Quick-Start Informatik 2011

Programmieren in Python
Teil 2

Organisatorisches

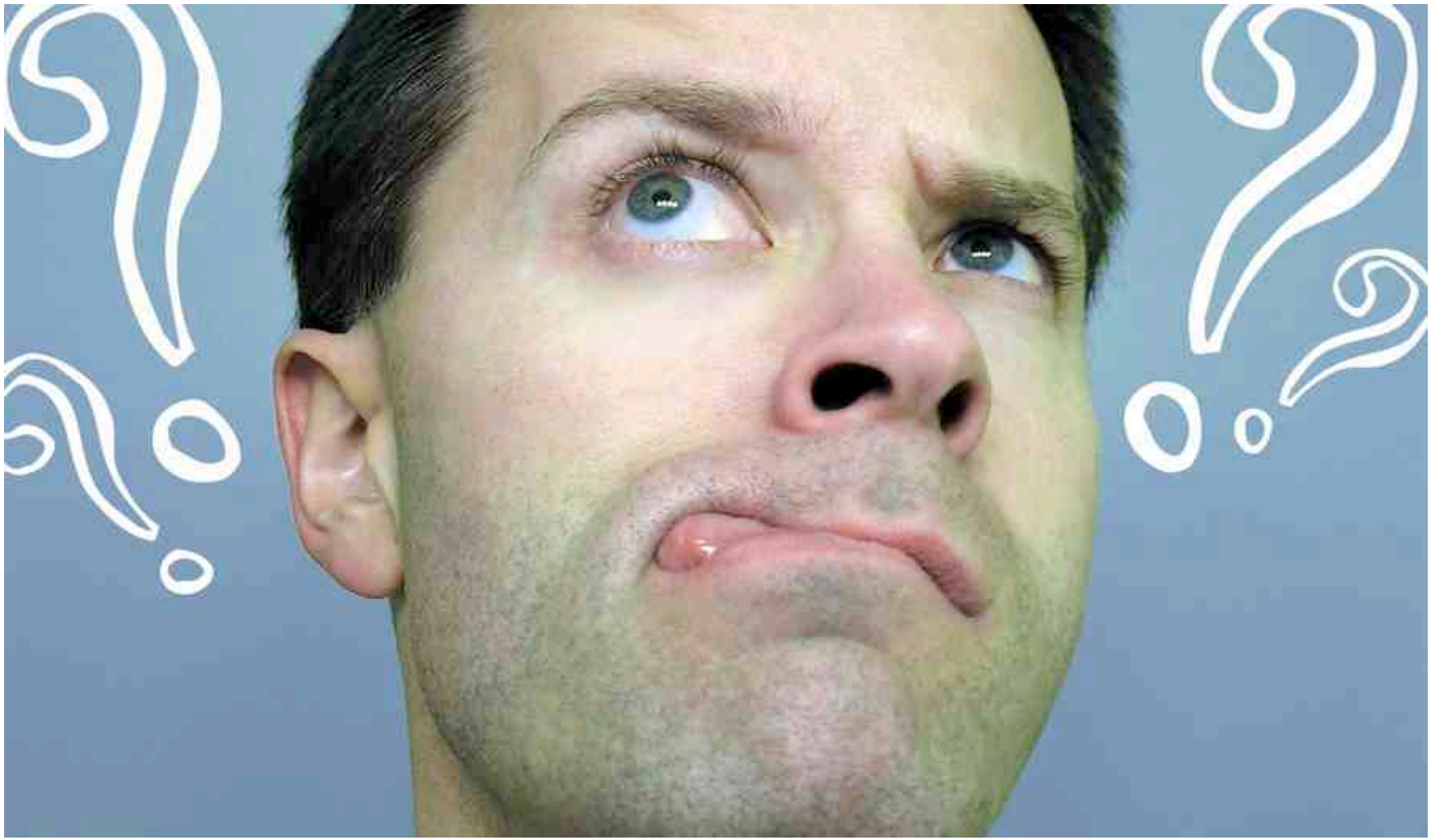
- Self-Assessment-Bögen
- Aufgabenblätter!

Rückblick

1. Programmieren: Wie geht das?
   (IDLE, REPL, .py-Dateien)
2. Daten und Datentypen
   1. Zahlen (`2, 1.5`)
   2. Vergleiche (`==`)
      und Wahrheitswerte (`bool`)
   3. Text (`"String"`)
3. Variablen (`name = "Wert"`)
4. Bedingungen stellen (`if`/`elif`/`else`)
5. Daten anfordern (`input()`)
   und Ausgeben (`print()`)

http://www.flickr.com/photos/kyriiee/4637080936

Fragen, fragen, fragen!



Forum: http://fsinf-forum.de/viewforum.php?f=45
Fragen an Tutoren: quick-start@lists.fsinf-frankfurt.de

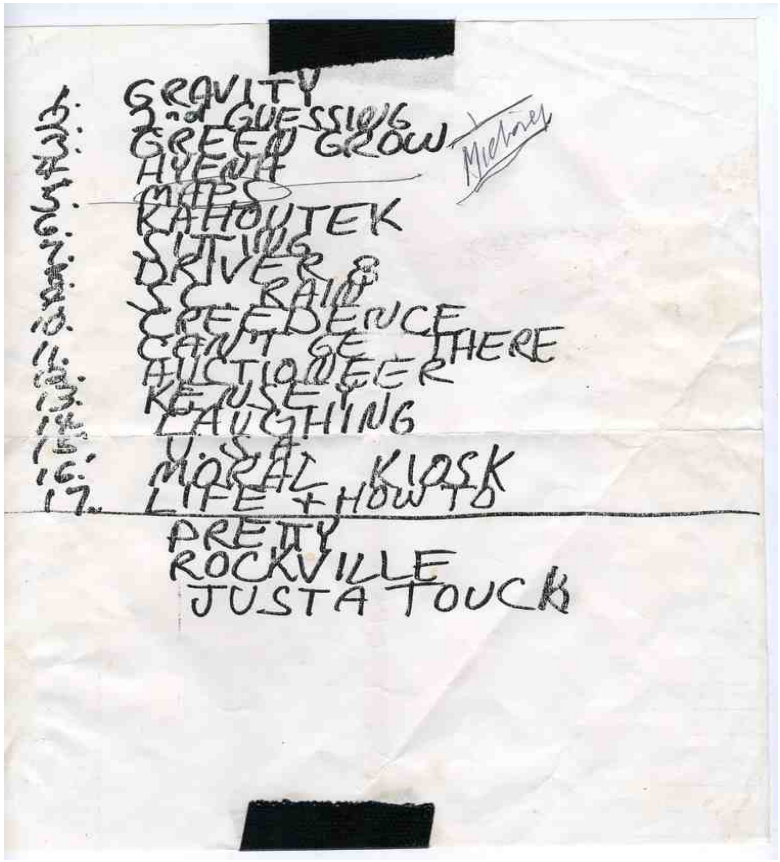
Agenda

1. Listen
2. Wörterbücher
3. Wiederholtes Ausführen
4. Eigene Funktionen
5. Sichtbarkeit von Variablen
6. Dateien lesen und schreiben
7. Externen Code laden
8. Turtle Graphics

Listen

Die Variable ist nicht genug...

Manchmal ist es mit einer Variable nicht getan: Man möchte einen ganzen Satz zusammengehöriger Daten speichern.



http://www.flickr.com/photos/27620885@N02/3018306107/

Listen zur Rettung!

http://www.flickr.com/photos/35134526@N03/3744804008

Listen

Die Python `list` kann mehrere Objekte zusammen speichern.

```
>>> list = []
>>> list1 = [1,2,3,42]
>>> list2 = ['Hello', 'world', '!']
>>> chaos_list = [5, 'test', True, "Ein
string"]
```

Einige Listenoperationen

Objekte hinzufügen:

```
>>> list = [1,2]
>>> list.append(3)   # 3 am Ende Einfügen
>>> list = list + [4,5]   # Listen anhängen
>>> list
[1, 2, 3, 4, 5]
```

Auf Objekte zugreifen:

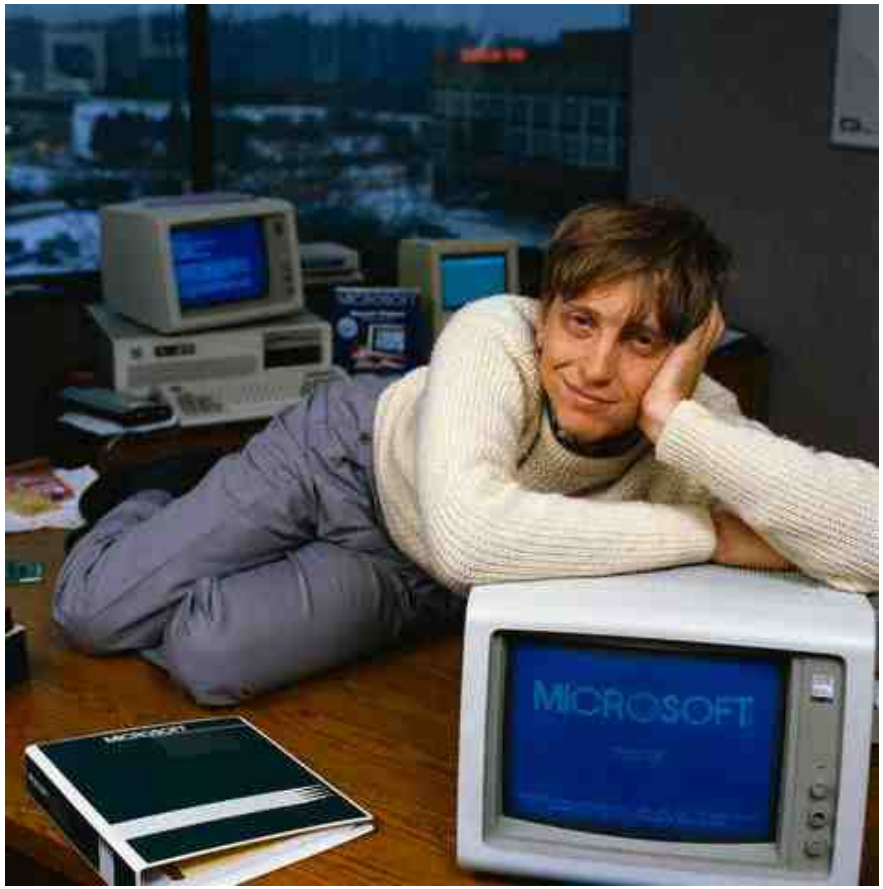
```
>>> list[0]   # erstes Listenelement
1
```

BTW: Die Zahl in den `[]` nennt man Index.

Informatiker können seltsam sein

Sie fangen bei 0 an zu zählen!

Das ist im ersten Moment vielleicht etwas seltsam, aber man gewöhnt sich daran...



Ganz so seltsam ist das nicht ...

Edsger Dijkstra erklärt warum wir bei Null zu zählen anfangen sollten:
http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html

Weitere Listenoperationen

Wollen wir wissen, wie viele Elemente in der Liste sind, verwenden wir `len()`:

```
>>> list = [1, 2, 3, 4, 5]
>>> len(list)
5
```

http://www.flickr.com/photos/lianakabel/266823190/

Listenelemente löschen wir mit dem Befehl `del` wieder aus der Liste.

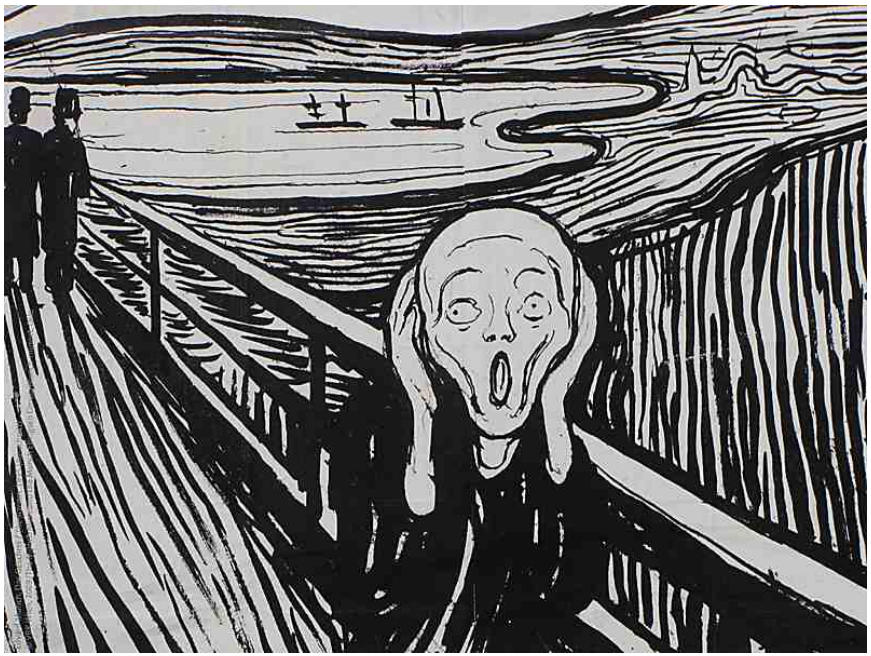
```
>>> del list[2]
>>> list  # Beachte: Elemente rücken auf!
[1, 2, 4, 5]
```

Hilfe!

Vielleicht fragst du dich langsam, wie du dir das alles merken sollst.

Python bietet dir eine `help()` Funktion als Gedächtnisstütze.

Gibst du z.B. `help(list)` ein, listet dir die Hilfe alle Methoden und eine kurze Beschreibung auf.



http://www.flickr.com/photos/bonys_bilder/4022993422/

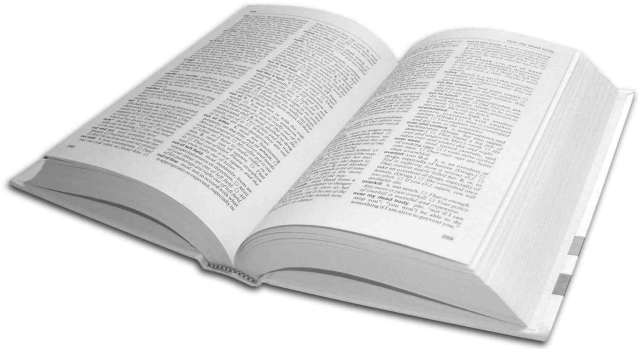
Wörterbücher

Wörterbücher

- Ähnlich wie Listen: Mehrere Elemente speichern
- Aber: Eigene Indizes definieren!

Beispiel:

```
>>> phonebook = {'Meier': '069123', 'Müller':
'069456'}
>>> phonebook['Müller']
'069456'
```



Hinzufügen:

```
>>> phonebook['Schmidt'] = '069789'
```

Löschen

```
>>> del phonebook['Meier']
```

Wiederholtes Ausführen

Wiederholtes Ausführen

Stellen wir uns einmal vor, wir wollen die Zahlen von 0 bis 99 ausgeben:

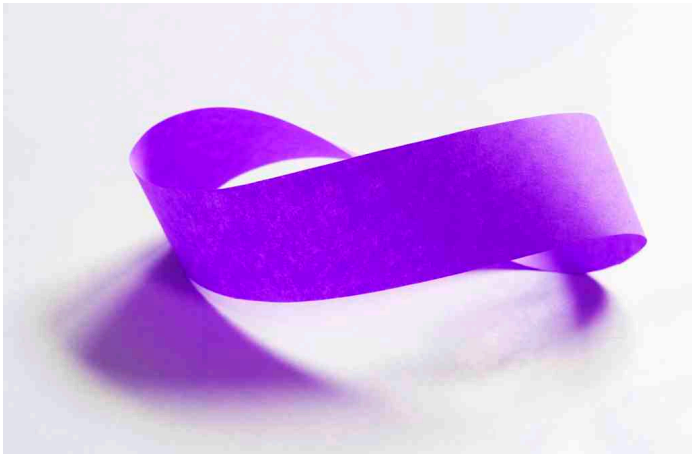
```
print 0
print 1
print 2
print 3
# usw...
print 99
```

Geht das nicht einfacher?

Wiederholtes Ausführen

Einfacher geht es mithlife einer while-Schleife:

```
x = 0
while x < 100:
    print(x)
    x = x + 1
```



http://www.flickr.com/photos/richardmayston/2878488106/

Wurde der Block ausgeführt, wird die Bedingung wieder geprüft. Solange sie wahr ist, wird er erneut ausgeführt.

Achtung: Du kannst eine Endlosschleife bauen, wenn du nicht vorsichtig bist!

Aus der Schleife ausbrechen

Mittels `break` lässt sich eine Schleife abbrechen:

```
x = 0
while True:
    if x == 3:
        break
    print(x)
    x = x + 1
```

http://www.flickr.com/photos/ladybeames/2896787167/

Hier wird eine vermeintliche Endlosschleife beendet.

Quizfrage: Was wird hier ausgegeben?

Einen Schritt überspringen

Mittels `continue` überspringt man den Rest des Blocks:

```
x = 0
while x < 100:
    x = x + 1
    if x % 2 != 0:
        continue
    print(x)
```

http://www.flickr.com/photos/rcameraw/453644988/

Hier werden alle geraden Zahlen ausgegeben, die ungeraden werden übersprungen.

Die for-Schleife

Die for-Schleife ist der zweieiige Zwilling der while-Schleife, der es liebt durch Listen zu laufen.

```
for x in ['Dies', 'ist', 'eine', 'Liste']:
print(x)
```

Die Schlüsselwörter `break` und `continue` lassen sich ebenfalls bei for-Schleifen verwenden.

Informatiker sprechen übrigens gerne von "iterieren", wenn sie "durch eine Liste von Dingen laufen".

Die range()-Funktion

Mit Hilfe der for-Schleife und der `range()` Funktion können wir das Zählen bis 100 noch eleganter gestalten:

`range()` erzeugt eine Liste aller Zahlen in einem Bereich:
Syntax: `range(start, stop, step)`

`start`: Anfang der Liste (optional)
`stop`: Ende der Liste (nicht eingeschlossen!)
`step`: Schrittweite (optional)

for-Schleife

Beispiele:

```
>>> range(3) # nur stop-Parameter
[0, 1, 2]

>>> range(1, 4) # start- und stop-Parameter
[1, 2, 3]

>>> range(0, 10, 2) # zus. step-Parameter
[0, 2, 4, 6, 8]
```

for-Schleife

Und so sieht das Zählen bis 100 als for-Schleife aus:

```
for x in range(100):
    print(x)
```

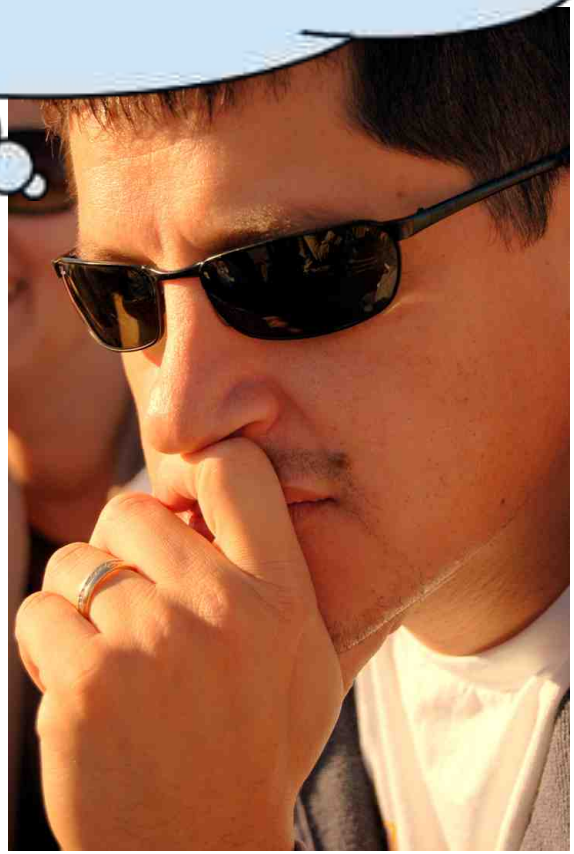
Funktionen

In der Zwischenzeit...

Unser geniales Programm von gestern hat für Furore auf dem Campus gesorgt...

In der Mensa...

Ich wünschte ich könnte in meinem Programm auch auf Teilbarkeit prüfen ...



In der Mensa...


Kannst du mir bitte deine Teilbarkeitstestfunktion schicken?

Wir haben keine Funktionen...

Bisher haben wir unseren Code nicht in Funktionen gesteckt. Hätten wir das getan, könnten wir und andere den Code an verschiedenen Stellen einfach wieder verwenden.

http://www.flickr.com/photos/24933501@N06/2352022690/

Funktionen

Für ihren Nutzer sind Funktionen etwas wie ein Computer:

Funktionen

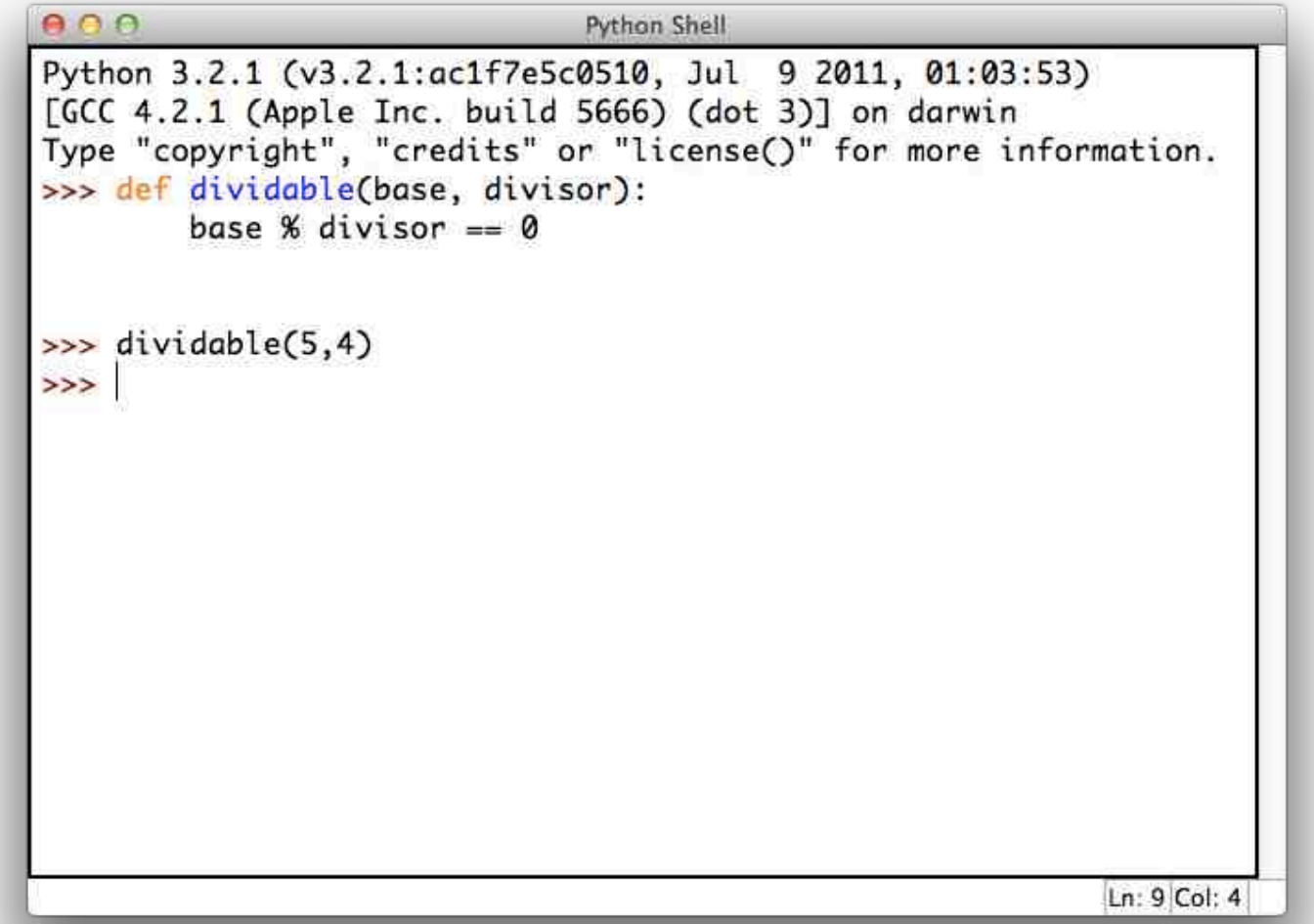
Hier ist unsere Teilbarkeitsfunktion:

```
def dividable(base, divisor):
    base % divisor == 0
```

- Das Schlüsselwort `def` leitet die Funktionsdefinition ein
- Dann kommt der Name der Funktion
- Innerhalb der Klammern steht die Eingabe (Parameter)
- Im folgenden Block stehen die Anweisungen

Funktionen

Aber wenn wir die Funktion ausführen:



Irgendetwas fehlt ...

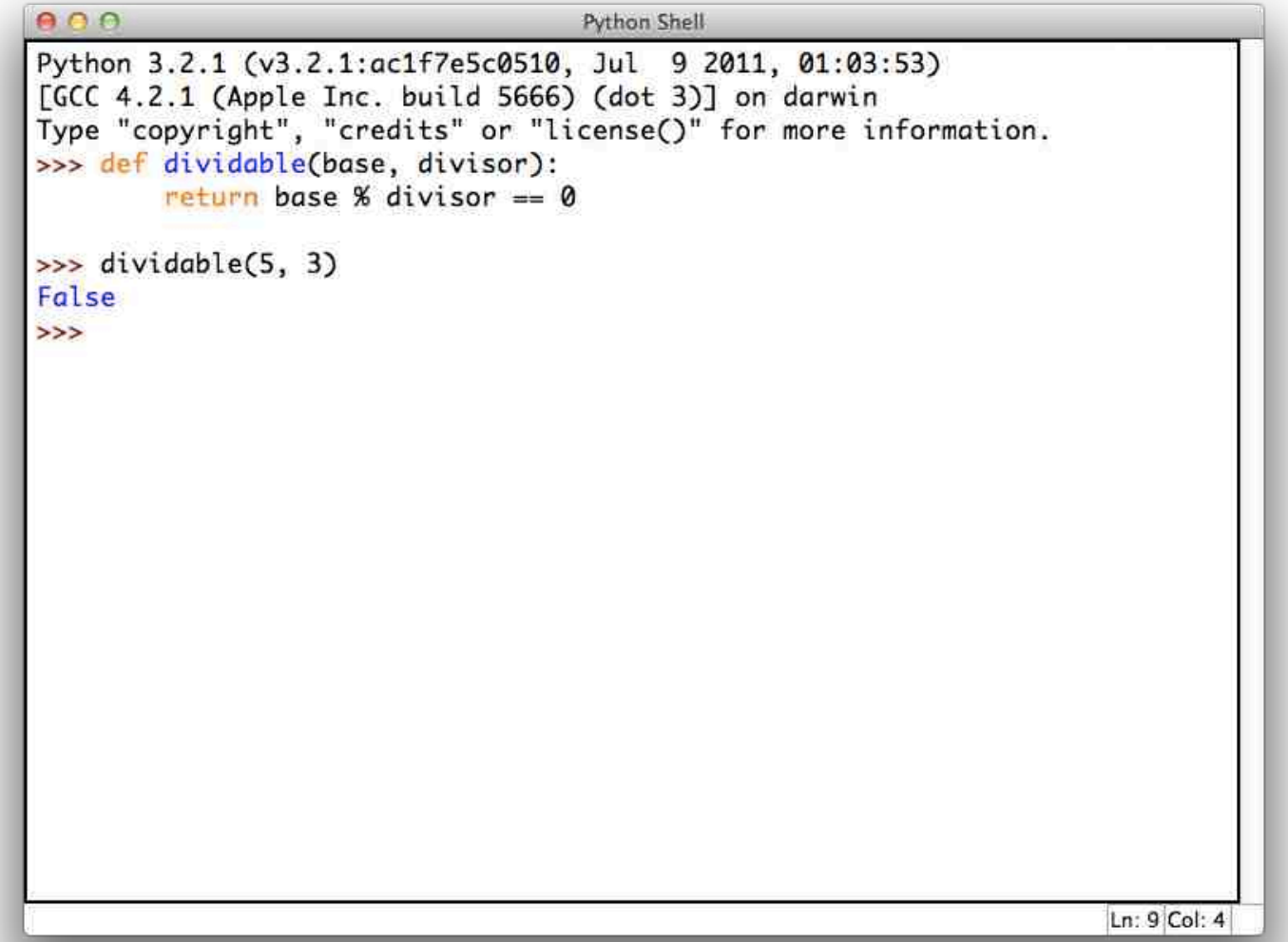
Return

Unsere Funktion gibt keinen Wert zurück!

Die `return` Anweisung sagt dem Interpreter, welchen Wert die Funktion dem Aufrufer **zurück geben** soll.

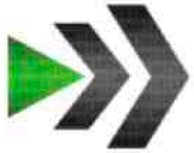
http://www.flickr.com/photos/gogreenasap/4368124634/

Return



Achtung: return beendet in jedem Fall die Ausführung der Funktion - auch falls danach noch Code steht!

print **vs.** return

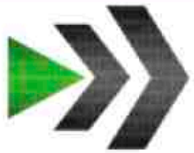


print	**return**

`def three():`
`    print(3)`
 	
`x = three()`
`print("x: " + str(x))`
	`def three():`
	`    return 3`
	
	`x = three()`
	`print("x: " + str(x))`

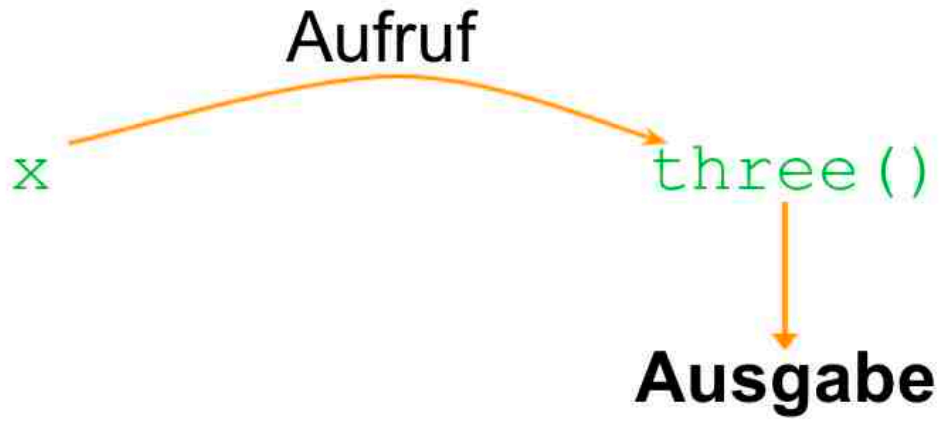
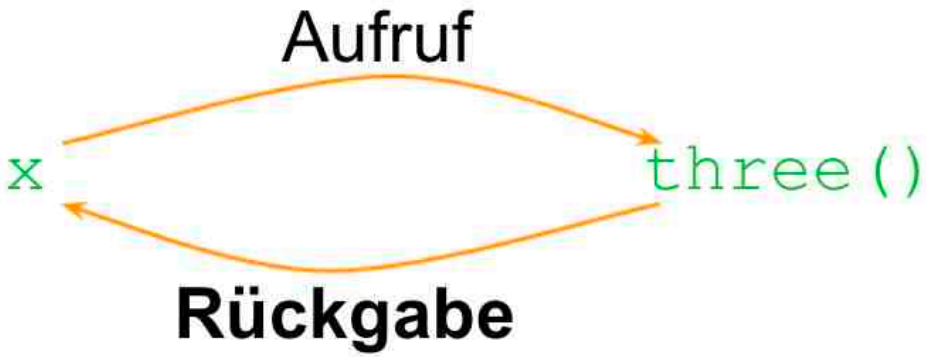
Ausgabe:
`3`
`x: None`
	Ausgabe:
	`x: 3`

print **VS.** return



print	**return**

`def three():`
`    print(3)`
 	
`x = three()`
	`def three():`
	`    return 3`
	
	`x = three()`

Sichtbarkeit von Variablen

Sichtbarkeit von Variablen

Stell dir vor, jemand verwendet diese Funktion von dir:

```
def do_something():
    x = 0
```

und in seinem Code steht folgendes:

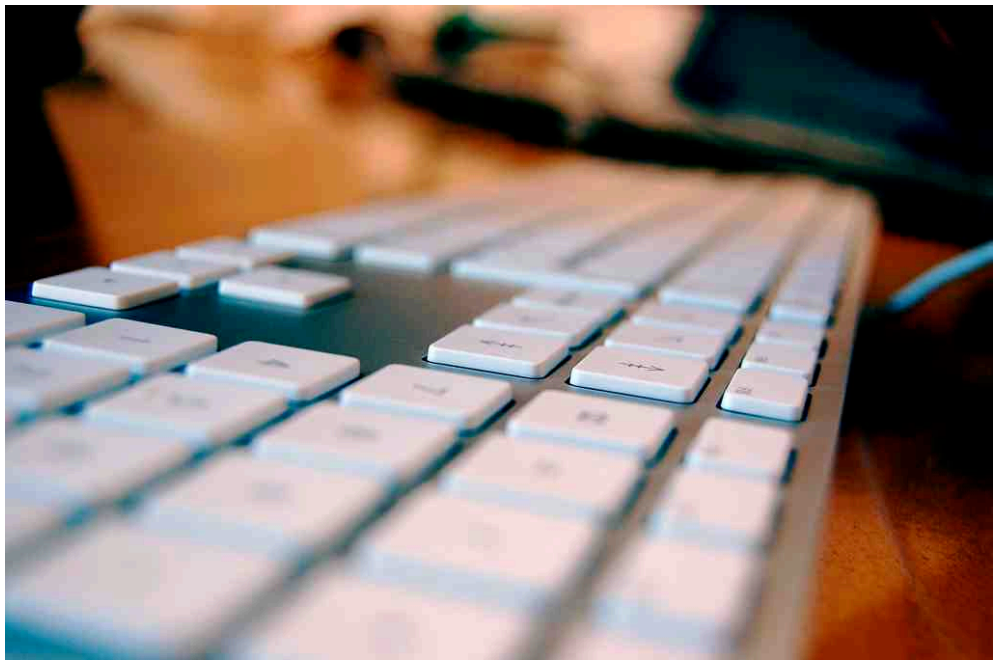
```
>>> x = 2
    #Mehr Code
>>> do_something()
>>> y = 5 / x
```

Yikes! Ob er wohl wusste, dass die Funktion **x** auf 0 setzt?

Sichtbarkeit von Variablen

Unsere Funktion ist schuld an seinem Fehler!
... oder?

Live Coding!



http://www.flickr.com/
photos/nez/1371111259/

Namensräume

Der Block der Funktion beginnt einen neuen Namensraum (Scope).

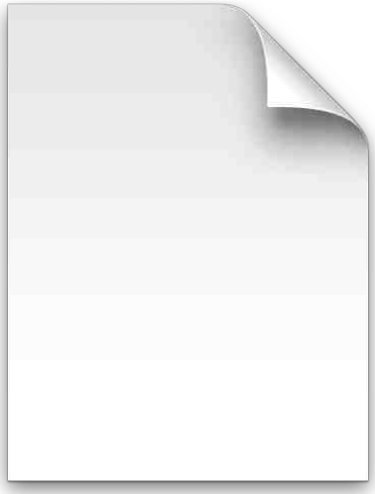
```
>>> def do_something(40):
        # Hier neuer Namensraum!
```

Die Sichtbarkeit von Variablen, die du dort zuweist, ist auf diesen Block beschränkt.

Dateien lesen und schreiben

Dateien lesen und schreiben

Der Befehl `open("dateiname.txt")` öffnet eine Datei. Einfach oder?



Durch die Zeilen einer Datei kannst du wie durch eine Liste mit der for-Schleife iterieren:

```
>>> with open("hallo.txt") as file:
        for line in file:
            print(line)
```

Dateien lesen und schreiben

Wollen wir nicht eine ganze Zeile lesen, können wir die `read()` Methode der Datei verwenden um anzugeben, wie viele Zeichen wir lesen möchten:

```
>>> with open("hallo.txt") as file:
        file.read(5)
'Hallo'
```

Beim erneuten Aufruf der `read()` Methode werden die nachfolgenden Zeichen ausgegeben.

Dateien lesen und schreiben

Versuchen wir aber in die Datei zu schreiben...

Python Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7 (r27:82500, Sep 24 2010, 18:49:26)
[GCC 4.4.4 20100630 (Red Hat 4.4.4-10)] on linux2
Type "copyright", "credits" or "license()" for more informa
tion.
>>> datei = open("hallo.txt")
>>> datei.write("Hallo, welt!")

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    datei.write("Hallo, welt!")
IOError: File not open for writing
>>>
```

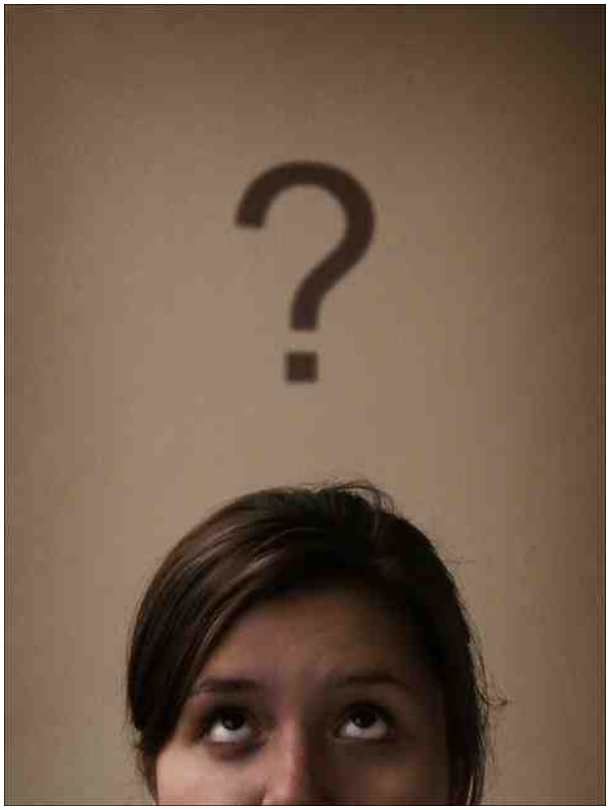
Dateien lesen und schreiben

Der Befehl `open("dateiname.txt")` öffnet eine Datei im Lesemodus. Damit wir schreiben können müssen wir das Python ausdrücklich sagen:

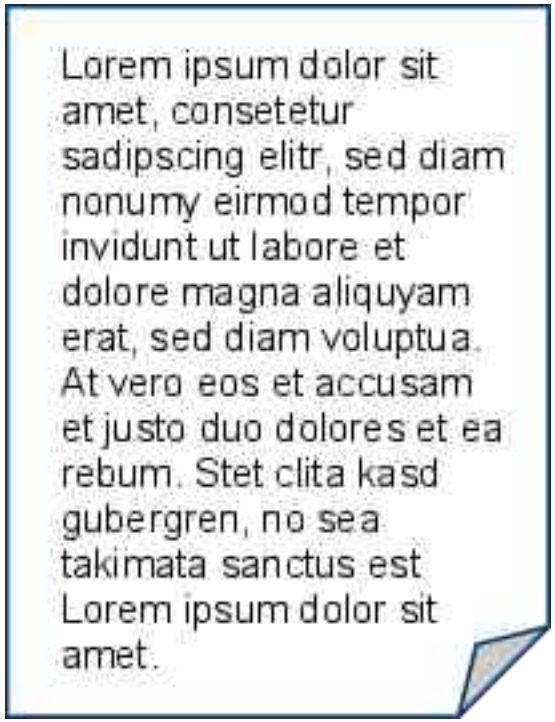
```
>>> with open("hallo.txt", "r+") as file:
        file.write("Hallo, Welt!")
```

Dateien lesen und schreiben

Aber wohin geht mein Text, wenn ich in die Datei schreibe?



http://www.flickr.com/photos/scabeater/3271864251/



An den Anfang?

Ans Ende?

Dateien lesen und schreiben

Der Cursor startet **am Anfang** der Datei.

Seine Position kannst du mit der `tell()`-Methode der Datei prüfen.

Mit der Methode `seek(position)` kannst du ihn auf den übergebenen Buchstaben setzen.

`seek(5)` setzt den Cursor z.B. auf den auf den sechsten (!) Buchstaben. Dieser und alle kommenden Buchstaben würden bei einem `write()` überschrieben.

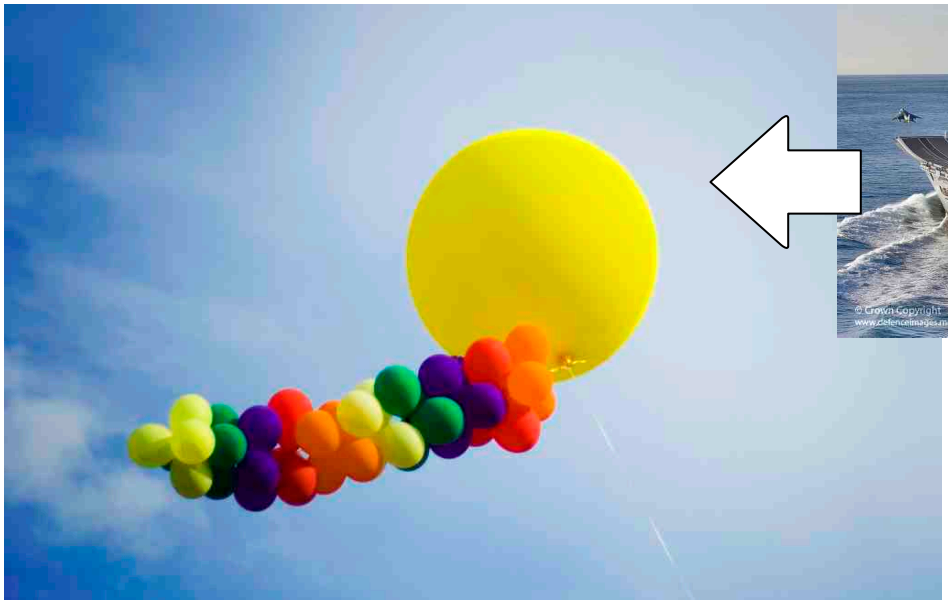
Module und Importe

Module und Importe

- Python liefert zunächst die wichtigsten Funktionalitäten
- Braucht man mehr: `import`
- Es wäre nicht praktikabel, von Anfang an alles zu laden

Module und Importe

- Python liefert zunächst die wichtigsten Funktionalitäten
- Braucht man mehr: import
- Es wäre nicht praktikabel, von Anfang an alles zu laden



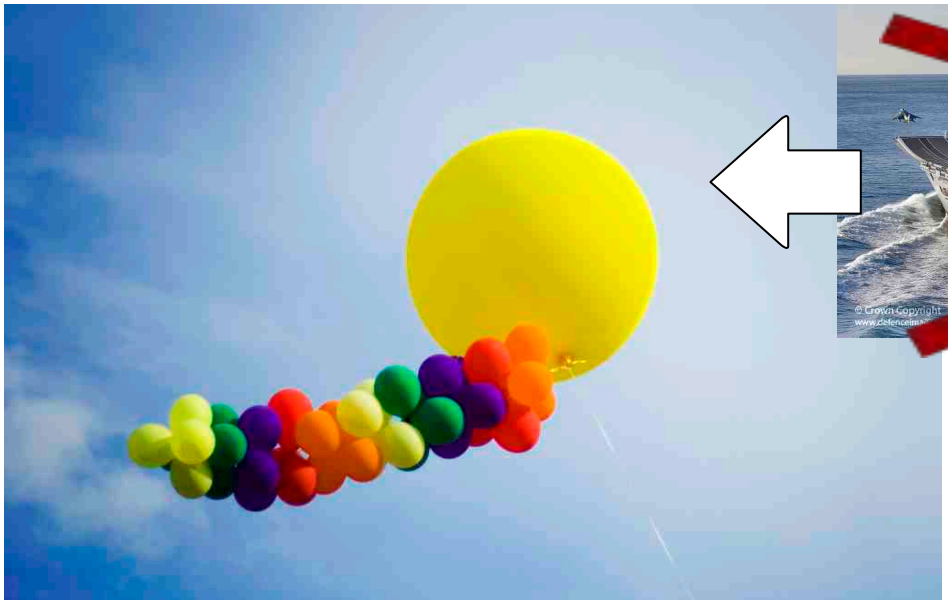
+

?

http://www.flickr.com/photos/underneath/3626513925/
http://www.flickr.com/photos/defenceimages/5035954229/
http://www.flickr.com/photos/defenceimages/5280822452/

Module und Importe

- Python liefert zunächst die wichtigsten Funktionalitäten
- Braucht man mehr: `import`
- Es wäre nicht praktikabel, von Anfang an alles zu laden



?

✓

http://www.flickr.com/photos/underneath/3626513925/
http://www.flickr.com/photos/defenceimages/5035954229/
http://www.flickr.com/photos/defenceimages/5280822452/
http://www.flickr.com/photos/sheph/452137142/

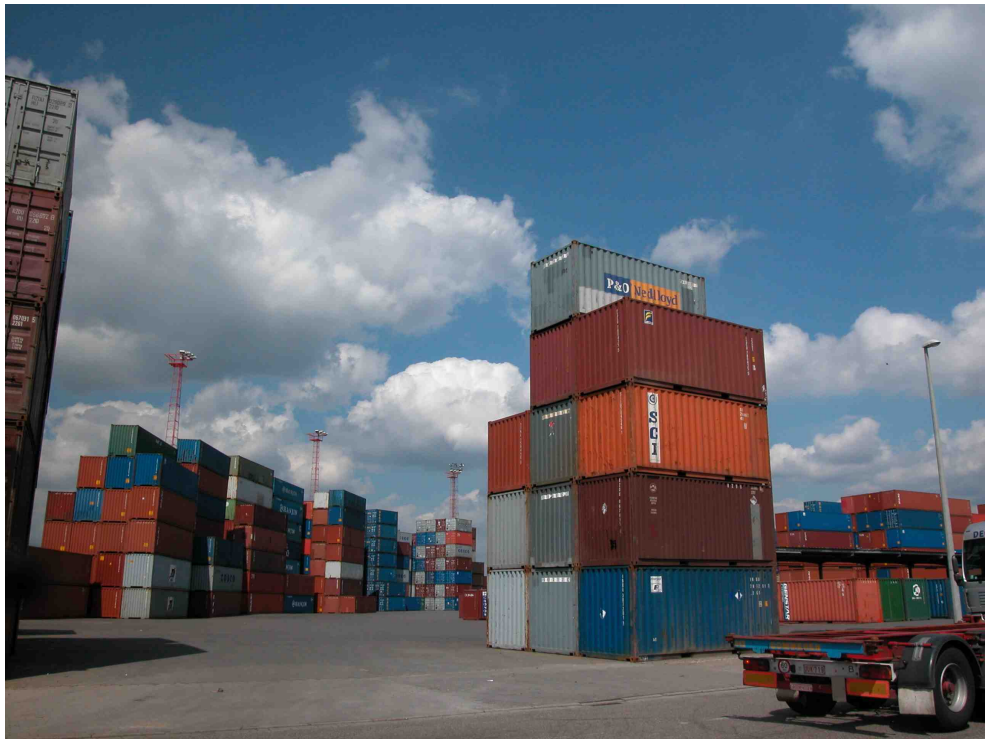
Module und Importe

Syntax:

1. `import MODULNAME`

oder

2. `from MODULNAME import NAME1, NAME2, ...`



Module und Importe

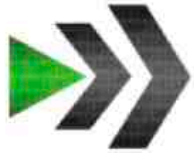
Beispiel zu 1.

```
>>> import random
>>> random.randint(0,10)
10
```

Beispiel zu 2.

```
>>> from random import *
>>> randint(0,10)
5
```

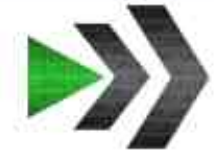
Module und Importe



Wann was benutzen? ("Faustregeln")

`import MODULE`	`from MODULE import ...`
• Normalerweise	• Bei häufiger Verwendung • Beim Testen auf der Konsole • **Achtung**: Gleicher Scope!

Module und Importe

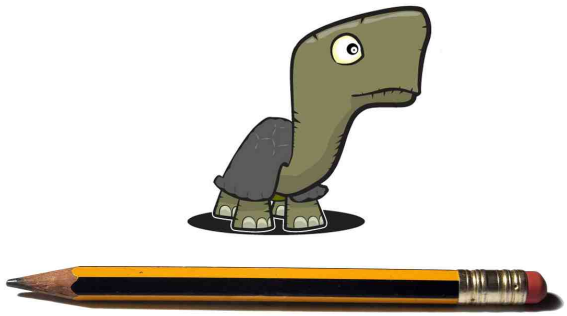


Nützliche Module

- `random`
  - Funktionen: `random()`, `randint()`, ...
  - http://docs.python.org/release/3.2/library/random.html

- `math`
  - Funktionen: `ceil()`, `floor()`, `exp()`, `sqrt()`, ...
  - http://docs.python.org/release/3.2/library/math.html

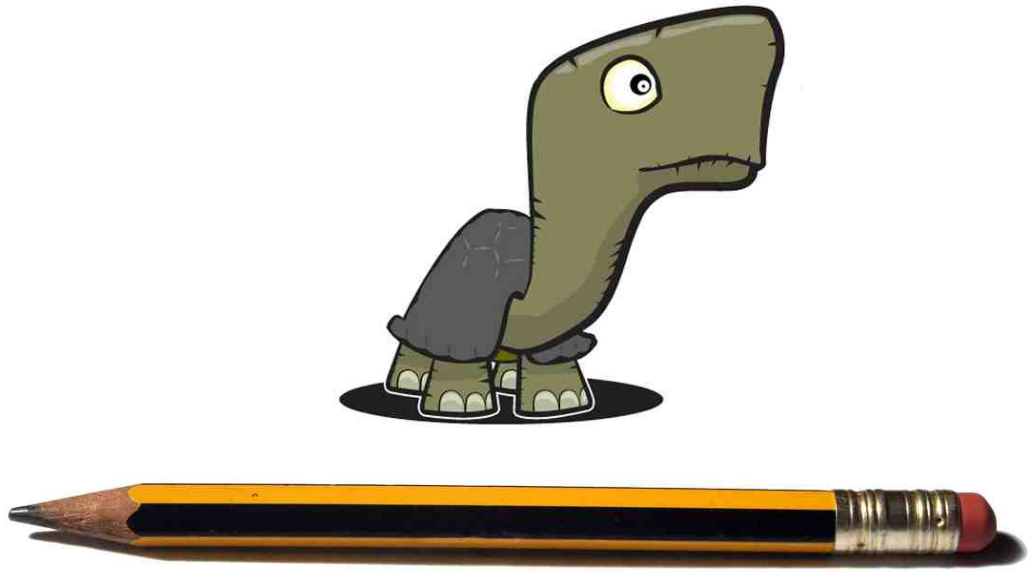
- `turtle` (Turtle Graphics)
  - Dazu gleich mehr...



Turtle Graphics

Turtle Graphics

- Entwickelt in den späten 60ern für die Sprache Logo
- Intuitiver Zugang
- Vorstellung: Schildkröte läuft mit Stift
- Heute: Verbreitung in verschiedenen Programmiersprachen, z.B. Python, Java, etc.



Turtle Graphics

Funktionen stecken im Modul `turtle`:

```
import turtle              # Möglichkeit 1, oder
from turtle import *       # Möglichkeit 2
```

Nützliche Funktionen:

`forward()`,
`left()`, `right()`, `penup()`, `pendown()`,
`pencolor("FARBE")`, uvm.

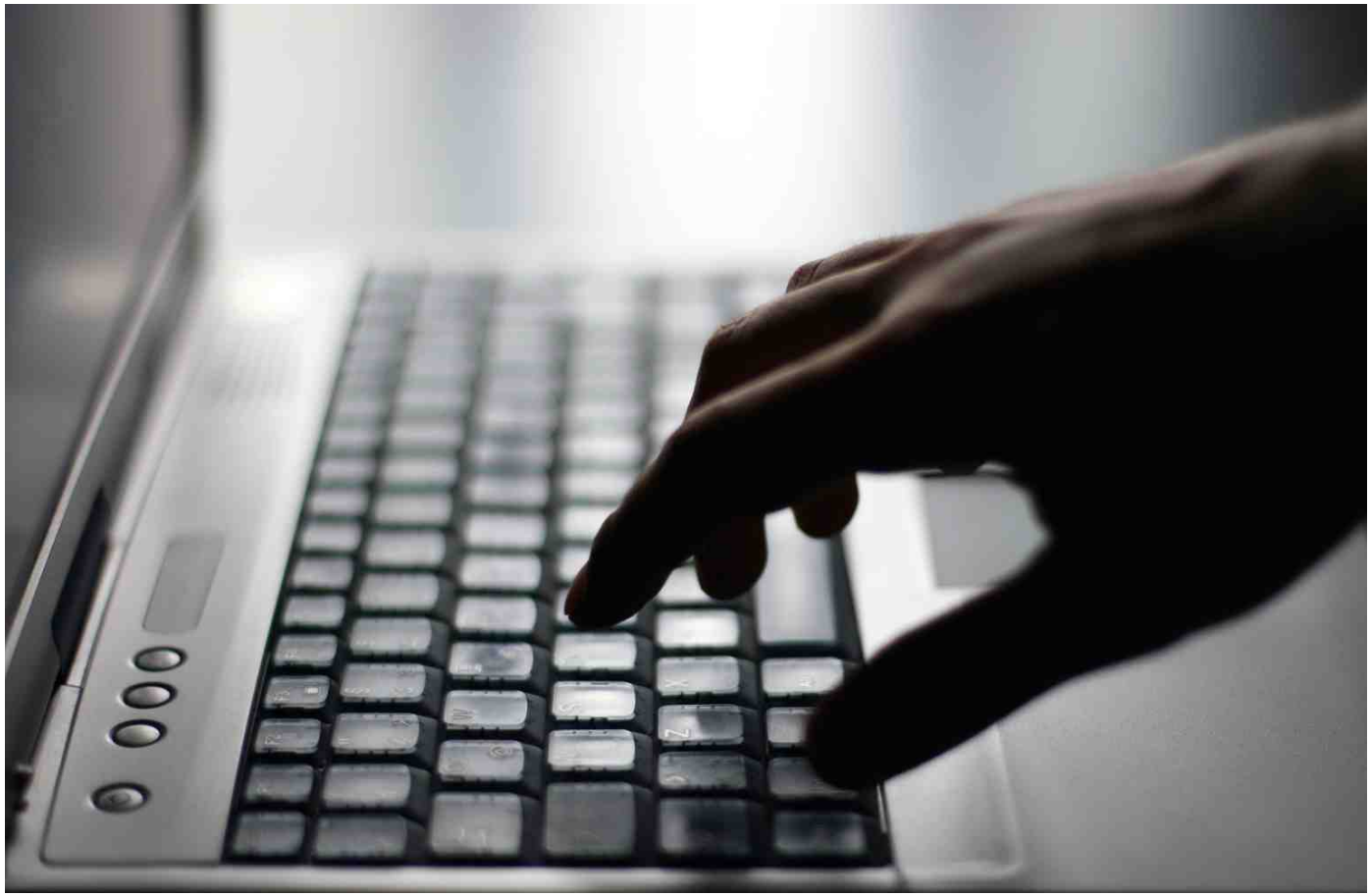
Insbesondere:

`speed(0)`, `delay(0)`, `exitonclick()`

Nachschlagen:
http://docs.python.org/release/3.2/library/turtle.html

Turtle Graphics

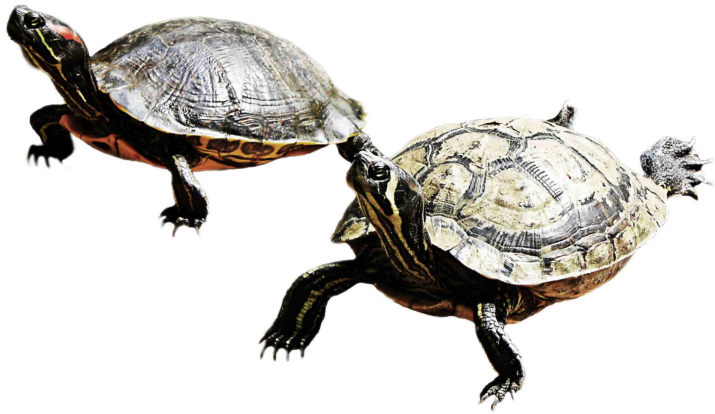
Live Demo, part 1



Turtle Graphics & Objekte

Bislang haben wir mit der Instanz `turtle` gearbeitet, die das Modul automatisch zur Verfügung stellt.

Wir können aber auch selbst beliebig viele Schildkröten anlegen!



Beispiel:

```
>>> messias = Turtle()
>>> messias.forward(10)
>>> killer_rabbit = Turtle()
>>> killer_rabbit.pencolor("red")
```

Turtle Graphics & Objekte

Live Demo, part 2

http://www.huslage24.de/artikelbilder/0199455_tasse_homer_l.jpg

Ab 14:00 Uhr

Übungen in den Fischer-Räumen / Raum 026!

Außerdem:

Während der Pause ist jemand in den Fischer-Räumen zur Betreuung!