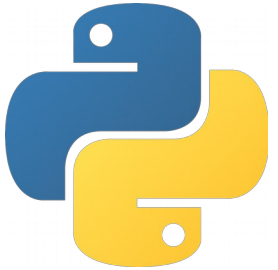
Python
Von 2.6 nach 3.x

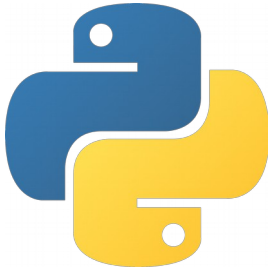


- Die wichtigsten Unterschiede
  - Im Folgenden werden die wichtigsten Unterschiede zwischen den Python-Versionen 2 und 3 aufgeführt

  - Hinweis
    - Mit Python 3 wurden auch viele subtile Änderungen, beispielsweise an den Schnittstellen vieler Module der Standardbibliothek, vorgenommen, die hier naheliegenderweise nicht alle erläutert werden können

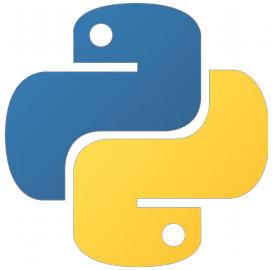
    - Antworten auf solch detaillierte Fragen finden Sie in den Online-Dokumentationen auf der Python-Website http://www.python.org

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Ein-/Ausgabe
    - In puncto Ein-/Ausgabe gab es zwei auffällige, aber schnell erklärte Änderungen
    - Das Schlüsselwort print aus Python 2 ist einer Built-in Function gleichen Namens gewichen
    - In der Regel brauchen hier also nur Klammern um den auszugebenden Ausdruck ergänzt zu werden

Python
Von 2.6 nach 3.x

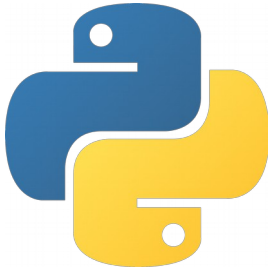


- Die wichtigsten Unterschiede
  - Ein-/Ausgabe

Python 2	Python 3
`>>> print "Hallo Welt"` `Hallo Welt`	`>>> print("Hallo Welt")` `Hallo Welt`
`>>> print "ABC", "DEF", 2+2` `ABC DEF 4`	`>>> print("ABC", "DEF", 2+2)` `ABC DEF 4`
`>>> print >> f, "Dateien"`	`>>> print("Dateien", file=f)`

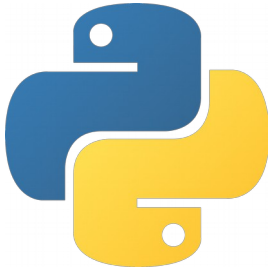
`>>> for i in range(3):`
`...  print i,`
`...`
`0 1 2`
	`>>> for i in range(3):`
	`...  print(i, end=" ")`
	`...`
	`0 1 2`

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Ein-/Ausgabe
    - Die zweite auffällige Änderung bezüglich Ein-/Ausgabe betrifft die Built-in Function input
    - Die input-Funktion aus Python 3 entspricht der raw_input-Funktion aus Python 2
    - Eine Entsprechung für die input-Funktion aus Python 2 gibt es in Python 3 als Built-in Function nicht, doch ihr Verhalten kann mithilfe von eval nachgebildet werden

Python
Von 2.6 nach 3.x



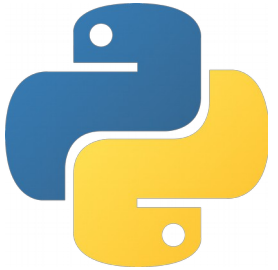
- Die wichtigsten Unterschiede
  - Ein-/Ausgabe

Python 2	Python 3

`>>> input("Ihr Wert: ")`
`Ihr Wert: 2**5`
`32`
	`>>> eval(input("Ihr Wert: "))`
	`Ihr Wert: 2**5`
	`32`

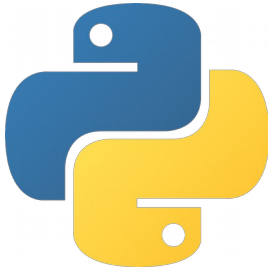
`>>> raw_input("Ihr Wert: ")`
`Ihr Wert: 2**5`
`'2**5'`
	`>>> input("Ihr Wert: ")`
	`Ihr Wert: 2**5`
	`'2**5'`

Python
Von 2.6 nach 3.x



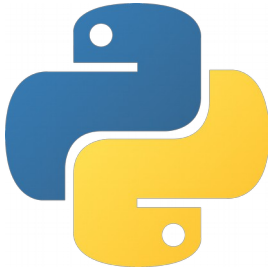
- Die wichtigsten Unterschiede
  - Iteratoren
    - Obwohl Python 2 das Iterator-Konzept bereits unterstützt, geben viele Funktionen, die für gewöhnlich zum Iterieren über eine bestimmte Menge verwendet werden, eine Liste der Elemente dieser Menge zurück, so beispielsweise die prominente Funktion range
    - Üblicherweise wird diese Liste aber nur in einer for-Schleife durchlaufen
    - Dies kann durch Verwendung von Iteratoren eleganter und speicherschonender durchgeführt werden

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Iteratoren
    - Aus diesem Grund geben viele der Funktionen und Methoden, die in Python 2 eine Liste zurückgeben, in Python 3 einen auf die Gegebenheiten zugeschnittenen Iterator zurück
    - Um diese Objekte in eine Liste zu überführen, können sie einfach der Built-in Function list übergeben werden
    - Damit verhält sich zum Beispiel die range-Funktion aus Python 3 gerade so wie die xrange-Funktion aus Python 2
    - Ein Aufruf von list(range(...)) in Python 3 ist äquivalent zur range-Funktion aus Python 2

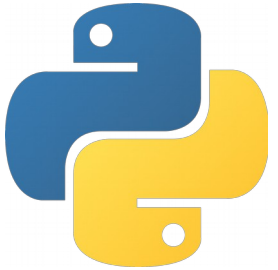
Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Iteratoren

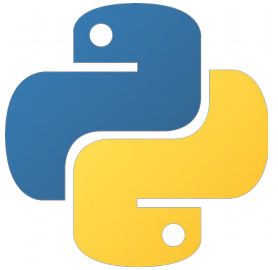
Python 2	Python 3
`>>> xrange(5)` `xrange(5)`	`>>> range(5)` `range(0, 5)`
`>>> range(5)` `[0, 1, 2, 3, 4]`	`>>> list(range(5))` `[0, 1, 2, 3, 4]`

Python
Von 2.6 nach 3.x



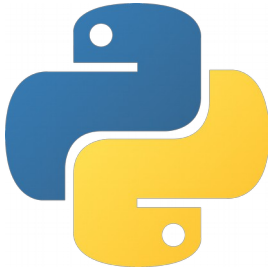
- Die wichtigsten Unterschiede
  - Strings
    - Die wohl grundlegendste Änderung in Python 3 ist die Umdeutung des Datentyps str
    - In Python 2 existieren zwei Datentypen für Strings: str und unicode
    - Während ersterer zum Speichern beliebiger Byte-Folgen verwendet werden kann, war letzterer für Unicode-Text zuständig
    - In Python 3 ist der Datentyp str ausschließlich für Text zuständig und mit dem unicode-Datentyp aus Python 2 vergleichbar

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Strings
    - Zum Speichern von byte-Folgen gibt es in Python 3 die Datentypen bytes und bytearray, wobei es sich bei bytes um einen unveränderlichen und bei bytearray um einen veränderlichen Datentyp handelt
    - In Python 3 existiert weder das u-Literal für Unicode-Strings noch der Datentyp unicode
    - Die Datentypen bytes und str sind in Python 3 klarer voneinander abgegrenzt, als es bei den Datentypen str und unicode in Python 2 der Fall ist

Python
Von 2.6 nach 3.x

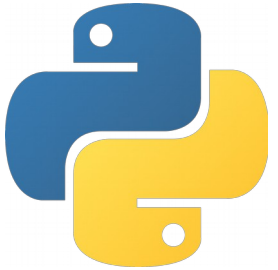


- Die wichtigsten Unterschiede
  - Strings
    - So ist es beispielsweise nicht mehr möglich, einen String und einen bytes-String ohne explizites Kodieren bzw. Dekodieren zusammenzufügen

Python 2	Python 3

`>>> u = u"Ich bin Unicode"`
`>>> u`
`u'Ich bin Unicode'`
`>>> u.encode("ascii")`
`'Ich bin Unicode'`
	`>>> u = "Ich bin Unicode"`
	`>>> u`
	`'Ich bin Unicode'`
	`>>> u.encode("ascii")`
	`b'Ich bin Unicode'`

Python
Von 2.6 nach 3.x



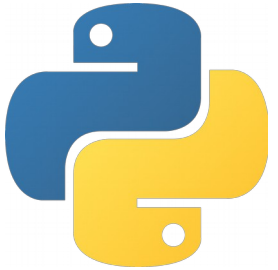
- Die wichtigsten Unterschiede
  - Strings

Python 2	Python 3

`>>> a = "Ich bin ASCII"`
`>>> a.decode()`
`u'Ich bin ASCII'`
	`>>> a = b"Ich bin ASCII"`
	`>>> a.decode()`
	`'Ich bin ASCII'`

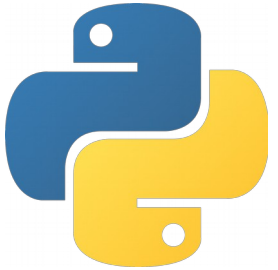
`>>> "abc" + u"def"`
`u'abcdef'`
	`>>> b"abc" + "def"`
	`Traceback (most recent call last):`
	`[...]`
	`TypeError: can't concat bytes to str`

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Strings
    - Die stärkere Abgrenzung von str und bytes in Python 3 hat Auswirkungen auf die Standardbibliothek
    - So dürfen Sie beispielsweise zur Netzwerkkommunikation nur bytes-Strings verwenden
    - Wichtig ist auch, dass der Typ der aus einer Datei eingelesenen Daten nun vom Modus abhängt, in dem die Datei geöffnet wurde
    - Der Unterschied zwischen Binär- und Textmodus ist in Python 3 also auch unter Betriebssystemen von Interesse, die diese beiden Modi von sich aus gar nicht unterscheiden

Python
Von 2.6 nach 3.x



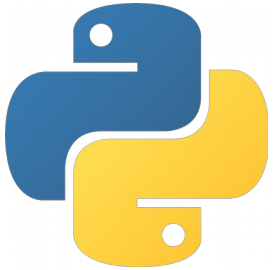
- Die wichtigsten Unterschiede
  - Ganze Zahlen

    - In Python 2 existieren zwei Datentypen zum Speichern von ganzen Zahlen
      - int für Zahlen im 32- bzw. 64-Bit-Zahlenbereich
      - long für Zahlen beliebiger Größe

    - In Python 3 gibt es nur noch einen solchen Datentyp namens int, der sich aber wie long aus Python 2 verhält

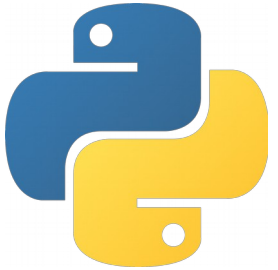
    - Die Unterscheidung zwischen int und long ist auch in Python 2 für den Programmierer im Wesentlichen schon uninteressant, da die beiden Datentypen automatisch ineinander konvertiert werden

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Ganze Zahlen
    - Eine zweite Änderung erfolgt in Bezug auf die Division ganzer Zahlen
    - In Python 2 wird in diesem Fall eine ganzzahlige Division (Integer-Division) durchgeführt, das Ergebnis ist also wieder eine ganze Zahl
    - In Python 3 ist das Ergebnis der Division zweier Ganzzahlen eine Gleitkommazahl
    - Für die Integer-Division existiert hier der Operator //

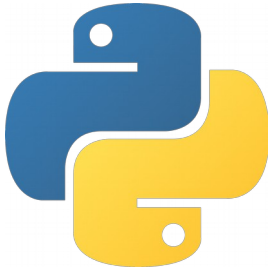
Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Ganze Zahlen

Python 2	Python 3
`>>> 10 / 4` `2`	`>>> 10 / 4` `2.5`
`>>> 10 // 4` `2`	`>>> 10 // 4` `2`
`>>> 10.0 / 4` `2.5`	`>>> 10.0 / 4` `2.5`

Python
Von 2.6 nach 3.x

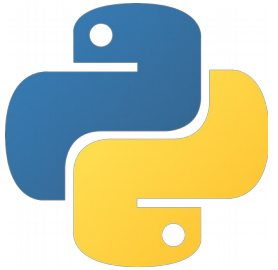


- Die wichtigsten Unterschiede
  - Exception Handling
    - Beim Werfen und Fangen von Exceptions wurden kleinere syntaktische Änderungen durchgeführt
    - Die alte und die neue Syntax werden in folgender Tabelle anhand eines Beispiels einander gegenübergestellt

Python 2	Python 3

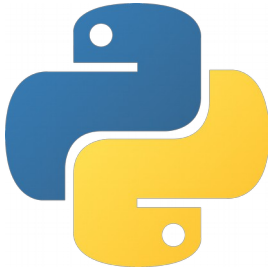
`try:`
`    raise SyntaxError, "Hilfe"`
`except SyntaxError, e:`
`    print e.args`
	`try:`
	`  raise SyntaxError("Hilfe")`
	`except SyntaxError as e:`
	`    print(e.args)`

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Standardbibliothek
    - Mit Python 3 wurde auch in der Standardbibliothek gründlich aufgeräumt
    - Viele Module, die kaum verwendet wurden, sind entfernt worden, andere umbenannt oder mit anderen zu Paketen zusammengefasst

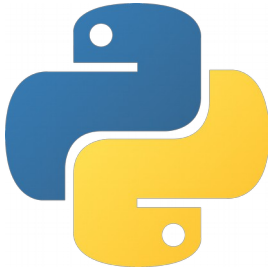
Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Standardbibliothek
    - Unbenannte Module

Python 2	Python 3
_winreg	*winreg*
ConfigParser	*configparser*
Copy_reg	*copyreg*
cPickle	*_pickle*
Queue	*queue*

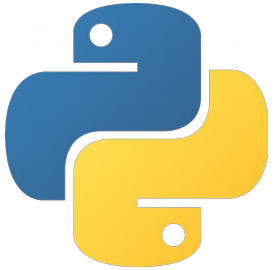
Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Standardbibliothek
    - Unbenannte Module

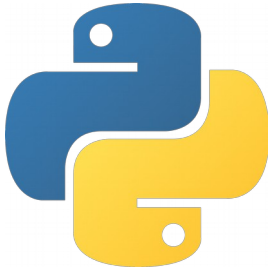
Python 2	Python 3
SocketServer	*socketserver*
markupbase	*_markupbase*
repr	*reprlib*
test.test_support	*test.support*
thread	*_thread*

Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Standardbibliothek
    - Unbenannte Module
      - Die meisten der oben aufgeführten Module wurden nicht thematisiert, da sie sehr speziell sind
      - Nähere Informationen zu ihnen finden Sie aber in der Online-Dokumentation von Python
      - Neben umbenannten Modulen wurden auch einige thematisch zusammengehörige Module zu Paketen zusammengefasst

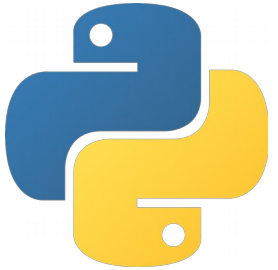
Python
Von 2.6 nach 3.x



- Die wichtigsten Unterschiede
  - Standardbibliothek
    - Verschobene Module

Paket in Python 3	Module aus Python 2
dbm	*anydbm, dbhash, dbm, dumbdbm, gdbm, whichdb*
html	*HTMLParser, htmlentitydefs*
http	*httplib, BaseHTTPServer, CGIHTTPServer, SimpleHTTPServer, Cookie, cookielib*
tkinter	abgesehen von *turtle* alle Module, die etwas mit Tkinter zu tun haben
urllib	*urllib, urllib2, urlparse, robotparse*
xmlrpc	*xmlrpclib, DocXMLRPCServer, SimpleXMLRPCServer*

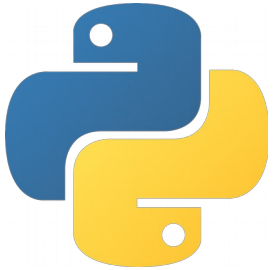
Python
Von 2.6 nach 3.x



- Neue Sprachelemente in Python 3
  - Mit Python 3 wurden eine Reihe neuer Sprachelemente in die Sprache aufgenommen, die wir in der folgenden Tabelle kurz auflisten möchten

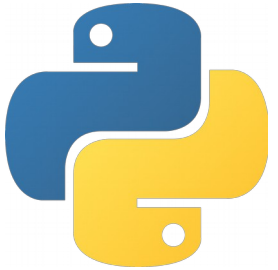
Sprachelement
Literal für Oktal- und Binärzahlen
Literal für Mengen
`nonlocal`-Anweisung
Dict- und Set Comprehensions
`with`-Anweisung
Function Annotations

Python
Von 2.6 nach 3.x



- Automatische Konvertierung
  - Um die Migration von Python 2 nach Python 3 auch bei größeren Projekten zu vereinfachen, gibt es in der Python 3-Distribution ein Tool namens 2to3
  - Das Tool 2to3 finden Sie im Unterverzeichnis Tools/scripts Ihrer Python-Distribution
  - Die Verwendung von 2to3 soll exemplarisch an folgendem Python 2-Beispielprogramm auf der nächsten Folie demonstriert werden

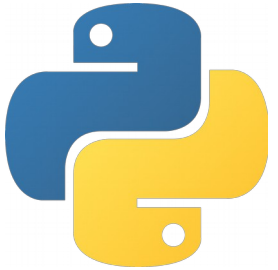
Python
Von 2.6 nach 3.x



- Automatische Konvertierung

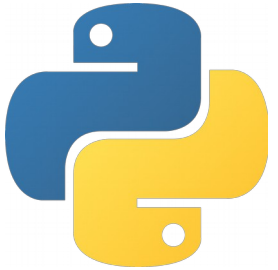
```
def getInput(n):
    liste = []
    for i in xrange(n):
        try:
            z = int(raw_input("Bitte eine Zahl eingeben: "))
        except ValueError, e:
            raise ValueError("Das ist keine Zahl!")
        liste.append(z)
        return liste
try:
    res = getInput(5)
    print res
except ValueError, e:
    print e.args[0]
```

Python
Von 2.6 nach 3.x



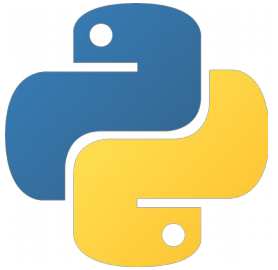
- Automatische Konvertierung
  - Dieses Programm liest mithilfe der Funktion getInput fünf Zahlen vom Benutzer ein und gibt eine mit diesen Zahlen gefüllte Liste aus
  - Wenn der Benutzer etwas eingibt, was keine Zahl ist, beendet sich das Programm mit einer Fehlermeldung
  - Sie sehen sofort, dass sich dieses Programm so nicht unter Python 3 ausführen lässt
  - Die Aufrufe von xrange, raw_input sowie die beiden except-Anweisungen verhindern dies

Python
Von 2.6 nach 3.x



- Automatische Konvertierung
  - Bereits bei den obigen 14 Quellcodezeilen ist es mühselig, den Code per Hand mit Python 3 kompatibel zu machen
  - Stellen Sie sich diese Arbeit einmal für ein größeres Projekt vor
  - Wir rufen 2to3 einmal mit dem Namen unseres Python-Programms als einzigem Parameter auf
  - Das Ergebnis sieht folgendermaßen auf der nächsten Folie aus

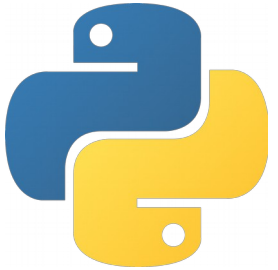
Python
Von 2.6 nach 3.x



- Automatische Konvertierung

```
--- test.py (original)
+++ test.py (refactored)
@@ -1,15 +1,15
  def getInput(n):
    liste = []
-   for i in xrange(n):
+   for i in range(n):
        try:
-           z = int(raw_input("Bitte eine Zahl eingeben: "))
-       except ValueError, e:
+           z = int(input("Bitte eine Zahl eingeben: "))
+       except ValueError as e:
            raise ValueError("Das ist keine Zahl!")
        liste.append(z)
    return liste
```

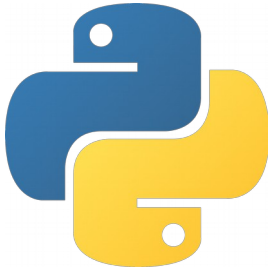
Python
Von 2.6 nach 3.x



- Automatische Konvertierung

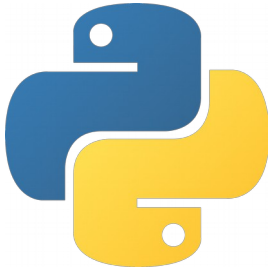
```
try:
    res = getInput(5)
-   print res
-except ValueError, e:
-   print e.args[0]
+   print(res)
+except ValueError as e:
+   print(e.args[0])
```

Python
Von 2.6 nach 3.x



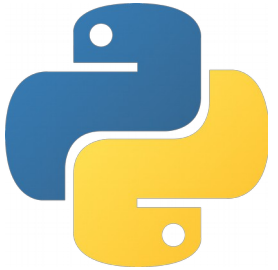
- Automatische Konvertierung
  - Das Konvertierungsprogramm ändert Ihre angegebenen Quellcodedateien standardmäßig nicht, sondern produziert nur einen diff-Ausdruck
  - Das ist eine spezielle Beschreibungssprache für die Unterschiede zwischen zwei Textstücken
  - Diesen diff-Ausdruck können Sie beispielsweise mithilfe des Unix-Programms patch in Ihre Quelldatei einpflegen
  - Alternativ erlauben Sie es dem 2to3-Script über den Kommandoschalter -w, die angegebene Quelldatei direkt zu modifizieren

Python
Von 2.6 nach 3.x



- Automatische Konvertierung
  - Der ursprüngliche Python 2-Code wird dabei als dateiname.py.bak gesichert
  - Wenn 2to3 mit dem Schalter -w und unserem obigen Beispielquellcode gefüttert wird, sieht der konvertierte Code hinterher so aus wie auf der folgenden Folie

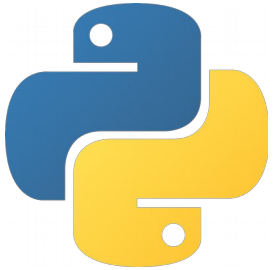
Python
Von 2.6 nach 3.x



- Automatische Konvertierung

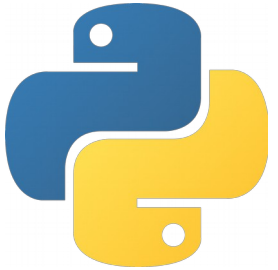
```
def getInput(n):
    liste = []
    for i in range(n):
        try:
            z = int(eval(input("Bitte eine Zahl eingeben: ")))
        except ValueError as e:
            raise ValueError("Das ist keine Zahl!")
        liste.append(z)
    return liste

try:
    res = getInput(5)
    print(res)
except ValueError as e:
    print(e.args[0])
```

Python
Von 2.6 nach 3.x



- Automatische Konvertierung
  - Sie sehen, dass die eingangs angesprochenen Stellen geändert wurden, und werden feststellen, dass der übersetzte Code unter Python 3 lauffähig ist
  - Statt einer einzelnen Programmdatei können Sie dem 2to3-Script auch eine Liste von Dateien oder Ordnern übergeben
  - Wenn Sie einen Ordner übergeben haben, wird jede Quelldatei in ihm oder einem seiner Unterordner konvertiert
  - Zum Schluss möchten wir noch auf die wichtigsten Kommandozeilenschalter zu sprechen kommen, mit deren Hilfe Sie das Verhalten von 2to3 an Ihre Bedürfnisse anpassen können (nächste Folie)

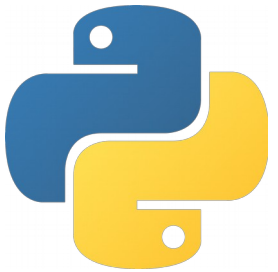
Python
Von 2.6 nach 3.x



- Automatische Konvertierung

Schalter	Alternativ	Beschreibung
`-d`	`--doctests_only`	Ist dieser Schalter gesetzt, so werden ausschließlich die in der angegebenen Quelldatei enthaltenen Doctests nach Python 3 konvertiert. Standardmäßig werden Doctests nicht angerührt. Näheres über Doctests erfahren Sie in Abschnitt 20.5.1 (S. 713).
`-f FIX`	`--fix=FIX`	Mit dieser Option geben Sie vor, welche sogenannten Fixes angewandt werden sollen. Bei einem Fix handelt es sich um eine bestimmte Ersetzungsregel, beispielsweise das Ersetzen von `xrange` durch `range`.
`-x NOFIX`	`--nofix=NOFIX`	Das Gegenstück zu `-f`. Hier bestimmen Sie, welche Fixes nicht angewandt werden dürfen.

Python
Von 2.6 nach 3.x



- Automatische Konvertierung

Schalter	Alternativ	Beschreibung
`-l`	`--list-fixes`	Durch Setzen dieses Schalters erhalten Sie eine Liste aller verfügbaren Fixes.
`-p`	`--print-function`	Wenn dieser Schalter gesetzt ist, werden `print`-Anweisungen nicht konvertiert. Das ist nützlich, wenn Sie `print` bereits in Python 2.6 wie eine Funktion geschrieben oder den entsprechenden Future Import `print_function` eingebunden haben. Das Programm `2to3` kann nicht von selbst entscheiden, ob es eine `print`-Anweisung mit Klammern versehen muss oder nicht.
`-w`	`--write`	Ist dieser Schalter gesetzt, so werden die Änderungen direkt in die untersuchte Quelldatei geschrieben. Ein Backup wird unter *dateiname.py.bak* angelegt.
`-n`	`--nobackups`	Wenn dieser Schalter gesetzt ist, wird auf das Anlegen der Backup-Datei verzichtet.