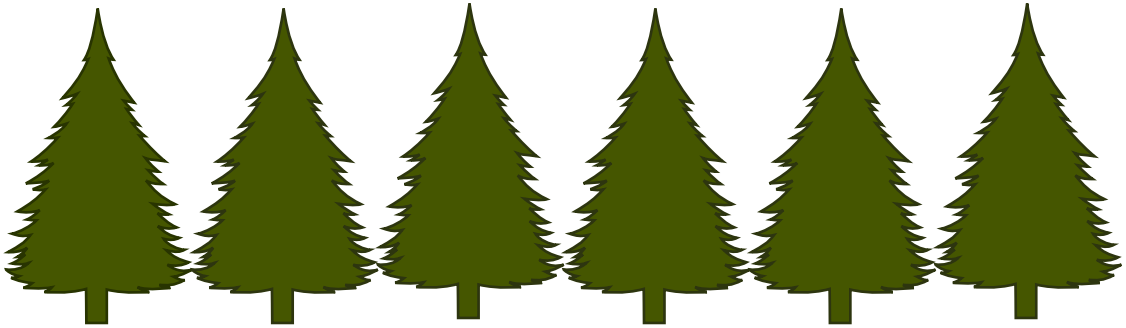
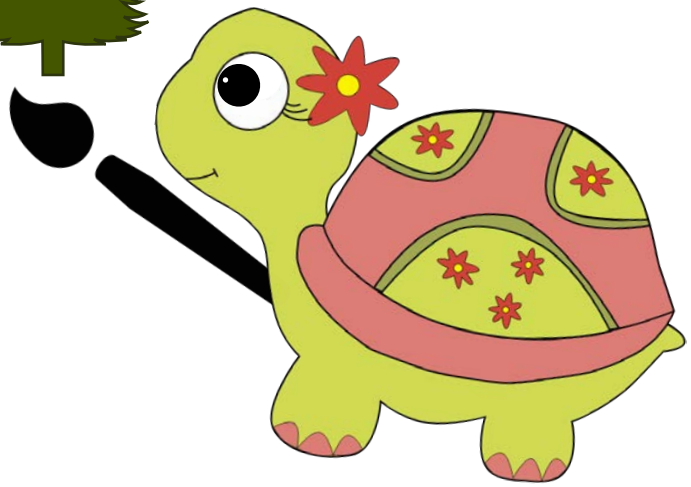
Funktionen ohne Parameter

Kira Grammel, Nina Ihde, Sebastian Serth & Selina Reinhard
Hasso-Plattner-Institut
Universität Potsdam

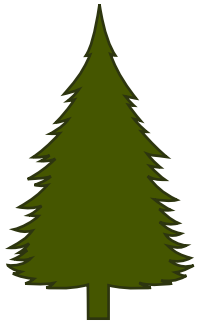
Funktionen

- Probleme zerlegen in Teilprobleme
- Beispiel:
  - Zeichne einen Wald aus mehreren Bäumen.





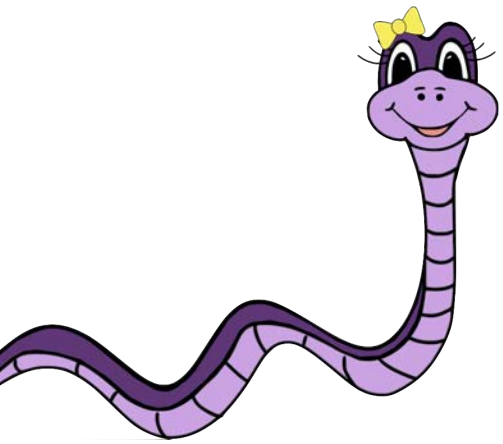
  - Teilprobleme: Zeichne einen Baum

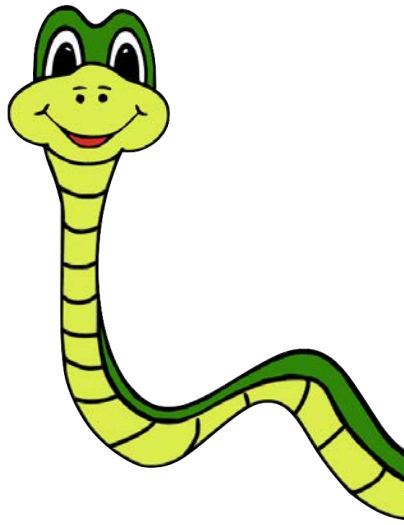


Funktionen

- Dienen als Strukturierungselement:
  Anweisungen werden zusammengefasst
- Funktionsname erklärt, was Unterprogramm macht

- Haben bereits vordefinierte Funktionen wie z. B. `print()` genutzt
- Nun werden wir eigene Funktionen schreiben





Funktionsdefinition

Schlüsselwort für Funktionen
Erstellen einer neuen Funktion

Funktionsname

```
1  def funktions_name():
2      # Anweisungen
```

Runde Klammern und den
Doppelpunkt nicht vergessen

Inhalt der Funktion

- Muss eingerückt sein! Nur der eingerückte Teil gehört zur Funktion
- Enthält die Anweisungen, die die Funktion ausführt

Einen Baum definieren

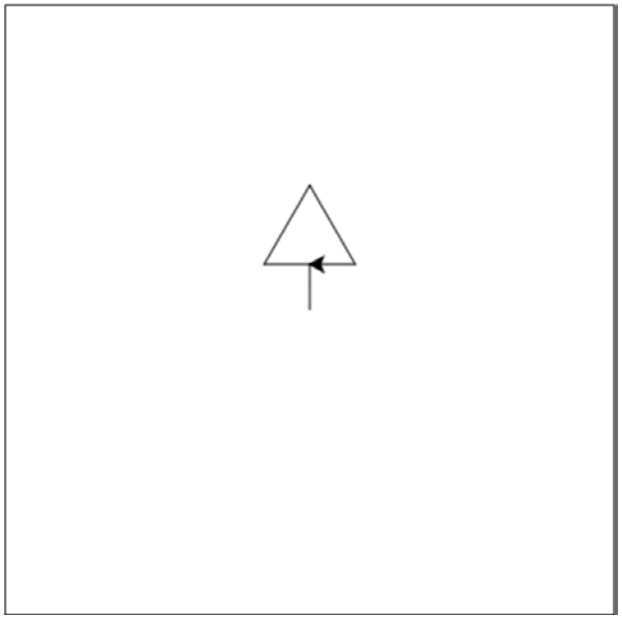
```
1  from turtle import *
2  def baum():
3      setheading(90)
4      forward(30)
5      left(90)
6      forward(30)
7      right(120)
8      forward(60)
9      right(120)
10     forward(60)
11     right(120)
12     forward(30)
```

Funktionsdefinition

- Reine Definition der Funktion
- Diese wird noch nicht aufgerufen und ausgeführt
- Daher gibt es keine Ausgabe

Einen Baum malen

```
1  from turtle import *
2  def baum():
3      setheading(90)
4      forward(30)
5      left(90)
6      forward(30)
7      right(120)
8      forward(60)
9      right(120)
10     forward(60)
11     right(120)
12     forward(30)
13 baum()
```



Funktionsaufruf

- `funktions_name()`
- Nun wird die Funktion aufgerufen und somit ausgeführt
- Verwenden der Funktion
- Schon viel verwendet z.B. `print()`

Zwei Bäume malen

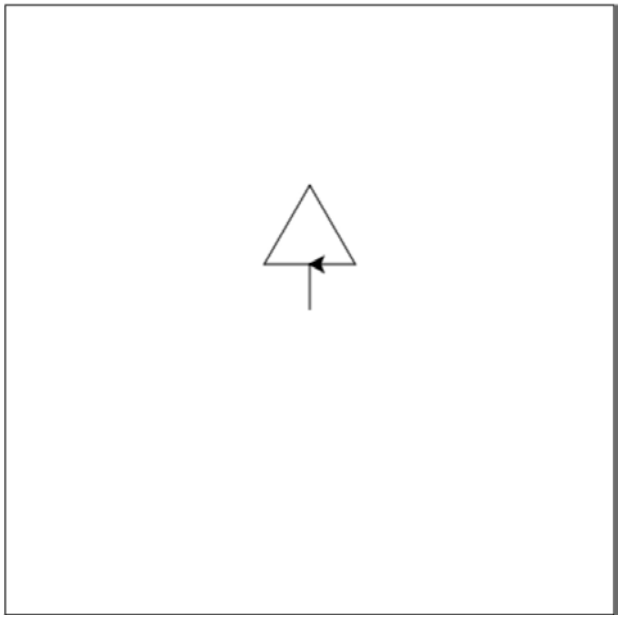
```
1  from turtle import *
2  def baum():
3      setheading(90)
4      forward(30)
5      left(90)
6      forward(30)
7      right(120)
8      forward(60)
9      right(120)
10     forward(60)
11     right(120)
12     forward(30)
13 baum()
14 baum()
```

Mehrfacher Funktionsaufruf

- Eine Funktion kann öfters aufgerufen werden
- Der nächste Baum wird einfach an die Stelle gemalt, wo der letzte Baum geendet hat

Funktionen mit Funktionsergebnis

```
 1 from turtle import *
 2
 3 aktuelle_position = position()
 4 print(aktuelle_position)
 5
 6 def baum():
 7     setheading(90)
 8     # mehr Aktionen ...
 9     forward(30)
10     return position()
11
12 aktuelle_position = baum()
13 print(aktuelle_position)
```



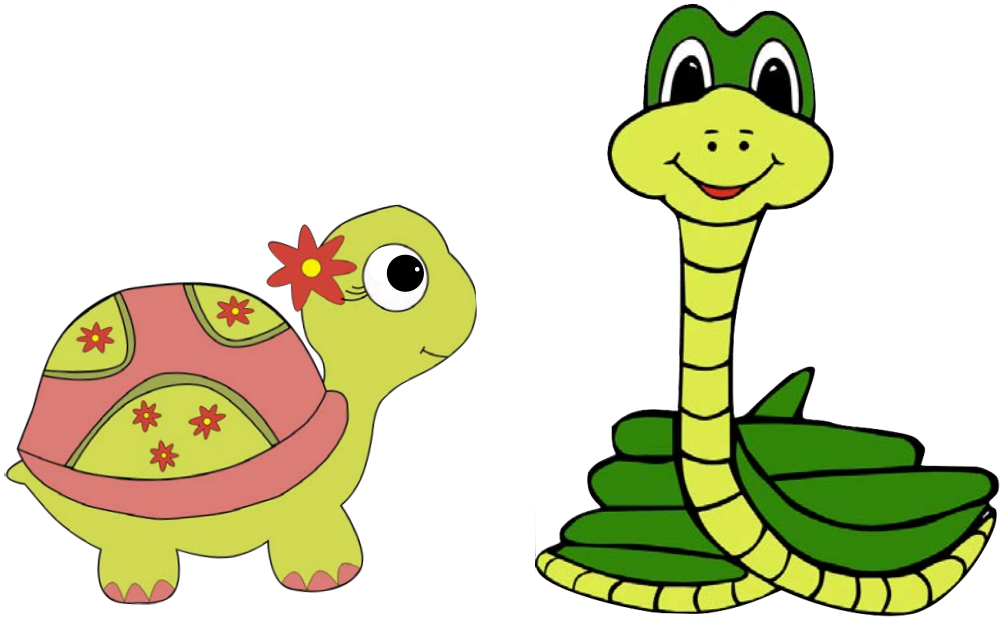
```
(0.00, 0.00)
(0.00,30.00)
```

return

- Heißt "zurückkommen"
- Legt Funktionsergebnis fest und gibt es zurück an die Stelle, wo die Funktion aufgerufen wurde

Funktionen

- Vorteile von Funktionen:
  - Keine Codedopplung
  - Wiederverwendbarkeit in anderen Programmen
  - Verständlicher und übersichtlicher
  - leichtere Pflege und Wartung des Codes



Zusammenfassung

- Funktionsdefinition
  - `def funktions_name():`
    `# Anweisungen`
- Funktionsaufruf:
  - `funktions_name()`
    (Funktion wird so ausgeführt, kann mehrfach aufgerufen werden)
- Mit `return` wird der Rückgabewert festgelegt
- Der Rückgabewert kann weiterverwendet werden