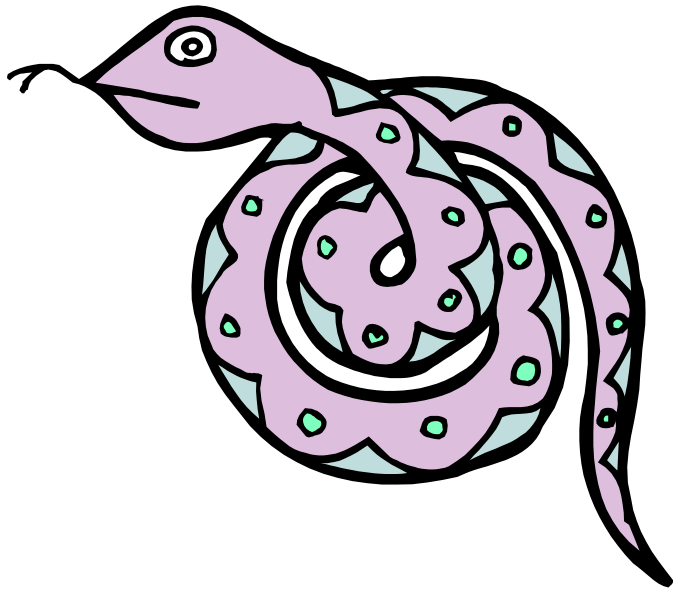
Grundlagen der Programmierung

VL 18: Skriptsprachen

Prof. Dr. Samuel Kounev
Jóakim von Kistowski
Norbert Schmitt

Inhalt

- Was ist eine Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen

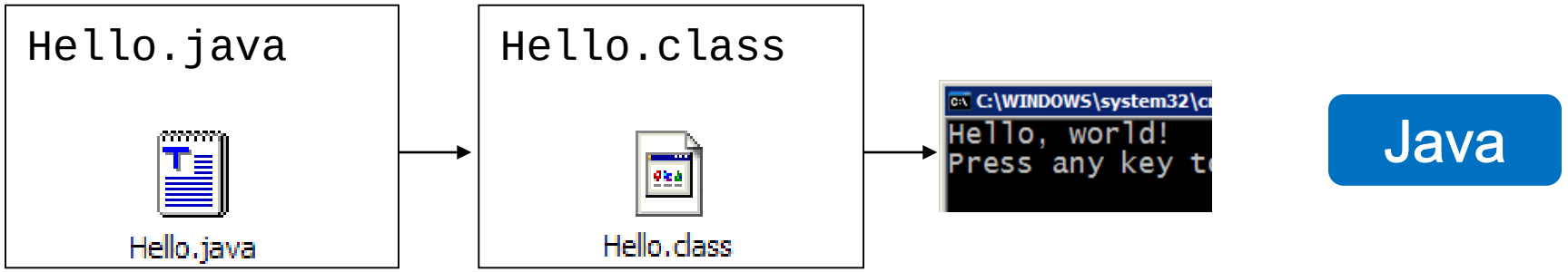


Was ist ein Skript?

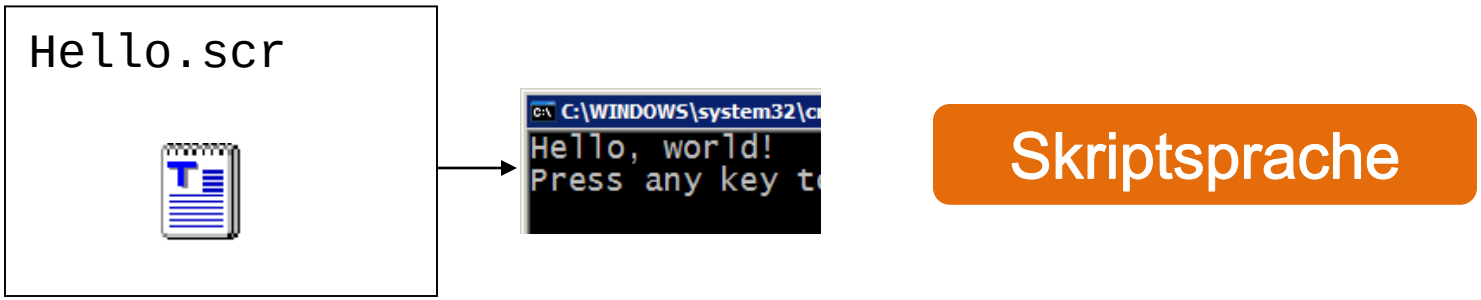
- Keine Kompilierung

Kompilieren

Ausführen



Interpretieren



Kompilieren vs. Interpretieren

Java

Kompilieren (javac Hello.java)

Hello.java

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello");
    thisDoesntMakeSense;
  }}
```

→

```
HelloWorld.java:4: error: not a statement
    thisDoesntMakeSense;
    ^
1 error
```

Skriptsprache

Interpretieren

Hello.scr

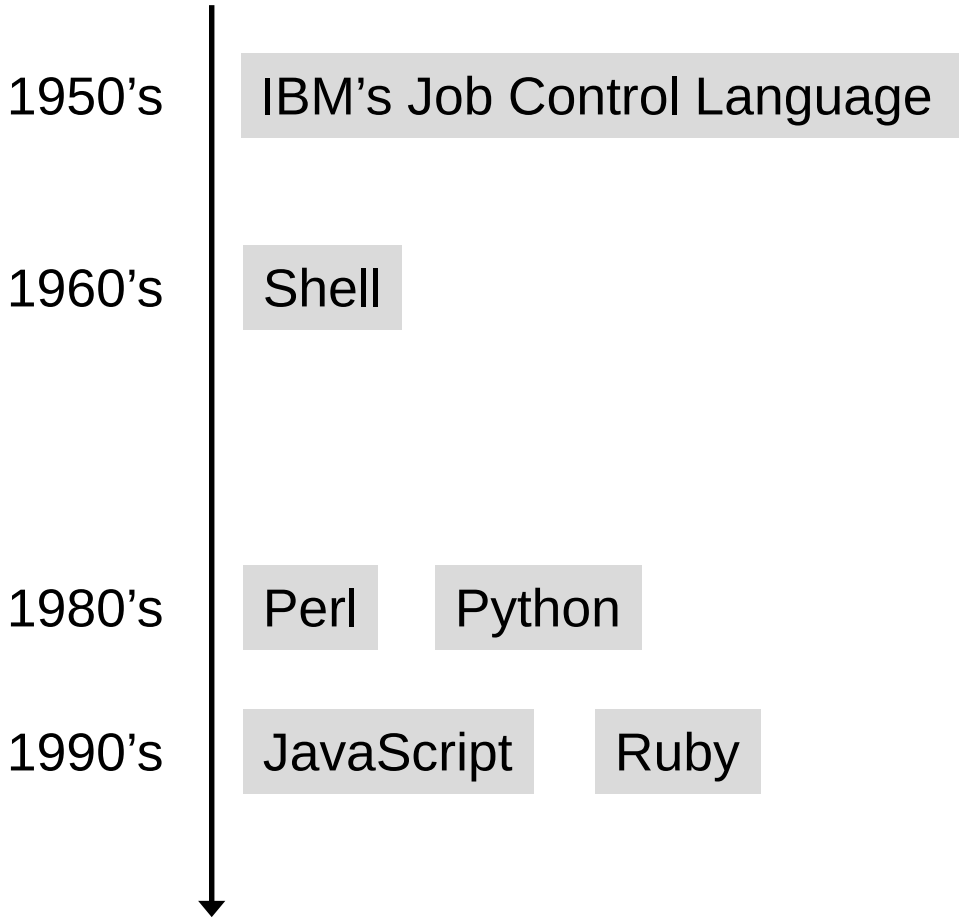
```
print "Hello"
thisDoesntMakeSense
```

→

```
Hello
Error: thisDoesntMakeSense
```

Skriptsprachen

- Beliebte Skriptsprachen
  - Shell
  - JavaScript
  - Perl
  - Ruby
  - **Python**



Was ist Python?

- Python ist eine Open-Source Skriptsprache
  - Benannt nach Monty Python
  - Download unter https://www.python.org/
  - Neueste „stable“ Version: 3.7.2

- Python ist
  - Objektorientiert
  - Modular
  - Plattformunabhängig
  - Effizient

Anwendungsszenarien

Natural language processing

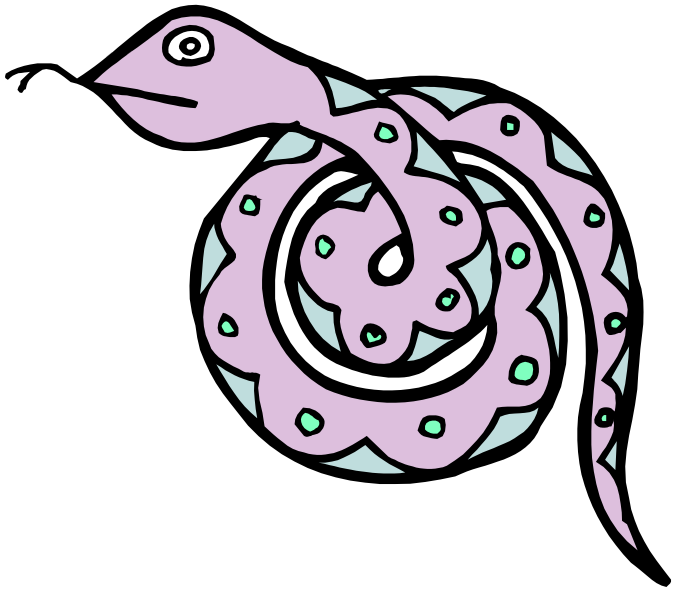
Data mining

Machine learning

Web programming

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Der Python-Interpreter

```
% python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 3+3
6
>>> Control + D
```

Ausgang

Eingabeauswertung

oder

Python 2

```
Summe.py
print 3+3
```

Ausgabe

Python 3

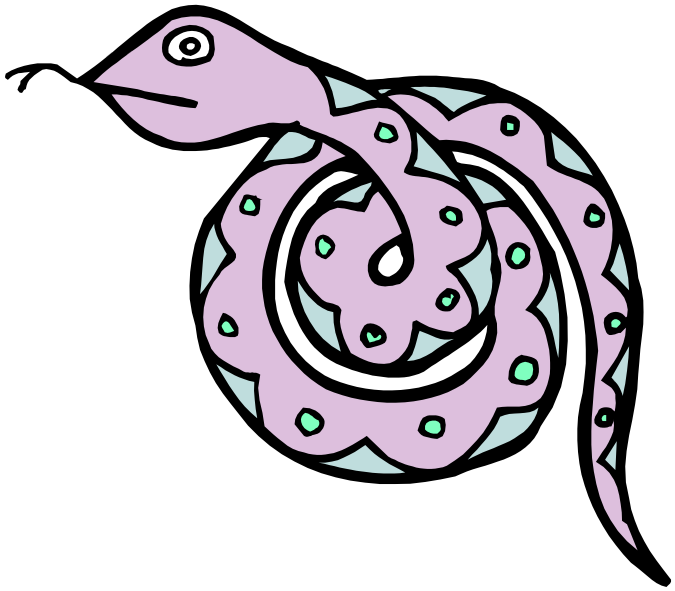
```
Summe.py
print(3+3)
```

Ausgabe

```
% python Summe.py
6
```

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Deklaration von Variablen

Java

testVar.java

```
public class testVar {
  public static void main(String[] args) {
    int testVar = 5;
    System.out.println(testVar);
  }}
```

→

```
% javac testVar.java
% java testVar
5
```

Python

testVar.py

```
testVar = 5
print (testVar)
```

→

```
% python testVar.py
5
```

In Python sind Variablen **nicht** an einen bestimmten Typ gebunden

Variablentypen und Typwechsel

Java		Python
int testVar = 5;	→	testVar = 5
String testVar = "test";	→	testVar = "test"
double testVar = 0.5;	→	testVar = 0.5

- Typwechsel

typwechsel.py

```
testVar = 5
print(testVar)
print(type(testVar))

testVar = 0.5
print(testVar)
print(type(testVar))

testVar = "test"
print(testVar)
print(type(testVar))
```

→

```
% python typwechsel.py
5
<class 'int'>
0.5
<class 'float‘>
test
<class 'str'>
```

Python ist **dynamisch Typisiert**

Mathematische Operatoren

```
X = 2
Y = 2

print (X * Y)
print (X / Y)
print (X % Y)
print (X + (X * Y) – X)
```

4

1.0

0

2 + 4 - 2 → 4

Python 2 Achtung! **Python 3**

Python 2		Python 3	
5 / 2	2	5 / 2	2.5
		5 // 2	2
5 / 2.5	2.0	5 / 2.5	2.0

Mathematische Operatoren(2)

- Python hat viele nützliche mathematische Befehle

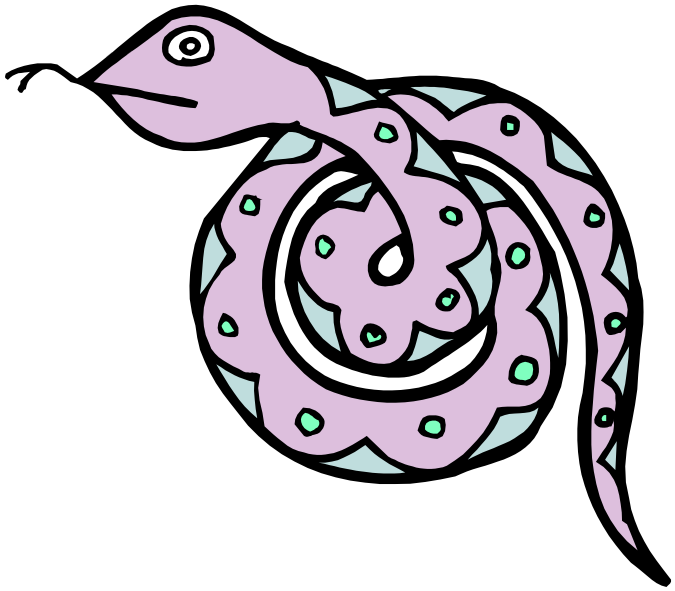
Befehl	Beschreibung
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1, value2***`)`	larger of two values
`min(`***value1, value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Konstante	Wert
`e`	`2.71828...`
`pi`	`3.14159...`

Um die mathematischen Befehle zu benutzen, wird eine Library benötigt (mehr später)

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Ausgabe mit *print*

- Der *print* Befehl
  - Syntax

Python 3

```
print ("Nachricht")
print (<Ausdruck>)
print (Ding1, Ding2, Ding3...)
```

```
print.py
print ("Hallo")

print (26 + 2)

gruess = "Hallo,"
jahre = 28

print (gruess, "ich bin", jahre, "Jahre alt“)
```

```
% python print.py
Hallo
28
Hallo, ich bin 28 Jahre alt
```

Eingabe mit *input*

Python 3

- Der *input* Befehl
  - Syntax

variable = input(**"Nachricht"**)

  - Der Rückgabewert ist immer vom Typ „str“ (String)

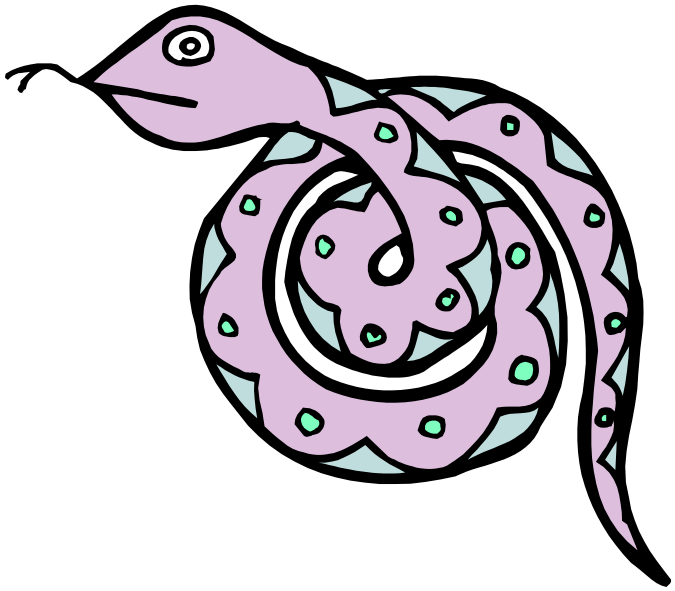
```
input.py
jahre = input("Wie alt bist du? ")
print ("Du bist", jahre, "Jahre alt. ")
```

```
% python input.py
Wie alt bist du? 28
Du bist 28 Jahre alt.
```

Die Variable *jahre* hat den Wert *'28'*

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Module und Importieren

- Python ist modular (genauso wie Java)
- Module werden mit *import* "importiert“
  - *import* erstellt eine Modulreferenz
  - eine Modulreferenz enthält Namen (Methoden, Variablen, …)

```
import.py

import math
nummer = input("Geben Sie eine Nummer ein: ")
print("Absolutwert: ", math.fabs(nummer))
```

Modulreferenz Name

```
% python import.py
Geben Sie eine Nummer ein: -4
Absolutwert:  4
```

- Ein Modul ist in einer Datei mit dem gleichen Namen plus die .py Endung definiert

Module und Importieren(2)

- Importieren ohne die Erstellung einer Modulreferenz

```
import.py
import math
nummer = input("Geben Sie eine Nummer ein: ")
print("log10: ", log10(nummer))
```

Die Modulreferenz fehlt

```
% python import.py
Geben Sie eine Nummer ein: 4
log10:
Traceback (most recent call last):
  File "import.py", line 5, in <module>
    print "log10: ", log10(nummer)
NameError: name 'log10' is not defined
```

Alle Namen, die das "math" Modul enthält, werden importiert

```
import.py
from math import *
nummer = input("Geben Sie eine Nummer ein: ")
print("log10: ", log10(nummer))
```

```
% python import.py
Geben Sie eine Nummer ein: 4
log10:  0.602059991328
```

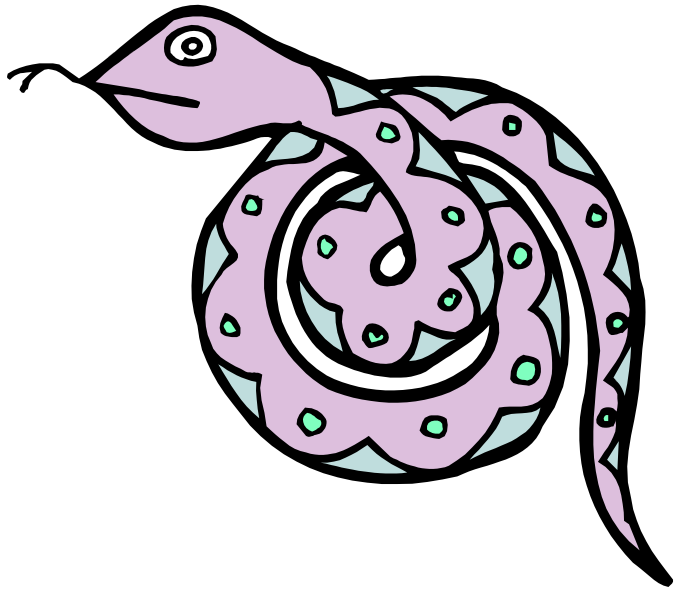
Oft verwendete Module

- *os:* OS-spezifische Funktionalitäten
- *os.path:* Verarbeitung von Verzeichnissen
- *math:* mathematische Befehle
- *random:* Erzeugung von zufälligen Werten

Mehr unter: https://docs.python.org/3/library/index.html
(für Python Version 3.7.2)

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Der *if* Befehl

- Syntax

if Bedingung**:**
· · · · Anweisung*1*
...
· · · · Anweisung*n*

Wenn man Anweisungen in eine Bedingung schreibt, **ist Einrückung notwendig**. Die Benutzung von Leerzeichen ist nicht notwendig, aber ist empfohlen wegen Konsistenz.

`if.py`

```
meter = 2.2
if meter > 2:
      print("Du bist sehr gross!")
```

→

```
% python if.py
Du bist sehr gross!
```

`if.py`

```
meter = 2.2
if meter > 2:
print("Du bist sehr gross!")
```

→

```
% python if.py
 File "if.py", line 4
   print "Du bist sehr gross!"
      ^
IndentationError: expected an
indented block
```

Die *if/else* Befehle

- Syntax

```
if Bedingung:
        Anweisung1
        ...
        Anweisungn
else:
        Anweisung1
        ...
        Anweisungn
```

`ifelse.py`

```
meter = 2.2
if meter > 2:
    print("Du bist sehr gross!")
else:
    print("Du bist 2 oder weniger Meter gross“)
```

→

```
% python ifelse.py
Du bist sehr gross!
```

if/elif/else genauso wie in Java

Operatoren

- Vergleichs-Operatoren

Operator	Bedeutung	Beispiel	Ergebnis
==	equals	1 + 1 == 2	wahr
!=	does not equal	3.2 != 2.5	wahr
<	less than	10 < 5	falsch
>	greater than	10 > 5	wahr
<=	less than or equal to	126 <= 100	falsch
>=	greater than or equal to	5.0 >= 5.0	wahr

- Logische Operatoren (kombiniert mit Vergleichs-Operatoren)

Operator	Beispiel	Ergebnis
and	9 != 6 and 2 < 3	wahr
or	2 == 3 or -1 < 5	wahr
not	not 7 > 0	falsch

Die *while* Schleife

- Syntax

while Bedingung**:**
· · · · Anweisung*1*
…
· · · · Anweisung*n*

Wenn man Anweisungen in eine Schleife schreibt, **ist Einrückung notwendig**. Die Benutzung von Leerzeichen ist nicht notwendig, aber ist empfohlen wegen Konsistenz.

`while.py`

```
nummer = 1
while nummer < 10:
    print(nummer)
    nummer = nummer * 2
```

→

```
% python while.py
1
2
4
8
```

Man kann *break* und *continue* genau so wie in Java nutzen

Die *for* Schleife

- Syntax

for Variable **in** Werte**:**
    Anweisung*1*
    ...
    Anweisung*n*

`for.py`

```
for nummer in (1,4,6):
    print(nummer)
```

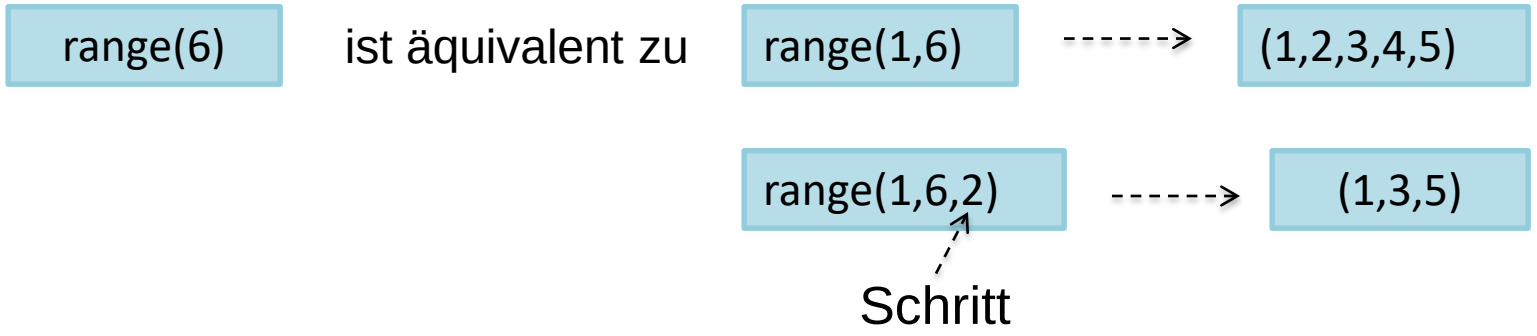
→

```
% python for.py
1
4
6
```

Das ist ein Tuple. Mehr über Tuples später.

Der *range* Befehl

- Der Range Befehl erzeugt eine Menge von Werten
- Man kann den *range* Befehl zusammen mit der *for* Schleife nutzen



`forrange.py`

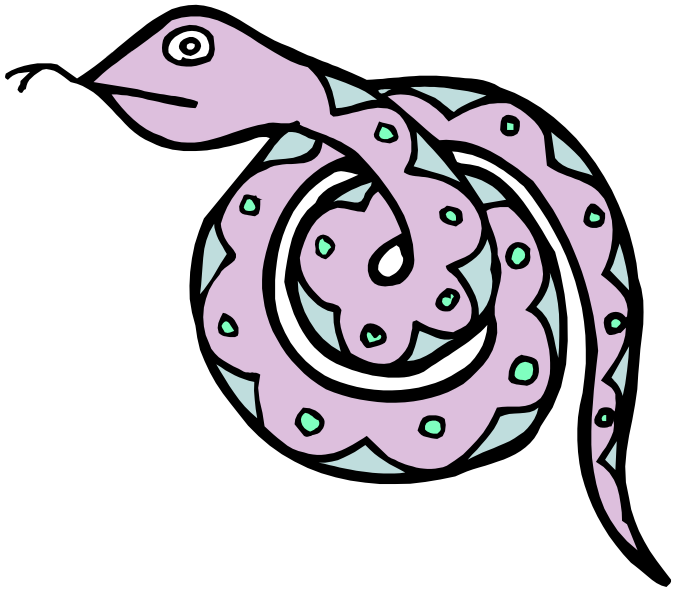
```
for nummer in range(1,6,2):
    print(nummer)
```

→

```
% python forrange.py
1
3
5
```

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Was sind Listen und Tuples?

- Listen und Tuples sind geordnete Folgen von Variabeln verschiedener Typen (untypisiert)
- Deklaration einer Liste

```
liste = [5, "abc", 23.8]
```

- Deklaration eines Tuples

```
tuple = (5, "abc", 23.8)
```

oder

```
tuple = 5, "abc", 23.8
```

- Zugriff auf Listen und Tuples

```
>>> liste = [5, "abc", 23.8]
>>> liste[0]
5
```

```
>>> tuple = (5, "abc", 23.8)
>>> tuple[2]
23.8
```

Listen und Tuples sind 0-indiziert

Listen vs. Tuples

- Listen und Tuples sind syntaktisch und funktional sehr ähnlich, aber trotzdem sind sie sehr unterschiedlich ...

- Listen sind **veränderbar**
  - Man **kann** Listen verändern

```
>>> liste = [5, "abc", 23.8]
>>> liste[0]=3
>>> print(liste)
[3, "abc", 23.8]
```

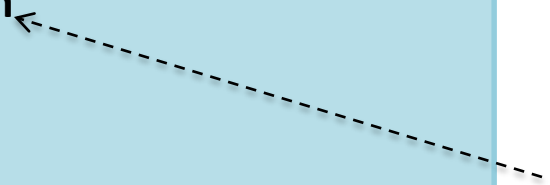
- Tuples sind **nicht veränderbar**
  - Man **kann** Tuples **nicht** ändern

```
>>> tuple = (5, "abc", 23.8)
>>> tuple[0]=3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
#Man kann Listen zu Tuples
#konvertieren (und umgekehrt)
>>> liste = [5, "abc", 23.8]
>>> tuple = tuple(liste)
>>> print(tuple)
(5, "abc", 23.8)
>>> liste = list(tuple)
>>> print(liste)
[5, "abc", 23.8]
```

Operationen auf Listen

```
>>> liste = [5, "abs", 23.8]
>>> liste.reverse() #kehrt eine Liste um
>>> print(liste)
[23.8, "abs", 5]
>>> liste.append(6) #fügt 6 hinzu
>>> print(liste)
[23.8, "abs", 5, 6]
>>> liste.insert(2, "test") #fügt 'test' auf index 2 hinzu
>>> print(liste)
[23.8, "abs", "test", 5, 6]
>>> liste.append([3,2]) #fügt die Liste [3,2] hinzu
>>> print(liste)
[23.8, "abs", "test", 5, 6, [3, 2]]
>>> liste.remove(5) #Entfernt das erste Vorkommen von 5
>>> print(liste)
[23.8, "abs", "test", 6, [3, 2]]
>>> liste.index(6) #Der Index vom ersten Vorkommen von 6
3
```



Kommentar

Tuples

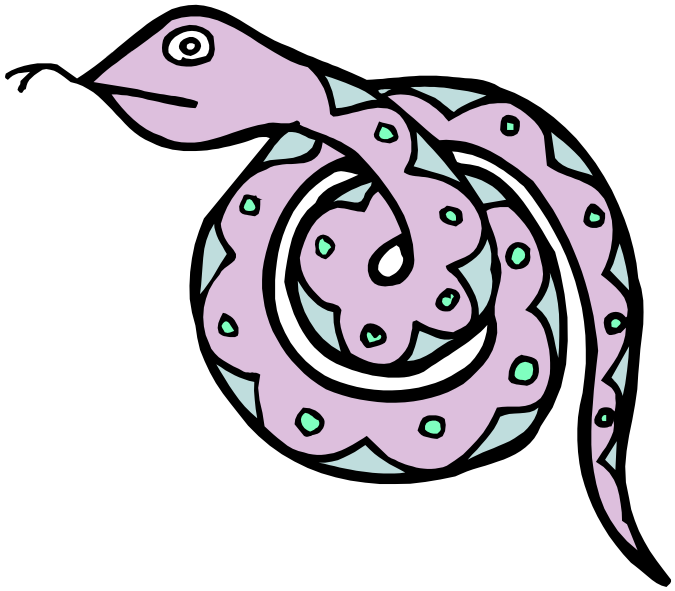
- Tuples können verwendet werden um mit mehreren Variablen auf einmal zu interagieren

```
>>> a = 1
>>> b, c = 2, "Hello"
>>> (b, c) = (2, "Hello")
>>> print(a)
1
>>> print(b)
2
>>> print(c)
Hello
>>> a, b = b, a
>>> print(a)
2
>>> print(b)
1
>>> a, b, c = b, c, a
>>> print(a, b, c)
1 Hello 2
```

mit und ohne Klammern möglich

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Deklaration von Funktionen

```
def Name (Argumente):
    Anweisung1
        ...
    Anweisungn
    [return expression]
```

Optional

Wenn man Anweisungen in eine Funktion schreibt, **ist Einrückung notwendig**. Die Benutzung von Leerzeichen ist nicht notwendig, aber ist empfohlen wegen Konsistenz.

`funktion.py`

```
def printme(str):
    print(str)
printme("Test")
```

→

```
% python funktion.py
Test
```

Funktionsargumente

- Funktionsargumente werden in Python **immer "by reference"** übergeben

```
wechseln.py

def wechseln(liste):
        liste.append([5,6])
        return
liste = [10, 20, 30]
wechseln(liste)
print(liste)
```

→

```
% python wechseln.py
[10, 20, 30, [5, 6]]
```

Die Referenz von liste wird überschrieben

```
wechseln.py

def wechseln(nummer):
        nummer = 5
        return
nummer = 1
wechseln(nummer)
print(nummer)
```

Achtung!

→

```
% python wechseln.py
1
```

nummer ist eine lokale Variable von *wechseln*

Rückgabewerte

- **Alle** Funktionen in Python haben ein Rückgabewert
  - Falls kein return-Wert spezifiziert ist, gibt eine Python-Funktion *None* zurück
  - *None* ist eine spezielle Konstante
    - genauso wie *null* in Java
    - Python gibt *None* **nicht** auf die Konsole aus
  - Dank Tuples kann eine Funktion auch mehrere Rückgabewerte haben

summeprodukt.py

```
def summe_und_produkt(a, b):
    return a + b, a * b

print(summe_und_produkt(2, 10))
print(summe_und_produkt(6, 4))
```

→

```
% python summeprodukt.py
(12, 20)
(10, 24)
```

Rückgabewerte (2)

- „Function overloading“ ist **nicht** möglich!
  - Zwei Funktionen können nicht gleichen Namen haben

Verwendung von Funktionen

- Eine Python-Funktion ist ein **Datentyp**
- Eine Python-Funktion kann
  - ein Funktionsargument
  - ein Teil von einer Liste
  - Ein Variablenwert
  - ...

  sein

spass.py

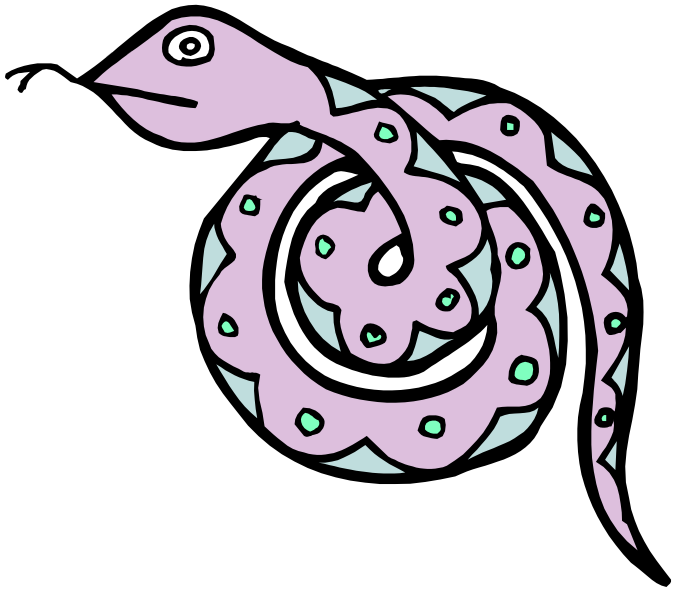
```
def square(y):
        return y*5
def multiply(x, y):
        return x*square(y)
print(multiply(3, 5))
```

→

```
% python spass.py
75
```

Inhalt

- Was ist ein Skript? Wie führt man ein Skript aus?
- Kurzüberblick Python
  - Grundlagen
    - Interpretation bei Python
    - Variablen und mathematische Operatoren
    - Eingabe/Ausgabe
    - Module
    - Bedingungen und Schleifen
  - Fortgeschrittene Themen
    - Listen und Tuples
    - Funktionen
    - Klassen



Deklaration von Klassen

Einrückung ist notwendig, genauso wie bei Schleifen und Bedingungen

```
class name:
    argument = wert
        ...
    def methode (self, argumente):
        Anweisung1
            ...
        Anweisungn
```

```
class Student:
        name = ""
        jahre = 0
        def __init__(self, Name, Jahre):
                self.name = Name
                self.jahre = Jahre
                return
        def get_jahre(self):
                return self.jahre
```

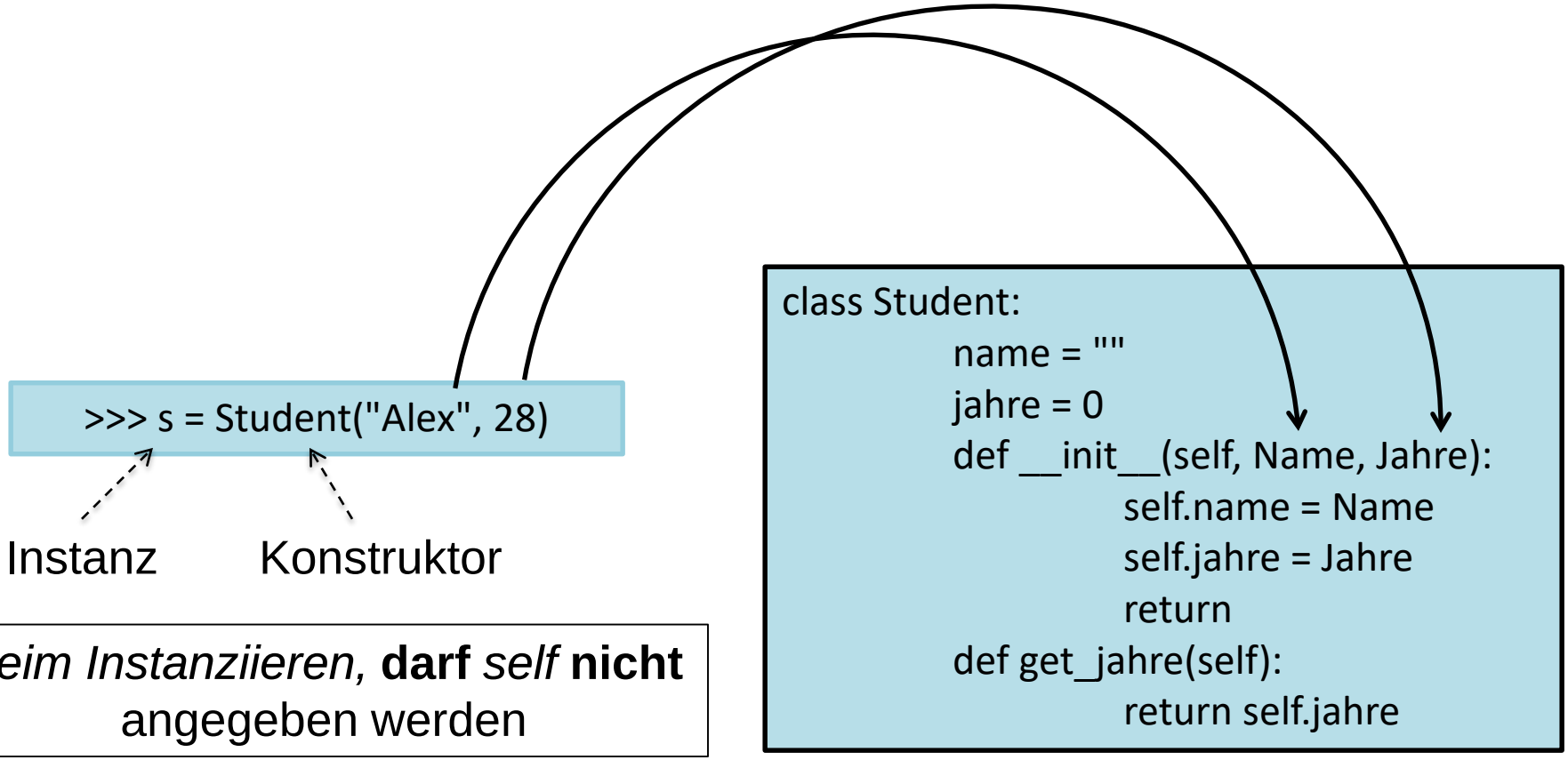
Die *__init__* Methode ist ein Konstruktor (optional)

Der erste Parameter einer Instanzmethode **muss** *self* sein

Java *this* in Java ist äquivalent zu *self in Python*

Instanziierung von Klassen

- Um eine Klasse zu instanziieren, muss man den Konstruktor der Klasse aufrufen



Beim Instanziieren, **darf** *self* **nicht** angegeben werden

Zugriff auf Methoden und Argumente

- Zugriff auf Methoden

```
>>> s = Student("Alex", 28)
>>> s.get_jahre()
28
```

Zur Erinnerung: self **darf nicht** angegeben werden

```
class Student:
        name = ""
        jahre = 0
        def __init__(self, Name, Jahre):
                self.name = Name
                self.jahre = Jahre
                return
        def get_jahre(self):
                return self.jahre
```

- Zugriff auf Variablen

```
>>> s = Student("Alex", 28)
>>> print(s.name)
Alex
```

Sichtbarkeit von Klassenvariablen, wie *private*, *public* und *protected* in Java, **existiert nicht** in Python

Vererbung

```
class Tier:
        art = ""
        def set_art(self,Art):
                self.art = Art

class Tiger (Tier):
        def __init__(self):
                set_art("Tiger")
                return
```

Tiger erbt von *Tier*

Tiger greift auf *Tier‘s* Methode *set_art und Tier‘s* Attribut *art* zu

```
>>> t = Tiger()
>>> print t.art
Tiger
```

Eine Kindklasse in Python **kann auf alle** Members und Methoden ihrer Elternklasse zugreifen

Multiple Vererbung ist möglich ----> class Kind **(Eltern1, Eltern2)**:

Überschreiben von Funktionen ist möglich

Ganz am Ende

- Der *pass* Befehl macht gar nichts 
  - Oft benutzt als Platzhalter für zukünftigen Code

```
For i in (1,1000):
    pass
```

```
class Student:
    pass
```

- Weitere wichtige Python Funktionalitäten
  - Regular expressions (siehe das *re* Modul)

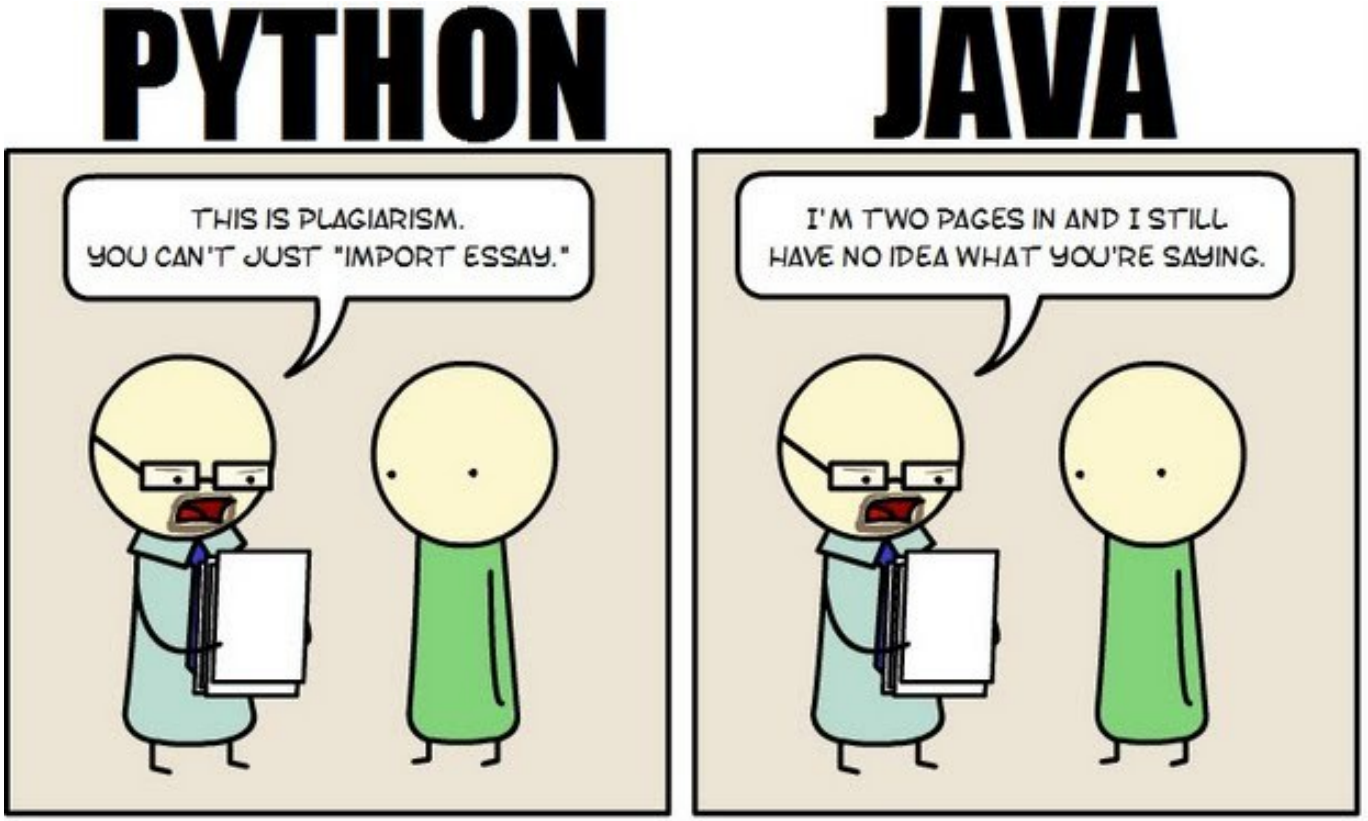
Mehr unter: https://docs.python.org/3.7/library/re.html
(für Python version 3.7.2)

  - Dictionaries (key-value mappings)

```
>> passwd = {"user1":"pass1", "user2":"pass2"}
>> passwd["user1"]
pass1
```

Viel, viel mehr unter: https://docs.python.org/3.7/tutorial/
(für Python version 3.7.2)

Fragen?


PYTHON
JAVA
THIS IS PLAGIARISM.
YOU CAN'T JUST "IMPORT ESSAY."
I'M TWO PAGES IN AND I STILL
HAVE NO IDEA WHAT YOU'RE SAYING.