Einführung in das Programmieren

Schleifen in python

Christoph Draxler, Vanessa Reichel
draxler@phonetik.uni-muenchen.de
vanessa.reichel@campus.lmu.de

14. November 2024

Prolog: Würfeln mit dem Computer

Hier schreiben wir einen elektronischen Würfel:

```
import random

def wuerfeln ():
  return random.randint(1, 6)
```

`random.randint(min, max)` gibt eine Zufallszahl im Bereich von *min* bis einschließlich *max* als ganze Zahl zurück.
`import random` in der ersten Zeile importiert die python-Library `random`, die diese Zufallsfunktion (und weitere) zur Verfügung stellt.

Schleifen

Schleifen sind *Kontrollstukturen* in Programmiersprachen.

- sie wiederholen festgelegte Code-Abschnitte
- ihr *Kopf* enthält die Schleifenbedingung, der *Rumpf* den auszuführenden Code
- im Kurs betrachten wir `for`- und `while`-Schleifen.

Beide Schleifenarten sind gleich *mächtig*, d.h. sie können gleich viel – man kann sie grob in *zählend* und *bedingt* unterteilen.

`for`-Schleife: zählend

Eine `for`-Schleife sieht wie folgt aus

```
for VARIABLE in LIST:
    ANWEISUNGEN
```

VARIABLE eine einfache Variable

LIST eine beliebige Liste

ANWEISUNGEN Folge von python Befehlen

Umgangssprachlich formuliert: für jeden Wert der Variable in der Liste führe die Anweisungen aus.

Beispiel

Schreibe 10-mal 'Hello World' in die Konsole. Die Schleife ist *zählend*, von $1 \ldots 10$

```
for i in [0,1,2,3,4,5,6,7,8,9]:
    print(i, "Hello World.")
```

Die Liste hätte man auch kurz mit `range(0, 10, 1)` definieren können.
Welchen Wert hat die Variable `i` nach Beendigung der Schleife?

range (start, stop, step):

start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

Mehrfach Würfeln

Die folgende Funktion würfelt n-mal und schreibt die gewürfelten Werte auf den Bildschirm:

```
def wuerfleNMal (n):
  for index in range(n):
    print(f"{wuerfeln()} gewürfelt.")
```

Probieren Sie es aus und würfeln Sie 10 mal.
Was gibt diese Funktion zurück?

Schleifen und Fallunterscheidungen

Im Rumpf von Schleifen kann beliebiger Code stehen – sehr häufig sind dies Fallunterscheidungen.

```
def grosseWerte (liste, schwelle) :
    ergebnis = []
    for element in liste:
        if element > schwelle:
            ergebnis.append(element)

    return ergebnis
```

Was macht die obige Funktion? Was gibt der Aufruf von `grosseWerte([3,1,5,6,2,4], 3)` zurück?
Überlegen Sie, bevor Sie die Funktion aufrufen: Funktioniert sie auch für nicht-numerische Werte? Nein

`while`-Schleife: bedingt

Eine `while`-Schleife sieht wie folgt aus

```
while BEDINGUNG:
    ANWEISUNGEN
```

BEDINGUNG eine logische Bedingung

ANWEISUNGEN Folge von Python-Befehlen

Umgangssprachlich formuliert: solange die Bedingung gilt, führe die Anweisungen aus.

Beispiel

Die gleiche Aufgabe wie vorher: schreibe etwas n-mal auf den Bildschirm. Benutze nun aber eine *bedingte* Schleife:

```
i = 10

while i > 0:
    print(i, ": Hello World")
    i = i - 1
```

Welchen Wert hat `i` nach Ende der Schleife? `0`

Paschwürfeln I

Solange würfeln, bis ein Pasch kommt.

```
def paschwuerfeln ():
  pasch = False
  while pasch == False:
    w1 = wuerfeln()
    w2 = wuerfeln()
    if w1 == w2:
      pasch = True
    else:
      print(f"{w1} ist ungleich" {w2})

  print(f"{w1}, {w2} ist ein Pasch")
```

`True` und `False` sind Wahrheitswerte.

Paschwürfeln II

Solange würfeln, bis ein Pasch kommt.

```
def paschwuerfeln2():
  while True:
    w1 = wuerfeln()
    w2 = wuerfeln()
    if w1 == w2:
      print(f"{w1}, {w2} ist ein Pasch")
      return w1
    else:
      print(f"{w1}, {w2} ist ungleich")
```

Hört diese Funktion jemals auf? Wenn ja, warum?

While Schleif hören auf entweder ist while False, oder mit einem break or return statement

Beispiel

Würfle solange, bis Du 10-mal eine bestimmte Zahl, z.B. die 6, gewürfelt hast, und gib die Anzahl Würfe zurück. Die Schleife ist *bedingt* – wir wissen nicht, wie häufig sie durchlaufen werden muss.

```
def wuerfleZahlNMal (zahl, anzahl):
    wuerfe = 0
    anzahlpassend = 0
    while anzahlpassend < anzahl:
        geworfen = wuerfeln()
        if geworfen == zahl:
            anzahlpassend = anzahlpassend + 1
        wuerfe = wuerfe + 1

    return  wuerfe
```

Wie häufig musstest Du würfeln?

Schleifen-Funktionen

Funktionen mit Schleifen sind fast immer gleich aufgebaut:

1. Initialisierung der Ergebnisvariablen mit bekannten Werten
2. Schleifendurchlauf mit Modifikation der Ergebnisvariablen; meist wird in der Schleife eine Ergebnisvariable schrittweise aufgebaut.
3. Abschluss durch Rückgabe der Ergebnisvariable

Versuchen Sie, die drei Phasen in den obigen Folien zu markieren.

Muster für Schleifenfunktionen

```
def schleifenFunktion (argument, liste):
    # Initialisierung: Ergebnisvariable auf Ausgangswert setzen
    ergebnis = ...

    for element in liste:
        # Schleifendurchlauf:
        # argument und aktuelles Element aus der Liste vergleichen
        if (argument, element):
            ergebnis = ...
        else:
            ...

    # Abschluss: Ergebnis zurückgeben
    return (ergebnis)
```

Merken Sie sich dieses Muster – Sie werden es häufig benötigen!

Wann numerischen Index, wann nicht?

Prinzipiell gibt es zwei Arten, durch die Liste zu iterieren:

1. `for element in liste:` Hier bekommt `element` in jedem Durchlauf den Wert des nächsten Listen*elements*
2. `for index in range(len(liste)):` Hier läuft `index` von 0 bis zur Länge der Liste - 1, d. h. man adressiert ein Element über seine Position in der Liste

Wenn man eine Liste vollständig durchlaufen will und dabei jedes Element verwenden will, dann nimmt man meist die erste Variante. Wenn man entweder a) mit dem `index` eine Berechnung durchführt, oder b) in mehreren Listen parallel auf Elemente an bestimmten Positionen zugreifen möchte, dann die zweite Variante.

`element` und `index` im Vergleich

In beiden Schleifen wird die Liste einmal durchlaufen:

```
liste = ['a', 'b', 'c']

for element in liste:
  print(element)
```

```
a
b
c
```

```
liste = ['a', 'b', 'c']

for index in range(len(liste)):
  print(f"{index} {liste[index]}")
```

```
0 a
1 b
2 c
```

`element` und `index` sind *sprechende* Variablennamen – der Name soll andeuten, ob das Listenelement direkt verwendet wird oder es über seinen Index adressiert wird.

Beispiel für Index

```
woerter = ["Einst", "stritten", "sich", "Nordwind"]
aussprachen = ["aInst", "StrIt@n", "zIC", "nO6tvInt"]

for index in range(len(woerter)):
   print(f"{woerter[index]}: {aussprachen[index]}")
```

```
Einst: aInst
stritten: StrIt@n
sich: zIC
Nordwind: nO6tvInt
```

`index` wird sowohl für den Zugriff auf die Liste `woerter` als auch `aussprachen` verwendet.

Beispiel: Minimum-Funktion

Diese Minimum-Funktion gibt **den** *kleinsten* Wert in einer Liste zurück. Es ist nur eine Liste zu durchlaufen, d.h. der Zugriff erfolgt mit `(element in liste)`, also ohne nummerischen Index.

```
def minimum (liste):
    # Initialisierung: Hilfsvariable min auf erstes Element in der Liste
    min = liste[0]

    for element in liste:
        # Schleifendurchlauf: prüfen, ob min größer ist als das aktuelle
        if min > element:
            min = element
        # kein else-Fall notwendig

    # Abschluss: min zurückgeben
    return min
```

Beispiel: Minimum-Funktion

Diese Minimum-Funktion gibt **alle** *kleinsten* Werte in einer Liste zurück.

```
def minimum (liste):
   # Initialisierung: Hilfsvariable min auf erstes Element in der Liste setzen
   min = liste[0]
   # allemins sammelt alle Vorkommen des kleinsten Elements
   allemins = []

   for element in liste:
      # Schleifendurchlauf: prüfen, ob min größer ist als das aktuelle Element
      # wenn ja, dann wird element das neue Minimum, und es bildet den
      # ersten Eintrag im Ergebnis allemins
      if min > element:
         min = element
         allemins = [element]
      elif min == element:
         allemins.append(min)

   # Abschluss: allemins zurückgeben
   return allemins
```

Schleifen abbrechen

Häufig möchte man eine Schleife vorzeitig verlassen, z. B. wenn man das gesuchte Element schon gefunden hat. Dazu dient der python-Befehl `break`.

```
def suchen (wert, liste):
  for element in liste:
    if wert == element:
      break

  # Code fortsetzen
  ...
```

`break` sollte nur verwendet werden, wenn nach der Schleife noch etwas Relevantes berechnet wird.
Achtung: `break` kann leicht zu unübersichtlichem Code führen!

Aufgaben

- Welche Schleifenart verwenden Sie, um ein Element in einer Liste zu suchen? Begründen Sie Ihre Entscheidung.
- Wie suchen Sie nach einem Element in einem python-Dictionary?
- Wie prüfen Sie, ob die Würfelfunktion tatsächlich gerecht würfelt?

Code lesen

Jemand hat Ihnen ein paar Zeilen Wundercode geschenkt. Was macht dieser Code?

```
def bs(array):
  for i in range(len(array)):
    for j in range(0, len(array) - i - 1):
      if array[j] > array[j + 1]:
        temp = array[j]
        array[j] = array[j+1]
        array[j+1] = temp
  return array

liste = [22, 27, 26, 25, 21, 16, 21, 23, 27, 23, 23, 27, 26, 29]
print(bs(liste))
```

Formatieren Sie den Code so, dass er besser lesbar wird und fügen Sie an den passenden Stellen Kommentare ein, damit Sie auch in Zukunft verstehen, wie dieser Code funktioniert.