RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

A Short Introduction to PostScript

Peter Fischer, ZITI, Uni Heidelberg

What is PostScript ?

- Postscript is a *language* to describe graphic objects (& text)
- It is a *vector format*
  - Shapes, characters,.. are defined in an *exact*, mathematical way → objects / characters can be scaled, magnified, rotated... without loss of quality
  - Other vector formats are, for instance: *pdf* (portable data format) and *svg* (scalable vector graphics)
- Postscript is a *programming* language
  - Complex graphics can be described quickly and efficiently
  - They can be *parameterized* and changed *easily*
- Postscript devices (printers) must be intelligent, because they must *interpret* the language
  - Otherwise, the host computer must do the translation. Most often using the (free) tool ‘ghostscript’

Why Use & Know About Postscript ?

- *Simple manual* generation of ***high quality*** graphics
- Graphics can be *parameterized*
- *Automatic* generation of graphics from within other programs
- Small files
- *Exact* dimensions

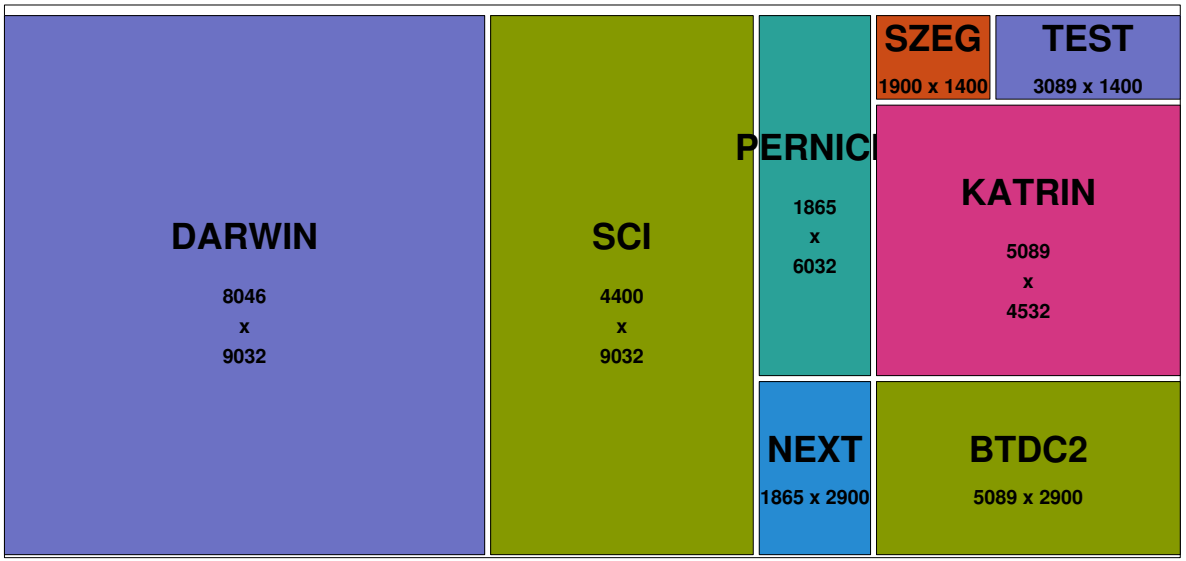
- Postscript is (still) common for LaTeX
- Sometimes, modification of available .ps or .eps files is required
  - Change a font
  - Modify colors or line width
  - Add water mark

- Many concepts are used in other languages (pdf, swift)

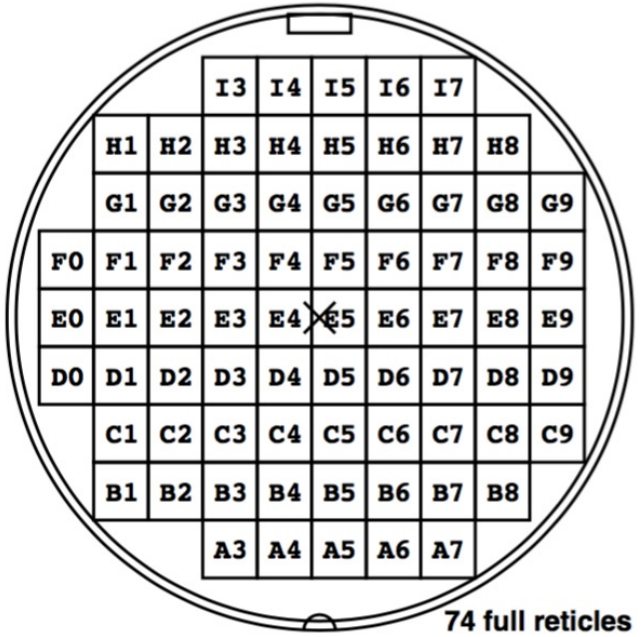
- Generating Graphics can be fun !

Examples

- Arrangement of chips in a 'reticle' ('to scale'):



Reticles on a wafer



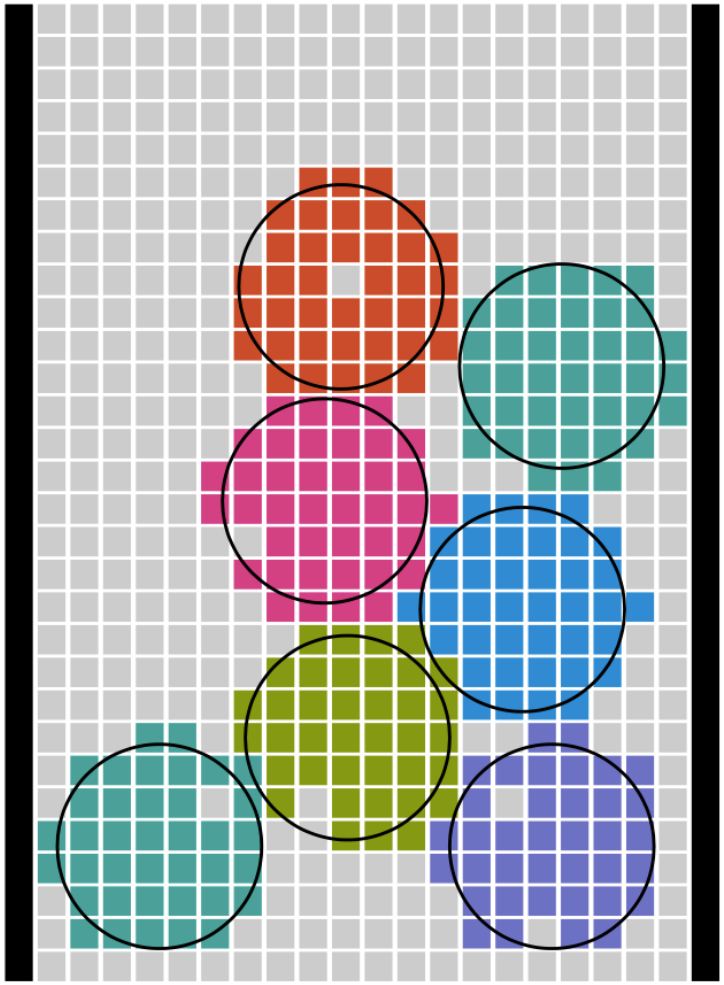
- Math exercises for your kids (with random generator):

7	8	5	1	x	5	4	6	6

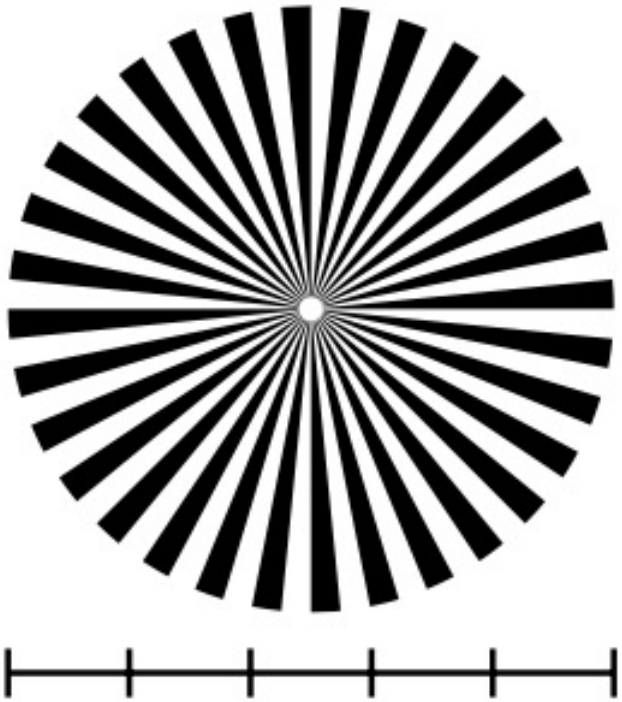
3	3	x	7	7	1

More Examples

- Illustration of a sensor readout

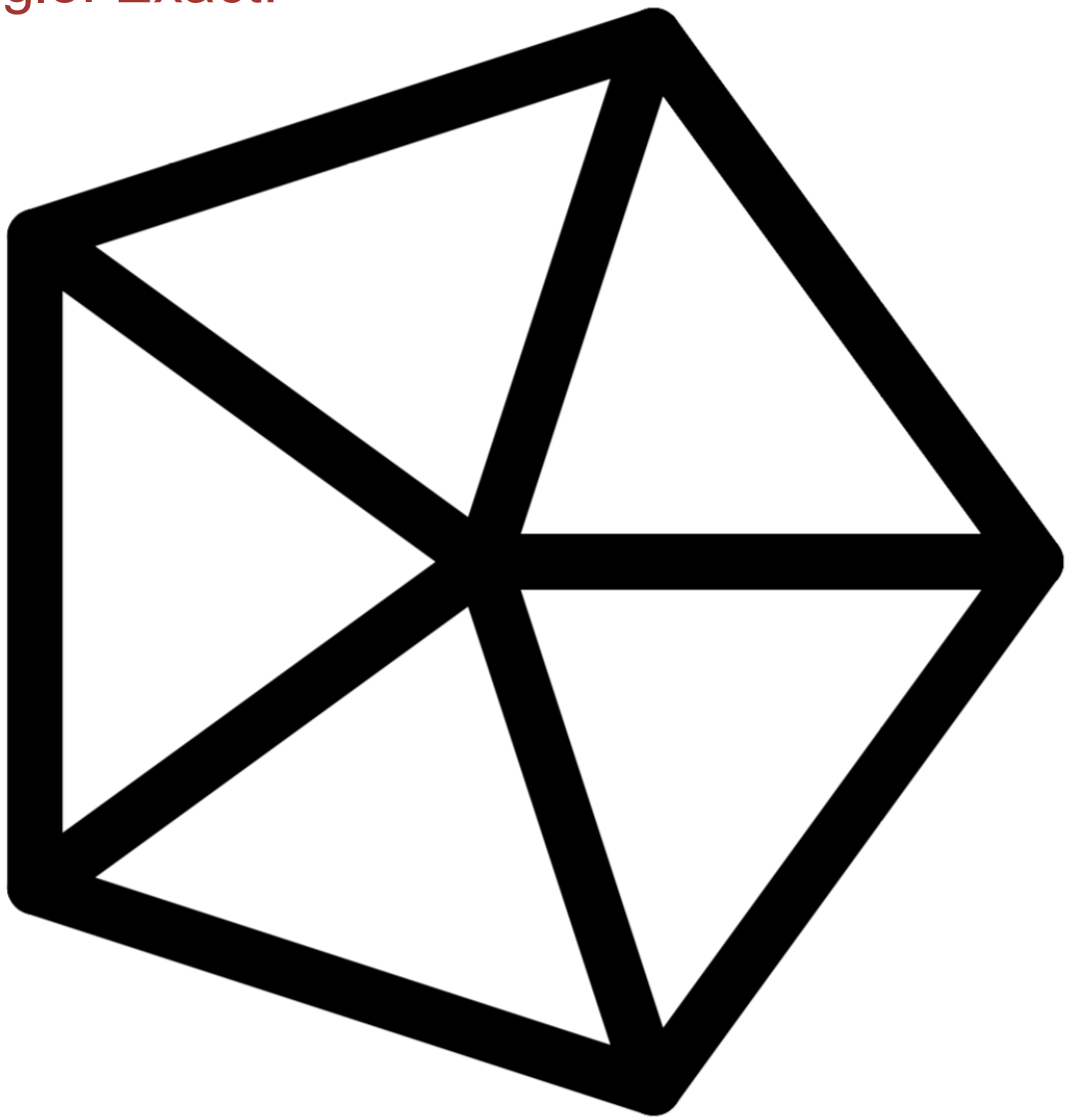


Siemensstern



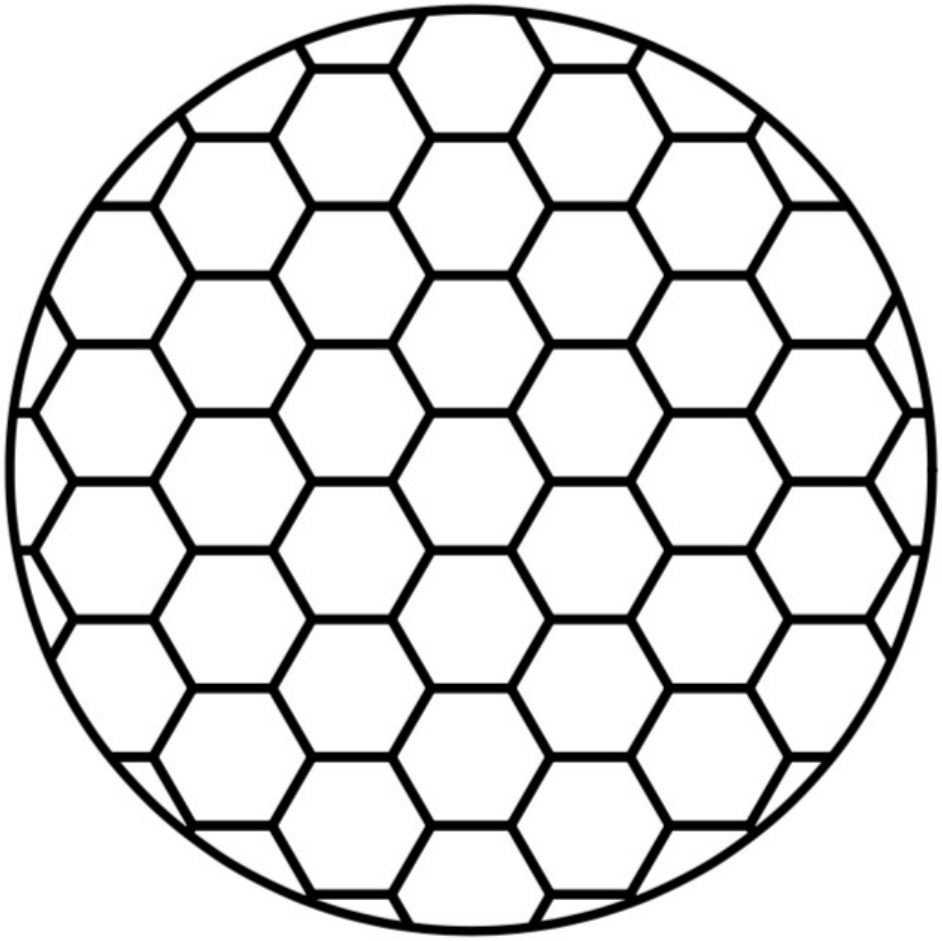
More Examples

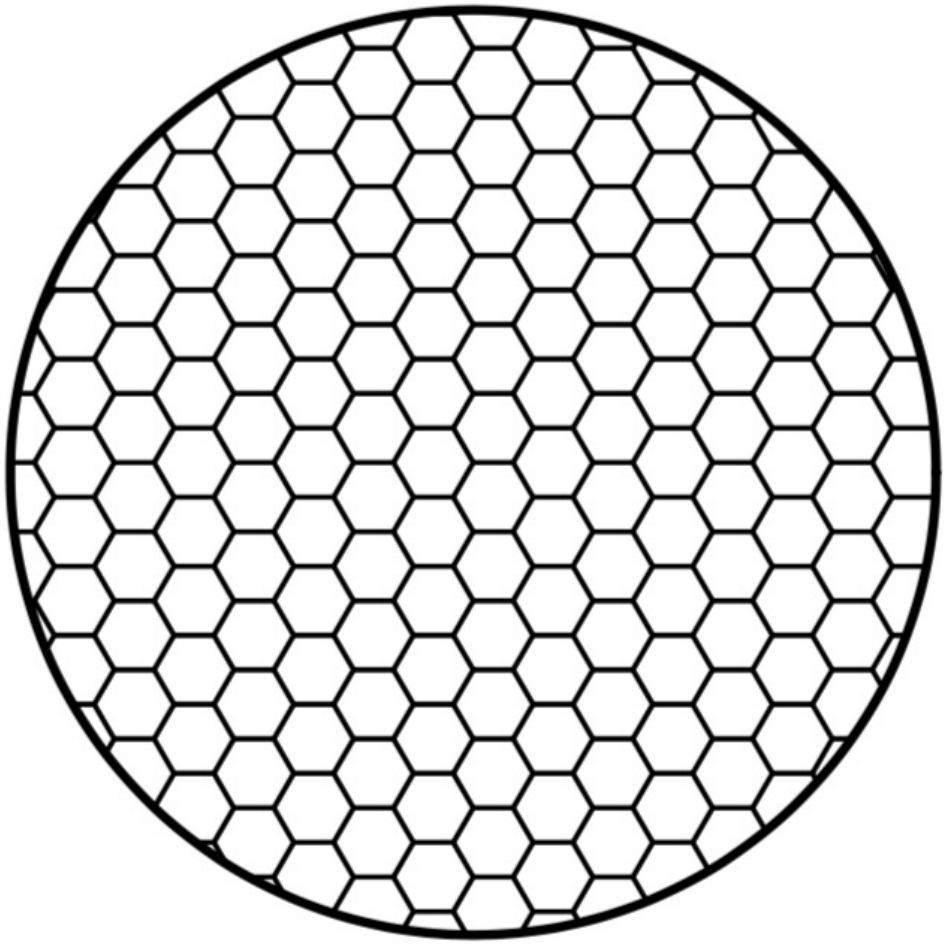
- Any Angle. Exact:



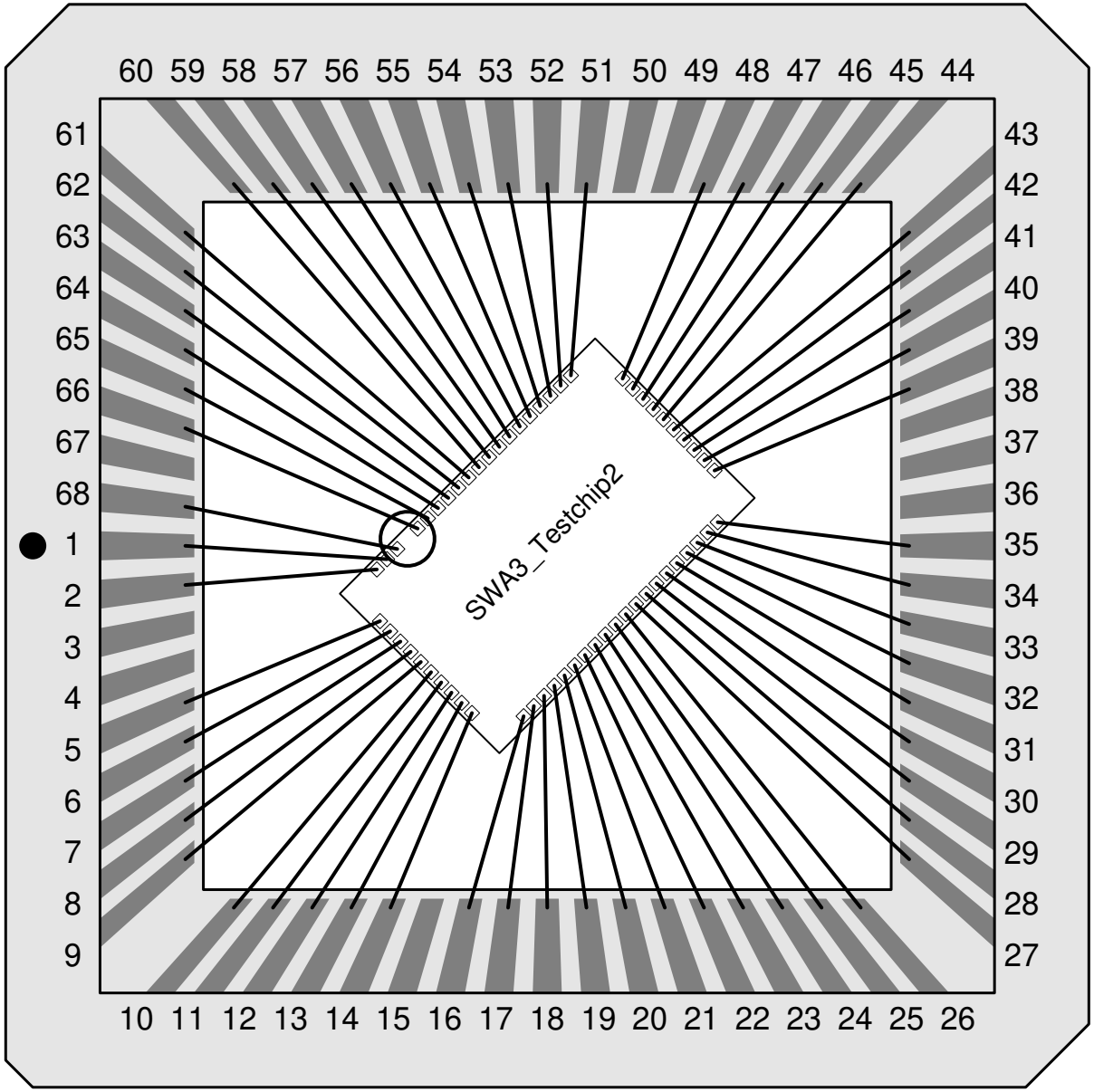
More Examples

- Exact Dimensions, exact clipping:





More Examples



What is the drawback ?

- Postscript is used less and less (replaced by pdf)
- Importing .eps in other documents is often difficult
  - It is simple in LaTeX (pdfLaTeX requires .pdf, but conversion from .eps → .pdf is simple and robust)
- Conversions often lead to quality loss.

- Debugging is difficult. Go step by step !

- Why not pdf?
  - pdf is *much* more complicated!
  - A ‘minimal’ pdf file is already lengthy
  - Hard to do ‘by hand’ because bytes need to be counted!

  - See the (short) intro to pdf later in the lecture…

.eps on Mac ≥ Sonoma

- Support for viewing .eps on Mac has been stopped ☹
- Completely! 🤬

- Really frustrating!!!!

- Way out – so far (any other ideas welcome):
  - Use editor ‘Visual Studio Code’
  - Use extension ‘eps-preview’
  - And maybe Postscript Language (syntax highlighting)

- To further process .eps files, convert to .pdf in linux shell using epspdf

Getting Information

- Postscript Language Reference Manual ('PLRM')
  - https://www.adobe.com/jp/print/postscript/pdfs/PLRM.pdf
  - Very complete, 'easy to read', 800 pages
- Language Tutorial and Cookbook (the 'Blue Book')
  - https://cupdf.com/document/poastscript-language-bluebook.html?page=10
  - (no pdf found any more…)
- Language Program Design (the 'Green Book'):
  - https://web.archive.org/web/20110613223722/http://partners.adobe.com/public/developer/en/ps/sdk/sample/GreenBook.zip
- Many Web sites of good quality (see Lecture Page)

Caveat

- These slides have grown over the years to ‘*my own reference manual*’ for PostScript:
  - When I forget syntax details, I find here all I need to do my drawings..
- This is *sometimes too detailed* information for a start…
  - There are many slides with titles *in parenthesis* which I will skip
  - …mostly….

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

LET'S GO….

Simple Example 1: Triangle + Circle

```
%!PS
 newpath
  10  10 moveto
 100  10 lineto
  50 100 lineto
 closepath

stroke

 100 100
 10
 0 360 arc fill

showpage
```

start a new shape

start at (10/10)

draw lines

connect to start

show outline

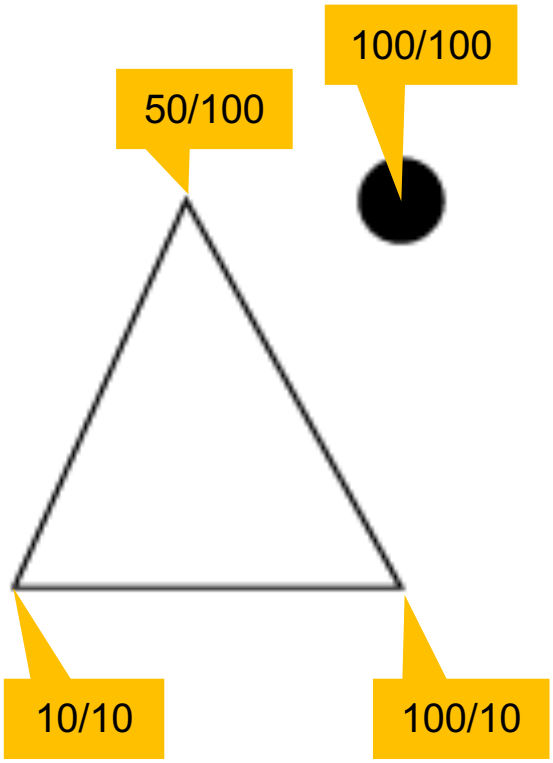
x/y = 100/100

radius = 10

start/stop angle

fill the arc

print everything

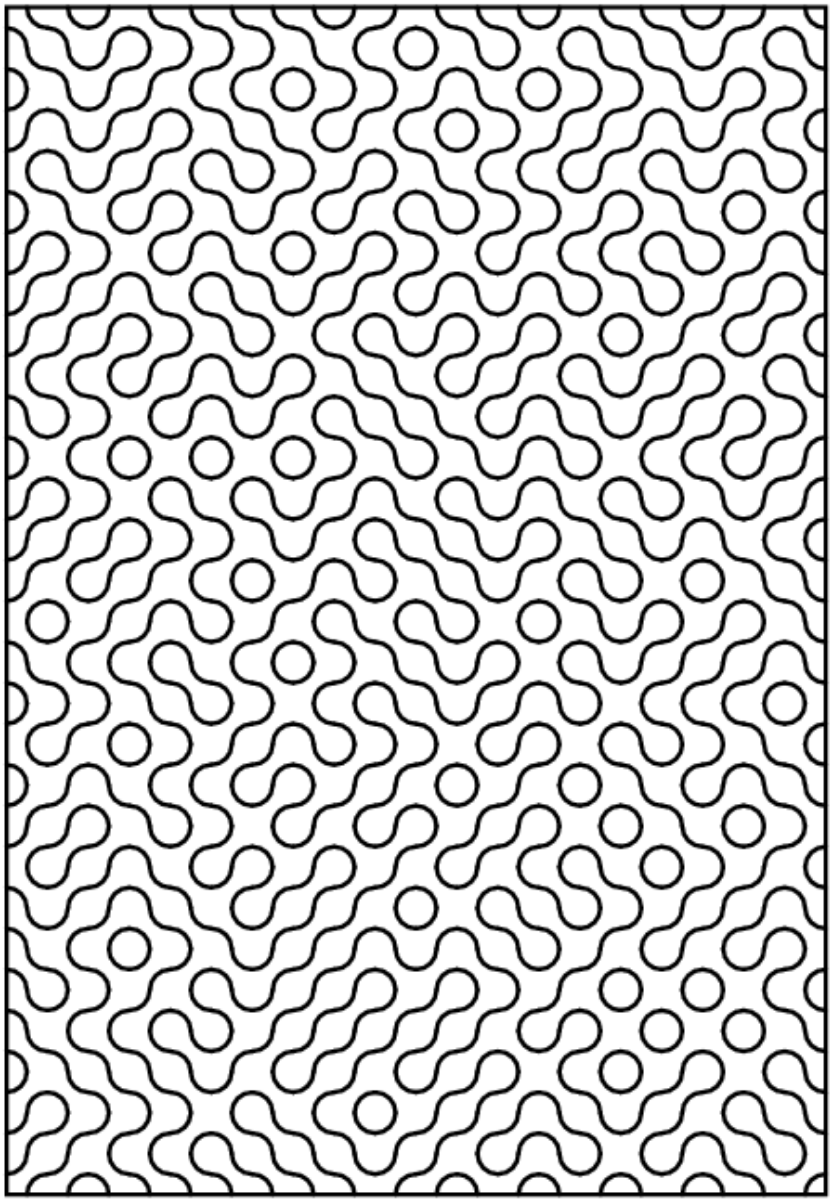


Viewing Postscript Files

- On Linux machines, files can be viewed with
  - gv (on the CIP Pool machines)
  - evince (on the CIP Pool machines)
  - ghostview, okkular, ShowView, GSView,…
  - ...there is always a viewer...
- On windows
  - Ghostview (must be installed, I do not know about new versions of Windows…)
- On MAC
  - Use Visual Studio Code + Extension
- Always need GhostScript to interpret the language
  - GhostScript is also used to convert ps/eps → pdf, png, jpg…

Advanced Example 2: Truchet Pattern

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 595 842

2.835 dup scale
5 4 translate 1 setlinecap
0 0 200 290 rectstroke
100 145 translate

/W 10 def /W2 { W 2 div } bind def

/DRAWUNIT {
  gsave  translate rotate
 W2 neg W2 neg W2   0  90 arc stroke
 W2 W2 W2 180 270 arc stroke
  grestore
} def

-95 W 95 {
  /x exch def
  -140 W 140 {
    /y exch def
    rand 4 mod 90 mul  x y  DRAWUNIT
  } for
} for

showpage
```



File Structure

- File **MUST** start with `%!PS` (may add PS - version number)
  - If forgotten, (most) printers will output (a lot of) ASCII stuff…
- PostScript is *CaseSensitive*!
- Blanks and Line breaks are *irrelevant*

- Comments
  - In-Line comments start with

```
% … commented code here …
```

  - Larger code blocks can be commented with

```
false {
  … commented code here …
} if
```

- Files have extension *.ps*
- To actually *print*, the file ***must*** end with `showpage`

eps Files

- .eps files contains some *additional* meta-information
- These 'encapsulated postscript files' have extension **.eps**
- .eps type is announced in first line by EPSF text:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 595 842
…
```

- All eps meta information is added as comment with '%%'

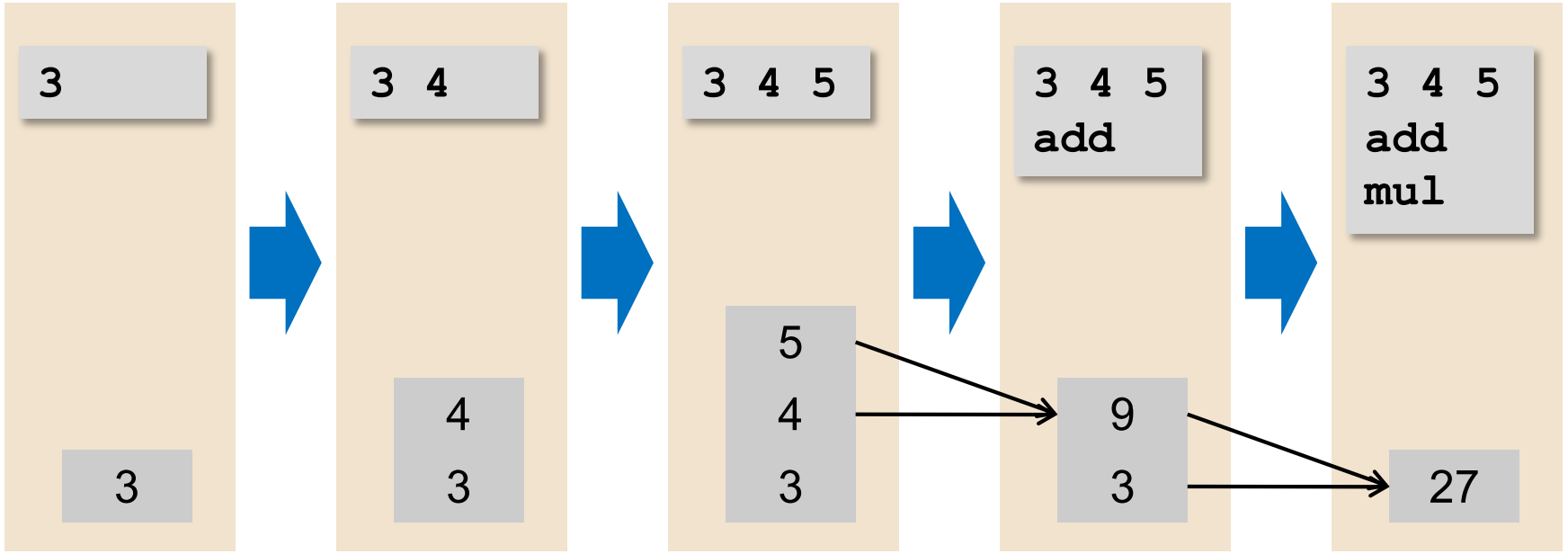
- Most important (and the only *required*) information: size of the viewing area = BoundingBox:
- parameters (in integer postscript units) are:

```
%%BoundingBox: x_botleft y_botleft x_topright y_topright
```

- **Best always use .eps !!!**

The Stack

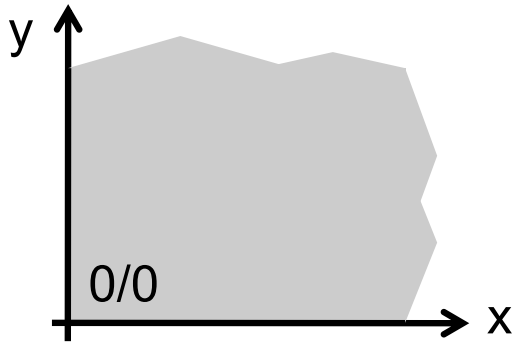
- PostScript uses
  - a stack (Last In - First out)
  - RPN (Reverse Polish Notation) = UPN (Umgekehrt Poln. Notation): Operands are put to stack **first**, operator is **last**
- Example `3 4 5 add mul` → (4+5) × 3



- Operators can have *1 or more* arguments

Coordinate System, Lengths and Points

- Origin (0/0) is ***BOTTOM LEFT***
- X is to the *RIGHT*
- Y is *UPWARD*



- 1 PostScript Unit = 1 Point = **1/72 inch = 0.353 mm**
  - (1 inch = 1 Zoll = 2.54 cm exactly)

- Convert *mm* to *point* by multiplying with 72 / 25.4 = 2.835..
- By defining the command (see later...)

```
/mm { 2.835 mul } def
```

you can just write

```
15 mm
```

in your code!

- Later we will use the **`scale`** command to change units...

The Page / Sheet Size

- ‘sheet’ size & orientation (in .ps) are undefined.
  - They depend on the ‘viewer’ or printer
  - (This is a drawback. This is better in .eps and .pdf!)

- The sheet size can be ‘fixed’ as a ‘bounding box’ using an eps command, see before…
  - `%!PS-Adobe-3.0 EPSF-3.0`
  - `%%BoundingBox: llx lly urx ury`

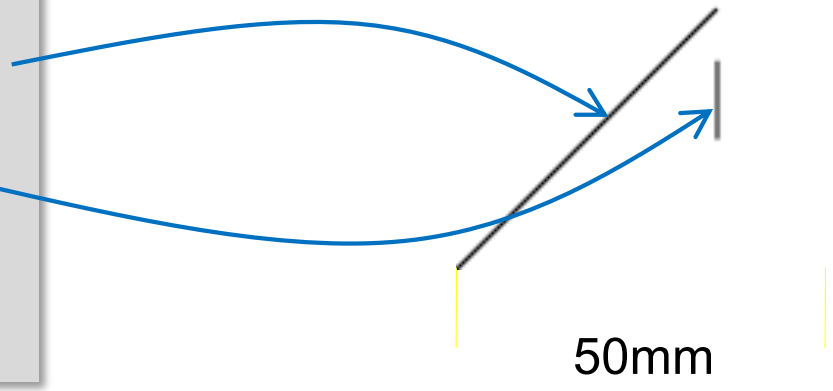
    (***`llx`*** = lower left x, … using *integer postscript* units)

- A4 (portrait) paper has
  - width = 210 mm = 595.28... points
  - height = 297 mm = 841.89... points
  - `%%BoundingBox: 0 0 595 842  % A4 portrait`

Hello World

- Shapes / Outlines are defined as *paths*.
  A *path* is a sequence of straight lines / bends / gaps / …
- **`x y moveto`** **moves** the ‚pen' to coordinate [x y]
- **`x y lineto`** **draws** a line **from the last point** to [x y]
- **`stroke`** executes the **path** drawing

```
%!PS
0 0 moveto
100 100 lineto
100 80 moveto
100 50 lineto
stroke
showpage
```



- Remember: 100 Units = 100 × 0.353 mm = 35.3 mm

- **`rmoveto`** and **`rlineto`** are *relative* to the *last* point
- Note: You **MUST** first move to *somex somey*!

Drawing and Filling Paths

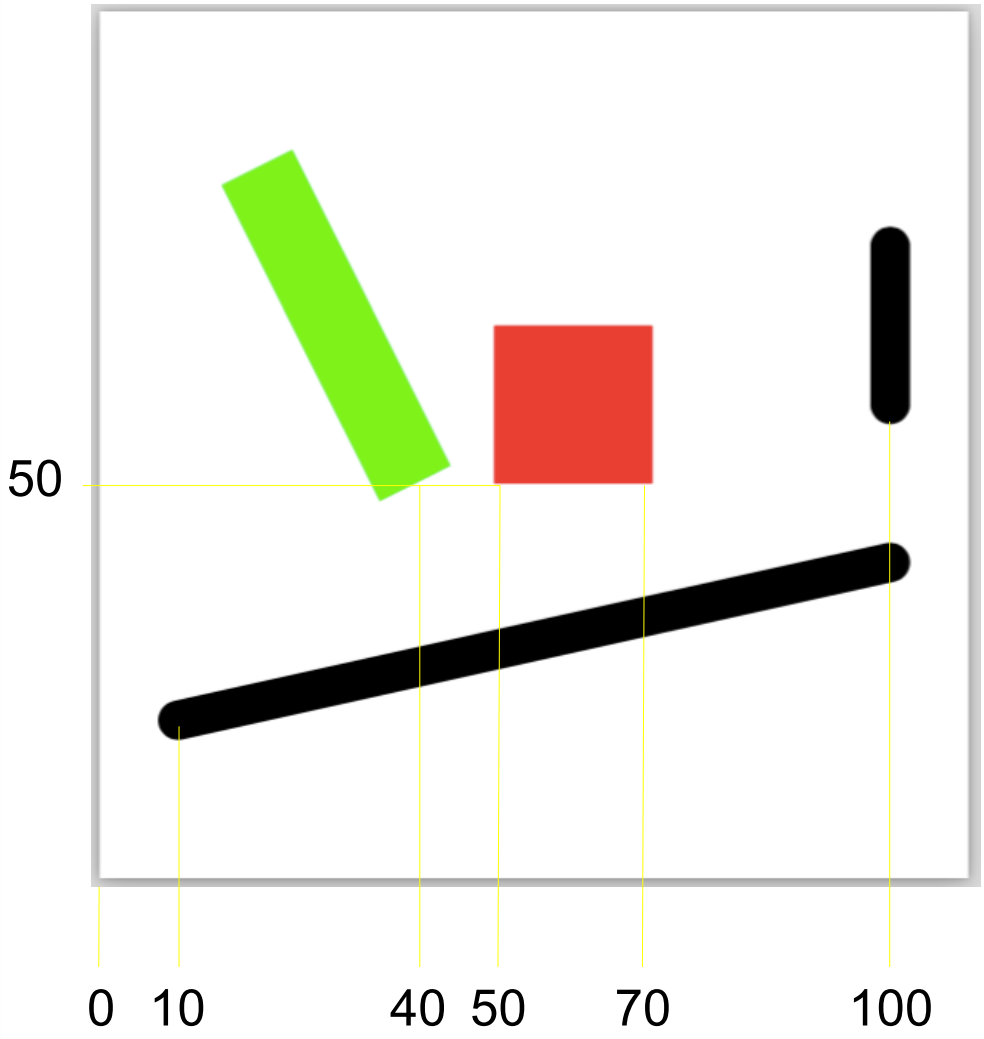
- A path *can* be started with **`newpath`**
- The command **`closepath`** connects the last active point to the starting point (see Example 1 on slide 10)
- A path can be used for further operations (e.g. clipping,…)
- Using a path is not always necessary

- To draw a path (or sequence of **`moveto`** / **`lineto`** commands)
  - **`stroke`** draws the **outline**
    - the *width* of the line can be set with **`value setlinewidth`**
    - the shape of the line *end* can be set with **`value setlinecap`**
    - the shape of corners is set with **`value setlinejoin`**.
  - **`fill`** fills the **inner part** with the presently selected color
- **`x y w h rectstroke`** is a shortcut to draw a rectangle

- Color can be set with **`r g b setrgbcolor`** (r,g,b = 0.0 … 1.0) or with **`g setgray`** (for gray values)

One More Example

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 110 110

10  20 moveto   100 40 lineto
100 60 moveto   100 80 lineto
5 setlinewidth  1 setlinecap
stroke

newpath
 50 50 moveto     20   0 rlineto
  0 20 rlineto   -20   0 rlineto
closepath
1 0 0 setrgbcolor
fill

40 50 moveto      20 90 lineto
0  setlinecap 10 setlinewidth
0 1 0 setrgbcolor
stroke
showpage
```



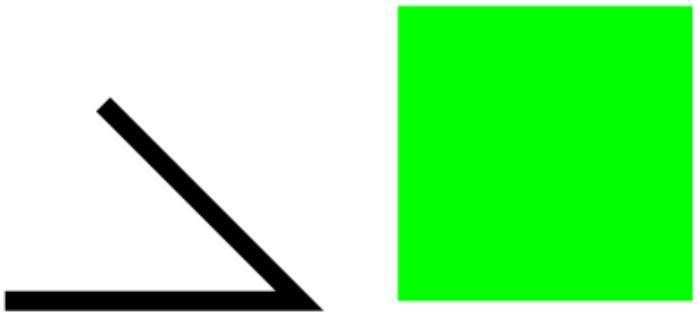
Working in Linux

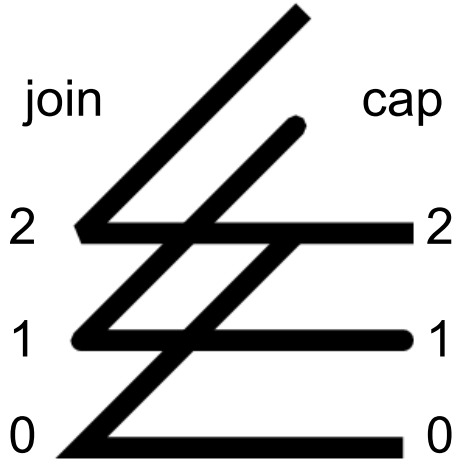
- Log in on one of the CIP Pools machines
- Start a terminal, for instance
  `Anwendungen->Systemwerkzeuge->XFCE-Terminal`
- (
  - Create a subdirectory with **`mkdir DIRNAME`**
  - Move to the subdirectory with **`cd DIRNAME`**
-)

- Edit your file for instance with **`gedit filename.eps &`**
- View your file with **`evince filename.eps &`**
- Or **`gv filename.eps &`**

Exercise 1

- Draw a line from (10,10) to (40, 10) to (20,30)
  - Change the width of the line
  - Play with shape of the line ends and the shape of the corners (use values 0…2 and a ‘thick’ line).
  - Can you find out the difference between cap = 0 and 2?
- Draw a square of 30 units size with its lower left corner at (50,10)
  - Use **`moveto`** and **`lineto`**
  - Use also **`newpath`** and **`closepath`**
  - Fill the square with green color





Mathematics

- PostScript knows several mathematical functions.
- Remember RPN: first **`operand(s)`**, then **`operator`**
  - **`x y sub`** → x - y. Also: **`add`**, **`mul`**, **`div`**, **`idiv`**, **`mod`**
  - **`x neg`** → -x
  - **`x abs`** → |x|. Also: **`round`**, **`floor`**
  - **`x sin`** → sin(x). Also: **`cos`**, (no **`tan`**), **`ln`**, **`log`**, **`sqrt`**
  - **`x y atan`** → arctan(x/y), 4 quadrants. ***Note*** 2 arguments!

- Angles are given (as floats) in *degrees* (i.e. 0…360)

- Examples:
  - (2 + 3) × 4 → **`2 3 add 4 mul`**
  - 2 + 3 × 4 → **`2 3 4 mul add`**
  - Sqrt (3 + 4) → **`3 4 add sqrt`**

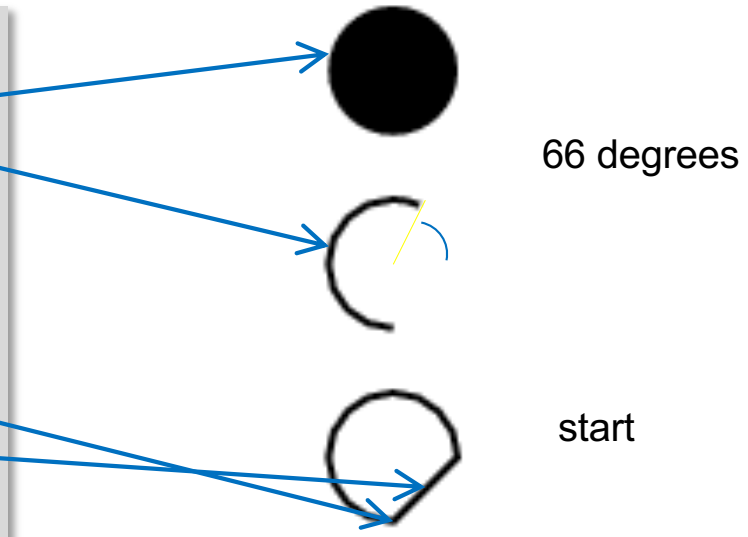
(Random Numbers)

- Random (integer) numbers can be obtained with
  - **`rand`** → random *integer* number
- A seed can be set with
  - **`value srand`**
- To obtain a different seed every time you ‘run’ (print) the postscript file, you can use a command that returns an integer time (in ms):
  - **`realtime`** → *integer* time value on stack
  - **`realtime srand`** → initialize with new value at each run

Drawing Arcs

- Arcs (parts of circles) are defined using
  `x y radius phistart phistop arc`
- Angles are in degrees, relative to *x-axis*
- **`arc`** turns *counter clock wise,* **`arcn`** turns *clock wise*
- They can be **`filled`** or **`stroked`**.
- Example:

```
%!PS
20 80 10 0 360  arc fill
20 50 10 66 270 arc stroke

newpath
20 20 10 0 270 arc
closepath
stroke

showpage
```



Defining Constants & Functions

- Defining a ‘fix’ constant:
  - **`/name value def`**
  - Example: **`/PI 3.141 def`**
- Defining a ‘calculated’ constant:
  - **`/name commands def`**
  - Example: **`/TWO_PI  PI 2 mul  def`**
- Defining a *function* (executed when called):
  - **`/name { commands } def`**
  - Example: **`/ADDFIVE { 5 add } def`**
    `3 ADDFIVE` → 8
- What happens?
  - The pair (name definition) is stored in a *dictionary* by **`def`**
  - (There are several different dictionaries for different tasks…)

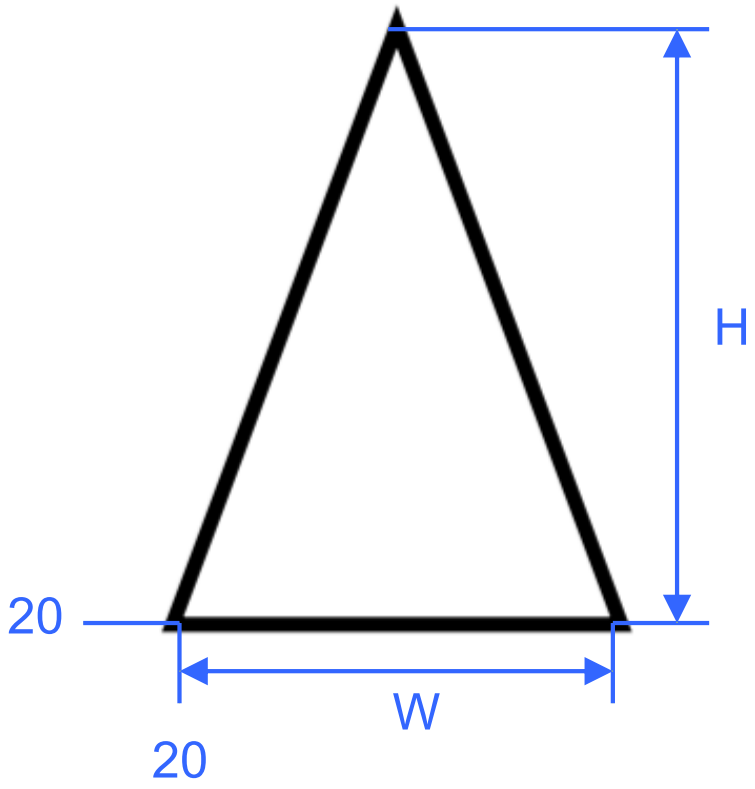
Example

- Understand in this example how the arguments of 'lineto' are constructed!
  - Note that the shift to (20,20) can be done more elegantly using the **`translate`** command, see later..

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 70 80

/W 30 def
/H 40 def

newpath
20 20 moveto
20 W add    20 lineto
20 W 2 div add    20 H add lineto
closepath
stroke

showpage
```

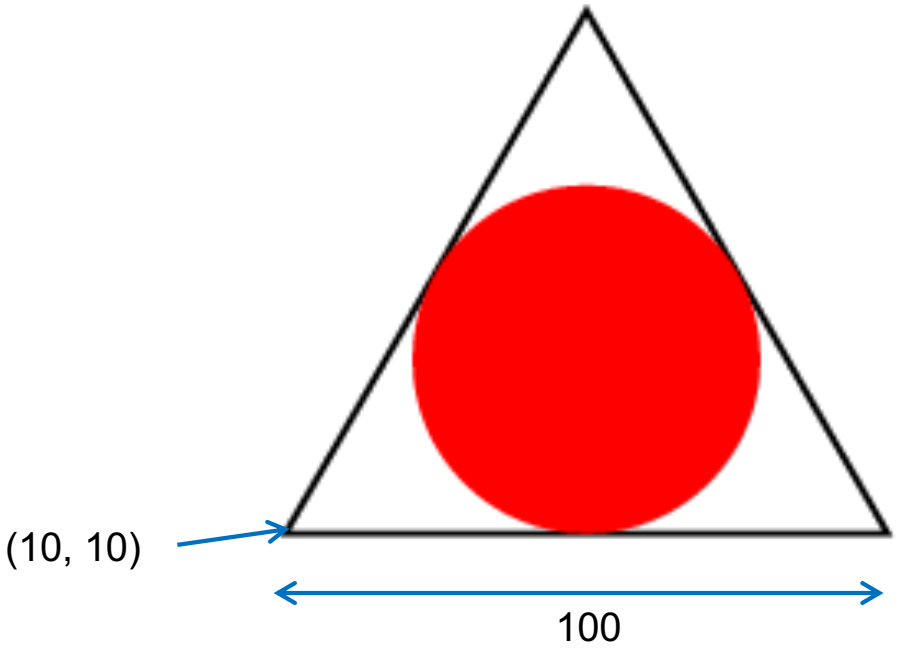
x coordinate of the top point

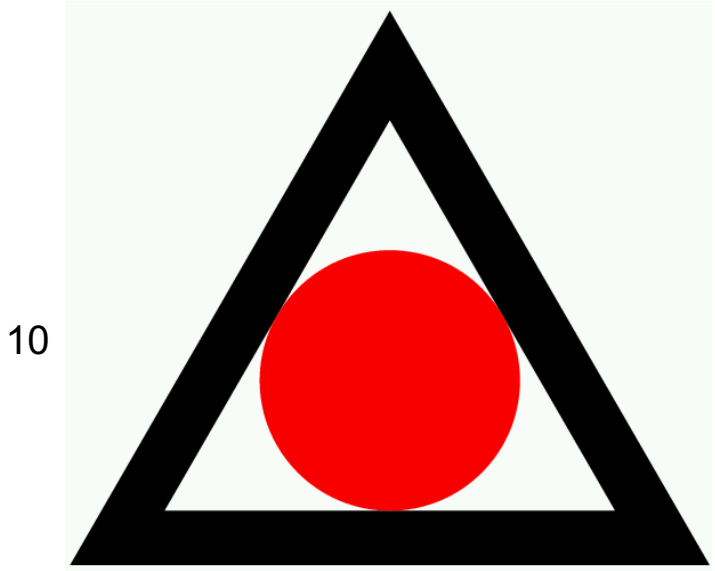
y coordinate of the top point



Exercise 2

- Draw a triangle with *equal* sides
  - Start at (10,10), side length =100. Use a variable: /W 100 def
  - You have to do so some simple math for find the height H. Do it in postscript!
- Make the lines wide (for instance 10 points)
- Add a red, filled circle in the center which *just* touches the lines

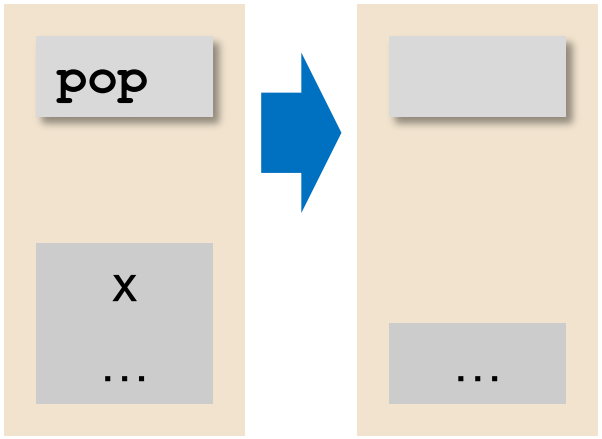




Manipulating the Stack: `pop`, `dup`, `exch`

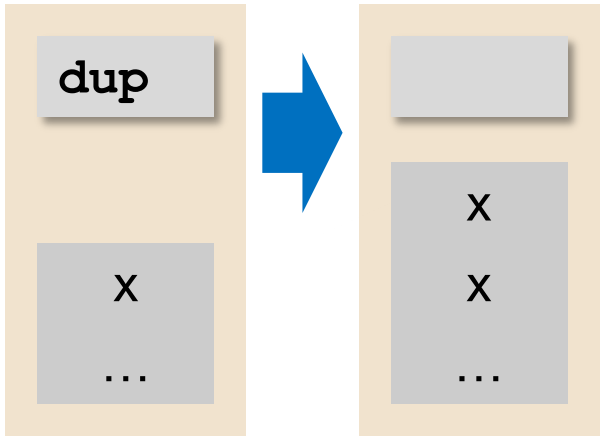
`pop`

drop top element:



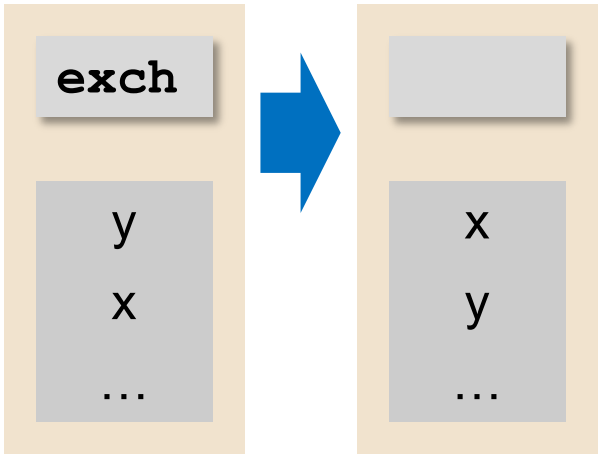
`dup`

duplicate top element:



`exch`

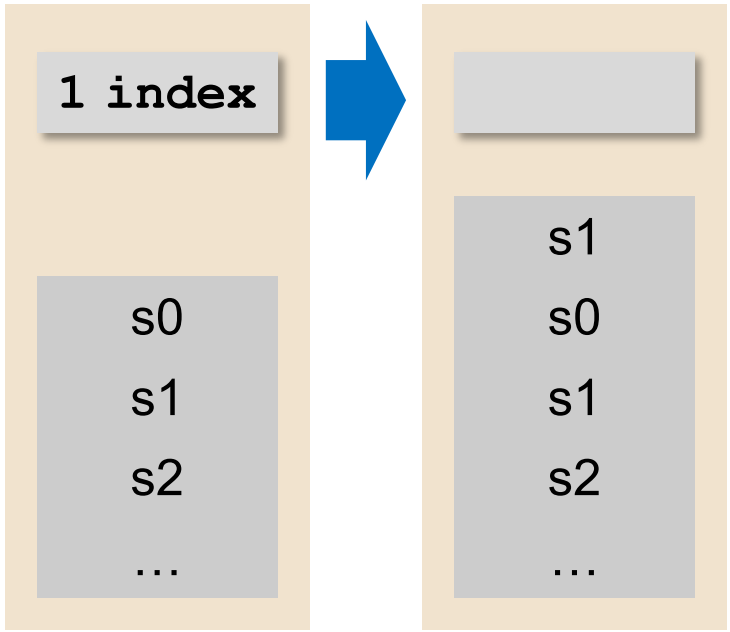
swap topmost elements:



(Manipulating the Stack: `index,copy`)

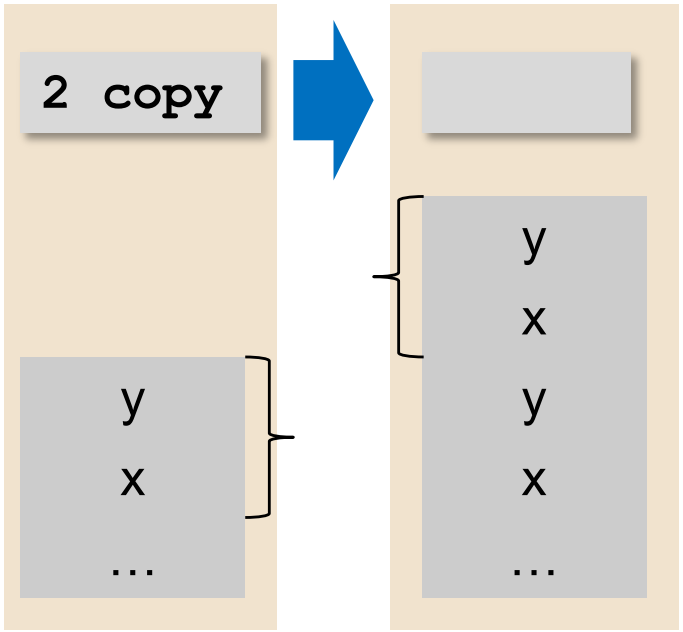
`n index`

copy t-th element (top index = 0):



`n copy`

duplicate *n* elements:

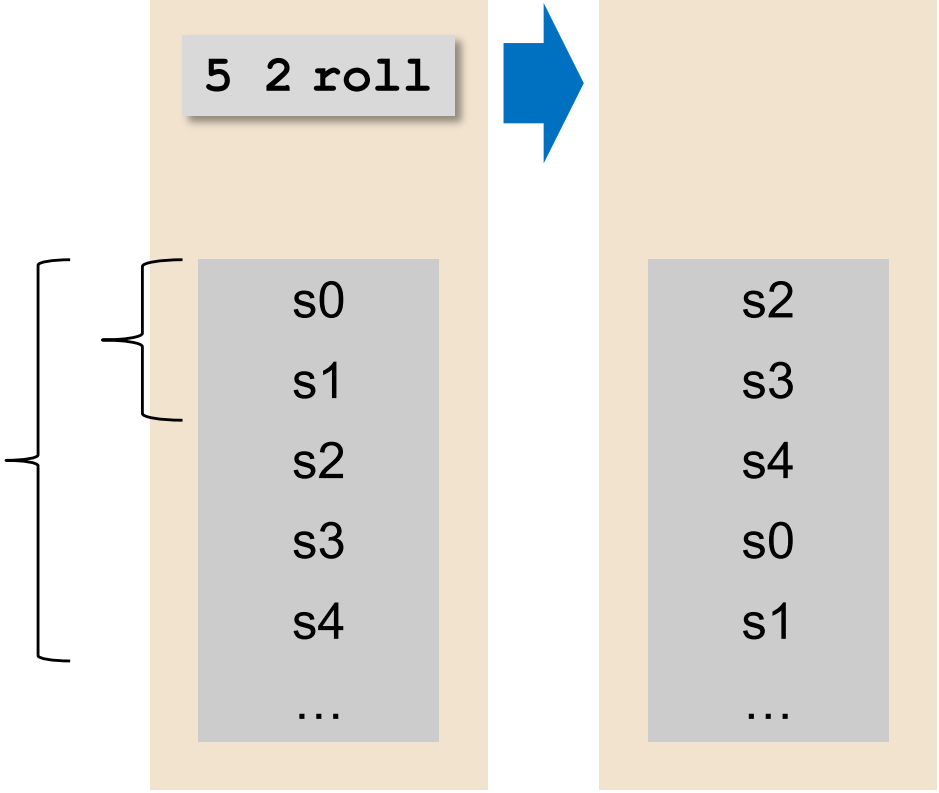


(Manipulating the Stack: `roll`)

```
n i roll
```

rotate n elements by i (upward)

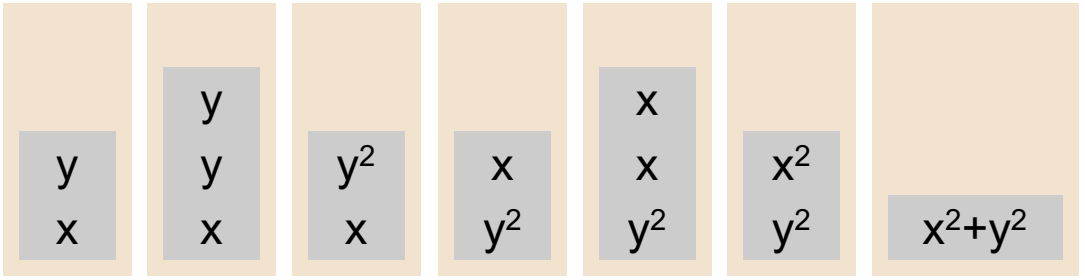
```
5 2 roll
```



Passing Values to Functions

- Parameters are passed *on the stack*
  - They can be used using stack manipulation commands
  - Example: Define DIST(x,y) = sqrt(x^2+y^2).
  - Assume x,y on stack:

```
/DIST {
   dup   %1
   mul   %2
   exch  %3
   dup   %4
   mul   %5
   add   %6
   sqrt
} def
```



- Usage: `3.2 1.7 DIST` → 3.6235
- Note: Functions can remove parameters or leave the stack intact. Stack over- / under-flows are very common mistakes!

Defining and Assigning Local Variables

- Values on the stack can be assigned to local variables:
  - **`/NAME exch def`**
  - (assume **`x`** is on the stack, then **`x /NAME exch`** leads to **`/NAME x`**, so that the **`def`** works normally)

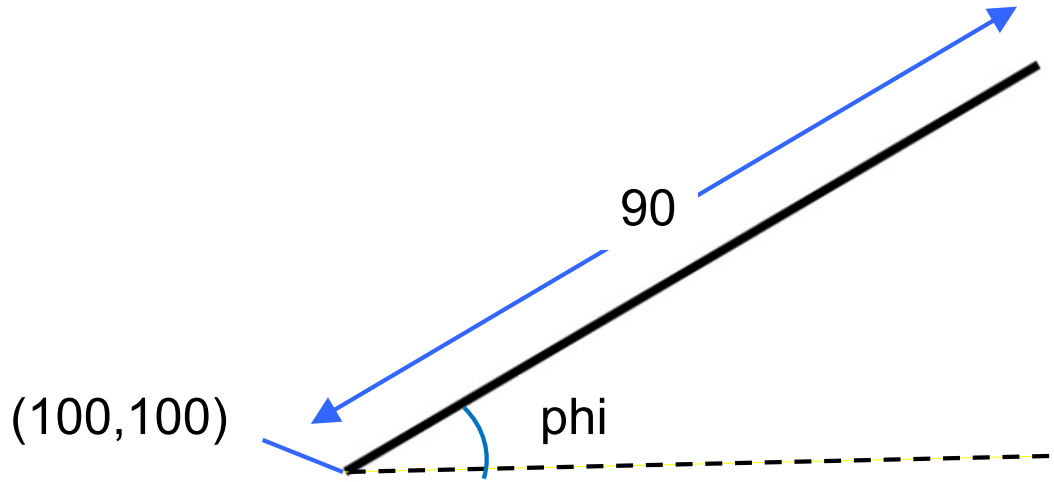
- Example: Define DIST(x,y) = sqrt(x^2+y^2)

```
/DIST {
   /y exch def   % topmost argument first!
   /x exch def   % now the stack is empty!
   x x mul       % on stack: x^2
   y y mul       % on stack: x^2 y^2
   add
   sqrt
} def
```

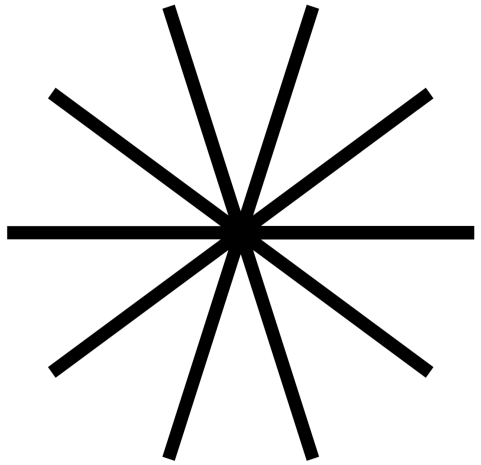
- This is much less efficient, because names must be looked up in a ‘Dictionary’. (Furthermore, the variables are global!)

Exercise 3a

- Define a function **`LINE`** which draws a line of length 90 in an angle phi (on stack), starting at 100/100:

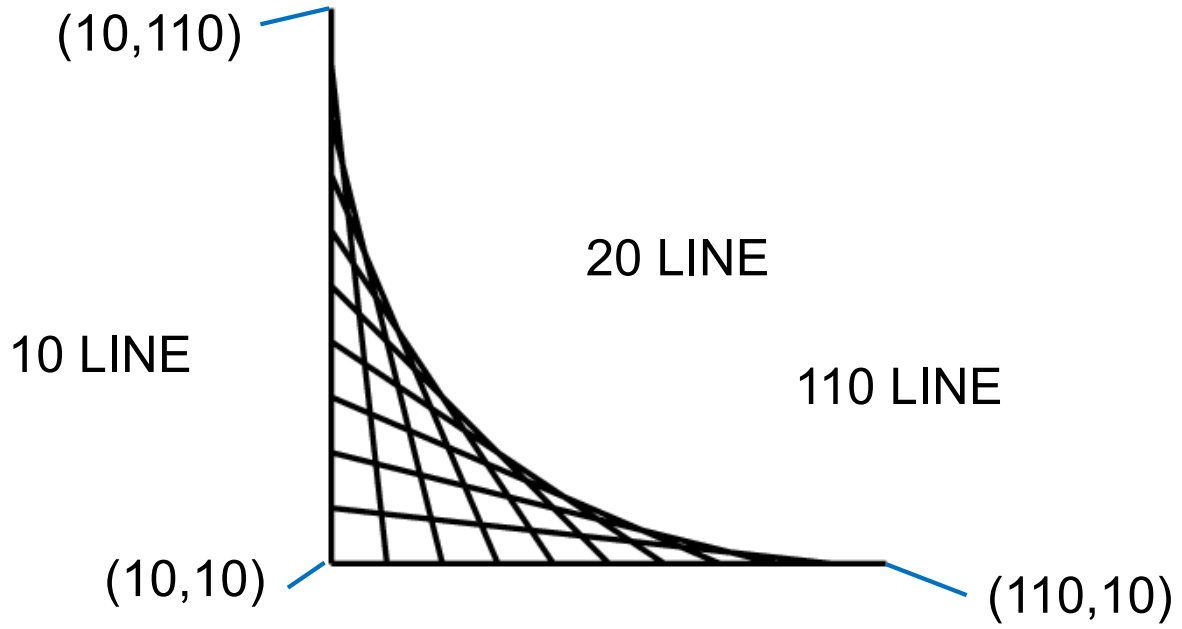


- Make a star by calling **`LINE`** several times



(Exercise 3b)

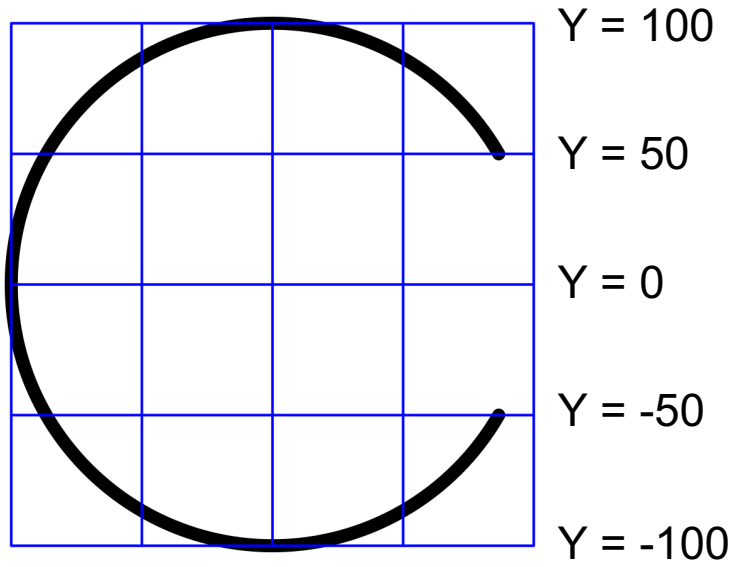
- Draw the following picture:



- First draw (a few) individual lines
- Next, define a function **`LINE`** which gets *one* value from the stack which indicates the *start of the line on the x-axis*.
- The drawing is then done by a sequence of **`LINE`** commands:
  `10 LINE  20 LINE  30 LINE` …

Exercise 3c

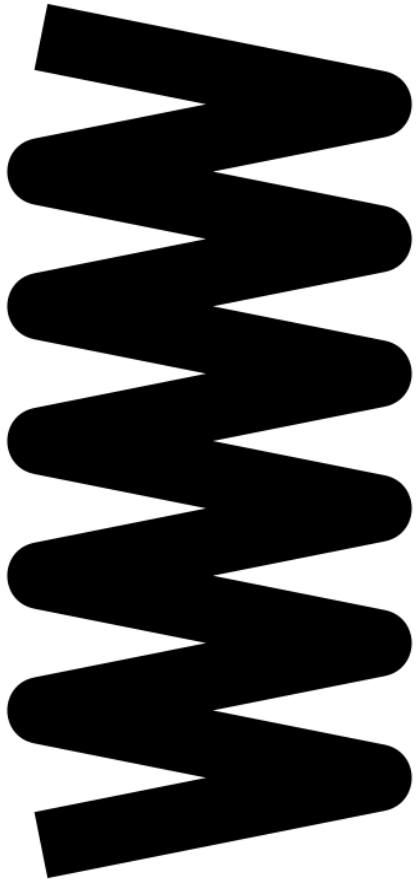
- There is no standard Arcus Cosine function available..
- In the internet, you find:
  arccos x = pi/2 - arctan (x / sqrt(1 - x^2))
- Define a function!
  - Remember that Postscript uses degrees, not radian
- Try it out:
  Draw a circle of radius 100 which ends exactly at y=50



Repeat

- Do something several times:
- **count { …*commands*… } repeat**

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 7 14

1 setlinejoin
1 1 moveto

6
{5 1 rlineto -5 1 rlineto}
repeat

stroke

showpage
```

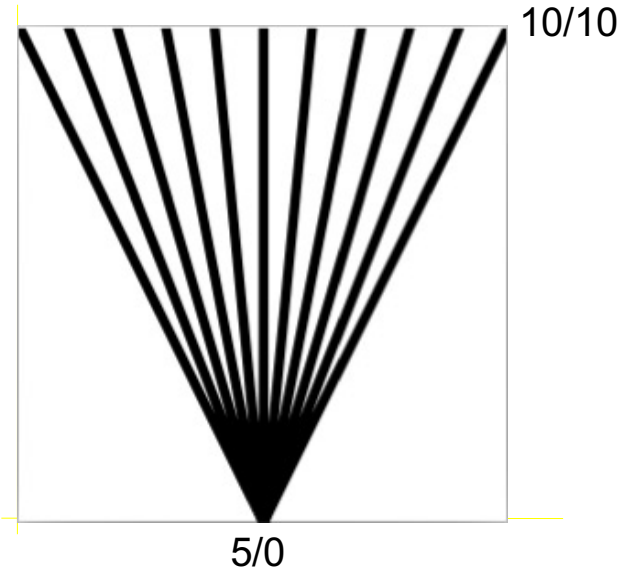


Loops

- If we want to so something different in each loop, we need a loop counter
- **`istart istep imax { …commands… } for`**
  - The loop value is put on the stack in each iteration (istart, istart+istep, istart+2 istep, …, *including* imax)
  - Then the commands are called
    They MUST consume (remove) the value from the stack
  - The loop variable can be assigned with **`/i exch def`**
- Example:

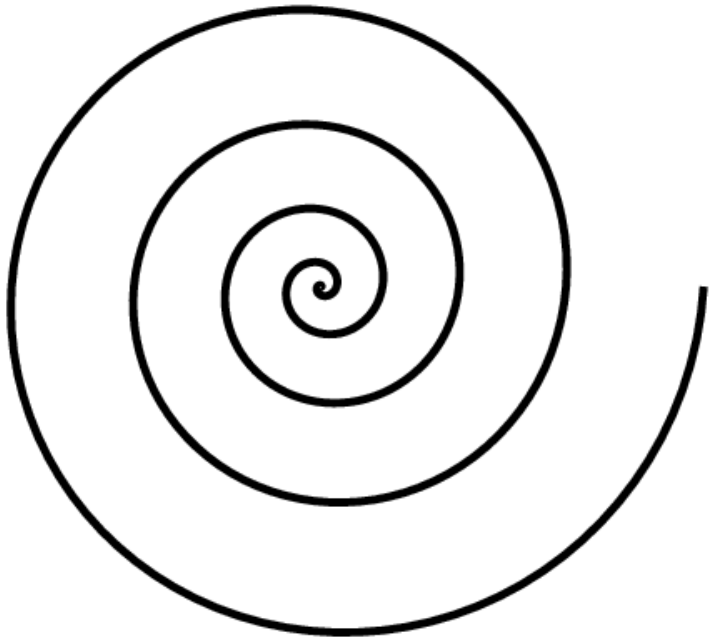
```
%!PS
0.2 setlinewidth
0 1 10 {
  5   0 moveto
      10 lineto
} for
stroke
showpage
```

Here we use the sweep variable which is still on the stack!!!



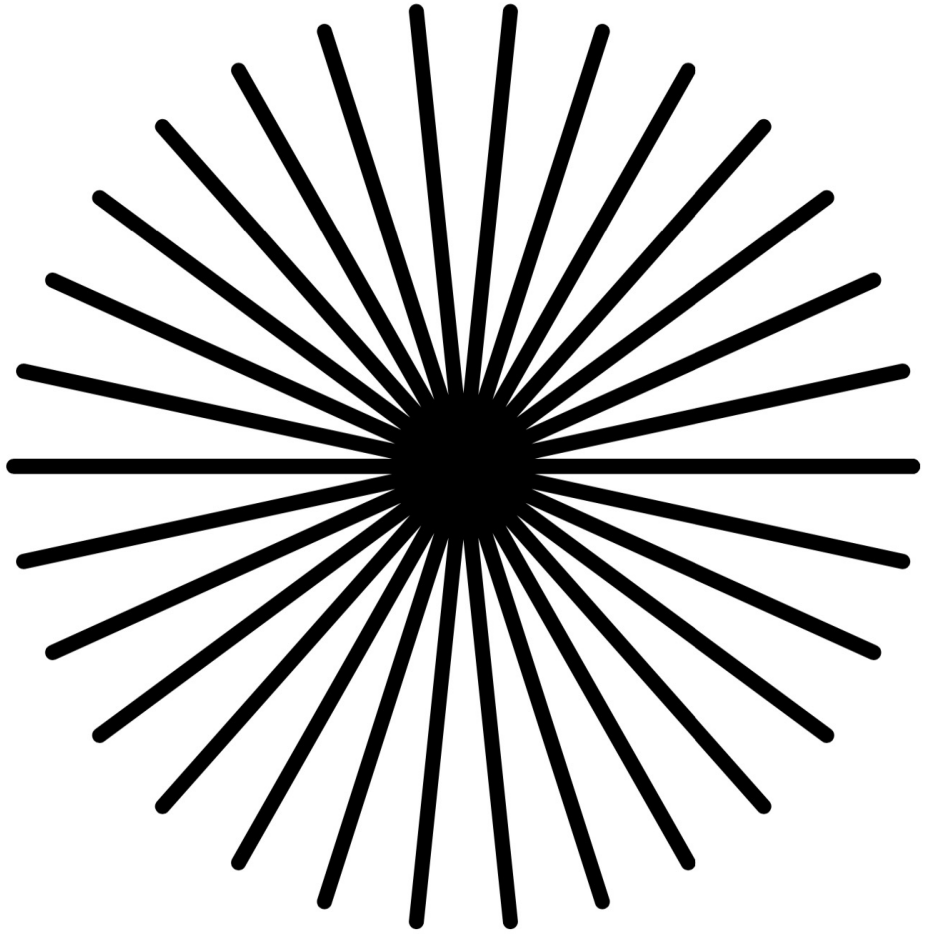
Loops: Another Example

```
%!PS
/X0   50 def          % center position
/Y0   50 def
/RMAX 48 def          % outer radius
/NTURN 5 def          % number of turns

/PHIMAX 360 NTURN mul def % maximal angle

X0 Y0 moveto          % start in center
0 10 PHIMAX {
  /phi exch def       % keep loop var.
                      % (drop from stack!)
  phi PHIMAX div      % get a value from 0 to 1
  dup mul RMAX mul    % square and scale
  dup                 % we need this for x and y
  phi cos mul X0 add % this is x
  exch                % get radius on top of stack
  phi sin mul Y0 add % this is y
  lineto              % draw a line
} for
stroke
showpage
```



Exercise 4a

- Modify exercise 3a using a for-loop for calling **`LINE`**
- Play with the increment

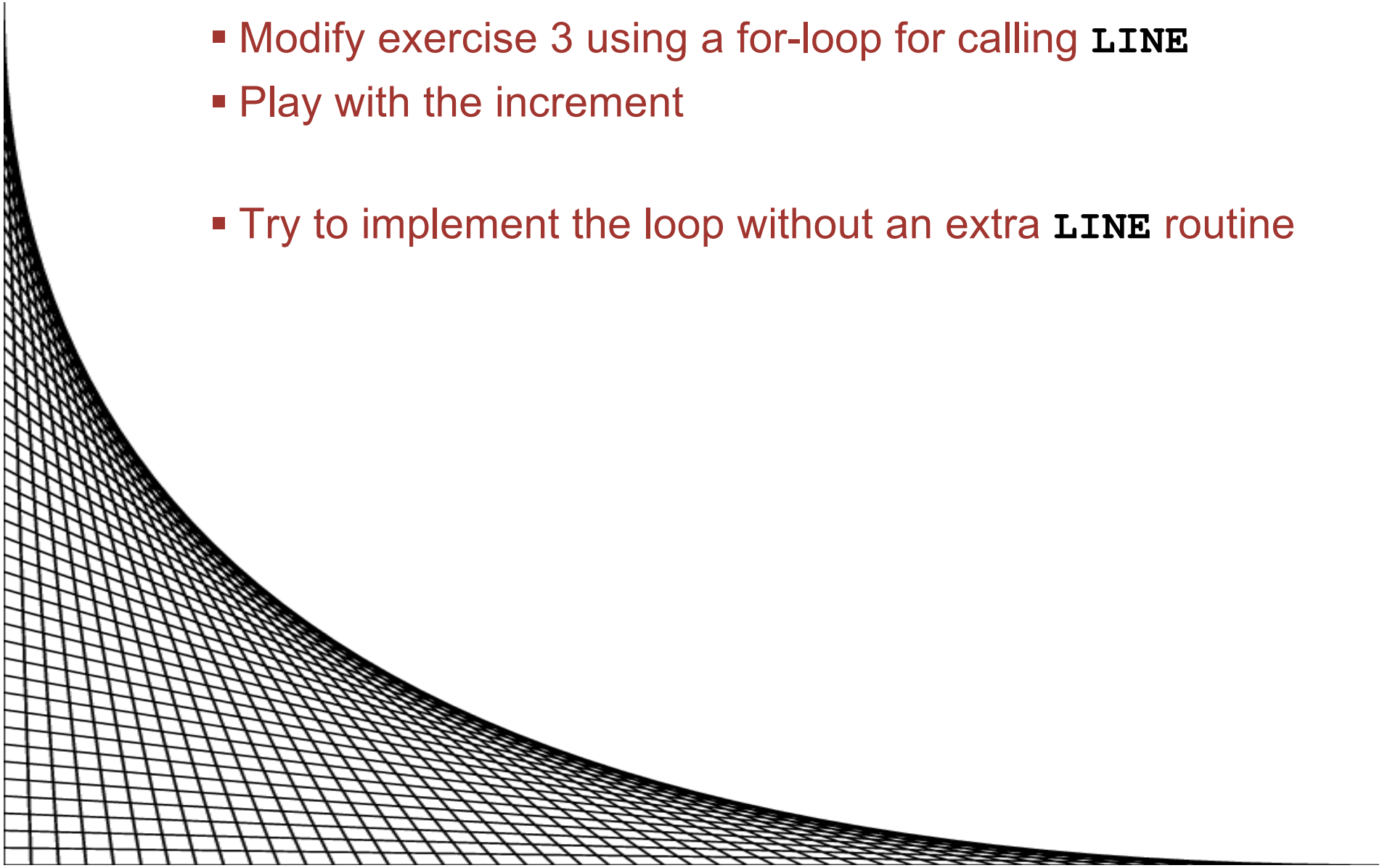


- Try to implement the loop without an extra **`LINE`** routine

Exercise 4b

- Modify exercise 3 using a for-loop for calling **`LINE`**
- Play with the increment

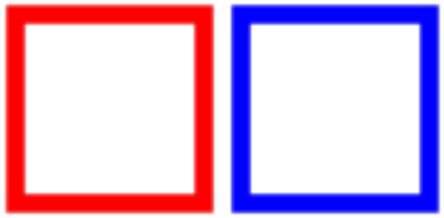
- Try to implement the loop without an extra **`LINE`** routine



Conditionals

- Conditional expression are possible
  - ***boolval*** **{…*commands*…} if**
  - ***boolval*** **{…*cmds_true*…} {…*cmds_false*…} ifelse**
- Boolean values can be
  - **true**
  - **false**
  - **x y eq**
  - **x y gt**
  - **bool1 bool2 or**
  - **bool not**
  - …

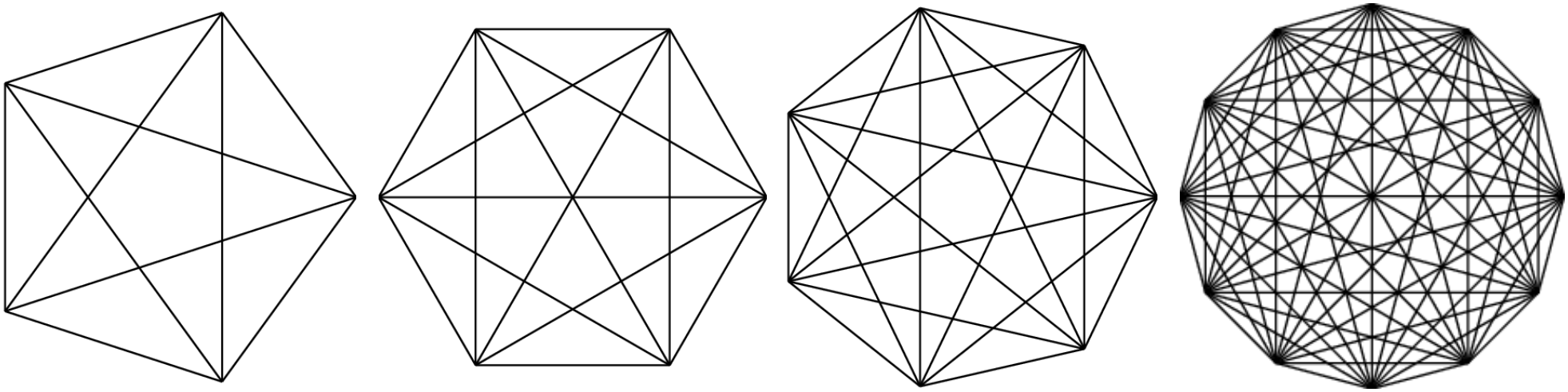
```
%!PS
/BOX { % Assume bool value on stack
 {1 0 0} {0 0 1} ifelse setrgbcolor
 0 0 10 10 rectstroke
} def

1  1 translate true  BOX
12 0 translate false BOX

showpage
```



- Can be used to comment out larger parts of code

Exercise 5

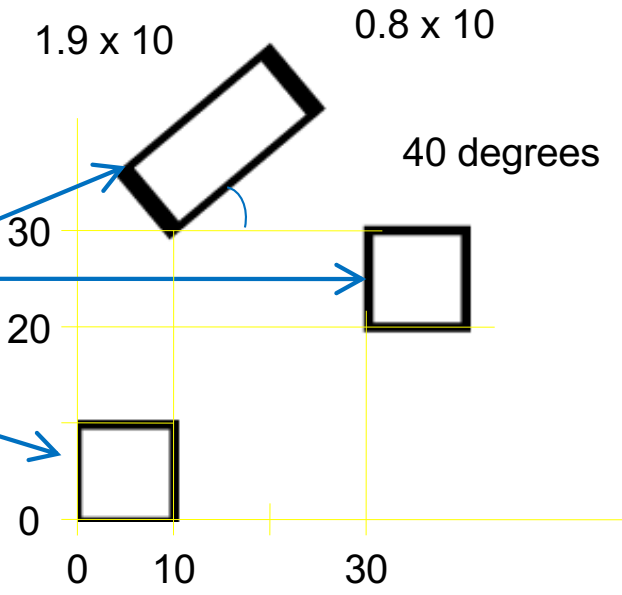
- This exercise is inspired by a problem in the ‘Mathekalender’ which offers a mathematics competition every year before Christmas at http://www.mathekalender.de
- Draw an N-fold polygon with all inner connections…
  - Use a double loop with 2 indices for the corners
  - Use a function to convert corner index to x/y (using trigonometry)



Translating and Scaling Things

- The coordinate system can be *translated*, *scaled* and *rotated* at any time.
- New transformations are 'added on top'
  - **`x y translate`**
  - **`x y scale`** % negative arguments are allowed → flip
  - **`phi rotate`** % angle in degree, as always

```
%!PS
/BOX {
  0 0 10 10 rectstroke
} def

BOX
30 20 translate BOX
-20 10 translate
40 rotate
1.9 0.8 scale
BOX
showpage
```



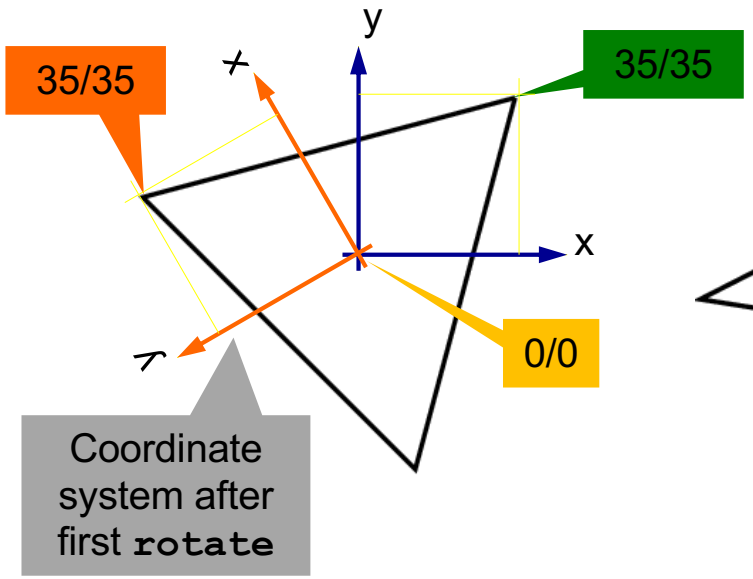
Applications of Coordinate Transformations

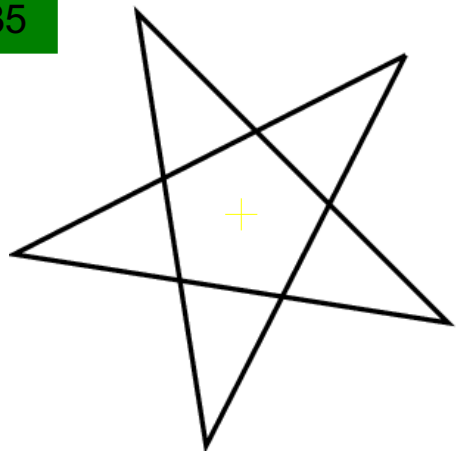
- Coordinate Transformations can simplify code *a lot*:

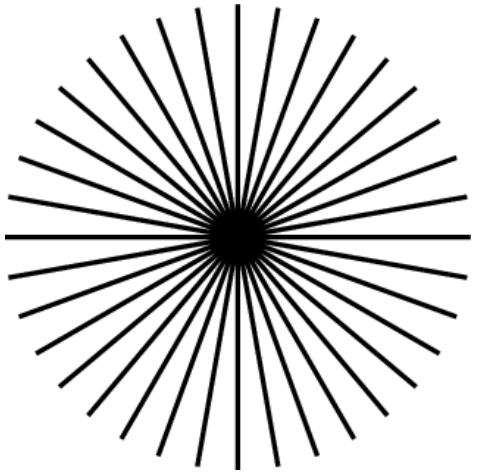
```
35 35 moveto
3 {
   120 rotate
   35 35 lineto
} repeat
stroke
```

```
35 35 moveto
5 {
   144 rotate
   35 35 lineto
} repeat
stroke
```

```
0 0 moveto
36 {
    50 0 lineto
     0 0 moveto
   10 rotate
} repeat
stroke
```







Converting Orientation and Units

- With

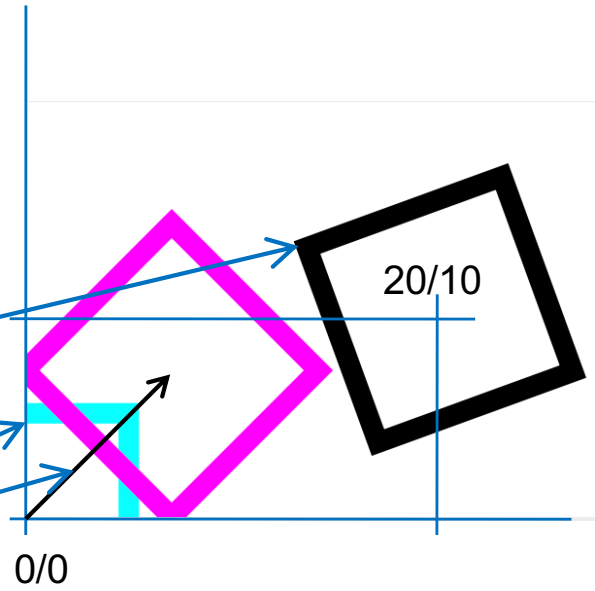
```
%!PS
2.835 dup scale          % now one unit is 1 mm
5 dup translate          % shift by 5/5 mm to center
0.1 setlinewidth         % line width is 0.1mm
newpath                  % draw a frame around A4
  0   0 moveto           % (in portait format)
  0 287 lineto
200 287 lineto
200   0 lineto
closepath
stroke
100 143.5 translate  % move origin to the center
```

  drawing can start in the center, in mm units.
- A frame is drawn around a A4 sheet.

Saving the Graphic State

- Temporary scaling / translating… operations often lead to 'corrupt' coordinate systems
- The graphics state can be remembered with **`gsave`** and restored with **`grestore`**
- Example:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 30 20
/BOX { -5 -5 10 10 rectstroke } def

gsave
20 10 translate
10 rotate
0 setgray BOX % black box
grestore
0 1 1 setrgbcolor BOX % magenta

45 rotate
10 0 translate
1 0 1 setrgbcolor BOX % pink
```



Re-using a Path

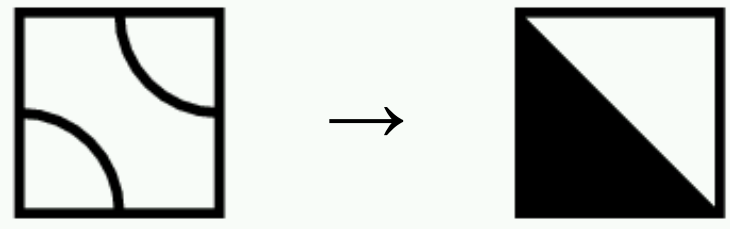
- A path is part of the graphics state
- It can be ‘saved’ with gsave

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 10 10

0.5 setlinewidth
1 1 translate 1 setlinejoin

newpath
0 0 moveto 5 0 lineto 5 5 lineto
closepath

gsave
0 1 0 setrgbcolor fill % green fill
grestore

1 0 0 setrgbcolor stroke % red outline
showpage
```

Exercise 6

- Understand how the Truchet Pattern on page 16 works
- Copy the code and play around
  - Change the number of tiles
  - Change the size of the tiles

- Replace the rounded tile by a triangle



Drawing Text

- Strings are delimited by `()`. Example: `(text)`
- A **font** must be selected before you can start drawing:
  - `/name findfont` — put font '*name*' to stack (height is 1 unit)
    (or `currentfont`)
    Some font names:
    - `Times-Roman`
    - `Helvetica-Bold`
    - `Courier`
  - `value scalefont` — resize the font (which must be on the stack)! (Result is on stack)
  - `setfont` — use it from now on (remove from stack)

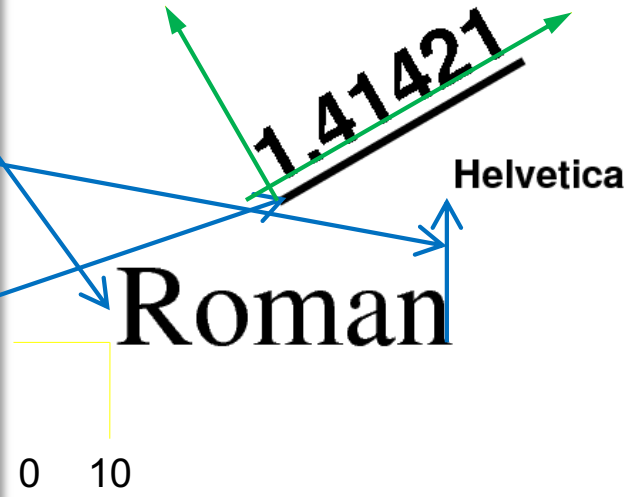
See also: selectfont

- Print a string (which is on the stack): `show`
  - start at current point. Current point moves to end of string!
  - Example: `0 0 moveto (Hello World) show`
- Convert a number to a string: `value 10 string cvs`
- Get width of a string: `strval stringwidth` (gets x *and* y)
  - Note: y is always zero and must often be `pop`ed

Drawing Text: Example

- Example:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 80 70

10 10 translate 0 0 moveto
/Times-Roman findfont
15 scalefont setfont (Roman) show

0 20 rmoveto
/Helvetica-Bold findfont
5 scalefont setfont (Helvetica) show

/x 2 sqrt 10 string cvs def

20 20 translate 30 rotate
0 0 moveto
currentfont 2 scalefont setfont
x show
0 -2 moveto
x stringwidth rlineto stroke

showpage
```



(Automatizing stuff) (PLRM page 316)

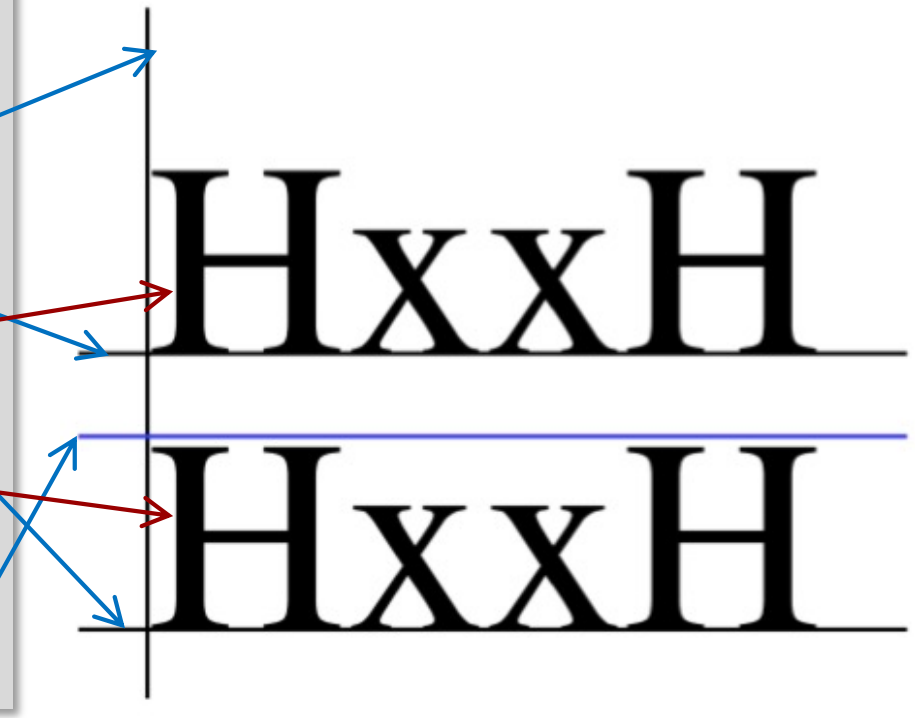
- If fonts are changed often, better define some functions to select them, for instance:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 80 70

/FSD { findfont exch scalefont def } bind def

/SMS {setfont moveto show} bind def
/MS  {moveto show} bind def

/F1 10 /Helvetica FSD
/F2 10 /Helvetica-Oblique FSD
/F3 10 /Helvetica-Bold FSD

(This is in Helvetica.) 10 78 F1 SMS
(And more in Helvetica.) 10 66 MS
(This is in Helvetica-Oblique.) 10 54 F2 SMS
(This is in Helvetica-Bold.) 10 42 F3 SMS

showpage
```

(A Detail: Font Size)

- Font height is from baseline to baseline
- **Character height** is ~ 0.7 x font height (depending on font)

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 70 60

10 10 translate

/FF 20 def
/Times-Roman findfont
FF scalefont setfont

0.3 setlinewidth
 0  -5 moveto 0 50 rlineto stroke
-5  FF moveto 60 0 rlineto stroke
-5   0 moveto 60 0 rlineto stroke

0 FF moveto (HxxH) show
0  0 moveto (HxxH) show

0 0 1 setrgbcolor
-5  FF 0.7 mul moveto 60 0 rlineto stroke

showpage
```



(Special Characters)

- The default character set does not contain special characters like ä,ö,ß,..
- This can be solved by assigning another character encoding to a font (see page 283 in PS Reference):

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 200 100

/Helvetica findfont dup length
dict begin {def} forall
/Encoding ISOLatin1Encoding def currentdict end
/HelveticaNew exch definefont pop

/HelveticaNew findfont  20 scalefont setfont
20 50 moveto
(äöü) show

/HelveticaNew findfont  30 scalefont setfont
20 20 moveto
(gßs) show

showpage
```

Test äöü
gßs

Exercise 7

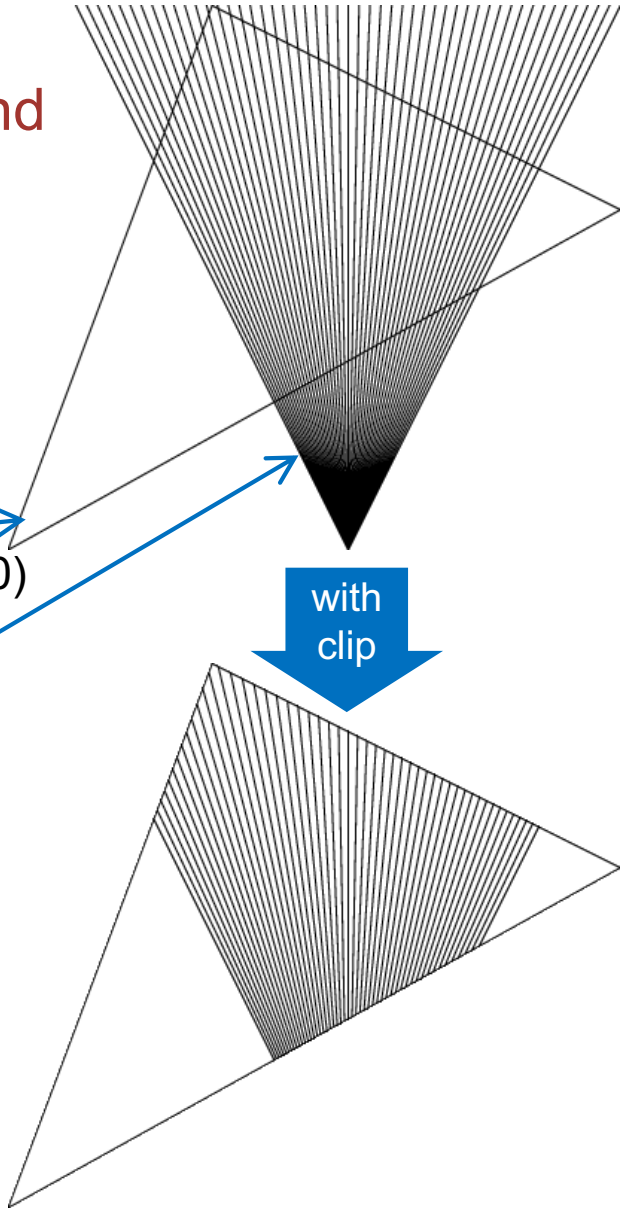
- Draw a box from (10,10) to (50,30)
- Print some text *centered* in the box
  - Use **`stringwidth`** to get the x- and y size of the text
  - Unfortunately, the y size is zero and cannot be used! Use the font height you have chosen instead.

(Advanced Topic: Clipping)

- A path can be used to restrict the drawing area using the **`clip`** command
- **`initclip`** clears the clipping path

construct clipping path

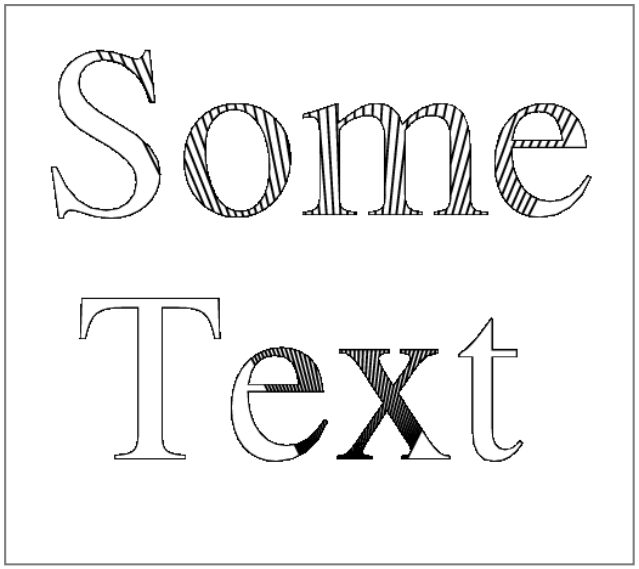
```
%!PS
0.2 setlinewidth

newpath 0 0 moveto
30 80 lineto 90 50 lineto
closepath
clip

0 2 100 {
  50 0 moveto 100 lineto
} for
stroke

showpage
```



(For fun: charpath)

- The outline of characters can be converted to a path using the **`charpath`** command. (Needs extra bool argument)
- Example using **`clip`**:

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 90 80
0.3 setlinewidth

/Times-Roman findfont
35 scalefont setfont
 5 50 moveto (Some) false charpath
10 15 moveto (Text) false charpath

clip

0 2 100 {
  50  0 moveto
    100 lineto
} for
stroke

showpage
```



(Advanced: Bit Maps)

- The command **`image`** draws a bit map in a *unit square*
  - To change size: scale in x- and y
- Parameters of **`image`** are:
  - Number of pixels in x
  - Number of pixels in y
  - Bits per pixel
  - A rotation matrix (not explained here..)
  - A function to get the values. Simplest case is a list of values
- Similar command is **`colorimage`**
  - It has some more parameters…

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 100 100

10 10 translate % move image to middle of page
80 80 scale     % make image one inch on a side
4 4 2 [4 0 0 4 0 0] {<fc1be400>} image
showpage
```

We need
4 x 4 x 2 = 32 bit

(Example for colorimage)

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 102 102

1 1 translate
100 100 scale

/picstr 6 string def

2 2 4 [2 0 0 -2 0 2]
{ currentfile picstr readhexstring pop}
false 3
colorimage
f000080f0088

showpage
```

Function to create a string of NX x NCOL characters

NX NY bits_per_col

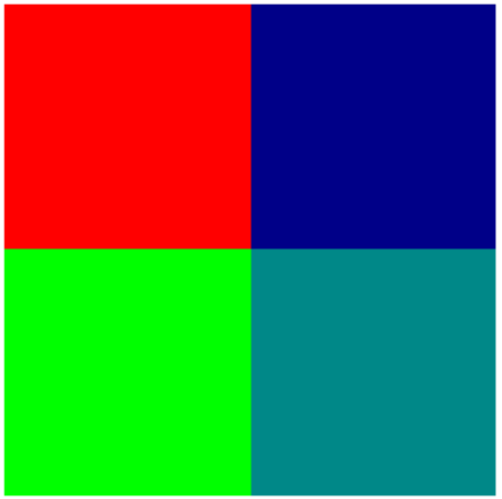
matrix

colors separate ?

NCOL

Function to retrieve data from file

Data:
2 x 2 x 4 x 3 bit
= 12 Byte



Stack Underflow

- A very common error is **stack underflow**
  - Not enough arguments on stack
- Debugging help:

Count
returns number of entries on stack

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox:  0 0 35 13

/PrintStackFillState
{count 10 string cvs show} def

/Helvetica findfont 15 scalefont setfont
1 1 moveto

1 2       PrintStackFillState
Add       PrintStackFillState
Dup       PrintStackFillState
pop pop   PrintStackFillState

showpage
```

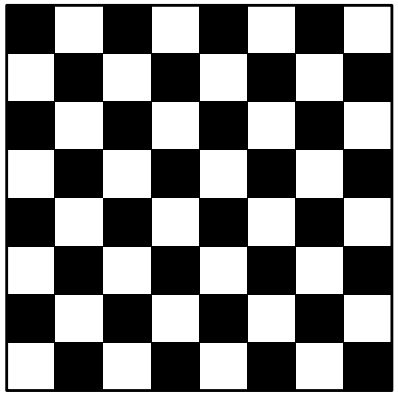
Output:

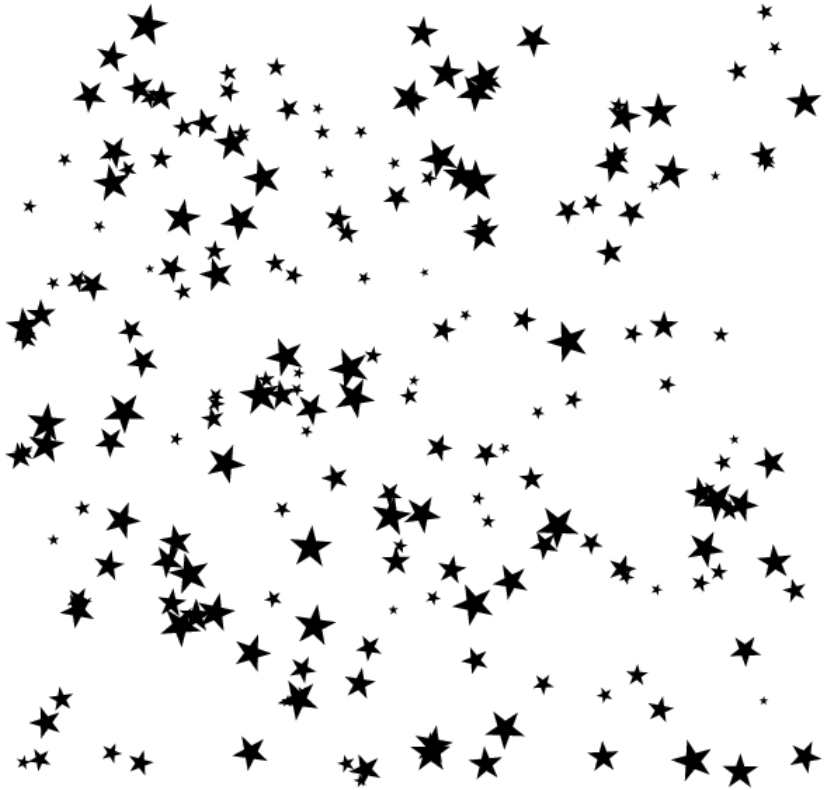
2120

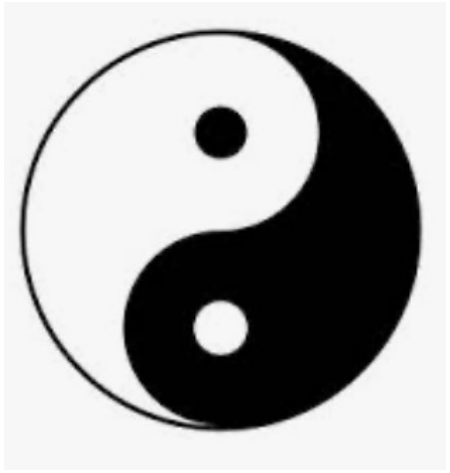
RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

NEED IDEAS TO TRY OUT YOUR NEW SKILLS ?

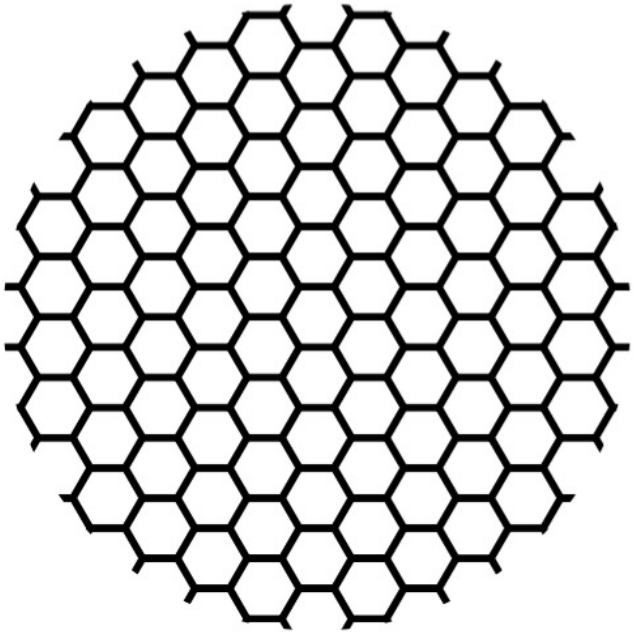
Inspirations

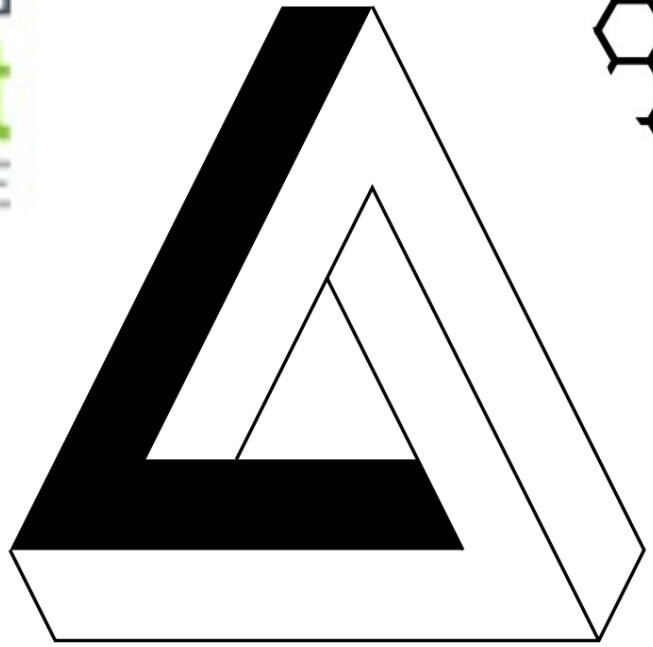




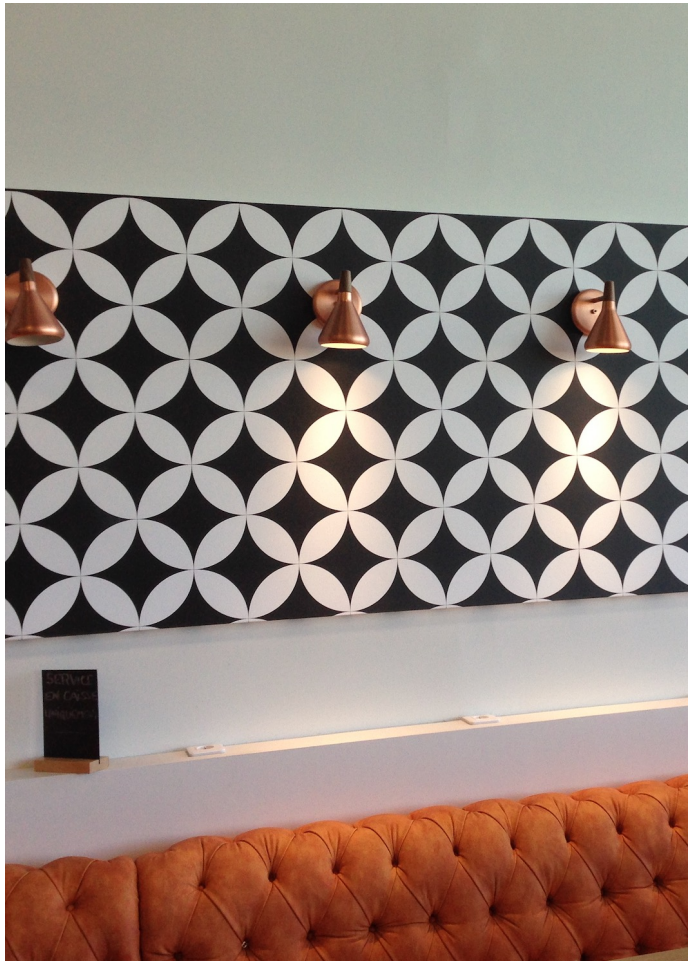


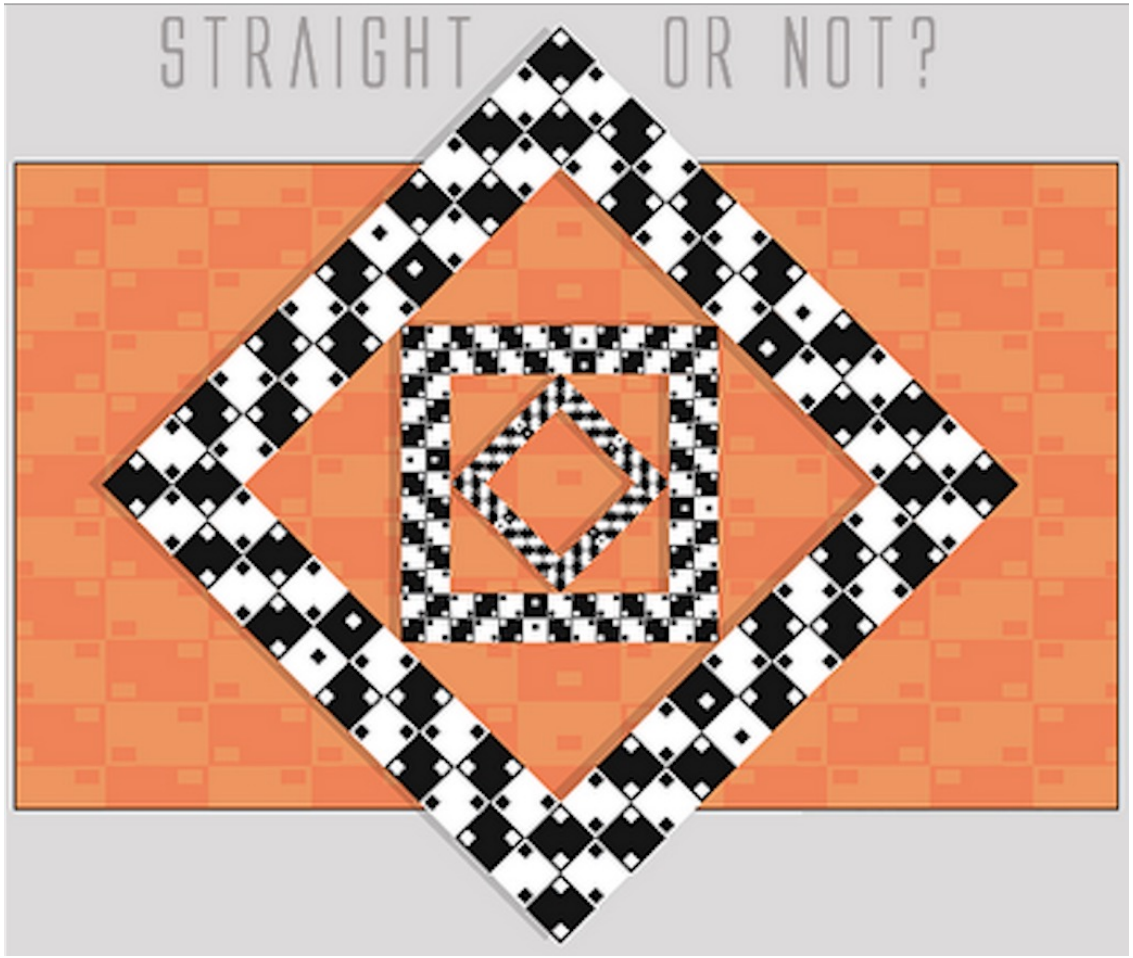
Inspirations





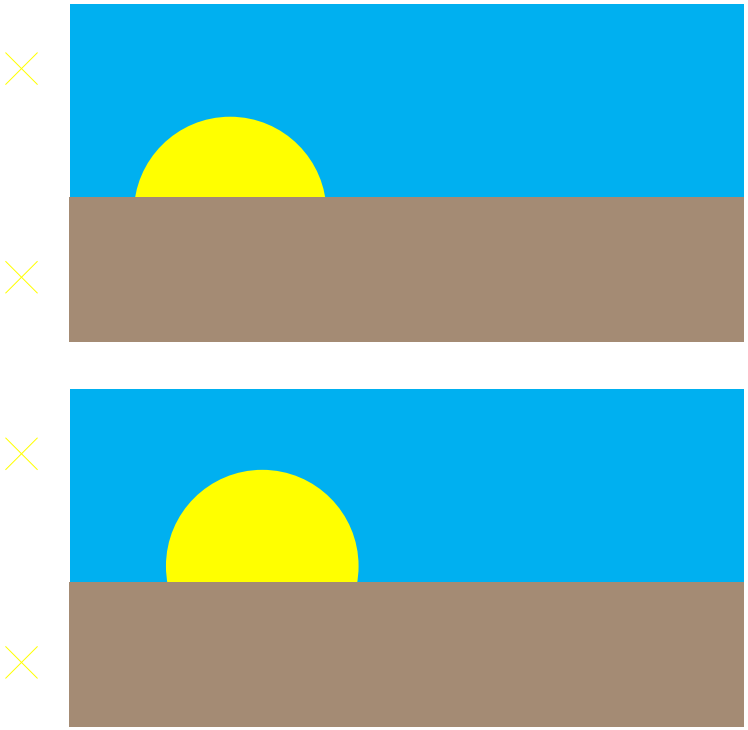
More Inspirations

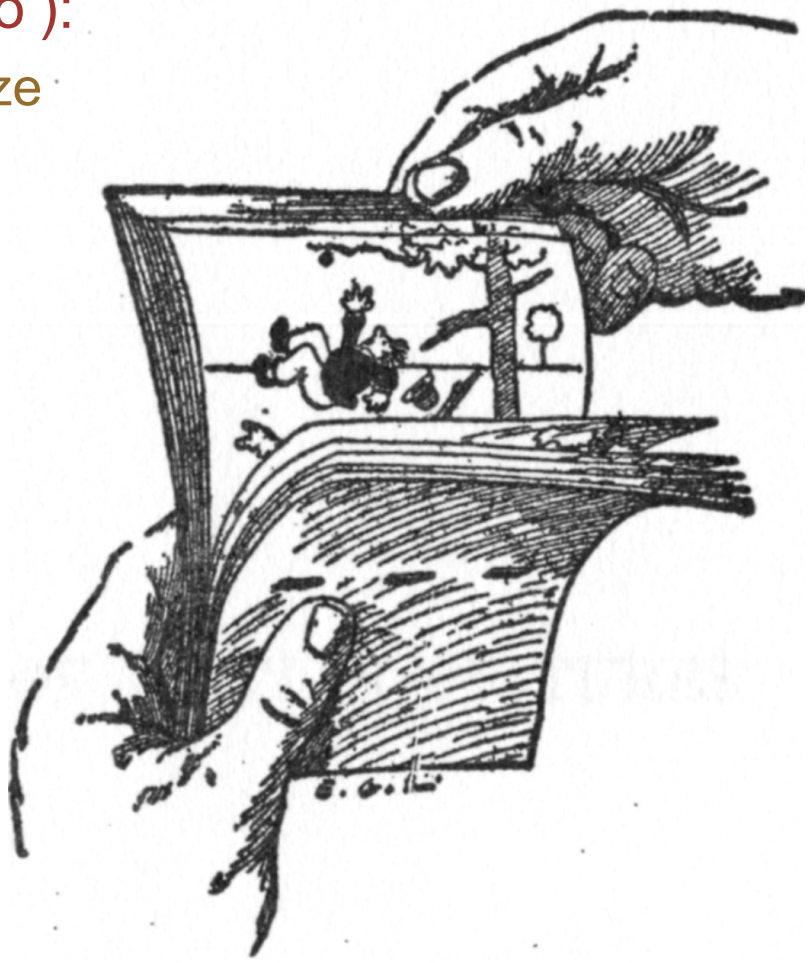




More Inspiration

- Make a Flipbook ('Daumenkino'):
  - Define a (small) sheet of unit size
  - Add alignment holes / pins / markers for later assembly
  - Add a scene drawing
  - Generate a series of sheets

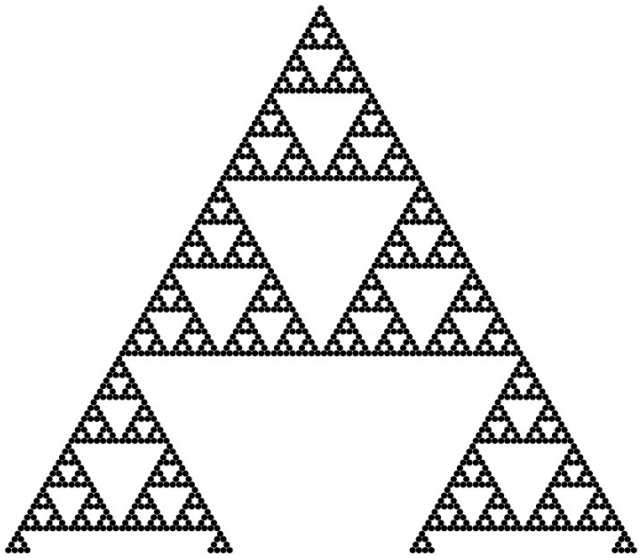




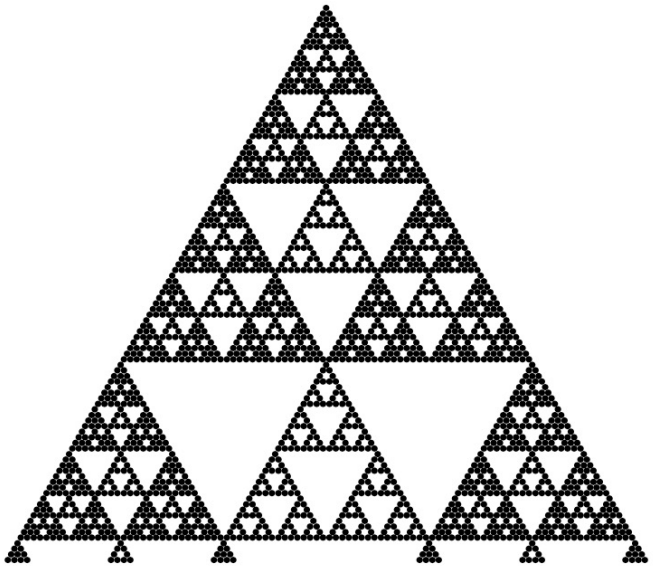
Wikipedia

More Inspirations

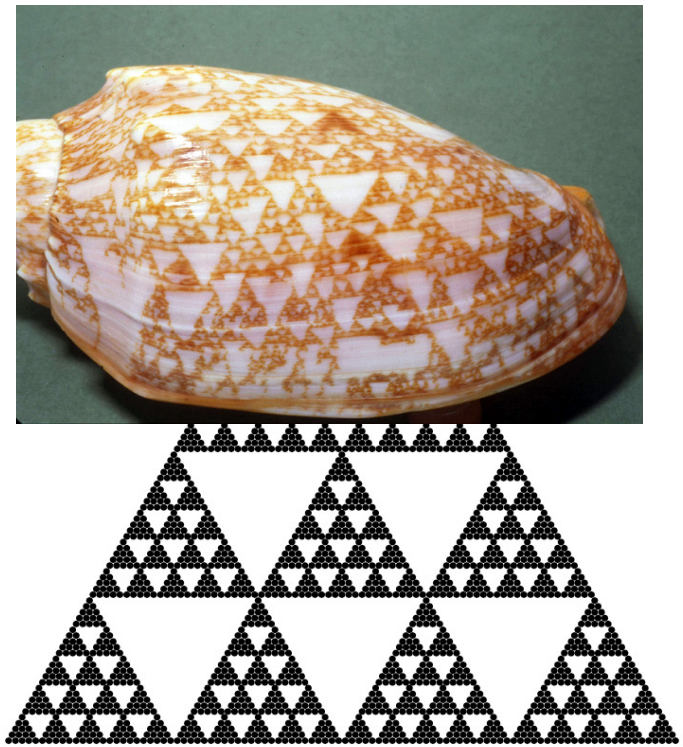
- Draw Pascal's Triangle (https://de.wikipedia.org/wiki/Pascalsches_Dreieck)
- Put a marker (circle, triangle,..) at the position of on all numbers which are multiples of some modulus K
- Examples:



K = 2



K = 4



K = 5

RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

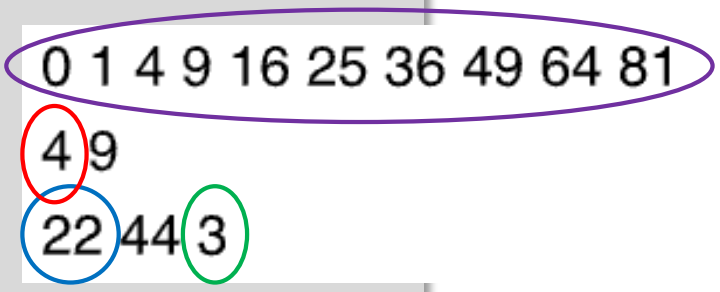
Misc

Data Structures: Arrays

- Arrays are like in other languages, but with RPN syntax:
  - ***n*** **`array`** → Create empty array with *n* elements
    This is now on stack.
  - **`[`*`val val`*** **`…]`** → initialized array. On stack
  - ***`arr`*** **`length`** → get length
  - ***`arr i val`*** **`put`** → set *i*-th element
  - ***`arr i`*** **`get`** → get *i*-th element
  - ***`arr proc`*** **`forall`** → execute *proc* for each element

Data Structures: Arrays - Example

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 300 100

% Define a function to print a number (and a blank)
/P { 10 string cvs show ( ) show } def
/Helvetica findfont 20 scalefont setfont   % select a font

1 1 moveto              % start show-ing at this position
/A [11 22 33] def       % define a named and initialized array
A 2 44 put              % overwrite value at index 2 (the '33')
A 1 get  P              % print value at index 1
A 2 get  P              % print value at index 2
A length P              % print the size

1 31 moveto
/N 10 def               % fix array size here
/B N array def          % define an empty array N elements
0 1 N 1 sub {dup dup mul B 3 1 roll put} for % fill it with squares
B 2 get  P              % check content
B 3 get  P              % check some more

1 61 moveto
B {P} forall            % Execute P for all elements

showpage
```



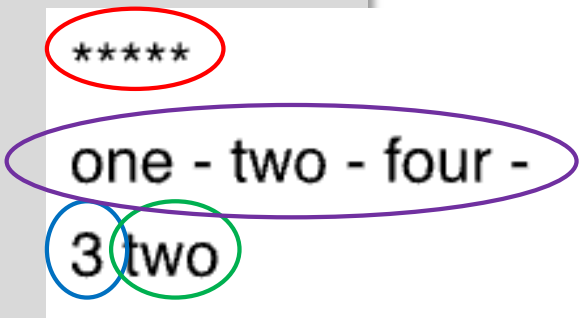
Data Structures: Dictionary

- Dictionaries contain (key/value) pairs
- Values are accessed through their key (dict. is 'searched')
- They are heavily used internally, for instance for fonts and procedures

- **`n dict`** → Create empty dictionary with a capacity of *n* elements. This is now on the stack.
- **`<<key val key val … >>`** → initialized dict. On stack
- **`d length`** → get length of dictionary *d*
- **`d key get`** → lookup *key* in dictionary *d*
- **`d key val put`** → fill (*key value*) to dictionary *d*
- **`d proc forall`** → execute *proc* for each element

Data Structures: Dictionary - Example

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 300 100

% Define a function to print a number (and a blank)
/P { 10 string cvs show ( ) show } def
/Helvetica findfont 20 scalefont setfont   % select a font

1 1 moveto
/A << 1 (one) 2 (two) 4 (four) >> def      % define a dictionary A
A length P                                 % check the size of A
A 2 get show                               % retrieve value for key '2'

1 31 moveto
A {show ( - ) show} forall % Execute something for all elements in A

1 61 moveto
/N 10 def                  % fix size here
/B N dict def              % define an empty dictionary with N elements
B 1 (*) put                % put a key-value pair
B 5 (*****) put            % put another key-value pair
B 5 get show               % retrieve value for key 5

showpage
```

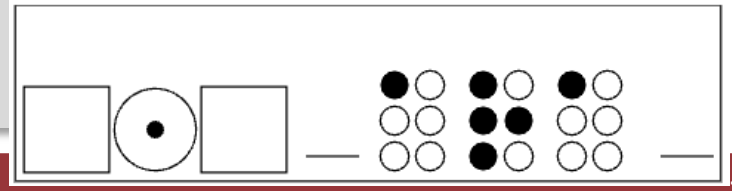


Defining Own Fonts

- You can define your own fonts (as drawing commands)
- See for instance PRLM, chapter 5.7.3

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 400 100

10 dict        % Define a new dictionary with some elements
begin          % push on dict stack -> we work with this now..
/FontType 3 def % Mandatory. See section 5.7 in PLRM
/FontMatrix [1 0 0 1 0 0] def % Mandatory.
/FontBBox [0 0 50 50] def % Mandatory.

/Encoding 256 array def % Declare an empty encoding array
0 1 255 {Encoding exch /.notdef put} for % assign /.notdef to all
Encoding 65 /ABraille put % 'A'
Encoding 66 /Box put % 'B'
Encoding 67 /Circles put % 'C'
Encoding 82 /RBraille put % 'R'

/CharProcs 6 dict def % Another (named) dictionary for glyphs
CharProcs begin % get it and push it on the dictionary stack

/.notdef {    % (normally) do nothing for undefined elements
10 10 moveto 40 10 lineto stroke     % do something for test
} bind def

/Circles {                % define drawing codes here....
  25 25 23 0 360 arc stroke
  25 25 5 0 360 arc fill
} bind def

/Box { 1 1 48 48 rectstroke } bind def

/ABraille {
  30 10 8 0 360 arc stroke 30 30 8 0 360 arc stroke
  30 50 8 0 360 arc stroke 10 10 8 0 360 arc stroke
  10 30 8 0 360 arc stroke 10 50 8 0 360 arc fill
} bind def
```

```
/RBraille {
  30 10 8 0 360 arc stroke 30 30 8 0 360 arc fill
  30 50 8 0 360 arc stroke 10 10 8 0 360 arc fill
  10 30 8 0 360 arc fill 10 50 8 0 360 arc fill
} bind def

end % End of CharProcs definitions

/BuildGlyph { % This stuff is a bit cryptic. See PLRM
  50 0                          % Width
  0 0 50 50                     % Bounding box
  setcachedevice                % ?? see PLRM
  exch /CharProcs get exch % Get CharProcs dictionary (see above)
  2 copy known not              % see if charname is known (??)
  {pop /.notdef}
  if
  get exec                      % Execute the code
} bind def

/BuildChar { % Language Level 1 compatibility, See PLRM
  1 index /Encoding get exch get
  1 index /BuildGlyph get exec
} bind def

currentdict                     % ??
end                             % End of dict definition (line 4/5)

/TestFont exch definefont pop % Define the font

1 1 398 99 rectstroke         % show bounding box
5 5 moveto
/TestFont findfont 1 scalefont setfont % get OUR OWN font
(BCB ARAx) show

showpage
```



Document File Structure

- The life of viewers, print managers, etc. is simplified by adding **Document File Structure** commands, as described in the '*PostScript Language Document Structuring Conventions Specification*' document (on Adobe Web Page)
- Main Document blocks are

• Comments	at beginning of document	Prolog
• Prolog	Procedure definitions	
• Setup	General setup	Script
• Pages	Content (opt.: PageSetup...)	
• Trailer	Cleanup	

- Structuring by
  - %%BeginProlog
  - %%EndProlog
  - ...

Document File Structure: Example

```
%!PS-Adobe-3.0 EPSF-3.0
%%BoundingBox: 0 0 200 300
%%Creator: Peter Fischer
%%Title: SomeTitle
%%CreationDate: Adate
%%Pages: 1
%%DocumentMedia: … descibe
  paper sizes here…
%%EndComments

/somecommand … def
%%EndProlog

%%BeginSetup
…
%%EndSetup

%%Page: 1 1
…
%%Trailer
…
%%EOF
```

Must be **one** block, i.e. without blank lines!

Columns (:) are important

Command Definitions

Setup before Drawing

Content of Page 1

Trailer

Multiple Pages

- Only possible in .ps, not in .eps
- **`%%Pages: N`** gives total number of pages
- **`%%Page: i i`** starts page i. i must be increasing per page
- Pages *should* be protected by save…restore
  (in case content redefines system commands)
- Example:

```
%!PS-Adobe-3.0
%%Pages: 2
%%EndComments

%%Page: 1 1
save …commands … restore

%%Page: 2 2
save …commands … restore

%%EOF
```

Error Messages

- GhostScript produces Error messages
  - To Clear the Console, Start Viewer from Scratch!
  - Scroll up to **First** error

- Some typical Errors
  - Stack underflow — command has too few arguments
  - No current point — `show` or `stroke` without point data
  - Variable not known — …
  - …

Converting to Other Formats

- Linux:
  - ps2eps
  - ps2pdf
  - epstopdf

- Convert
  - http://www.imagemagick.org/script/convert.php

Interactive Mode

- GhostScript can be run interactively
  - List stack using **`stack`** command
  - Not further covered here…

```
GPL Ghostscript 8.60 (2007-08-01)
Copyright (C) 2007 Artifex Software, Inc.  All rights :
This software comes with NO WARRANTY: see the file PUB
GS>2 3 4
GS<3>stack
4
3
2
GS<3>add
GS<2>stack
7
2
GS<2>/SUM2 { add dup mul } def
GS<2>SUM2
GS<1>stack
81
GS<1>_
```

History

- PostScript has been developed by Adobe
  - Level1 1984
  - (Display PostScript) 1988
  - Level 2 1991
  - PostScript 3 1997/98
- Info at http://www.adobe.com/devnet/postscript.html

- pdf is an ‚extension' of PostScript
  - All graphics possibilities are preserved
  - transparency
  - Better page structure (can scroll to pages without code analysis)
  - Interactive elements
  - …
  - But: No programming language!

Source: wikipedia

(atend)

- Some %%-Parameters can be deferred to the trailer section with (atend) instead of parameter
  - quite useful if page size or #pages not know at the beginning
- Example:

```
%!PS-Adobe-3.0
%%Pages: (atend)
%%EndComments

%%Page: 1 1
save …commands … restore

%%Page: 2 2
save …commands … restore

%%Trailer
%%Pages: 2

%%EOF
```

What Else ?

- Other PostScript features are
  - Control structures: loop, forall, …
  - Access to external files
  - Arrays
  - Dictionaries
  - …
- Crazy Stuff (from the web)
  - Raytracer
  - Cellular Automaton
  - Henon Attractor
  - …