Wir programmieren mit *"Turtle Graphics"*

Klaus Kusche

Programmmieren

= dem Computer sagen, was er tun soll

= ein Programm schreiben

Programm

= ein “Text” mit ***Befehlen*** an den Computer

... geschrieben in einer

“Programmiersprache”

Die Programmiersprache ...

legt fest:

- ***Welche Befehle*** **es gibt**
- ***Wie sie aussehen***

  (daran muss man sich genau halten!)
- ***Was sie bedeuten*** **oder bewirken**

  (z.B. rechnen, etwas anzeigen,
  etwas in einer Datei speichern,
  den Benutzer etwas eintippen lassen...)

Welche Programmiersprache?

Es gibt sehr *viele Programmiersprachen*:

Java, JavaScript, Python, C, SQL, ...

Heute: Eine Sprache *speziell für diesen Kurs*

für "Turtle Graphics"

==> Unsere Befehle

... jagen eine "Schildkröte" über den Bildschirm

... die dabei einen Strich zeichnet

Unsere “Schildkröte” …

- *sieht man nicht!*
- hat immer

eine Position,
auf der sie gerade sitzt
(am Anfang: *In der Fenstermitte*)

- und

eine Richtung,
in die sie gerade schaut und geht
(am Anfang: *Nach rechts*)

Unsere ersten Befehle: Gehen

- **`geh`** *schritte*

  Zum Beispiel: **`geh 10`**

  Die Schildkröte geht
  10 Schritte in die aktuelle Richtung
  und *zeichnet dabei einen Strich*

  Das ganze Fenster ist 40 x 30 Schritte groß
  `geh -10` *oder* **`geh rueckw 10`** *ist auch erlaubt, geht rückwärts*
  `geh heim` *geht zurück an den Startpunkt (Fenstermitte)*

- **`spring`** *schritte* dasselbe, aber *ohne Strich*

Unsere ersten Befehle: Drehen

- **`dreh`** *grad* oder **`dreh rechts`** *grad*

  Zum Beispiel: **`dreh 90`**

  Ändert die Richtung der Schildkröte
  um 90 Grad *im Uhrzeigersinn*

  ==> Das nächste **`geh`** oder **`spring`**
  geht *in die neue Richtung*!

- **`dreh`** –*grad* oder **`dreh links`** *grad*

  dasselbe, aber *gegen den Uhrzeigersinn*

Das erste Programm

```
programm
    dreh 60
    geh 10
    dreh 120
    geh 10
    dreh 120
    geh 10
    stop
prog_ende
```

Der Computer fängt hier an

… und macht *der Reihe nach*

… *einen Befehl nach dem anderen*

“fertig, aber lass das Fenster offen”
Ohne **stop** geht es nach 2 sec zu

Ende des Programms

Schleifen

Computer machen oft pfeilschnell
Millionen mal immer wieder dasselbe...

... aber dieselben Zeilen Millionen mal in das Programm tippen wäre mühsam!

==> Es gibt Befehle, die die Befehle dahinter ***wiederholen***, d.h. *mehrmals machen*

Diese Befehle heißen "Schleifen" weil das Programm dort "im Kreis läuft"

Die *“mach x Mal”*-Schleife

…

`mach` *zahl* **`mal`** Wiederholt *zahl* Mal

befehle … alle diese Befehle

…

`mach_ende` … bis hierher

… … und macht dann ganz normal weiter

Kannst Du damit ein Zwölfeck zeichnen?
(Dreh-Winkel in den Ecken 30 Grad, Seitenlänge 3)

Variablen

Programme müssen sich oft *Zahlen merken*

*Solche **Speicherplätze für Zahlen***
*heißen **“Variablen”***

Man gibt jeder solchen Variable
*einen **sinnvollen Namen***

Bitte für die Namen nur Buchstaben und **_** *verwenden,*
keine Umlaute und kein ß !

Unsere Variablen können Kommazahlen speichern
Kommazahlen schreibt man englisch mit **.** *statt* **,** *!*

Variablen verwenden

Variablen (und Rechnungen mit Variablen) können in Befehlen *statt Zahlen* verwendet werden:

Das Programm nimmt dann die Zahl, die zuletzt in der Variable gespeichert wurde

Beispiele:

```
geh seitenl
dreh 360 / ecken
```

Mit Variablen rechnen

Beispiele für Rechnungen:

```
speichere 360 / ecken in winkel
add 0.5 zu seitenl
sub 1 von anzahl
mul seitenl mit 2
div winkel durch 3
```

add, **sub**, **mul** und **div** *speichern* ihr Ergebnis wieder *in derselben Variable*.

Die *"solange"*- Schleife

```
programm
   speichere 0.1 in seitenl
   solange seitenl < 3 mach
     geh seitenl
     dreh 10
     mul seitenl mit 1.02
   mach_ende
   stop
prog_ende
```

Die Zähler-Schleife

`zaehler` *var* **`von`** *anfang* **`bis`** *ende* **`mach`**
    befehle ...
`mach_ende`

... *zählt* mit der *Variable* *var* von *anfang* bis *ende*

... und wiederholt *für jeden Wert* der Variable einmal die Befehle

- Mit **`runter_bis`** statt **`bis`** kann man *rückwärts* zählen
- Mit **`schritt`** *xxx* nach **`bis`** *ende* kann man statt um 1 jedesmal *um xxx weiterzählen*

Beispiel Zähler-Schleife

```
programm
   zaehler winkel von 0 bis 360
                              schritt 4 mach
     dreh winkel
     geh 12
     spring heim
   mach_ende
prog_ende
```

Funktionen

Um das Programm *übersichtlicher* zu machen, kann man ein Programmstück

- *außerhalb* von **`programm`** und **`prog_ende`** schreiben
- und ihm ***einen eigenen Namen*** geben

Die Informatiker sagen zu solchen *benannten Programmstücken* ***"Funktion"***, bei uns heißen sie **`figur`**.

Beispiel einer `figur`

```
figur quadrat
  mach 4 mal
    geh 0.5
    dreh 90
  mach_ende
figur_ende
```

Das gehört *vor* **`programm`** !

Da wird *noch nichts gezeichnet*:

Wir erklären dem Rechner nur, welche Befehle er für ein **`quadrat`** ***machen soll!***

Verwendung einer `figur`

Jedesmal, wenn man dann irgendwo
zwischen **`programm`** und **`prog_ende`**
den Befehl **`figur quadrat`** hinschreibt,
werden *an dieser Stelle*
die Befehle der Figur ausgeführt!

Das nennt der Informatiker
"aufrufen" der Funktion.

Versuche, in der **`solange`***-Schleife deiner Spirale jedesmal ein solches Quadrat zu zeichnen!*

“Parameter”

Unser **`quadrat`** ist noch *unpraktisch*:
Es kann nur ein Quadrat *fixer Größe* zeichnen.

Man kann auch Figuren programmieren,
*die **jedesmal andere Zahlen** verwenden.*

Der Informatiker nennt die Werte,
die man erst beim Aufruf
einer Funktion festlegt,
“Parameter” der Funktion.

figur-Parameter (1)

Man schreibt hinter den Figur-Namen in **( )**

- …beim ***Programmieren*** der Figur:

  Die *Liste der änderbaren Variablen*

  Beispiel: Wir wollen eine **figur**
  für ein **n**-Eck mit Seitenlänge **sl** schreiben
  figur n_eck(n, sl)

  Das **mach**, das **geh** und das **dreh** in der Figur
  müssen dann natürlich **n**, **sl** und **360/n**
  statt **4**, **0.5** und **90** verwenden!

figur-Parameter (2)

Man schreibt hinter den Figur-Namen in **( )**

- …beim ***Verwenden*** der Figur:

  Die *Werte*, die für *diese Variablen*
  beim Zeichnen *eingesetzt* werden sollen

- Also **`figur n_eck(12, seitenl)`**

  wenn wir in unserer Spirale ein Zwölfeck
  mit Seitenlänge **`seitenl`** zeichnen wollen

Jede Figur hat ihre eigenen Variablen!

Für Fortgeschrittene (1)

- Mit **`speichere 10 in @verz`** machst du das Zeichnen *schneller* (das ist die *Verzögerung* in Millisekunden *pro Strich*).
- Der *Befehl* **`farbe`** *rot*`,` *grün*`,` *blau* *ändert die Farbe*.
  Die drei Farben haben Werte zwischen 0 und 100.
  Beispiele: 0, 0, 0: schwarz, 100, 100, 100: weiß, 100, 0, 0: rot,
  0, 0, 100: blau, 100, 50, 0: orange, 0, 100, 50: blaugrün usw.

  Mit den *Variablen* **`@rot @gruen @blau`**
  kann man die Farbwerte *einzeln abfragen* und auch *ändern*.
- Der *Befehl* **`richtung`** *grad* setzt die Richtung auf *grad* (0 ist rechts).
  Die *Variable* **`@richtung`** liefert die *aktuelle Richtung* in Grad,
  `@x` und **`@y`** liefern die *aktuelle Position*,
  `@entfernung` liefert die aktuelle *Entfernung vom Mittelpunkt*,
  `@max_x` und **`@max_y`** liefern / ändern die *Koordinatensystem-Größe*.

Für Fortgeschrittene (2)

- ***Befehlszeile:***
  Mit **`@1`** bis **`@9`** bekommt man die beim Starten des Programms *auf der Befehlszeile angegebenen Zahlenwerte* (der Reihe nach).
  Nicht angegebene Werte sind 0.

- ***Rechnungen:***
  - Er beachtet die *Vorrangregeln* und kennt **`( )`**, ein Vorzeichen- **`-`**, **`^`** als Rechenzeichen für *"hoch"* sowie **`wurzel(…)`**.
  - Es gibt **`|`** … **`|`** für den *Absolutbetrag*.
  - Es gibt **`sin(…)`**, **`cos(…)`** und **`tan(…)`**. Sie rechnen in *Grad*.
  - **`@pi`** liefert die *Konstante Pi*.
  - **`zufall(`***von*`,`** *bis***`)`** liefert eine *zufällige Kommazahl* zwischen *von* und *bis*.

- ***Vergleiche:*** **`> < >= <= = <>`** **`und oder nicht`**

Für Fortgeschrittene (3)

- Der *Befehl* **`schwarz`** löscht alle Striche im Fenster.
- Für Informatiker: Es gibt die üblichen *Befehle*
  - **`wenn`** *vergleich* **`dann`** *befehle* **`sonst`** *befehle* **`wenn_ende`**
    (den **`sonst`**-Teil kann man weglassen)
  - **`wiederhole`** *befehle* **`bis`** *vergleich*
- Der *Befehl* **`marke`** *speichert die aktuelle Position und Richtung*.
  Es können *mehrere* Marken gespeichert werden.

  `geh marke` kehrt *zur zuletzt gespeicherten Marke* zurück
  und *entfernt* diese, ebenso **`spring marke`**.

  Will man *mehrmals* zur *selben* Marke zurückkehren,
  muss man sie *nach jeder Rückkehr wieder frisch speichern*.

Für Fortgeschrittene (4)

- Für Informatiker:
  Es gibt *selbstgeschriebene Funktionen zum Rechnen:*

  rechnung *name* **(** *parameter* **)**
  befehle
  ergebnis *ergebnis_wert*
  rech_ende

  Die *befehle* können entfallen,
  wenn die gesamte Berechnung im *ergebnis_wert* steckt.

- Mit **@***name* kann man *eigene globale Variablen* erzeugen,
  d.h. Variablen, die vom *eigentlichen Programm* und
  allen Figuren und Funktionen ***gemeinsam*** verwendet werden.