

# Programmieren in Python - Teil 1

Christian Dietrich

December 9, 2008

Schleifen

Funktionen

Sichtbarkeit von Variablen

# Die while-Schleife

```
1 i = 0
2 while i < 5:
3     print i
4     i = i + 1
```

- ▶ i wird auf 0 initialisiert
- ▶ while prüft, ob i kleiner 5 ist und führt dann den Block aus
- ▶ Im Block: i wird um eins erhöht
- ▶ Rücksprung vor das while, die Bedingung wird erneut geprüft
- ▶ Der Block wird solange ausgeführt, bis die Bedingung False zurückgibt

# while - Anwendungsbeispiel

```
1 tekla = ["Pouty", "Treu", "Hat einen Papagei"]
2 i = 0
3 while i < len(tekla):
4     print i, tekla[i]
5     i += 1
```

- ▶ tekla ist eine Liste
- ▶ i wird auf 0 initialisiert
- ▶ Der Schleifenkoerper wird solange durchlaufen, bis i = len(tekla) - 1  
Also fuer jedes Listenelement einmal

# Die for - Schleife

```
1 tekla = ["Pouty", "Treu", "Hat einen Papagei"]
2 for i in tekla:
3     print i
```

- ▶ tekla ist eine Liste
- ▶ Der Schleifenkoerper wird fuer jedes Element des Liste durchlaufen
- ▶ Bei jedem Durchlauf enthaelt i das aktuelle Listenelement

# for - Anwendungsbeispiel

```
1 print range(0, 23)
2 for i in range(0, 23):
3     print i, i * i
```

- ▶ range() gibt eine Liste von Zahlen zurueck
- ▶ Der Schleifenkoerper wird fuer jedes Element des Liste durchlaufen
- ▶ Es wird fuer jede Zahl der Liste das Quadrat ausgegeben

## for + while: Fibonacci

```
1 fib = [1, 1]
2 while fib[-1] < 100:
3     fib.append(fib[-1] + fib[-2])
4
5 for i in range(0, len(fib)):
6     print "Fibonacci Zahl" i, fib[i]
```

- ▶ fib wird als eine Liste initialisiert
- ▶ Solange das letzte Element von fib kleiner als 100 ist laeuft die Schleife
- ▶ Bei jedem Durchlauf wird die Summe des letzten und vorletzten Listenelements an die Liste angehaengt

## for + while: Fibonacci

```
1 fib = [1, 1]
2 while fib[-1] < 100:
3     fib.append(fib[-1] + fib[-2])
4
5 for i in range(0, len(fib)):
6     print "Fibonacci Zahl", i, fib[i]
```

- ▶ range() erstellt eine Liste, in der jedes Element ein Indice von fib ist
- ▶ Die Schleife wird also fuer jedes Element von fib durchlaufen
- ▶ i ist dabei immer der aktuelle Indice
- ▶ Die print Zeile gibt den Indice und die Fibonacci Zahl aus

# Funktionsdefinitionen

```
1 def fib(n):
2     print "Das Argument n ist", n
3     fib = [1, 1]
4     while len(fib) < n:
5         fib.append(fib[-1] + fib[-2])
6     print "Der Rueckgabewert ist", fib[n - 1]
7     return fib[n - 1]
8
9 print fib(10)
```

- ▶ def ist Beginn der Funktionsdefinition
- ▶ fib ist der Funktionsname
- ▶ (n) ist Liste der zu uebergebenen Argumente
- ▶ return springt aus der Funktion und gibt fib[n - 1] zurueck

# Globale und lokale Variablen

```
1 globalVar = "globaler Wert"
2
3 def dont_change(a):
4     print "dont_change:", a
5     globalVar = a
6     print "lokaler Wert von globalVar:", globalVar
7
8 print globalVar
9 dont_change("lokal")
10 print globalVar
```

- ▶ globalVar ist eine globale Variable
- ▶ in dont\_change wird globalVar ueberlagert
- ▶ die globale Variable globalVar bleibt unveraendert

# Globale und lokale Variablen

```
1 globalVar = "globaler Wert"
2
3 def change(a):
4     print "change:", a
5     global globalVar
6     globalVar = a
7     print "lokaler Wert von globalVar:", globalVar
8
9 print globalVar
10 change("lokal")
11 print globalVar
```

- ▶ globalVar ist eine globale Variable
- ▶ in change wird globalVar Sichtbar gemacht
- ▶ die globale Variable globalVar wird veraendert

# Raetsel

```
1 def fak(n):
2     i = 1
3     for x in range(1, n + 1):
4         i *= x
5     return i
6
7 def k_from_n(n,k):
8     return fak(n) / (fak(k) * fak(n-k))
9
10 def binominal(n):
11     a = []
12     i = 0
13     while i <= n:
14         a.append(k_from_n(n,i))
15         i += 1
16     return a
17
18 print binominal(3)
```

Erklaeren **wie** es funktioniert!