

# Algorithmen und Programmieren II

## Einführung in Python (Teil 2)



SS 2012

Prof. Dr. Margarita Esponda

# Variablen

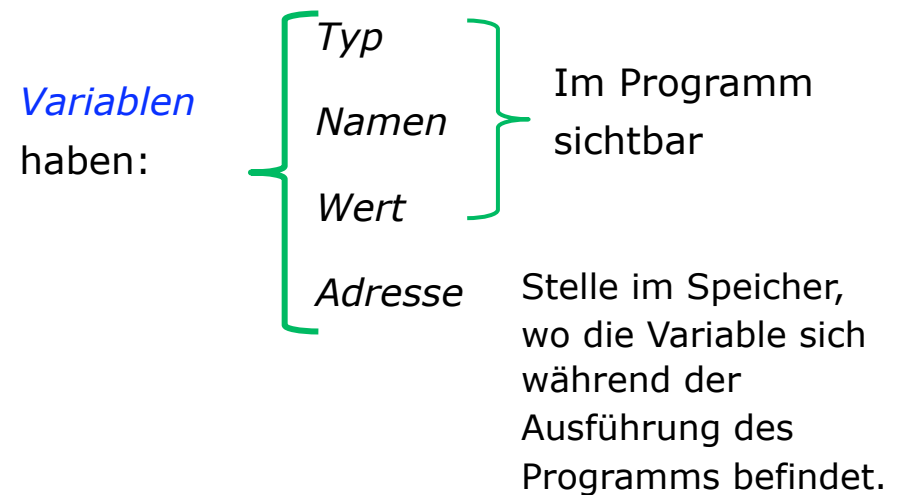
## Funktionale Programmiersprachen

Eine Variable stellt nur den symbolischen Namen von einem Wert oder einem Ausdruck, der zu einem Wert ausgewertet werden kann, dar.

**Der Wert einer Variablen kann nicht verändert werden.**

## Imperative Programmiersprachen

*Variablen* sind Stellen im Speicher, in denen Werte abgelegt werden können. Variablen speichern Zustände.



*Variablen* müssen normalerweise vor der erstmaligen Benutzung deklariert werden.



## Datentyp einer Variablen

- \* Python hat **dynamische Datentypen**, ist aber streng typisiert.
- \* Der Datentyp einer Variable wird erst zur Laufzeit festgelegt.
- \* Im Gegensatz zur statischen Typisierung, bei der der Datentyp einer Variable explizit deklariert werden muss, wird der Typ einer Variablen aus dem Typ des Werts **zur Laufzeit abgeleitet**.
- \* **Quelle einiger schwierig zu findender Fehler.**
- \* Streng typisiert.

## Datentyp einer Variablen

- In **C** wird damit die **minimale Speichergröße**, die eine Variable braucht, festgelegt.
- In modernen Programmiersprachen wird damit der **Wert-Bereich**, den eine Variable annehmen kann, beschränkt.
- **Python** hat ein dynamisches Typsystem

Der Datentyp von Variablen wird in Python erst zur Laufzeit festgelegt und **kann** während der Ausführung des Programms **verändert werden**.



## Variablendeklarationen in C und in Java

*Typ*      *Name*      *Wert*

```
int      x;  
int      y      = 10;  
double   area = 0;
```

← Semikolon

In Java sagt der Datentyp unter anderem, wie viel Speicherplatz benötigt wird, um die entsprechende Variable speichern zu können.

Symbolischer Name einer Speicheradresse

Der Wert 0 ist der Inhalt der Speicheradresse.

## Kommentare in Python

```
""" Blockkommentare:  
    können sich über mehrere Zeilen  
    erstrecken.  
"""  
  
# von hier aus bis Ende der Zeile wird dieser Text ignoriert  
a = 4 # Kommentar
```



## Eingabe und Ausgabe

```
input ( Prompt )  
...  
print ( Ausdrücke )
```

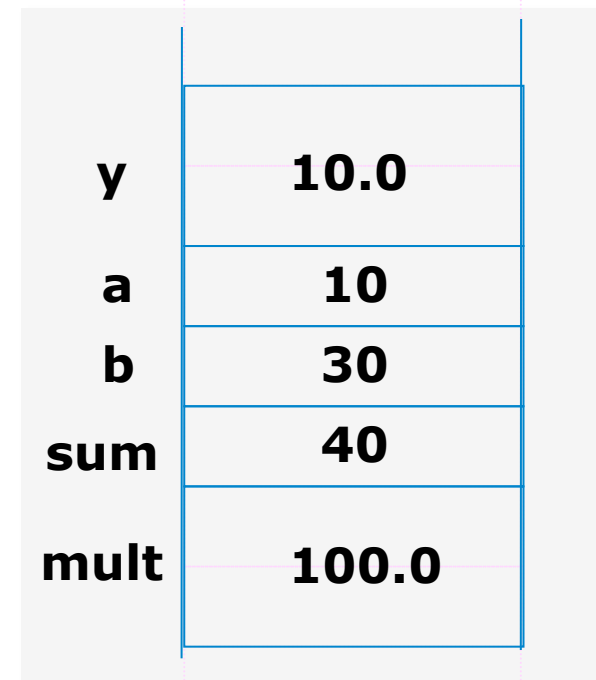
Die *input*-Funktion liest Zeichenketten aus der Standardeingabe ein und gibt diese als Rückgabewert der Funktion zurück.

Die *print*-Funktion kann Zeichenketten, Zahlenwerte oder zusammengesetzte Datenstrukturen ausgeben.

# Dynamisches Typsystem

Python Virtuelle Maschine **PVM**

```
y = 10.0  
a = 10  
b = 30  
sum = a+b  
mult = a*y
```



Speicher

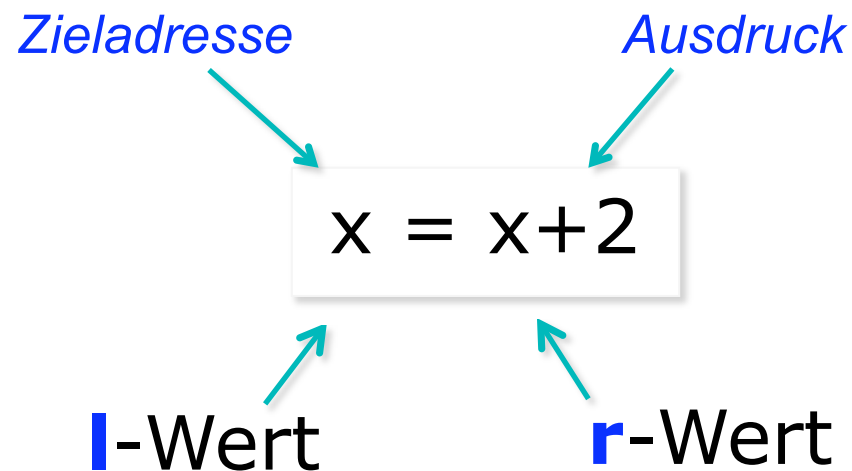
# Zuweisungen



Zuweisungen sind destruktive Anweisungen.

Die Adresse mit dem symbolischen Namen **A** wird mit dem Wert des Ausdrucks **B** überschrieben.

## Zuweisungen



Speicherbereich der Variable `x`

Aktueller Wert der Variable `x`

Zwei verschiedene semantische Bedeutungen des gleichen Symbols. ☹

## Multiple-Zuweisungen

### Python-Zucker ☺

```
a, b, c = 2, 3, 5  
print ( a, b, c )
```

```
>>>  
2 3 5
```

```
a = b = c = 2  
print ( a, b, c )
```

```
>>>  
2 2 2
```

```
x = [a, b] = [1, 2]  
print ( a, b, x )
```

```
>>>  
1 2 [1, 2]
```

# Datentypen in Python

Datentyp	Beispiel	Bemerkungen
Integer	<code>a = 3</code>	
Float	<code>x = 1.0</code>	
Boolean	<code>w = True</code>	
Complex	<code>c = 2 + 3j</code>	
String	<code>t = "Text" oder t = 'Text'</code>	nicht veränderbar
Liste	<code>l = [ 5, 3, 1, 6, 8 ]</code>	veränderbar
Tuple	<code>p = ( 35, 0, "Name")</code>	nicht veränderbar
Dictionary	<code>d = { 1:'a', 2:'b', 3:'c'}</code>	veränderbar

# Ausdrücke

Ein Ausdruck wird in der Informatik als eine Berechnungsvorschrift verstanden, die einen eindeutigen Wert darstellt.

Die einfachsten Ausdrücke in Python sind Konstanten beliebiger Datentypen

**Beispiel:** `1` `3.4` `"hello"` (**Literale**)

Ausdrücke haben keinen **|**-Wert und können nicht auf der linken Seite einer Anweisung stehen.

**Beispiele:**

**`a+b = c`** oder **`3 = c`**

# Arithmetische Operatoren

Operator		Beschreibung
<b>+</b>	unär	ändert nichts an der Zahl
<b>-</b>	unär	ändert das Vorzeichen der Zahl
<b>**</b>	binär	Power-Operator
<b>*</b>	binär	Multiplikation
<b>+</b>	binär	Addition
<b>-</b>	binär	Subtraktion
<b>/</b>	binär	Division
<b>%</b>	binär	Modulo (Restbildung)
<b>//</b>	binär	Division (immer ganzzahlig)



# Ausdrücke

Ausdrücke haben einen **Wert** und einen **Datentyp**

Beispiele:

	<b>Ausdruck</b>	<b>Wert</b>	<b>Type</b>
<b>type casting</b>	<code>int("12345")</code>	<b>12345</b>	<b>int</b>
	<code>int(0xff)</code>	<b>255</b>	<b>int</b>
	<code>int(3.14)</code>	<b>3</b>	<b>int</b>
	<code>(3*11//5)</code>	<b>6</b>	<b>int</b>
	<code>8%2</code>	<b>0</b>	<b>int</b>
	<code>8/3</code>	<b>2.66..</b>	<b>float</b>
	<code>float(3)</code>	<b>3.0</b>	<b>float</b>
	<code>2**(2+1)</code>	<b>8</b>	<b>int</b>
	<code>pow(2,3)</code>	<b>8</b>	<b>int</b>

## weitere Operationen ...

```
>>> help ( int )
Methods defined here:
|  __abs__(...)
|  x.__abs__() <==> abs(x)
|  __add__(...)
|  x.__add__(y) <==> x+y
|  __and__(...)
|  x.__and__(y) <==> x&y
|  __bool__(...)
|  x.__bool__() <==> x != 0
|  __ceil__(...)
|  Ceiling of an Integral returns itself.
|  __divmod__(...)
|  x.__divmod__(y) <==> divmod(x, y)
|  __eq__(...)
|  x.__eq__(y) <==> x==y
|  __float__(...)
|  x.__float__() <==> float(x)
|  __floor__(...)
|  ...
```

```
>>> import math
>>>> math.pi, math.e
(3.1415926535897931, 2.7182818284590451)
>>> help (math)
FUNCTIONS
  acos(...)
    acos(x)
      Return the arc cosine (measured in radians) of x.
  acosh(...)
    acosh(x)
  ...
  radians) of x.
  ...
```

## Import-Anweisung

Die `import`-Anweisung ermöglicht in Python-Skripten den Zugang auf vorprogrammierte Module

Einige interessante Module sind:

<b>math:</b>	exp, sin, sqrt, pow
<b>string:</b>	digits, whitespace
<b>sys:</b>	stdin, stderr, argv
<b>os:</b>	system, path
<b>re:</b>	split, match

## Höhere Datenstrukturen

Python unterstützt vier *sequentielle höhere Datentypen*

**Listen** (dynamic arrays)

**Tuples** (immutable lists)

**Dictionaries** (hash tables)

## Listen

Eine *Liste* ist eine **veränderbare** Sammlung von Objekten verschiedener Datentypen.

Beispiele:

```
[] # Leere Liste
```

```
liste = [1, 3.5, "Zeichenkette", 10]
```

```
a = [ 'Hi', 4, 0.234 ]
```

## Zeichenketten- Listen- und Tupel-Operatoren

<b>Operator</b>	<b>Funktion</b>	<b>Beispiel</b>	<b>Wert</b>
<b>+</b>	Verkettung	"Eins" + " Zwei"	"Eins Zwei"
<b>*</b>	Wiederholung	2 * "Eins"	"EinsEins"
<b>in</b>	enthalten in	1 in [1,2]	<b>TRUE</b>
<b>not in</b>	nicht enthalten in	3 not in [1,2]	<b>TRUE</b>
<b>x[i]</b>	Indizierung, liefert das Element aus x an der Position i	x = "hallo" x[1]	a
<b>x[i:j]</b>	liefert aus x die Elemente von i bis j-1 zurück	x = "hallo" x[0:3]	hal

## Höhere Datenstrukturen in Python

Beispiele:

```
>>> a = [1,2,3,4]
```

```
>>> a[2]
```

```
3
```

```
>>> a[2] = 10
```

```
>>> a
```

```
[1, 2, 10, 4]
```

```
>>> b = a
```

```
>>> b
```

```
[1, 2, 10, 4]
```

```
>>> a[1] = 2000
```

```
>>> b
```

```
[1, 2000, 10, 4]
```

```
>>> a = [7,6,5,4]
```

```
>>> a
```

```
[7, 6, 5, 4]
```

```
>>> b
```

```
[1, 2000, 10, 4]
```

Hier wird nur eine Speicheradresse kopiert.

Hier wird eine neue Liste erzeugt.

b zeigt aber auf die alte Liste.

Beispiele:

```
>>> a = [1,2,3,4]
```

```
[1, 2, 3, 4]
```

```
>>> b = a[:]
```

```
>>> b
```

```
[1, 2, 3, 4]
```

```
>>> a[1] = 2000
```

```
>>> b
```

```
[1, 2, 3, 4]
```

```
>>> a
```

```
[1, 2000, 3, 4]
```

```
>>> a[-1]
```

```
4
```

```
>>> a[-2]
```

```
3
```

Hier wird die Liste vollständig kopiert.



# Tupel

Ein *Tupel* ist eine Sammlung von Objekten verschiedener Datentypen zu einem Objekt.

Im Gegensatz zu einer Liste ist die Folge der Elemente dabei **nicht veränderbar**.

Beispiele:

<code>()</code>	<b># Leeres Tupel.</b>
<code>(1,)</code>	<b># Tupel mit einem Element.</b>
<code>(1,2,3)</code>	<b># Tupel mit drei Elementen.</b>
<code>(1, 'Eins')</code>	<b># Tupel mit zwei Elementen.</b>

`number = (1, 'eins')`

## Höhere Datenstrukturen in Python

### Beispiele:

```
>>> a = (1,2,3)
>>> b = a
>>> b
(1, 2, 3)
>>> a[2]
3
>>> len(a)
3
>>> a == b
True
>>> a[0] = 5
Traceback (most recent call last):
  File "<pyshell#76>", line 1, in <module>
    a[0] = 5
TypeError: 'tuple' object does not support
item assignment
```

```
>>> a = (5,6,7)
>>> b
(1, 2, 3)
>>> a+b
(5, 6, 7, 1, 2, 3)
>>> a = a+a
>>> a
(5, 6, 7, 5, 6, 7)
>>> b
(1, 2, 3)
>>> a*3
(5, 6, 7, 5, 6, 7, 5, 6, 7, 5, 6, 7, 5, 6, 7, 5, 6, 7)
```

## Höhere Datenstrukturen

**Dictionaries** *Dictionaries* sind eine Sammlung von Schlüssel- und Wertpaaren.

Ein Dictionary ist also eine Liste aus Schlüssen (*keys*), denen jeweils ein Wert (*value*) zugewiesen ist.

**Beispiele:**

`{}` **# Leeres Wörterbuch (*Dictionary*)**

`{ 1:'Goethe', 2:'Schiller', 3:5.67 }`

`atomic_num = {'None' : 0, 'H' : 1, 'He' : 2}`

# Höhere Datenstrukturen

## Dictionaries

### Beispiele:

```
>>> synonyms = {}
>>> synonyms['pretty'] = 'beautiful'
>>> synonyms['shy'] = 'timid'
>>> synonyms['easy'] = 'facile'
>>> synonyms
{'shy': 'timid', 'easy': 'facile', 'pretty': 'beautiful'}
>>> synonyms['easy']
'facile'
>>> 'pretty' in synonyms
>>> True
```

## Höhere Datenstrukturen Dictionaries

- Beliebige Datentypen können kombiniert werden.
- Sehr effizient implementiert mit Hilfe von Hashtabellen.

### Beispiele:

```
>>> dic = {}
>>> dic[1] = 'Hi'
>>> dic['a'] = 'Hallo'
>>> dic[3.1416] = 'pi'
>>> dic[(1,2,3,4)] = 'Reihe'
>>> dic
{'a': 'Hallo', 1: 'Hi', (1, 2, 3, 4): 'Reihe', 3.1416: 'pi'}
```

## 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 eine Variablen-Adresse gespeichert wird.

 Python

## Vergleichsoperatoren

alle binär

$<$	Kleiner
$>$	Größer
$<=$	Kleiner oder gleich
$>=$	Größer oder gleich
$==$	Gleichheit
$!=$	Ungleichheit

## 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 



# if-else-Anweisung

```
bit = True
```

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

```
bit = 0
```

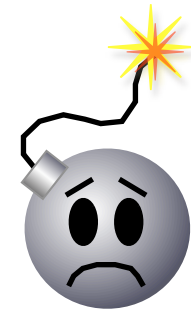
```
if bit:  
    print(True)  
else:  
    print(False)
```

```
bit = "Hi"
```

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



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



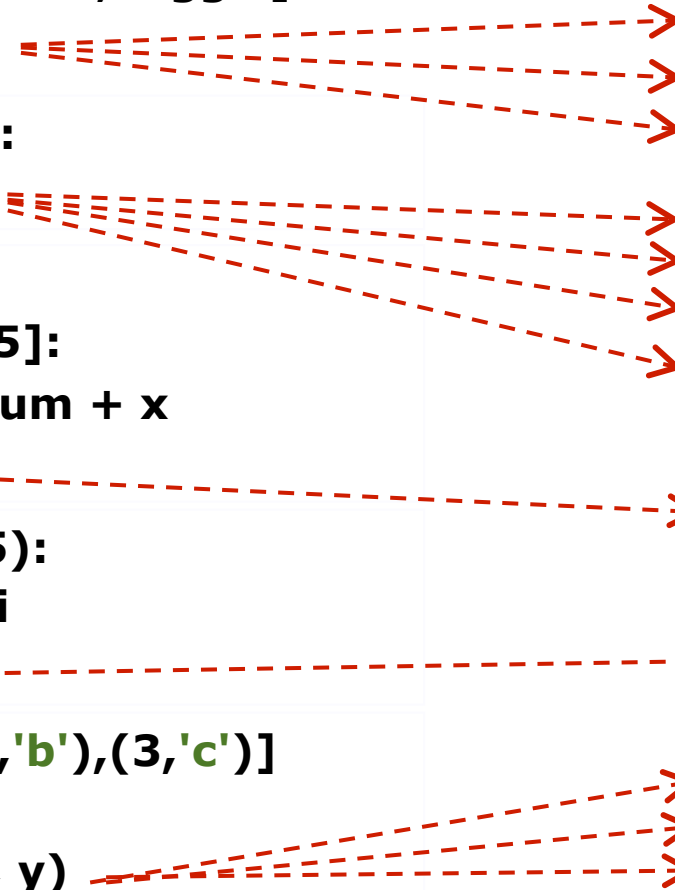
True

False

True

## for-Schleifen

```
for x in ['spam', 'bla', 'eggs']:  
    print (x)  
  
for x in [1,2,3,4]:  
    print (x)  
  
sum=0  
for x in [1,2,3,4,5]:  
    sum = sum + x  
print( sum)  
  
for i in range(1,5):  
    sum += i  
print( sum)  
  
Dic = [(1,'a'), (2,'b'),(3,'c')]  
for (x,y) in Dic:  
    print (x, y)
```



```
>>>  
spam  
bla  
eggs  
  
>>>  
1  
2  
3  
4  
  
>>>  
15  
  
>>>  
25  
  
>>>  
1 a  
2 b  
3 c
```

## 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

<b>Operator</b>		<b>Beschreibung</b>
<b>not</b>	unär	logische Negation
<b>or</b>	binär	logisches Oder
<b>and</b>	binär	logisches Und

## Bit-Operatoren

<b>Operator</b>		<b>Beschreibung</b>
<b>~</b>	unär	bitweise Inversion (Negation)
<b>&lt;&lt;</b>	binär	nach Links schieben
<b>&gt;&gt;</b>	binär	nach Rechts schieben
<b>&amp;</b>	binär	bitweise UND
<b> </b>	binär	bitweise ODER
<b>^</b>	binär	bitweise Exklusives Oder

## while-Anweisung

```
# Berechnet alle Quadratzahlen bis n  
  
n = int(input( "n = " ))  
  
zaehler = 0  
while zaehler<=n:  
    print (zaehler*zaehler)  
    zaehler = zaehler + 1
```

## while-Schleifen

```
import random

money = int ( input ( 'money? ' ) )

random.seed()

while money>0:
    if random.randint (0,1) == 0:
        money += 1
    else:
        money -= 1
    print(money)

print( 'You are a great loser! ' )
```

Der Zufallsgenerator wird mit einem internen Startwert initialisiert.