Werkzeuge der Informatik

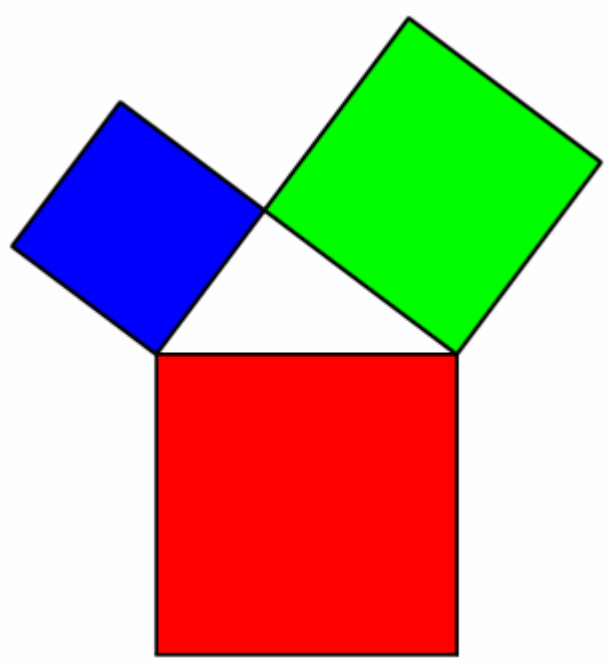
Einführung in PostScript®

Prof. Dr. Kai Hormann

Institut für Informatik

TU Clausthal

23.01.2009



Übersicht

- Wiederholung
  - einfache Zeichenoperationen
  - Zeichenattribute
  - lineare Transformationen
- Definition von Makros
- arithmetische Operationen
- einfache Schleifen
- Ausgabe von Text

Definition und Zeichnen von Pfaden

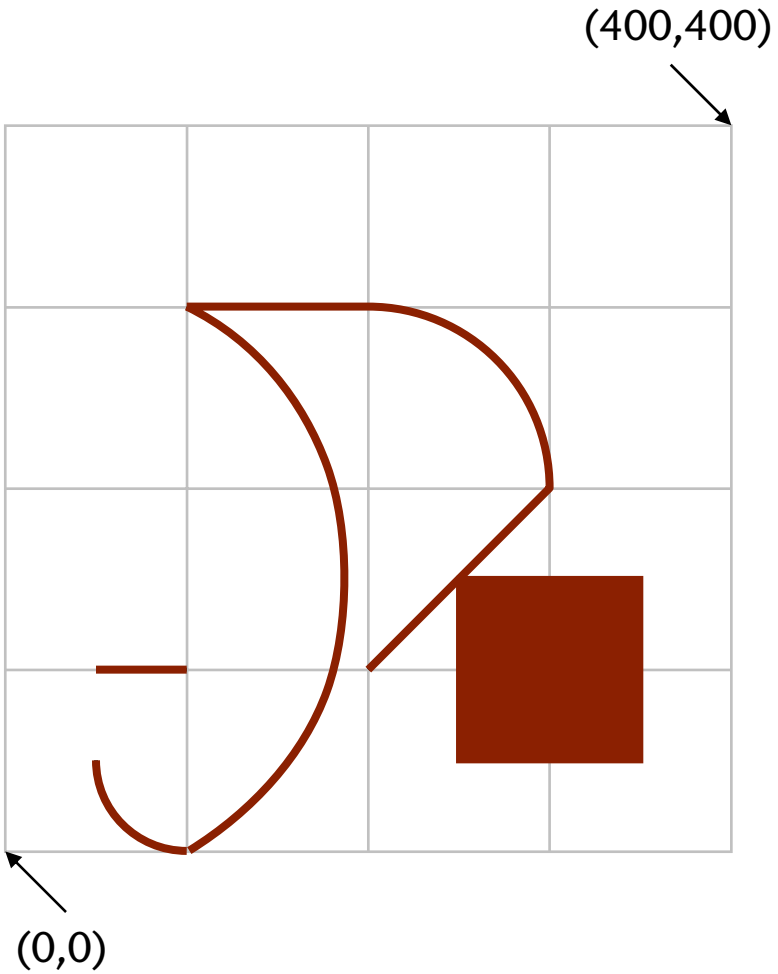
- Beginn eines Pfades `newpath`
- Stift bewegen ohne Zeichnen `moveto`, `rmoveto`
- Stift bewegen mit Zeichnen
  - gerade Liniensegmente `lineto`, `rlineto`
  - Kreisbögen `arc`, `arcn`
  - kubische Bézierkurven `curveto`
- Schließen eines Pfades `closepath`
- Zeichnen des Pfades
  - Umriß `stroke`
  - gefüllt `fill`

Definition und Zeichnen von Pfaden

- Beispiel

```
newpath
200 100 moveto
300 200 lineto
200 200 100 0 90 arc
-100 0 rlineto
200 250 200 50 100 0 curveto
100 50 50 270 180 arcn
0 50 rmoveto
100 100 lineto
stroke
```

```
newpath
250 50 moveto
100 0 rlineto
0 100 rlineto
-100 0 rlineto
closepath
fill
```



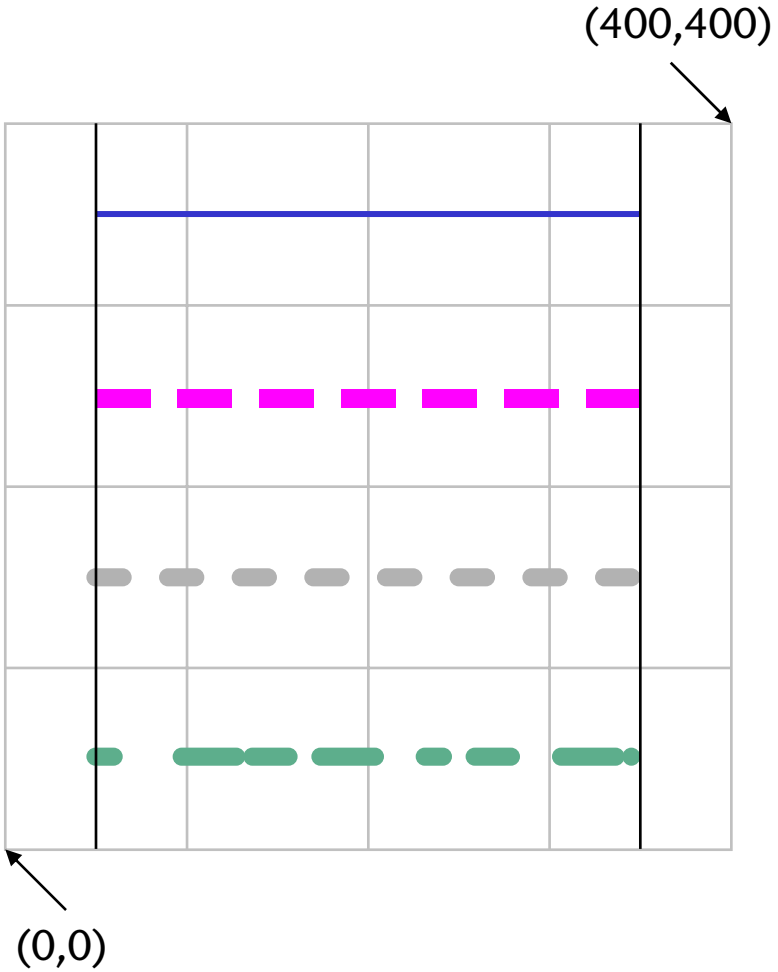
Verändern der Zeichenattribute

- Setzen der Sitft- oder Füllfarbe
  - farbig (rot, grün, blau) `setrgbcolor`
  - Grauton `setgray`
- Setzen der Linienattribute
  - Liniendicke `setlinewidth`
  - gestrichelte Linien `setdash`
  - Anfang und Ende der Linie `setlinecap`
  - Aussehen an Ecken `setlinejoin`

Verändern der Zeichenattribute

- Beispiel

```
0 0 1 setrgbcolor
50 350 moveto 300 0 rlineto
stroke
1 0 1 setrgbcolor
10 setlinewidth
[30 15] 0 setdash
50 250 moveto 300 0 rlineto
stroke
0.5 setgray
1 setlinecap
[15 25] 0 setdash
50 150 moveto 300 0 rlineto
stroke
0 0.5 0 setrgbcolor
[10 20 20 30 30] 40 setdash
50 50 moveto 300 0 rlineto
stroke
```



Lineare Transformationen

- Verschieben `translate`
- Drehen (um den Ursprung) `rotate`
- Skalieren `scale`
- Dabei gilt
  - Transformationen verändern das lokale Koordinatensystem
  - wirken auf **alle** nachfolgenden Objekte
- Zwei Sichtweisen
  - zeichne Objekt im aktuellen lokalen Koordinatensystem
  - zeichne Objekt in globalem Koordinatensystem und wende alle Transformationen rückwärts an

Lineare Transformationen

- Beispiel

```
100 100 25 0 360 arc
fill
```

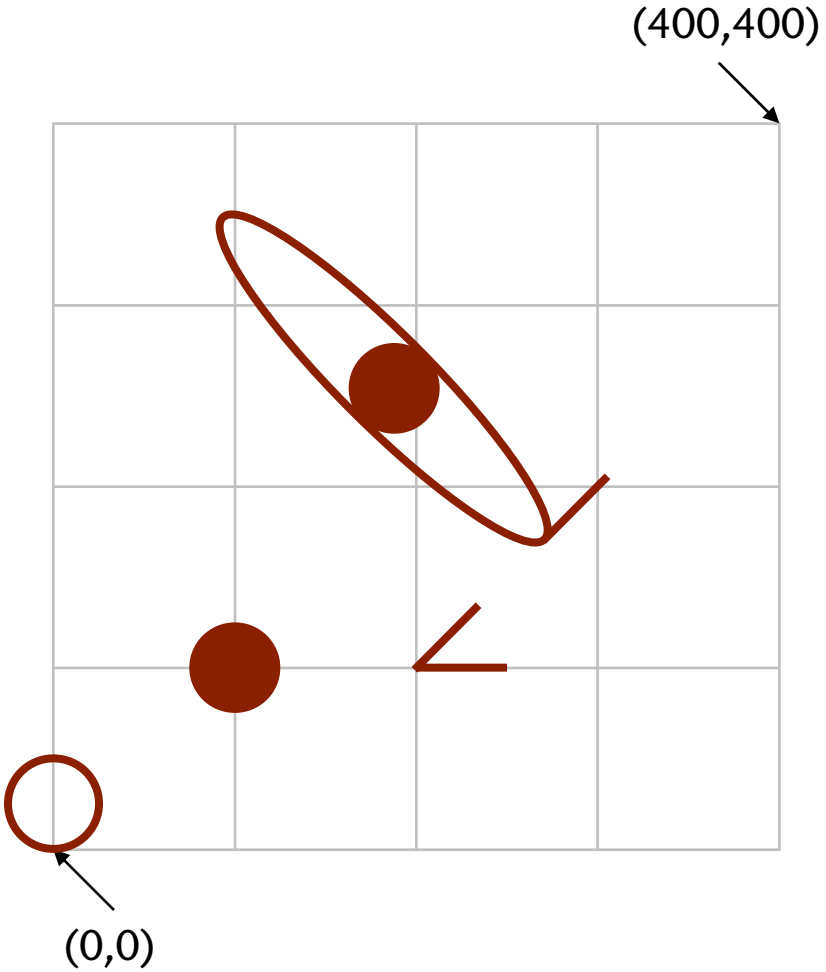
```
200 100 translate
0 0 moveto 50 0 rlineto
stroke
```

```
45 rotate
0 0 moveto 50 0 rlineto
stroke
```

```
100 0 translate
0 0 moveto 50 0 rlineto
stroke
```

```
1 5 scale
0 25 25 0 360 arc
stroke
```

```
1 0.2 scale
0 125 25 0 360 arc
fill
```



Lineare Transformationen

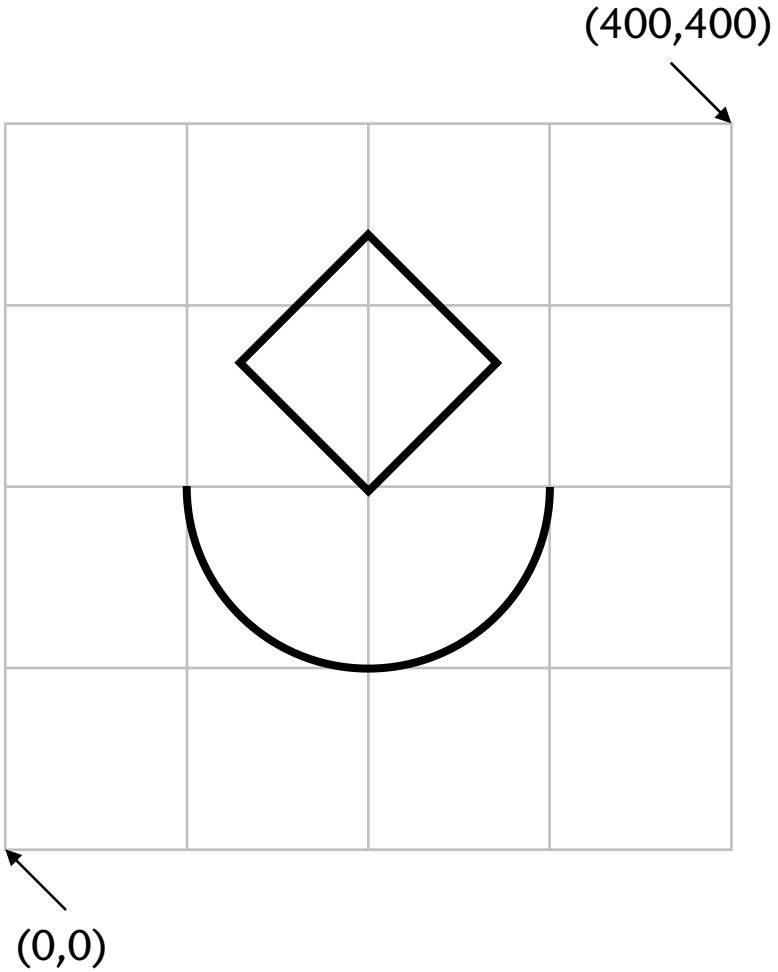
- Transformation(en) nur für ein Objekt
  - Transformation(en) ausführen
  - Objekt zeichnen
  - inverse Transformation(en) (in umgekehrter Reihenfolge) ausführen
- Alternative (besser)
  - Speichern des aktuellen Zustands `gsave`
    - inklusive aller anderen Graphikattribute
  - Wiederherstellen des Zustands `grestore`

Lineare Transformationen

- Beispiel

```
200 200 translate
45 rotate
newpath
0 0 moveto 100 0 rlineto
0 100 rlineto -100 0 rlineto
closepath stroke
-45 rotate
0 0 100 180 360 arc stroke
```

```
200 200 translate
gsave
45 rotate
newpath
0 0 moveto 100 0 rlineto
0 100 rlineto -100 0 rlineto
closepath stroke
grestore
0 0 100 180 360 arc stroke
```



Makros

- mehrfaches Ausführen derselben Befehlssequenz
- Beispiel
  - rotes Quadrat mit schwarzem Rand
- Sequenz als Makro zusammenfassen

```
/square {
  newpath
  0 0 moveto  100 0 lineto
  100 100 lineto  0 100 lineto
  closepath
} def
```

- allgemeine Form

  `/makroname { Befehlssequenz } def`

```
1 0 0 setrgbcolor

square

fill

0 setgray

square

stroke
```

Makros

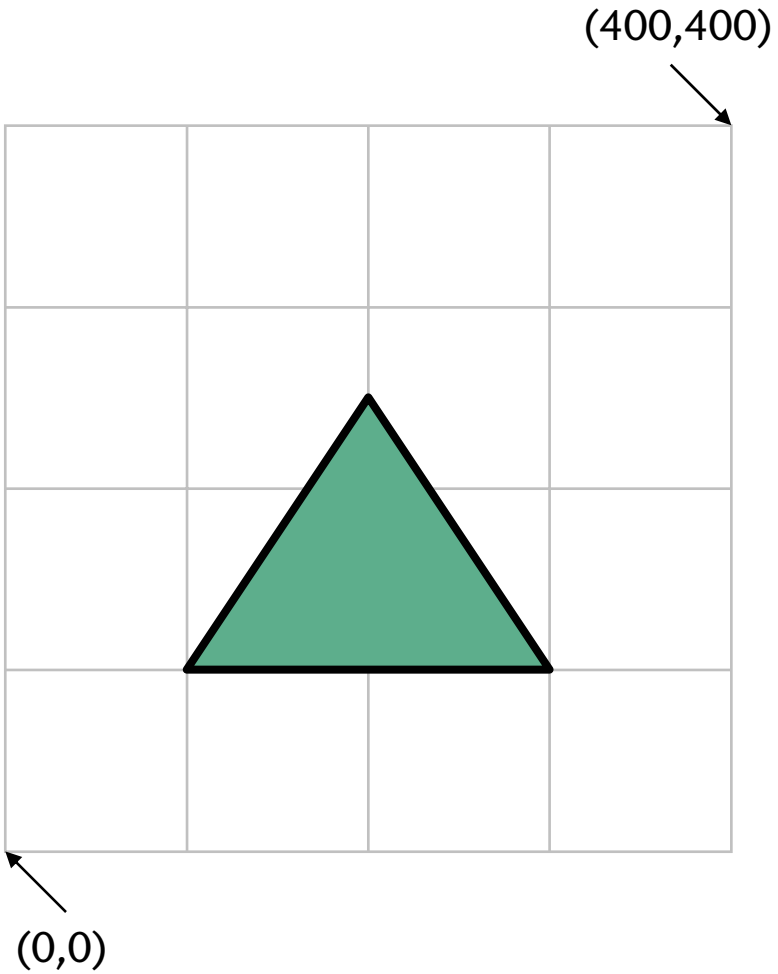
- Beispiel

```
/green {0 1 0 setrgbcolor} def
/black {0 setgray} def

/triangle {
  newpath
  0 0 moveto
  200 0 lineto
  100 150 lineto
  closepath
} def

%%EndProlog
```

```
100 100 translate
green triangle fill
black triangle stroke
```



Makros

- Nachteil des Quadrat-Makros
  - feste Größe von 100 × 100
- wünschenswert wäre ein Makro für Quadrate beliebiger Größe n × n
- 1. Lösung
  - definiere Makro „n“ je nach Bedarf

    `/n 50 def` oder `/n 100 def` etc.
  - benutze Makro „n“ im Quadrat-Makro
- eleganter wäre allerdings der Aufruf

  `50 square` oder `100 square` etc.

```
/square {
  newpath
  0 0 moveto
  n 0 lineto
  n n lineto
  0 n lineto
  closepath
} def
```

```
100 100 translate
/n 50 def
square fill

100 0 translate
/n 100 def
square stroke

0 150 translate
/n 50 def
square fill
```

Makros mit Argumenten = Funktionen

- genaue Funktionsweise von `def`
  - holt die beiden obersten Elemente vom Stapel
  - definiert zweitoberstes durch oberstes
- neuer Befehl `exch`
  - vertauscht die beiden obersten Elemente des Stapels
- 2. Lösung
  - verschiebe die Definition des Makros „n“ in das Quadrat-Makro
- benutze `1 dict begin` ... `end` zur Vermeidung von Namenskonflikten

```
/square {
 1 dict begin
  /n exch def
  newpath
  0 0 moveto
  n 0 lineto
  n n lineto
  0 n lineto
  closepath
 end
} def
/n 50 def

100 100 translate
50 square fill

100 0 translate
100 square stroke

0 150 translate
50 square fill
```

mehr Stapel-Operationen

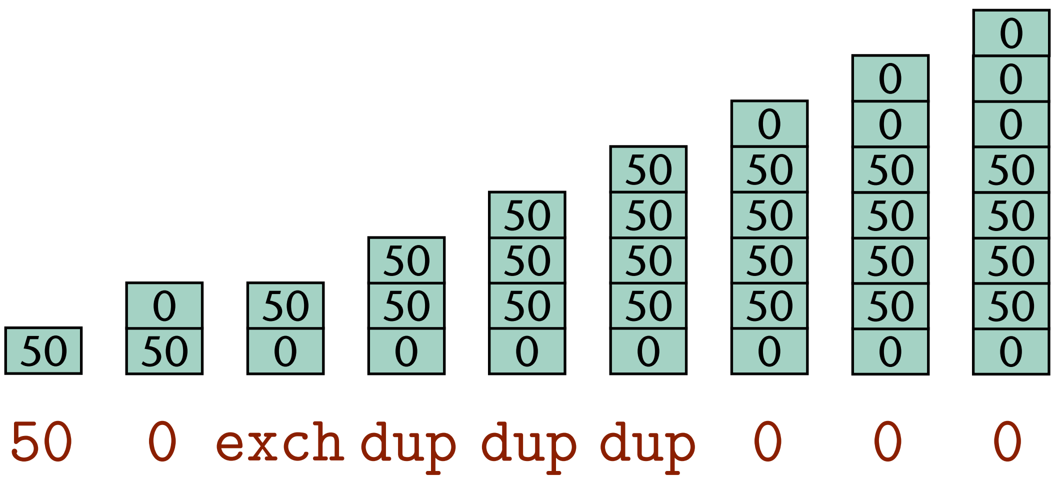
- neuer Befehl `dup`
  - legt ein Kopie des obersten Stapelelements auf den Stapel
- 3. Lösung
  - geschicktes Verdoppeln und Vertauschen
  - kommt ohne lokale Variable aus

```
/square {
  newpath
  0 exch dup dup
  dup 0 0 0
  moveto lineto
  lineto lineto
  closepath
} def
```

```
50 square
```

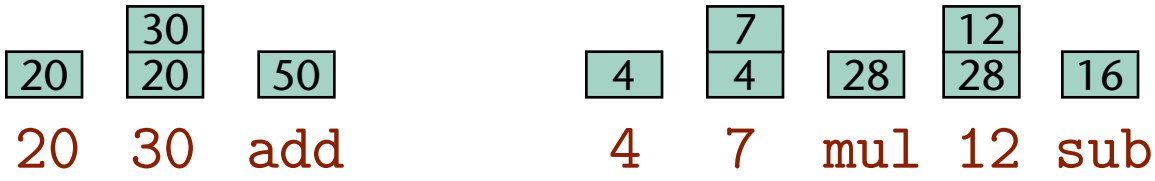
=

```
newpath
0 0 moveto
50 0 lineto
50 50 lineto
0 50 lineto
closepath
```



Rechnen in PostScript

- Arithmetische Operatoren
  - holen Operanden vom Stapel
  - legen Ergebnis wieder auf den Stapel
- Beispiel



- wichtigste Operatoren

`x y add` → $x+y$, `x y sub` → $x-y$, `x neg` → $-x$,

`x y mul` → $x*y$, `x y div` → x/y, `x abs` → $|x|$,

`x sin` → $\sin(x)$, `x cos` → $\cos(x)$, `x sqrt` → $\sqrt{x}$,

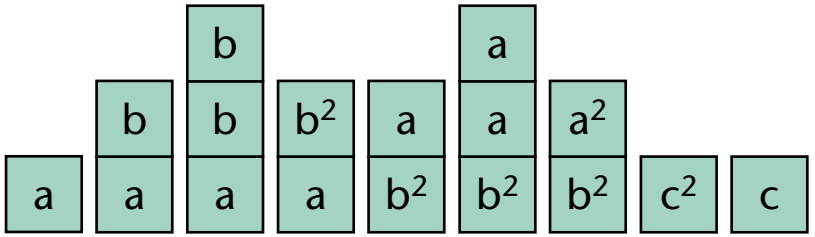
`x y atan` → $\arctan(x/y)$, `x floor` → $\lfloor x \rfloor$

Rechnen in Postscript

- Beispiel 1

```
/hypotenuse {
  dup mul exch dup mul add sqrt
} def

a b hypotenuse
```

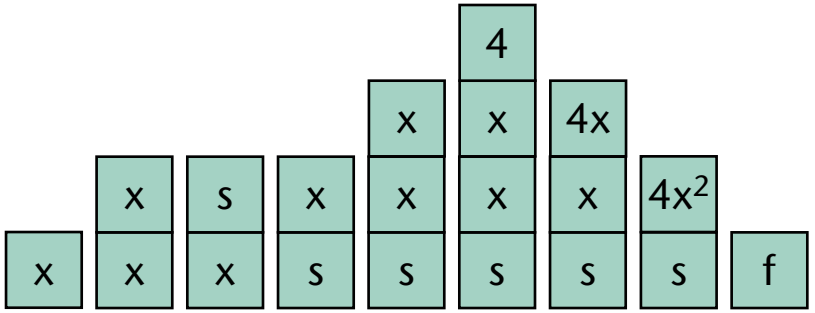


berechnet $c = \sqrt{a^2 + b^2}$

- Beispiel 2

```
/f {
  dup sin exch dup 4 mul mul add
} def

x f
```



berechnet $f(x) = 4x^2 + \sin(x)$

Zusammenfassung

- mit den Makros

```
/red   {1 0 0 setrgbcolor} def
/green {0 1 0 setrgbcolor} def
/blue  {0 0 1 setrgbcolor} def
/black {0 setgray} def
```

und den Funktionen

```
/square { ... } def

/fsquare {
  dup square fill
  black square stroke
} def

/hypotenuse { ... } def
```

können wir nun ganz leicht die Pythagoras-Figur erstellen

```
/pythagoras {
 3 dict begin
  /b exch def
  /a exch def
  /c a b hypotenuse def
  gsave
   0 c neg translate
   green c fsquare
   0 c translate
   b a atan rotate
   red a fsquare
   a 0 translate
   -90 rotate
   blue b fsquare
  grestore
 end
} def

200 300 translate
100 150 pythagoras
```

Schleifen in PostScript

- einfachste Art der Schleife

  `n { Befehlssequenz } repeat`

  führt die gegebene Befehlssequenz n-mal aus

- Beispiel

```
/square {
  newpath
  0 exch dup dup dup 0 0 0
  moveto lineto lineto lineto
  closepath
} def
```

=

```
/square {
  newpath
  0 exch dup dup dup 0 0 0
  moveto 3 {lineto} repeat
  closepath
} def
```

- weitere Schleifen- und Verzweigungsbefehle

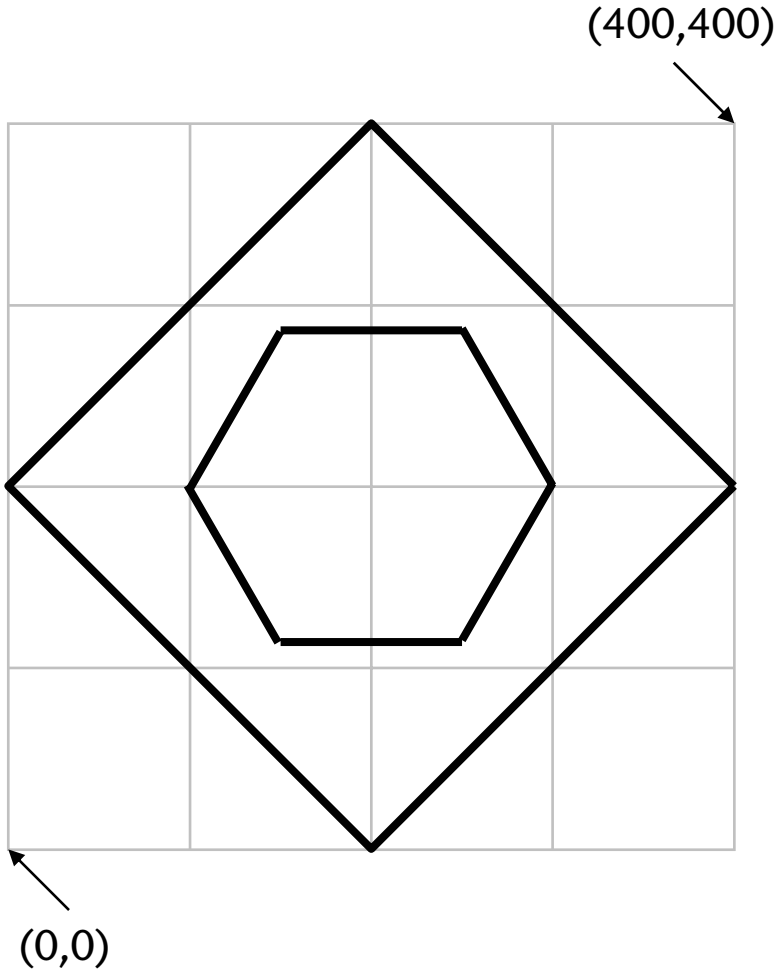
  `for`, `if`, `ifelse`, `loop`, `exit`, ... fast wie in C/C++

Schleifen in PostScript

- Beispiel 1

```
/n-gon {
 3 dict begin
  /l exch def
  /n exch def
  /a 360 n div def
  newpath
  l 0 moveto
  n {
    a rotate
    l 0 lineto
  } repeat
  closepath
 end
} def
```

```
200 200 translate
4 200 n-gon stroke
6 100 n-gon stroke
```

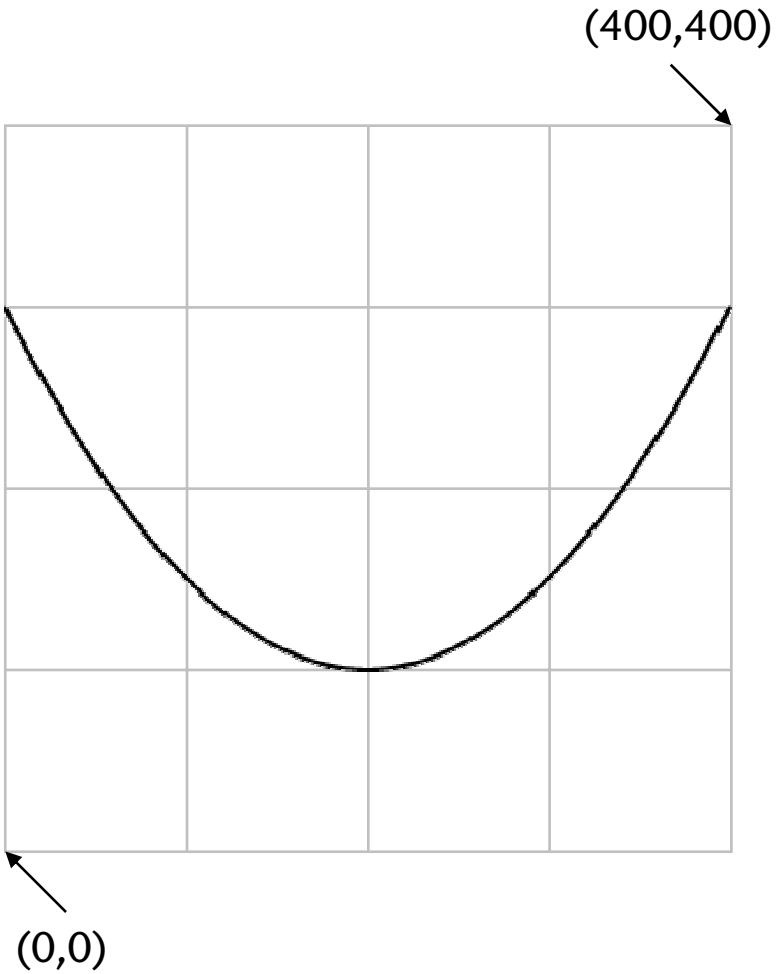


Schleifen in PostScript

- Beispiel 2

```
/f {dup mul} def
/plot-f {5 dict begin
  /n exch def
  /max exch def
  /min exch def
  /x min def
  /dx max min sub n div def
  x dup f moveto
  n {
    /x x dx add def
    x dup f lineto
  } repeat
  stroke
end} def
```

```
200 100 translate 200 200 scale
0.01 setlinewidth
-1 1 100 plot-f
```



Textausgabe in PostScript

- Setzen eines Zeichensatzes „Font“ in Größe p

```
/Font findfont p scalefont setfont
```

  - Verfügbare Zeichensätze: „Times“ und „Helvetica“, optional mit Ergänzung „-Oblique“ (kursiv) oder „-Bold“ (fett)

- Ausgabe der Zeichenkette „text“

```
(text) show
```

- Umwandlung einer Zahl „x“ in eine Zeichenkette

```
/str 100 string def
x str cvs
```

- Umwandlung der Zeichenkette „text“ in einen Pfad

```
(text) false charpath
```

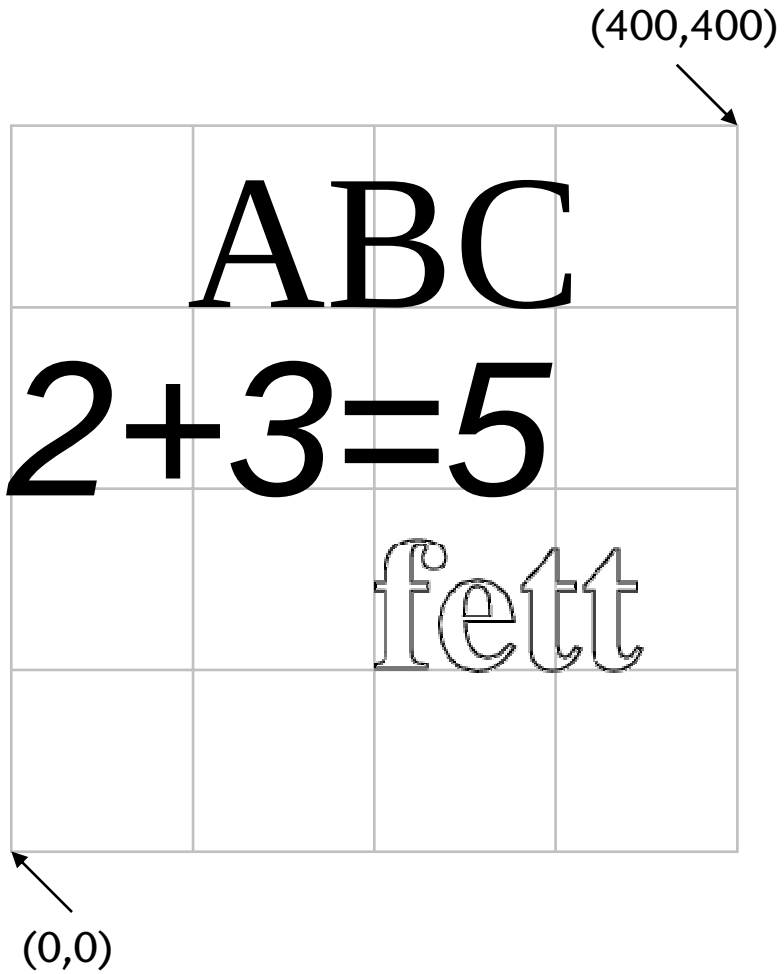
Textausgabe in PostScript

- Beispiel

```
/Times findfont
100 scalefont setfont
100 300 moveto
(ABC) show

/Helvetica-Oblique findfont
100 scalefont setfont
0 200 moveto
/str 10 string def
2 str cvs show (+) show
3 str cvs show (=) show
5 str cvs show

/Times-Bold findfont
100 scalefont setfont
200 100 moveto
(fett) false charpath
stroke
```



Zusammenfassung

- Definition von Makros

  `/makroname { Befehlssequenz } def`

- wichtigste Stapeloperatoren

  `exch`, `dup`

- arithmetische Operatoren

  `add`, `sub`, `mul`, `div`, `sin`, ...

- einfachste Art der Schleife

  `n { Befehlssequenz } repeat`

- Text- und Zahlenausgabe

  `(text) show` bzw. `n 100 string cvs show`