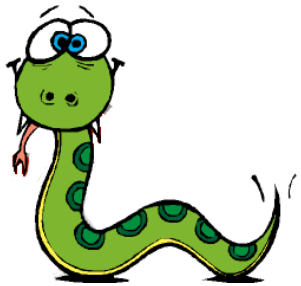
Freie Universität Berlin

Objektorientierte Programmierung

Python (Teil 2)



SoSe 2020

Prof. Dr. Margarita Esponda

Konzepte imperativer Programmierung

Die eigentliche Geschichte der Programmiersprachen begann mit:

- dem Konzept der **von Neumann-Maschine**, die **1945** die Notwendigkeit eines **gespeicherten Programms** postulierte
- und mit dem Konzept des *"conditional control transfer"*
  - ***If-then***-Anweisung
  - ***looped***-Anweisungen (***for***- und ***while***-Schleifen)
  - ***Subroutines*** (**Funktionen**)

Freie Universität Berlin

Fallunterscheidung

if-else-Anweisung

```
a = int(input( "Zahl = " ))

if a<0:
      print ("a ist negativ" )
else:
      print ( "a ist positiv" )
```

```
a = int(input( "Zahl = " ))

if a<0:
      print ( "a ist negativ" )
elif a==0:
      print ( "a ist gleich 0" )
else:
      print ( "a ist positiv" )
```

Einrücken anstatt Klammern

Freie Universität Berlin

if-else-Anweisung

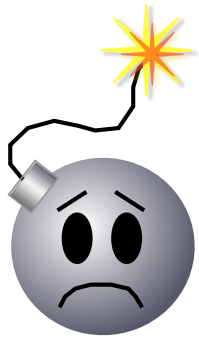
```
bit = True

if bit:
      print( "Hi!" )
else:
      print( "No!" )

bit = 0

if bit:
      print(True)
else:
      print(False)

bit = "Hi"

if bit:
      print(True)
else:
      print(False)
```

Hi!

False

True

Zahlen und andere Datentypen werden wie in **C** als Wahrheitswerte interpretiert



for-Schleifen

```
for x in ['spam', 'bla', 'eggs']:
        print (x)
```

```
>>>
spam
bla
eggs
```

```
for x in [1, 2, 3, 4]:
        print (x)
```

```
>>>
1
2
3
4
```

```
sum=0
for x in [1, 2, 3, 4, 5]:
        sum  = sum + x
print( sum)
```

```
>>>
15
```

```
for i in range(1, 5):
        sum += i
print( sum)
```

```
>>>
25
```

```
Dic = [(1,'a'), (2,'b'),(3,'c')]
for (x,y) in Dic:
        print (x, y)
```

```
>>>
1  a
2  b
3  c
```

Freie Universität Berlin

for-Schleife

```
for Ausdruck in Sequenz :
    Anweisungen
```

Bei der for-Schleife in Python wird nicht über eine Folge von Zahlen, sondern über Elemente einer Sequenz iteriert.

```
#  Erzeugt alle Kombinationen von zwei Zeichen aus text

text = input (  "text = " )

for  i in text:
     for  j in text:
          print ( i+j )
```

```
text = abc
aa
ab
ac
ba
bb
bc
ca
cb
cc
```

Logische Operatoren

Operator		Beschreibung
not	unär	logische Negation
or	binär	logisches Oder
and	binär	logisches Und

Bit-Operatoren

Operator		Beschreibung
~	unär	bitweise Inversion (Negation)
<<	binär	nach Links schieben
>>	binär	nach Rechts schieben
&	binär	bitweise UND
|	binär	bitweise ODER
^	binär	bitweise Exklusives Oder

Freie Universität Berlin

while-Anweisung

```
while Ausdruck :
        Anweisungen
```

```
#  Berechnet alle Quadratzahlen bis n

n = int(input( "n = " ))

zaehler = 0
while zaehler<=n:
        print  (zaehler*zaehler)
        zaehler = zaehler + 1
```

Imperatives Programmieren

Grundlegende Operation: die **Zuweisung**

- Speicherinhalte werden verändert und damit der Zustand der gesamten Maschine.

Kontrollfluss-Anweisungen

- bedingte Sprünge:

  if-then-else-Anweisung und **Loop**-Anweisungen (**for**- und **while**-Schleifen).

- unbedingte Sprünge:

  GOTO-, **break**-, **return**-Anweisung usw.

for- vs. while-Schleifen

```
summe = 0

for i in range(1, 100):
        summe += i  # summe = summe + i
print( summe )
```

```
summe = 0

i = 1
while i<100:
        summe += i   # summe = summe + i
        i += 1       # i = i + 1
print( summe )
```

```
n = int(input('n= '))

y = n+1
while (not isPrime(y)):
        y = y + 1
print('next prime > ', n, 'is: ', y)
```

?

Python Programm ohne Funktionen

```
n = int(input( 'Integer number: ' ))

result = 0
while result**3 < abs(n):
    result = result+1

if result**3 != abs(n):
    print(n,'is not a perfect cube ')
else:
    if n<0:
        result = -result
    print ('The cube root is', result)
```

```
n = int(input( 'Integer number: ' ))

for result in range(0, abs(n)+1):
    if result**3 == abs(n):
        break

if result**3 != abs(n):
    print(n,'is not a perfect cube')
else:
    if n<0:
        result = -result
    print ('The cube root is', result)
```

Freie Universität Berlin

Grundlegende Elemente von imperativen Programmen

- Definitionen von Datentypen
- Deklarationen von Variablen unter Verwendung vordefinierter Datentypen
- Zuweisungen
- Ausdrücke
- Anweisungen für den Kontrollfluss innerhalb des Programms
- **Gültigkeitsbereich von Variablen (*locality of reference*)**
- **Definition von Prozeduren, Subroutinen und Funktionen**
- **Parameter-Übergabe**

heute

Parameter-Übergabe in Python

call-by-value

Beim Aufruf eine Funktion wird in Python nur eine Kopie der Referenzen der jeweiligen Parameter-Objekte übergeben.

Innerhalb der Funktionen werden die Objekte mittels ihrer Referenzen für die Berechnungen verwendet.

Zuweisungen auf **nicht** veränderbare Variablen verursachen das Erzeugen von neuen Objekten.

Zuweisungen auf veränderbare Variablen haben Auswirkung auf die originalen Variablen des aufrufenden Programmteils.