestore } def                  % Up and restore

300 400 moveto                       % Move to a good place to start
3.0 3.0 scale                        % Scale up so we see something
FractArrow                           % Start the show!
stroke                               % Stroke the lines
showpage                             % Print it.. Poor printer!
```

Exempel

Gråskalor

- För att ändra gråskalor används `setgray`

```
%!

/inch {72 mul} def      % Convert inches->points (1/72 inch)

/box   %  <x> <y> box  --
{
   newpath
   inch moveto
   0 inch 1 inch rlineto
   1 inch 0 inch rlineto
   0 inch -1 inch rlineto
   closepath fill
} def

0 1 10 { dup 10 div setgray          % Calc and set grayvalue
         300 exch box stroke } for    % Paint box

showpage
```

Exempel

90–talet sänds i färg!

- Färg sätts med `sethsbcolor` eller `setrgbcolor`

```
<h> <s> <b> sethsbcolor
<r> <g> <b> setrgbcolor
```

```
%!
/n 32 def                                          % Number of boxes
/unit {500 n div mul} def                          % Width / boxes

/box   %  <x> <y> box  --
{  newpath moveto
   0 unit 1 unit rlineto 1 unit 0 unit rlineto
   0 unit -1 unit rlineto closepath fill stroke
} def

50 50 translate                                    % Start a bit up right

0 1 n { /x exch def                                % Loop x values
        0 1 n { /y exch def                        % Loop y values
                x n div y n div 0 setrgbcolor      % Set color to x, y, 0
                x unit y unit box                  % Draw a box at x, y
              } for
      } for

showpage
```

Exempel

Lite slump

- Slumptal kan skapas med `rand` (0 < x < 2^31):

```
%!PS-Adobe-3.0

/pac
{
   gsave
   0 0 0 setrgbcolor
   moveto currentpoint translate
   0 0 10 30 -30 arc
   0 0 lineto
   gsave
      200 200 0 setrgbcolor
      fill
   grestore stroke grestore
} def

0 1 500 {                     % Loop 500 times
          pop                 % Throw away the for-value
          rand 620 mod        % Generate random x coordinate (0-619)
          rand 800 mod        % Generate random y coordinate (0-799)
          pac } for

showpage
```

Exempel

Randomania!

- Lite mer slump:

```
%!PS-Adobe-3.0

/pac
{   gsave 0 0 0 setrgbcolor
    moveto currentpoint translate          % get x and y from stack
    rotate                                 % get angle from stack
    0 0 20 30 -30 arc 0 0 lineto gsave
    setrgbcolor fill                       % get 3 color values from stack
    grestore stroke grestore
} def

0 1 1500 { pop
           rand 255 mod 255 div            % Red color (0-1)
           rand 255 mod 255 div            % Green color (0-1)
           rand 255 mod 255 div            % Blue color (0-1)
           rand 360 mod                    % Rotate angle (0-360)
           rand 620 mod                    % x coordinate
           rand 800 mod                    % y coordinate
           pac } for


showpage
```

Exempel

Text

- Text är grafik!
- Först måste man öppna och välja en font
  `<namn> findfont <storlek> scalefont setfont`
- Text skrivs ut med: `(string) show`

```
%!

/Times-Roman findfont 60 scalefont setfont
50 100 moveto (This is a text!) show

/Courier findfont 60 scalefont setfont
50 200 moveto (This is a text!) show

/Helvetica findfont 60 scalefont setfont
50 300 moveto (This is a text!) show

showpage
```

Exempel

Roterad text

- Text roteras precis som annan grafik:

```
%!

/Times-Roman findfont 50 scalefont setfont
300 350 moveto

0 10 360 { gsave
           rotate (This is a text!) show
           grestore
         } for

showpage
```

Exempel

Typsnitt

- Det finns 35 standardtypsnitt:

```
AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bookman-Demi
Bookman-DemiItalic
Bookman-Light
Bookman-LightItalic
Courier-Bold
Courier-BoldOblique
Courier
Courier-Oblique
Helvetica-Bold
Helvetica-BoldOblique
Helvetica-NarrowBold
Helvetica-NarrowBoldOblique
Helvetica
Helvetica-Oblique
Helvetica-Narrow
Helvetica-NarrowOblique
NewCenturySchlbk-Bold
NewCenturySchlbk-BoldItalic
NewCenturySchlbk-Italic
NewCenturySchlbk-Roman
Palatino-Bold
Palatino-BoldItalic
Palatino-Italic
Palatino-Roman
Symbol
Times-Bold
Times-BoldItalic
Times-Italic
Times-Roman
ZapfChancery-MediumItalic
ZapfDingbats
```

Generera Postscript

- Det är ganska lätt att generera Postscript från program genom att först definiera de funktioner som behövs för de features man behöver och sedan använda detta.
- ```
  #!/bin/bash

  echo "%!\n"
  echo "/Times-Roman findfont 10 scalefont setfont"
  l=750

  while read i; do
      if expr $l '<' 50 > /dev/null; then echo "showpage"; l=750; fi
      echo "72 $l moveto ($i) show"
      l=`expr $l '-' 10`
  done
  ```
  ```

Låt tolken göra jobbet!

```
%!PS-Adobe-3.0
/LM 15 def  /RM 578 def  /TM 760 def   /BM 10 def     % Margins
/fontsize 14 def  /ypos TM def   /lineheight fontsize def

/newline
 { currentpoint exch pop lineheight sub BM lt   % check if below bottom
   { showpage LM TM moveto /ypos TM def }
   { ypos lineheight sub /ypos exch def LM ypos moveto } ifelse
} bind def

/prtstr
 { dup stringwidth pop currentpoint pop  % get string width and current point
   add RM gt                    % sum > right margin?
   {newline} if                 % if so, next line
   show } bind def

/Courier findfont fontsize scalefont setfont
LM TM moveto

(This ) prtstr (text ) prtstr (will ) prtstr (be ) prtstr (formatted ) prtstr
(by ) prtstr (the ) prtstr (PostScript ) prtstr (environment. ) prtstr (It )
prtstr (will ) prtstr (break ) prtstr (lines ) prtstr (when ) prtstr (they )
prtstr (reach ) prtstr (the ) prtstr (right ) prtstr (margin ) prtstr (and )
prtstr (break ) prtstr (the ) prtstr (pages ) prtstr (when ) prtstr (the ) prtstr
(text ) prtstr (reaches ) prtstr (the ) prtstr (bottom. ) prtstr
```

Exempel

Objektorientering, PyPS

```
#!/usr/bin/python
import ps

text = """                        KÖP EN KATT! FÅ 21 GRATIS!
Just det! Du hörde rätt! Köp denna vackra katt för bara 85 kronor (inkl. porto) och
du får hennes kull på 27 bedårande kattungar helt gratis! En underbar kattkamrat som
"""

# Create a new A4-sized document
my_doc = ps.document( title='Test Document', papertype='a4' )

page = ps.page()                            # Create a page
c = ps.container_grid( 3, 3 )               # Create a 3*3 container
c.add_item( 0, 0, ps.text( text ))          # First square is a normal text
c.add_item( 1, 0, ps.text( text, font='ZapfChancery-MediumItalic' ))
c.add_item( 2, 0, ps.image() )                            # An image
c.add_item( 0, 1, ps.text( text, font='Palatino-BoldItalic' ))
c.add_item( 1, 1, ps.eps(file='eps/illusion.epsi') )    # An EPS file
c.add_item( 2, 1, ps.text( text, font='Bookman-DemiItalic' ))
c.add_item( 0, 2, ps.text( text, font='AvantGarde-Book' ))
c.add_item( 1, 2, ps.text( text, font='Helvetica' ))
c.add_item( 2, 2, ps.mono_text( text ))
page.add_item(c)                            # Add container to page
my_doc.add_page( page )                     # Add page to document

print my_doc.generate_ps()                  # Generate PostScript
```

Exempel

Korrekta dokument

- Postscript–dokument bör innehålla en del meta–data om dokumentet:

```
%!PS-Adobe-3.0
%%Creator: Python-ps 0.1, (C) 2000 Anders Karlsson <pugo@pugo.org>, License: GPL
%%CreationDate: Tue Jul 18 18:27:28 2000
%%Title: Test Document
%%Pages: 7
%%PageOrder: Ascend
%%BoundingBox: 0 0 595 842
%%Orientation: Portrait
%%EndComments
%%BeginProlog
% --> Prolog med funktionsdeklarationer
%%EndProlog

%%Page: 1 1
%%BeginPageSetup
% --> Här kan man stoppa deklarationer som rör sidan
%%EndPageSetup
% --> Postscript för sida

%%Trailer
%%EOF
```

Läsa filer

- Postscript kan läsa filer:

```
%!PS-Adobe-3.0

/ypos 760 def     % current y-position
/newline
{ currentpoint exch pop 14 sub  % calc y-val
    ypos 14 sub                 % Go left/down
    /ypos exch def 10 ypos moveto
} def

/Courier findfont 14 scalefont setfont
10 760 moveto

/infile (/etc/passwd) (r) file def
/buff 128 string def

{ % loop
  infile buff readline
  { show newline }
  { show infile closefile exit } ifelse
} bind loop
```

Exempel

Mer om filer

- Filaccess anges med:

```
r    - read only
w    - write only, skriv över om den redan finns
a    - write only, addera om den redan finns
r+   - read and write
w+   - read and write, skriv över om den redan finns
a+   - read and write, addera om den redan finns
```

- Speciella filer:

```
%stdin          - standard in
%stdout         - standard out
%stderr         - standard error
%statementedit  - statement editor file
%lineedit       - line editor file
```

- Postscript använder filter för att läsa och omvandla filer i olika format

Länkar

- Adobes sidor om Postscript:
  `http://www.adobe.com/products/postscript/`
- Bibeln: Adobe Postscript Language Reference Manual:
  `http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`
- Sammanfattning av funktioner:
  `http://www.colorado.edu/ITS/docs/scientific/gs/GS.html`
- Mathematical Postscript:
  `http://www.math.ubc.ca/people/faculty/cass/graphics/text/www/`