Unterabschnitt 4

Funktionen

Grundlegende Bemerkungen

- Die bisherigen Programme sollten “einzigartige” Probleme lösen und waren zum einmaligen Ausführen gedacht
- Ein wesentliches Konzept der Informatik ist dagegen, Probleme in leichter zu lösende Teilprobleme zu zerlegen
- Hierzu sollte aus Effizienzgründen auf schon bekannten Teillösungen aufgebaut werden können
- Einfaches “Rüberkopieren” von Code-Schnipseln führt unübersichtlichen Programmen (Wartbarkeit)
- **Funktionen** fassen einen Codeblock, der häufig wiederverwertet wird und ein spezifisches Problem löst, zusammen
- Beispiele: Sortieren, Ausgabe in einem bestimmten Format, einen Satz in Wörter zerteilen (Tokenisieren) ...
- Wesentlicher Vorteil: Andere können auf einmal geschriebene Funktionen zurückgreifen, ohne deren innere Abläufe zu kennen

Funktionen Aufrufen

- Funktionen tragen Namen (ähnlich wie Variablen)
- Funktionen werden durch ihren Namen gefolgt von runden Klammern aufgerufen: `print()`
- Um abstrakte Probleme lösen zu können, müssen Funktionen mit unterschiedlichen Eingaben versorgt werden können
- Diese Eingaben (**Argumente**) werden mit dem Funktionsaufruf übergeben

Funktion Aufrufen

```
1 print("Beispiel") # Print-Funktion wird mit
      einem Argument ausgerufen
2
3 # Fuer eine Funktion ist jeweils festegelegt,
      wie viele Argumente uebergeben werden duerfen
4 # Dies kann jedoch tw. flexibel sein
5
6 print("Ein", "Beispiel")
7 len("Beispiel")
8 len("Ein", "Beispiel") # FEHLER
```

Funktionen definieren

```
1 #Funktionsdefinition
2 def pretty_print(x):
3     print("~~~~" + x + "~~~~")
4
5 # Funktionsaufruf
6 pretty_print("Hallo") # ~~~~Hallo~~~~
```

- **Funktionskopf**:
  - Kontrollwort `def`
  - Funktionsname (frei wählbar, aber siehe Variablennamen)
  - **Parameter** (Platzhalter für Argumente) in Klammern gefolgt von Doppelpunkt
- **Funktionskörper**
  - Der eigentliche Code der ausgeführt wird
  - Eingerückt (per Konvention 4 Leerzeichen)
  - Abschließend eine Leerzeile

Rückgabewert von Funktionen

- Damit ein Programm mit dem “Ergebnis” eine Funktion weiterechnen kann, muss diese einen Wert zurückgeben
- Rückgaben werden im Funktionskörper mit dem Kontrollwort `return` markiert
- Der Funktionsaufruf endet sobald der erste `return`-Befehl erreicht ist
- Achtung: Unterschied von Rückgabe und Ausgabe!

```
1  def addiere_1(x):
2      x = x +1
3      return x   # Rueckgabe
4
5  y = addiere_1(5)
6  print(y)   # Ausgabe
```

Übung: Funktion zum Quadrieren einer Zahl

Schreiben Sie eine Funktion, die ein Zahl entgegennimmt, diese quadriert und das Ergebnis **zurückgibt**.

Übung: Funktion zum Quadrieren einer Zahl (Lösung)

```
1  def quadriere(x):
2      x = x ** 2
3      return x
4
5  y = quadriere(2)
6  print(y)
```

Übung: Funktion zum Addieren von Zahlen

Schreiben Sie eine Funktion, die zwei Zahlen entgegennimmt, diese addiert und das Ergebnis **zurückgibt**.

Übung: Funktion zum Addieren von Zahlen (Lösung)

```
1  def summe(x,y):
2      z = x + y
3      return z
4
5  # x = 2
6  # y = 5
7  # z = summe(x,y)
8  # print(z)
9
10 print(summe(2,5))
```

Übung: Naiver Tokenizer

Schreiben Sie eine Funktion, die einen String (z.B. einen Satz) entgegen nimmt und diesen in eine Liste von Wörter zerlegt. Diese Liste soll zurückgegeben werden. Die Funktion soll mindestens für die Eingaben ”Anne liebt ihren Kaffee heiß” und ”Anne liebt heißen Kaffee” funktionieren.

Übung: Naiver Tokenizer (Lösung)

```
1  def zerlegen(s):
2      w = s.split(" ")
3      return w
4
5  s1 = "Anne liebt ihren Kaffee heiss"
6  s2 = "Anne liebt heissen Kaffee"
7  print(zerlegen(s1))
8  print(zerlegen(s2))
```

Achtung: Computer denken nicht mit!

```
1  def foo():
2      print("foooooooo")
3      foo()
```

Achtung: Dynamisches Typensystem!

- Anders als in anderen Programmiersprachen wird der Datentyp der Argumente und der Rückgabe erst zur **Laufzeit** bestimmt
- Dies kann mitunter überraschende Effekte haben, da der Rückgabewert von der Verarbeitung innerhalb des Funktionskörpers abhängt

```
1  def add(x, y):
2      return x + y
3
4  print(add(3,4))           # 7
5  print(add("3", "4"))      # 34
6  print(add("3", 4))        # FEHLER
```

Variablenskopus

- Variablennamen haben jeweils einen bestimmten Gültigkeitsbereich (Variablenskopus)

```
 1  def pretty_print(x):
 2      b = "foo"
 3      print("~~~~" + x + "~~~~")
 4  pretty_print("bar")
 5  print(b) # Fehler: b nur innerhalb der Funktion,
        nicht global definiert
 6
 7  # Andersherum funktioniert es aber
 8  b = "foo"
 9  def pretty_print(x):
10      print(b)
11      print("~~~~" + x + "~~~~")
12  pretty_print("bar")
```