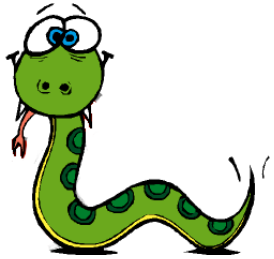
Freie Universität Berlin

Algorithmen und Programmieren II

Einführung in Python (Teil 3)



SS 2012

Prof. Dr. Margarita Esponda

Weiter mit imperativen Grundkonzepten

- Funktionen, Prozeduren und Subroutinen
- Parameter-Übergabe
- Gültigkeitsbereich von Variablen
- Lebenszeit von Variablen

Freie Universität Berlin

Funktionen

Funktionen sind das wichtigste Konzept in der Welt der höheren Programmiersprachen.

Funktionen sind ein grundlegendes Hilfsmittel, um Probleme in kleinere Teilaufgaben zerlegen zu können.

Sie ermöglichen damit eine bessere Strukturierung eines Programms sowie die Wiederverwertbarkeit des Programmcodes.

Gut strukturierte Programme bestehen typischerweise aus vielen kleinen, nicht aus wenigen großen Funktionen.

Freie Universität Berlin

Funktionen in Python

Eine Funktionsdefinition startet mit dem

def-Schlüsselwort

Funktionsname

Argumente

$$\pi = 4\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}$$

```
def  pi_leibniz ( k ):
    sum = 0
    for n in range( 0, k ):
        sum = sum +((-1)**n)/(2.0*n+1)
    return sum*4
```

Das Einrücken entscheidet, was zur Funktion gehört bzw. wann die Funktionsdefinition zu Ende ist.

Die return-Anweisung gibt das Ergebnis der Funktion zurück

Freie Universität Berlin

Python-Funktionen

etwas genauer:

```
def   Funktionsname ( Arg1, Arg2 , . . .  ) :
            Anweisung1
            Anweisung2
            . . .
            Anweisungn
```

Eine Anweisung der Form:

return *Ergebniswert*

befindet sich an beliebiger Stelle und beliebig oft in dem Funktionsrumpf; sie beendet die Ausführung der Funktion mit der Rückgabe eines Ergebniswertes.

Freie Universität Berlin

Funktionsdokumentation

Funktionen können einen Dokumentationstext beinhalten, der als Blockkommentar in der ersten Zeile der Funktion geschrieben werden muss.

Beispiel:

```
def  fact (n) :
      """ Berechnet die Fakultätsfunktion der Zahl n """
      if n==0:
                  return 1
      else:
                  return n*fact(n-1)
```

```
>>> print( fact(5) )
    120
>>> help (fact)
Help on function fact in module __main__:
fact(n)
      Berechnet die Fakultätsfunktion der Zahl n
```

Freie Universität Berlin

Funktionsargumente können sehr flexibel angegeben werden.

```
""" Argumente von Funktionen """

def fun( a=1,  b=3,  c=7 ):
        print ('a=', a, 'b=', b, 'c=', c)

fun( 30,  70 )
fun(20, c=100)
fun(c=50, a=100)
fun(20)
fun(c=30)
fun()
```

Ausgabe:

```
>>>
  a= 30  b= 70  c= 7
  a= 20  b= 3  c= 100
  a= 100  b= 3  c= 50
  a= 20  b= 3  c= 7
  a= 1  b= 3  c= 30
  a= 1  b= 3  c= 7
```

Die Reihenfolge der Argumente kann verändert werden.
Nur die Argumente, die benötigt werden, können angegeben werden.

Funktionen

Funktionen verdienen ihre Namen, wenn:

- diese keine Seiteneffekte beinhalten
- die **Eingabe** nur durch die **Argumente** erfolgt
- die **Ausgabe** nur mit Hilfe von **return**-Anweisungen stattfindet
- und zwischendurch keinerlei Ein-/Ausgabe verwendet wird.

Funktionen

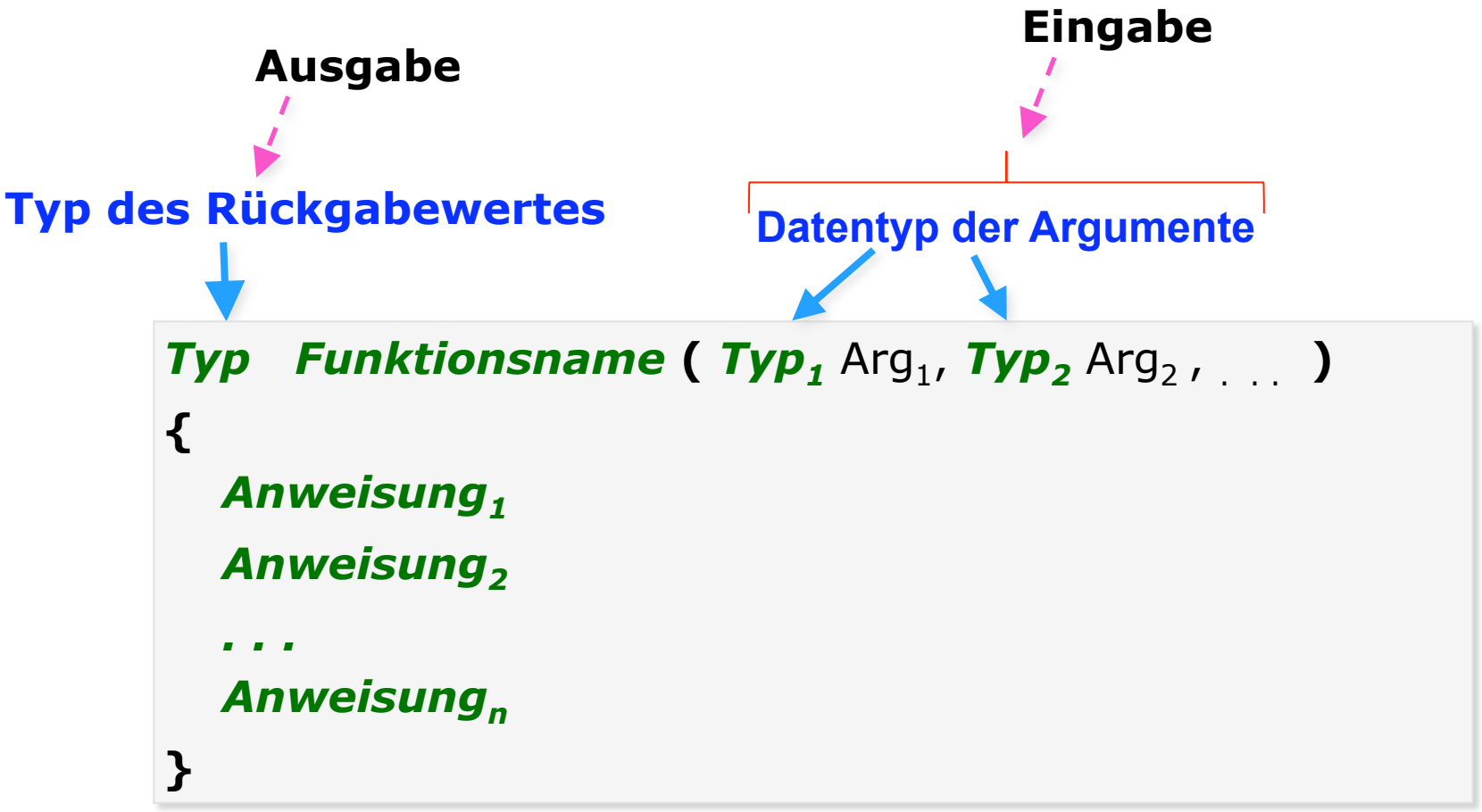
Funktionen sollen möglichst nur lokale Variablen benutzen.

Gut definierte Funktionen können innerhalb von Ausdrücken angewendet werden.

Sonst sollen sie Subroutinen, Prozeduren oder Methoden heißen.

Freie Universität Berlin

Funktionen in C




```
Typ   Funktionsname ( Typ1 Arg1, Typ2 Arg2 , . . .  )
{
   Anweisung1
   Anweisung2
   . . .
   Anweisungn
}
```

Freie Universität Berlin

Formale und aktuelle Parameter

Die formalen Parameter einer Funktionsdefinition sind **Platzhalter**.

Beim Aufruf der Funktion werden die formalen Parameter durch reale Variablen ersetzt, die den gleichen Typ wie die formalen Parameter haben müssen.

Freie Universität Berlin

Funktionen in Python

Funktionen

```
def quadrat( zahl ):
    return zahl*zahl

def teiler( a, b ):
    return a%b == 0
```

keine saubere Funktion

Anwendung innerhalb eines Ausdrucks

```
def test_funktionen():
    a = int( input('a='))
    b = int( input('b='))
    print( teiler(quadrat(a), quadrat(b)) )
```

Freie Universität Berlin

Funktionen

Der Funktionsbegriff wird innerhalb vieler Programmiersprachen sehr **unpräzise** verwendet.

In C, Python und vielen Programmiersprachen spricht man von Funktionen, obwohl sie oft keine Funktionen im mathematischen Sinn sind.

In einigen Programmiersprachen unterscheidet man zwischen Funktionen und Prozeduren (*Subroutines*) wie z.B. VB (VBA)

In Python muss weder der Datentyp des Rückgabewertes noch der Datentyp der Argumente deklariert werden.

Freie Universität Berlin

Geltungsbereich und Lebenszeit von Variablen

- Der Geltungsbereich einer Variablen ist der Bereich innerhalb des Programms, in dem diese sichtbar ist.

- Lebenszeit ist die Zeit, die eine Variable im Speicher existiert.

Freie Universität Berlin

Geltungsbereich von Variablen in Python

```
def foo(x,y):
        print(x,y)

def foo2(a,b):
        print(a,b,x,y)


x = 100
y = 200


foo(1,2)
foo2(1,2)
```

Ausgabe:

```
>>>
1 2
1 2 100 200
>>>
```

Freie Universität Berlin

Geltungsbereich und Variablen in Python

Beispiel:

```
x = 100
z = 7
def fun( a=1, b=3, c=7 ):
        x = 6
        print ('a= ', a, 'b= ', b, 'c= ', c)
        while ( x>3 ):
                y = 4
                print( y, x )
                print( z )
                x = x - 1
        print ( y, x )

print( x )
fun()
print( y )
```

Modulare Variablen

Lokale Variablen innerhalb der Funktionsdefinition

Geltungsbereich von Variablen in Python

Beispiel:

```
x = 100
z = 7
def fun( a=1, b=3, c=7 ):
        x = 6
        print('a=', a, 'b=', b, 'c=', c)
        while ( x>3 ):
                y = 4
                print( y, x )
                print( z )
                x = x - 1
        print ( y, x )

print( x )
fun()
print( y ) # Laufzeitfehler
```

Ausgabe:

```
>>>
100
a=1 b=3 c=7
4 6
7
4 5
7
4 4
7
4 3
Traceback (most recent call last):
  File "example.py", line 16, in
<module> print(y)
NameError: name 'y' is not
defined
```

Freie Universität Berlin

Geltungsbereich von Variablen in Python

global-Spezifizierer

```
def foo3():
        global x
        global y
        x = 0
        y = 0

x = 100
y = 200
print (x, y)
foo3()
print (x, y)
```

Ausgabe:

```
>>>
100 200
0 0
>>>
```

global-Spezifizierer

Beispiel:

```
def func(x, y):
    global g
    g = 3
    v = 6
    x, y = y, x
    print(g, v, x, y)

g, v, x, y = 10, 20, 30, 40

func(0,1)
print(g, v, x, y)

func(x, y)
print(g, v, x, y)
```

Mit dem global-Spezifizierer werden Bezeichner dem globalen Namensraum zugeordnet.

guter Programmierstil?

sinnvolle Anwendung?

Ausgabe: >>>

3 6 1 0

3 20 30 40

3 6 40 30

3 20 30 40

Freie Universität Berlin

Gültigkeitsbereich von Variablen in Python

Lokaler

Variablennamen sind innerhalb einer Methode oder Funktion definiert.

Modularer

Die Variablen sind innerhalb eines Moduls (Skriptdatei).

Eingebauter Geltungsbereich

Innerhalb der Python-Interpreter vordefinierte Namen, die immer gültig sind.

Freie Universität Berlin

Verschachtelte Funktionen

Funktionen können innerhalb anderer Funktionen definiert werden.

```
def percent (a, b, c):
    def pc(x): return (x*100.0) / (a+b+c)
    return (pc(a), pc(b), pc(c))

print (percent (2,4,4))
print (percent (1,1,1))
```

```
>>>
(20.0,  40.0,  40.0)
(33.3333333333,  33.3333333333,  33.3333333333)
>>>
```

Freie Universität Berlin

Parameter-Übergabe in Funktionen

call-by-value

- Ausdrücke werden zuerst ausgewertet und dann nur der Ergebniswert an die Funktionen übergeben.
- Einzelne Variablen werden kopiert und nur eine Kopie als Parameter weitergegeben.
- Der Inhalt der originalen Variablen des Aufrufers bleibt unverändert.

Freie Universität Berlin

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.

Freie Universität Berlin

Typsystem von Python

Python ist eine objektorientierte Programmiersprache im weiten Sinn.

In Python wird alles durch Objekte repräsentiert.
Jedes Objekt besitzt eine Identität.

Die Identität eines Objekts kann mit der Standardfunktion id() abgefragt werden.

Freie Universität Berlin

Wert- vs. Referenz-Semantik

- Wert-Semantik

  Ein Ausdruck wird ausgewertet und das Ergebnis direkt in eine Variablen-Adresse gespeichert.

- Referenz-Semantik

  Ein Ausdruck wird zu einem Objekt ausgewertet, dessen Speicheradresse in einer Variablen-Adresse gespeichert wird.

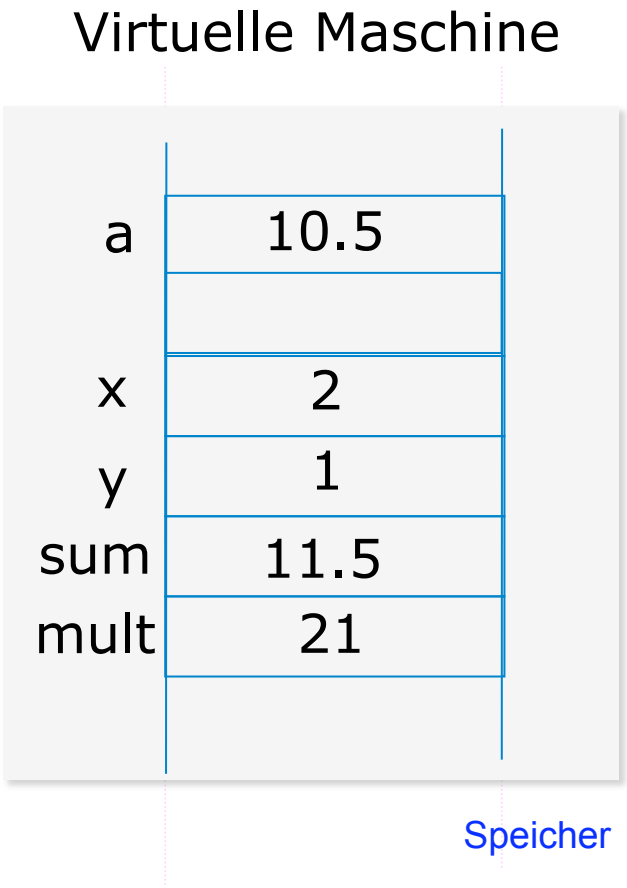
Python

Freie Universität Berlin

Dynamisches Typsystem von Python

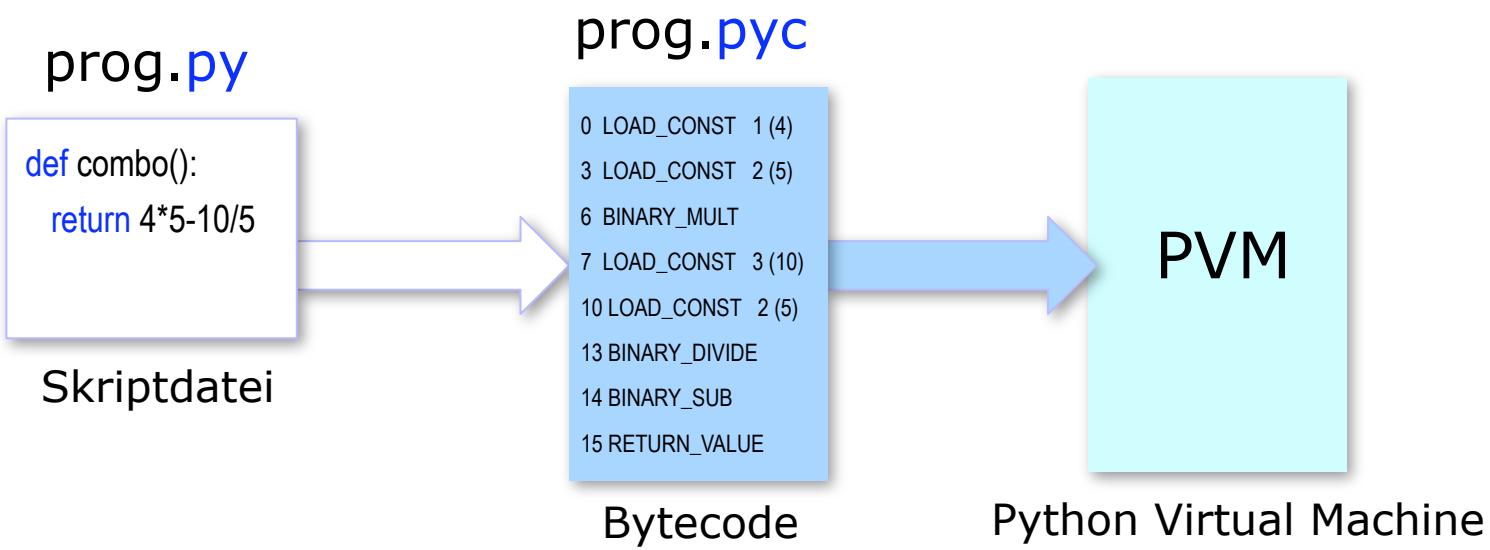
nur die halbe Wahrheit!

```
y = 1
x = 2
a = 10.5
sum = y+a
mult = a*x
```



Freie Universität Berlin

Python Virtual Machine



Cpython Standard aus www.python.org **PVM**

Jython Übersetzung auf Java-Bytecode **JVM**

IronPython für Microsoft .Net Framework **CLR**

Freie Universität Berlin

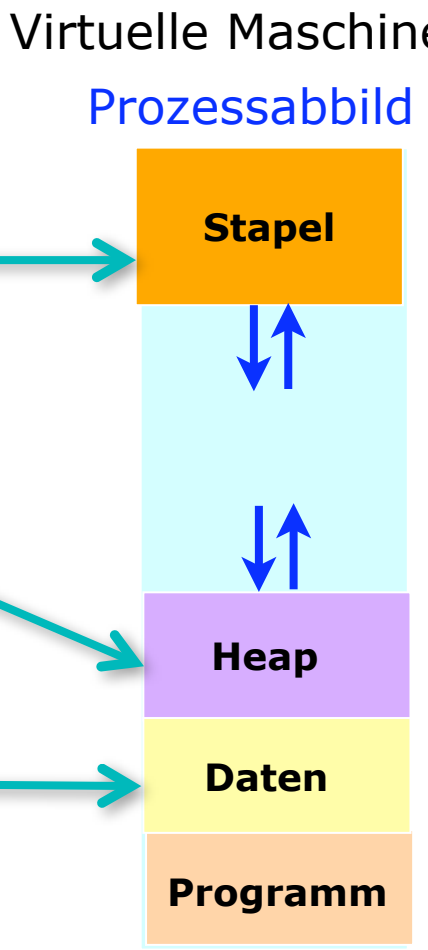
Programm in Ausführung (Prozess)

Ausführungsstapel

Parameter von Funktionen, *return*-Adressen und lokale Variablen werden hier gespeichert.

Daten, die während der Programmausführung dynamisch erzeugt werden, werden hier verlagert.

Statische Datenstrukturen, die am Anfang der Programmausführung erzeugt werden.

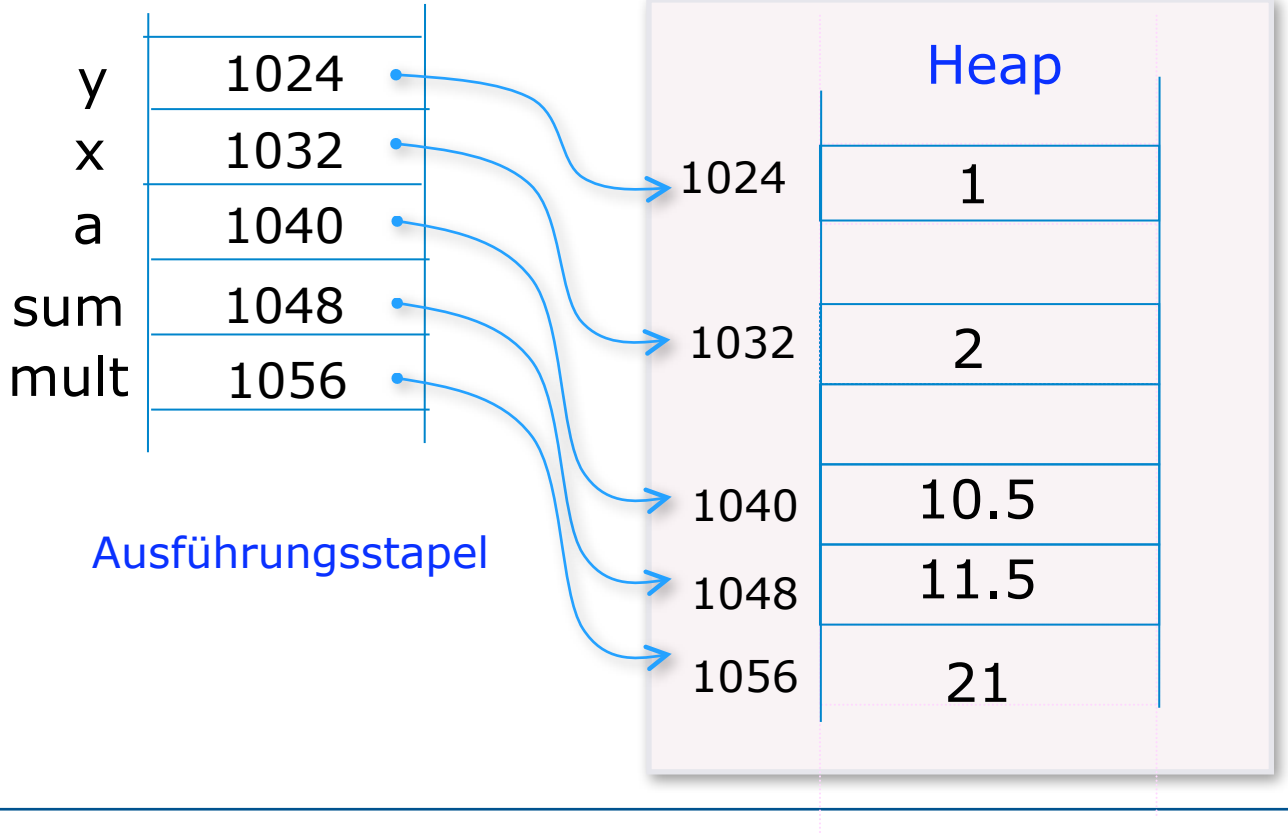


Freie Universität Berlin

Dynamisches Typsystem von Python

Python arbeitet nur mit Referenzen

```
y = 1
x = 2
a = 10.5
sum = y+a
mult = a*x
```



Freie Universität Berlin

Python arbeitet nur mit Referenzen

Änderbare (*mutable*)

- Listen
- Dictionary

Unveränderbar (*unmutable*)

- Integer
- Boolean
- Complex
- Float
- String
- Tupel

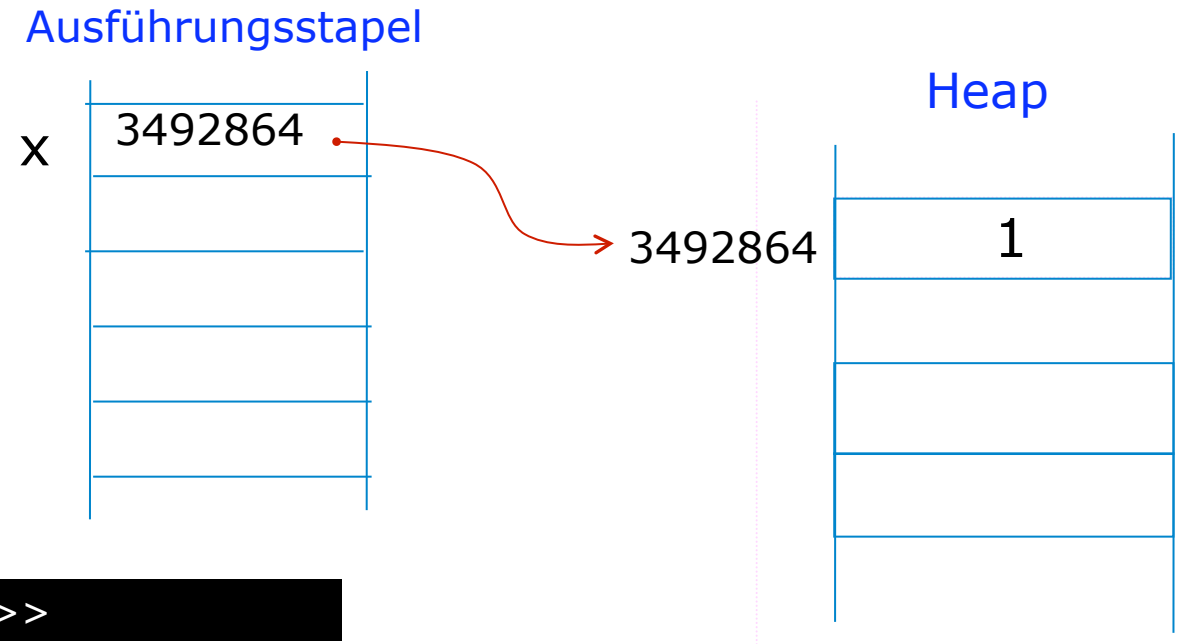
Freie Universität Berlin

Dynamisches Typsystem von Python

Python arbeitet nur mit Referenzen

```
x = 1
y = 1
z = x
print (id(x))
print (id(y))
print (id(z))

x = 3
print (id(x))
print (id(y))
print (id(z))
```



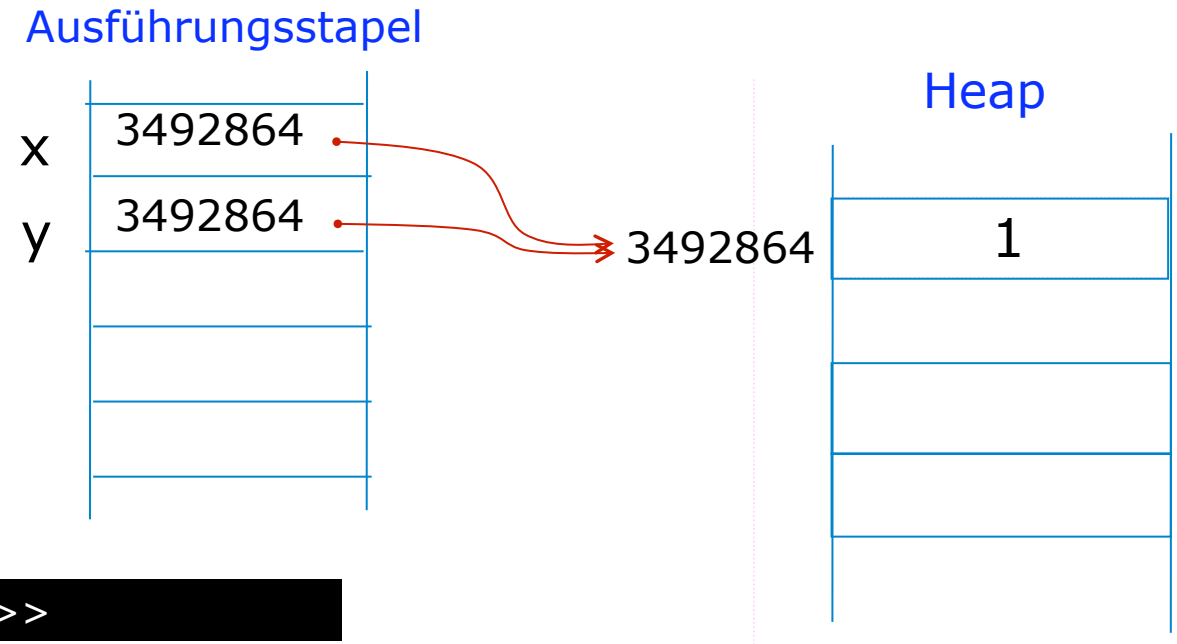
Freie Universität Berlin

Dynamisches Typsystem von Python

Python arbeitet nur mit Referenzen

```
x = 1
y = 1
z = x
print (id(x))
print (id(y))
print (id(z))

x = 3
print (id(x))
print (id(y))
print (id(z))
```



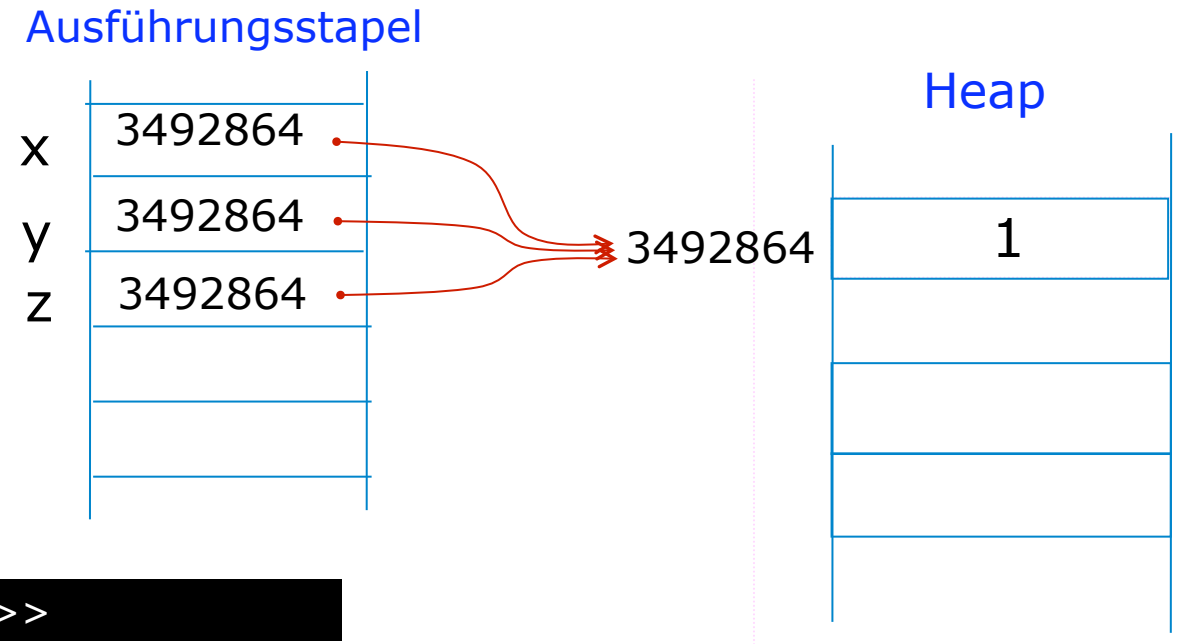
Freie Universität Berlin

Dynamisches Typsystem von Python

Python arbeitet nur mit Referenzen

```
x = 1
y = 1
z = x
print (id(x))
print (id(y))
print (id(z))

x = 3
print (id(x))
print (id(y))
print (id(z))
```



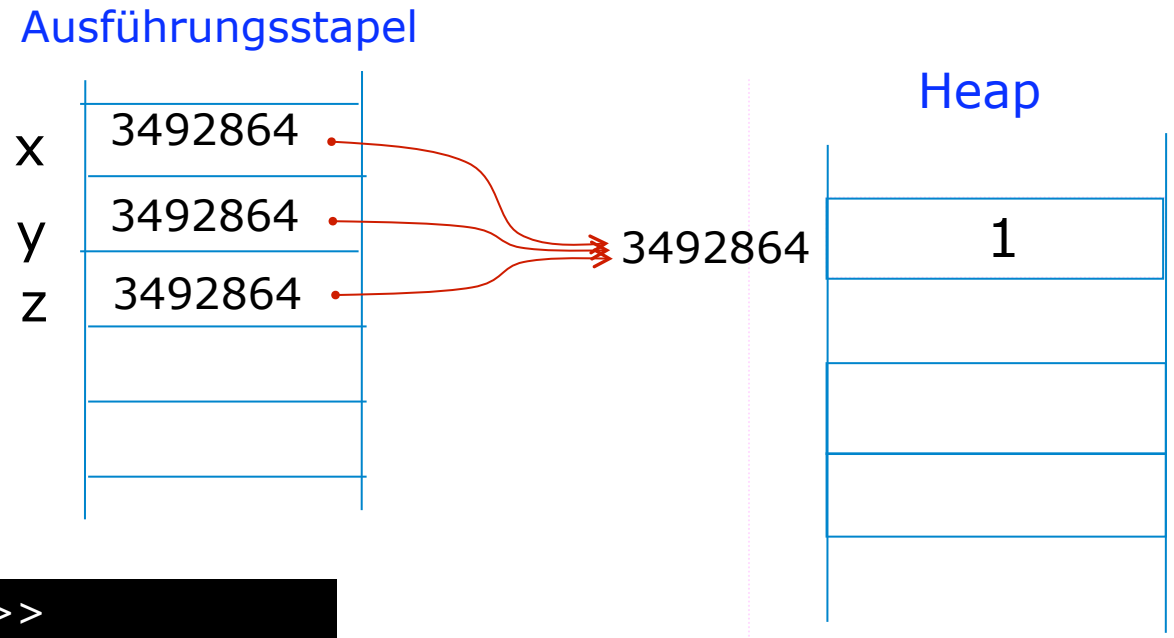
Freie Universität Berlin

Dynamisches Typsystem von Python

Python arbeitet nur mit Referenzen

```
x = 1
y = 1
z = x
print (id(x))
print (id(y))
print (id(z))

x = 3
print (id(x))
print (id(y))
print (id(z))
```




```
>>>
3492864
3492864
3492864
```

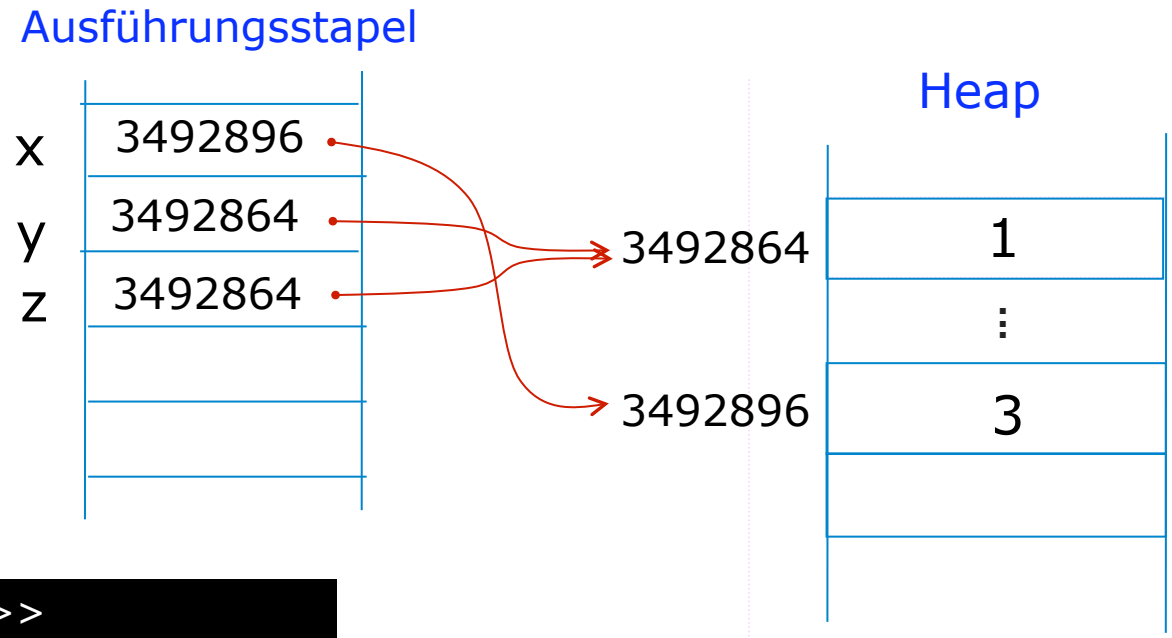
Freie Universität Berlin

Dynamisches Typsystem von Python

Python arbeitet nur mit Referenzen

```
x = 1
y = 1
z = x
print (id(x))
print (id(y))
print (id(z))
```

```
x = 3
print (id(x))
print (id(y))
print (id(z))
```




```
>>>
3492864
3492864
3492864
```

Freie Universität Berlin

Änderbare und unveränderbare Objekte

```
x = 1
y = 2
m = [[x,y],[x,y]]
print (m)
```
→ [[1, 2], [1, 2]]

```
x = 7
y = 10
print (m)
```
→ [[1, 2], [1, 2]]

```
m = [[x,y]*2]
print (m)
```
→ [[7, 10, 7, 10]]

```
r = [2,3,4,5,6]
print(r)
print(id(r))
```
→ [2, 3, 4, 5, 6]
→ 20047992

```
r.append(10)
print(r)
print(id(r))
```
→ [2, 3, 4, 5, 6, 10]
→ 20047992

Freie Universität Berlin

Variablen

Imperative Programmiersprachen

Variablen sind Speicherbehälter

Eigenschaften

- *Typ*
- *Namen*
- *Wert*

Im Programm sichtbar

- *Adresse*

In vielen imperativen Programmiersprachen (*Assembler*) noch sichtbar und direkt manipulierbar

In modernen Programmiersprachen nicht mehr.

Freie Universität Berlin

Zeiger- und Referenz-Datentypen

Zeigervariablen und Referenzvariablen in imperativen Programmiersprachen stellen Variablen, die Speicheradressen beinhalten, dar.

Zeiger- und Referenz-Variablen haben die Macht der Indirekten Adressierung.

Mit Hilfe von Zeiger- und Referenz-Datentypen können dynamische Datenstrukturen erzeugt werden.

Dynamische Datenstrukturen, die erst zur Laufzeit entstehen, befinden sich in dem heap-Bereich eines Prozesses (Programm in Ausführung).

Freie Universität Berlin

Zeigervariablen

Ähnlich wie bei der GOTO-Anweisung sind Zeigervariablen ein sehr diskutierter Datentyp in der Welt der imperativen Programmiersprachen.

Zeiger als expliziter Datentyp kommt vor allem in maschinennahen Programmiersprachen wie z. B. Assembler, C oder C++ vor. Zeiger dürfen hier auf beliebigen Speicherpositionen stehen und können mit Hilfe von arithmetischen Operationen manipuliert werden.

In Java und Python sind Referenz-Variablen intern vorhanden, aber für den Programmierer nicht explizit sichtbar.

In C++ und C# gibt es die Möglichkeit explizit mit Zeigern zu arbeiten oder nur implizit, wie in Java, mit Referenzen arbeiten.

Freie Universität Berlin

Parameter-Übergabe mit unveränderbaren Datentypen

```
def changeDouble( d = 1.3 ):
        d = 2
        print( d )
        print( id (d) )

a = 2.5
changeDouble( a )
print( a )
print( id ( a ) )
```

Ausgabe?

```
2
3492880

2.5
8466964
```

Freie Universität Berlin

Parameter-Übergabe mit unveränderbaren Datentypen

```
def changeDouble( d=1.3 ):
   # d = 2
    print( d )
    print( id(d) )

a = 2.5
changeDouble( a )
print( a )
print( id(a) )
```

Ausgabe?

```
2.5
8466964

2.5
8466964
```

Freie Universität Berlin

Parameter-Übergabe mit veränderbaren Datentypen

```
def  changeList ( list=[1, 2, 3, 4] ):
    list[0] = 100
    print( list )
    print( id(list) )

a = [5, 7, 8]
changeList(a)
print(a)
print( id(a) )
```

Ausgabe?

[100, 7, 8]
6488192

[100, 7, 8]
6488192

Freie Universität Berlin

del-Anweisung

Die Bindung eines Variablennamens zu einem Objekt wird aufgehoben.

Das Objekt bleibt noch im Speicher, bis keine Variable mehr auf dieses Objekt zeigt, und wird dann vom *Garbage Collector* beseitigt.

Beispiel:

```
a = 100
del a
print (a)
```

→ Laufzeitfehler!

Freie Universität Berlin

is-Anweisung

Der Ausdruck

a is b

liefert genau dann den Wahrheitswert True,
wenn a und b identisch sind.

Beispiel:

```
>>> a = [100, 200, 300]
>>> b = a
>>> a is b
True
```

```
>>> a = [100, 200, 300]
>>> b = [100, 200, 300]
>>> a is b
False
```

Freie Universität Berlin

pass-Anweisung

Die pass-Anweisung bewirkt nichts. Sie wird als Platzhalter bei Verzweigungen aus rein syntaktischen Gründen benötigt, um Einrückungsfehler während der Entwicklungsphase zu vermeiden.

Beispiel:

```
x = int (input("x="))
if x>0:
    pass
else:
    print( "x is negativ" )
```

from-Anweisung

Beispiel:

```
from math import sin, con
print(sin(0))
```

Freie Universität Berlin

exec-Anweisung

Die exec-Anweisung wird verwendet, um Python-Anweisungen auszuführen, die in einem String oder in einer Datei gespeichert sind.

Python-Skripte können zur Laufzeit erzeugt werden und mittels der exec-Anweisung ausgeführt werden.

Beispiel:

```
>>> exec( 'print("Hello")' )
Hello
>>> exec( 'print(2*3**2)' )
18
```

Freie Universität Berlin

Verwendung von "*Dictionaries*"

In Python gibt es keine switch-case-Anweisung wie in **C** und in **Java**, aber diese Anweisungen können mit Hilfe von *Dictionaries* simuliert werden.

```
def zero(): print( "You typed zero. " )
def sqr(): print( "n is a perfect square " )
def even(): print( "n is an even number " )
def prime(): print( "n is a prime number " )

options = {0 : zero,
           1 : sqr,
           4 : sqr,
           9 : sqr,
           2 : even,
           3 : prime,
           5 : prime,
           7 : prime,
          }
options[4]()
```

Freie Universität Berlin

break-Anweisung

Die break-Anweisung wird verwendet, um die Ausführung einer Schleife vorzeitig zu beenden.

```
while True:
        s = input( 'Text eingeben: ' )
        if s == 'end':
                break
        print ('Die Laenge des Texts ist', len(s))

print ('Tchüss.')
```

Freie Universität Berlin

continue-Anweisung

Die **continue**-Anweisung wird verwendet, um die restlichen Anweisungen der aktuellen Schleife zu überspringen und direkt mit dem nächsten Schleifen-Durchlauf fortzufahren.

```
while True:
        s = input('Text eingeben: ')
        if s == 'kein print':
                continue
        print ('Die Laenge des Texts ist', len(s))
```

Freie Universität Berlin

Neue Anweisungen in Python

Wort	kurze Erläuterung
from	Teil einer import-Anweisung
global	Verlegung einer Variablen in den globalen Namensraum
is	test auf Identität
pass	Platzhalter, führt nichts aus

exec	Ausführung von Programmcode

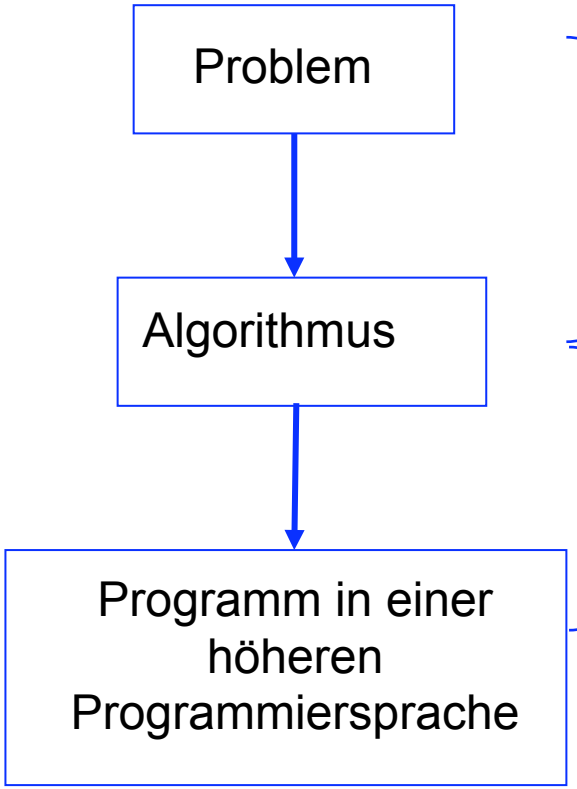
Freie Universität Berlin

Reservierte Wörter in Python

help> keywords

and	**else**	**import**	**raise**
assert	**except**	**in**	**return**
break	**exec**	**is**	**try**
class	**finally**	**lambda**	**while**
continue	**for**	**not**	**yield**
def	**from**	**or**	
del	**global**	**pass**	
elif	**if**	**print**	

Effiziente Lösung von Problemen



Wesentlicher Teil der Lösung eines Problems. Sehr kreative Phase

Programmierung oder Codierung des Algorithmus in einer bestimmten Programmiersprache. Einfache Phase.