

Migration auf Python 3

Rainer Grimm

Training, Coaching und
Technologieberatung

www.ModernesCpp.de

Migration auf Python 3

Neue Feature in Python 3

Aufräumarbeiten in Python 3

Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

Migration auf Python 3

Neue Feature in Python 3

Aufräumarbeiten in Python 3

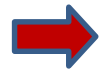
Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

Bedarfsauswertung

Vielen Datenstrukturen in Python 3 werden *lazy*.



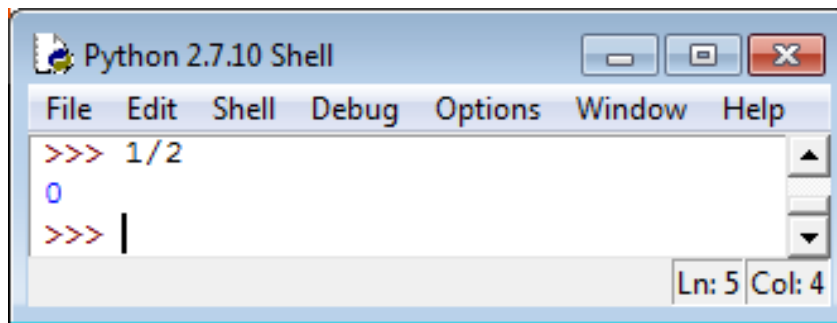
Der Rückgabewert von Listen, Dictionaries, oder den Funktionen `range`, `map` und `filter` ist keine neue Liste, sondern ein Generator.

- *Lazy evaluation* (Bedarfsauswertung) spart Zeit und Speicher.
- Ein Generator gibt auf Anfrage einen neuen Wert zurück und kann unendlich sein.
- Wird ein Generator in eine Liste verpackt, erzeugt er alle Werte.
 - `list(range(1, 7)) == [1, 2, 3, 4, 5, 6]`

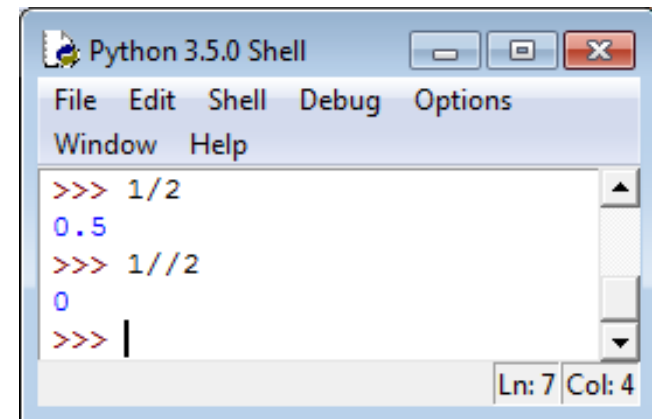
True Division versus Floor Division

Python 3 unterstützt:

- True Division: $1/2 == 0.5$
- Floor Division: $1//2 == 0$



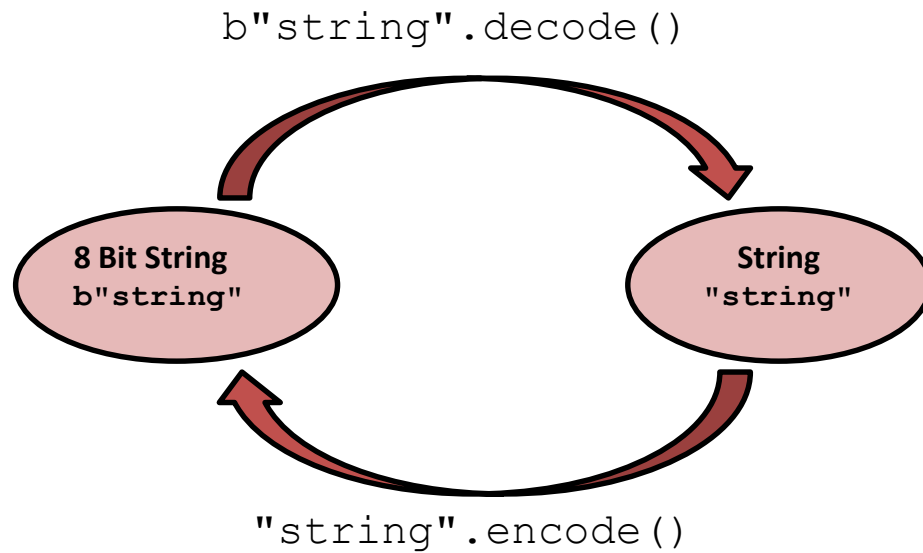
A screenshot of a Python 2.7.10 Shell window. The window title is "Python 2.7.10 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The command prompt shows the input `>>> 1/2` followed by the output `0`. The cursor is on the next line. The status bar at the bottom right shows "Ln: 5 Col: 4".



A screenshot of a Python 3.5.0 Shell window. The window title is "Python 3.5.0 Shell". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The command prompt shows the input `>>> 1/2` followed by the output `0.5`. The next line shows the input `>>> 1//2` followed by the output `0`. The cursor is on the next line. The status bar at the bottom right shows "Ln: 7 Col: 4".

Unicode

Typ	Python 2	Python 3
8 Bit String	<code>"string"</code>	<code>b"string"</code>
Unicode String	<code>u"string"</code>	<code>"string"</code>



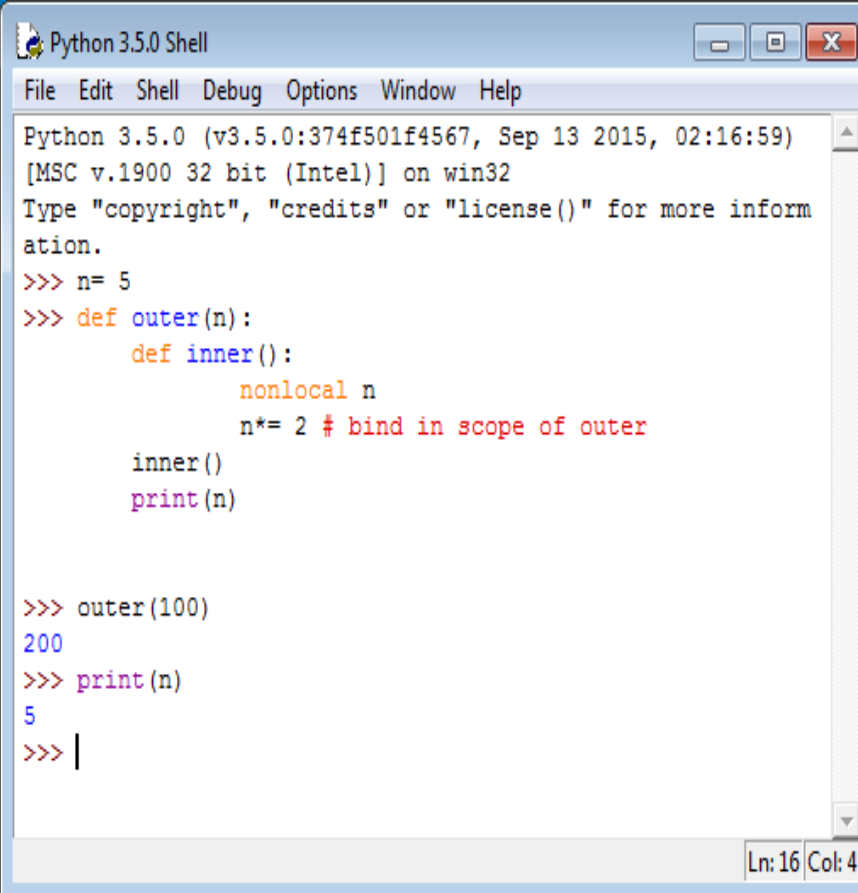
Das Unicode Sandwich



- **bytes** → **str** (Eingabe)
- **str** (Datenverarbeitung)
- **str** → **bytes** (Ausgabe)

nonlocal

`nonlocal` erlaubt es, auf den umgebenden Bereich schreibend zuzugreifen.



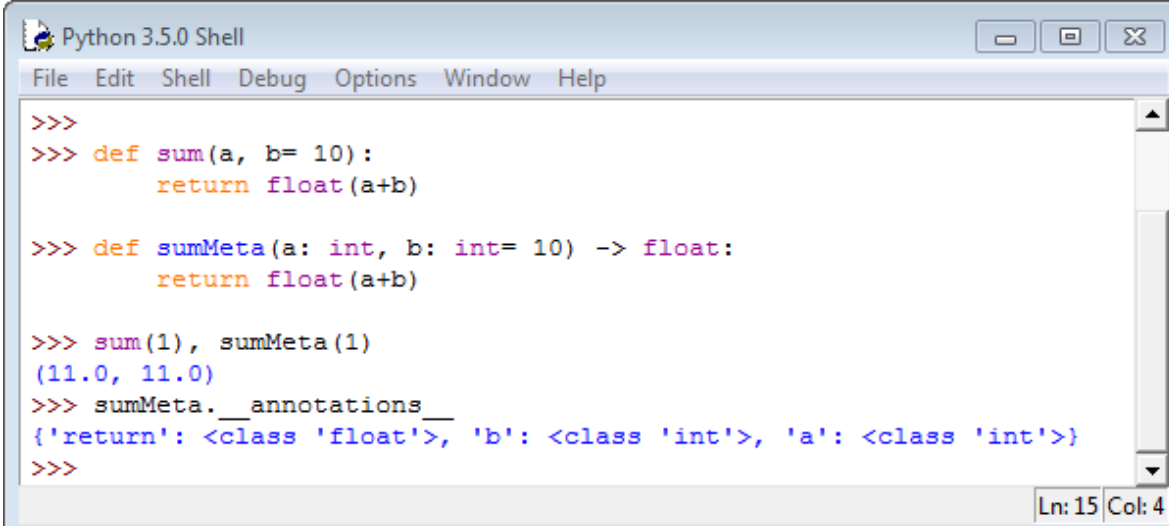
```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59)
[MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> n= 5
>>> def outer(n):
    def inner():
        nonlocal n
        n*= 2 # bind in scope of outer
    inner()
    print(n)

>>> outer(100)
200
>>> print(n)
5
>>> |
```

Ln: 16 Col: 4

Annotationen an Funktionen

- Annotation an Funktionen erlauben es, Metadaten an Funktionen zu binden.
- Diese Metadaten können zur
 - Dokumentation oder zur
 - Typprüfung verwendet werden.
- Die Metadaten lassen sich mit dem Attribut `__annotations__` ansprechen.



```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
>>>
>>> def sum(a, b= 10):
>>>     return float(a+b)
>>>
>>> def sumMeta(a: int, b: int= 10) -> float:
>>>     return float(a+b)
>>>
>>> sum(1), sumMeta(1)
(11.0, 11.0)
>>> sumMeta.__annotations__
{'return': <class 'float'>, 'b': <class 'int'>, 'a': <class 'int'>}
>>>
```

Ln: 15 Col: 4

Migration auf Python 3

Neue Feature in Python 3

Aufräumarbeiten in Python 3

Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

old style Klassen entfernt

- Python 2 unterstützt
 - old style Klassen (Default)
 - new style Klassen
- Python 3 unterstützt
 - new style Klassen

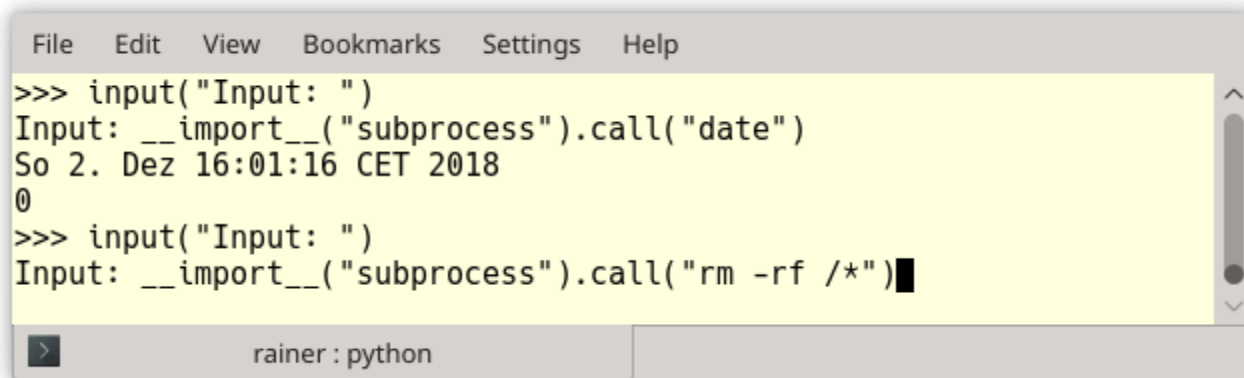
```
class OldStyle:  
    def getName(self):  
        ...
```

```
class NewStyle(object):  
    def getName(self):  
        ...
```

new style Klassen müssen in Python 2 explizit angefordert werden  Ableiten von `object`

input und raw_input

Funktion	Python 2	Python 3
raw_input	raw_input	input
input	input	REMOVED



```
File Edit View Bookmarks Settings Help
>>> input("Input: ")
Input: __import__("subprocess").call("date")
So 2. Dez 16:01:16 CET 2018
0
>>> input("Input: ")
Input: __import__("subprocess").call("rm -rf /*")
```

rainer : python

Vergleiche

- Python 2 unterstützt *comparison* und *rich comparison*.

- Comparison**

- Die interne Funktion `__cmp__` wird für alle Vergleiche verwendet.
- Rückgabewerte von `__cmp__(a, b)`:

```
a < b : -1  
a == b: 0  
a > b : 1
```

- Rich Comparison**

Operator	Interne Funktion
<	<code>__lt__</code>
>	<code>__gt__</code>
<=	<code>__le__</code>
>=	<code>__ge__</code>
==	<code>__eq__</code>
!=, <>	<code>__ne__</code>

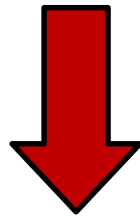
Python 3



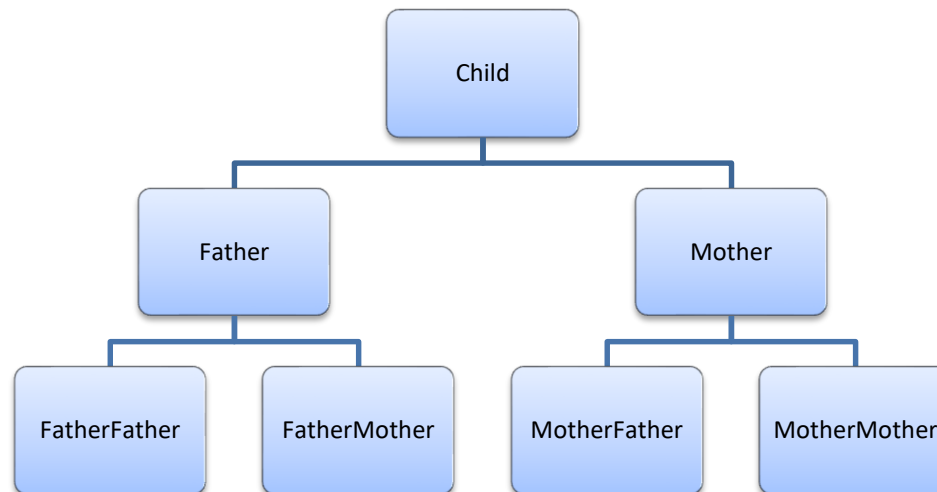
- Entfernt den <> Operator
- Unterstützt nur noch rich comparison

Methodensuchpfad

Der Methodensuchpfad (Method Resolution Order MRO) folgt in Python 2 der Strategie: Tiefensuche.



Der Methodensuchpfad folgt in Python 3 der Strategie: Breitensuche.



print ist eine Funktion

Die Anweisung `print` wird in Python 3 eine Funktion.

- **Syntax:** `print(*args, sep = ' ', end = "\n", file = sys.stdout)`

Beispiele	Python 2	Python 3
Allgemeine Form	<code>print "x = ",5</code>	<code>print("x = ", 5)</code>
Zeilenumbruch	<code>print</code>	<code>print()</code>
Unterdrücken des Zeilenumbruchs	<code>print x,</code>	<code>print(x, end = "")</code>
Unterdrücken des Zeilenumbruchs ohne Leerzeichen		<code>print(1, 2, 3, 4, 5, sep = "")</code>
Umleitung der Ausgabe	<code>print >> sys.stderr, "err"</code>	<code>print("err", file = sys.stderr)</code>

print ist eine Funktion

Die `print` Funktion kann mit Python 3 überladen werden.

```
import sys
```

```
def print(*args, sep=' ', end="\n", file = sys.stdout):  
    __builtins__.print(*args, sep = sep, end = end, file = file)  
    __builtins__.print(*args, sep = sep, end = end, file = open("log.file", "a"))
```


Migration auf Python 3

Neue Feature in Python 3

Aufräumarbeiten in Python 3

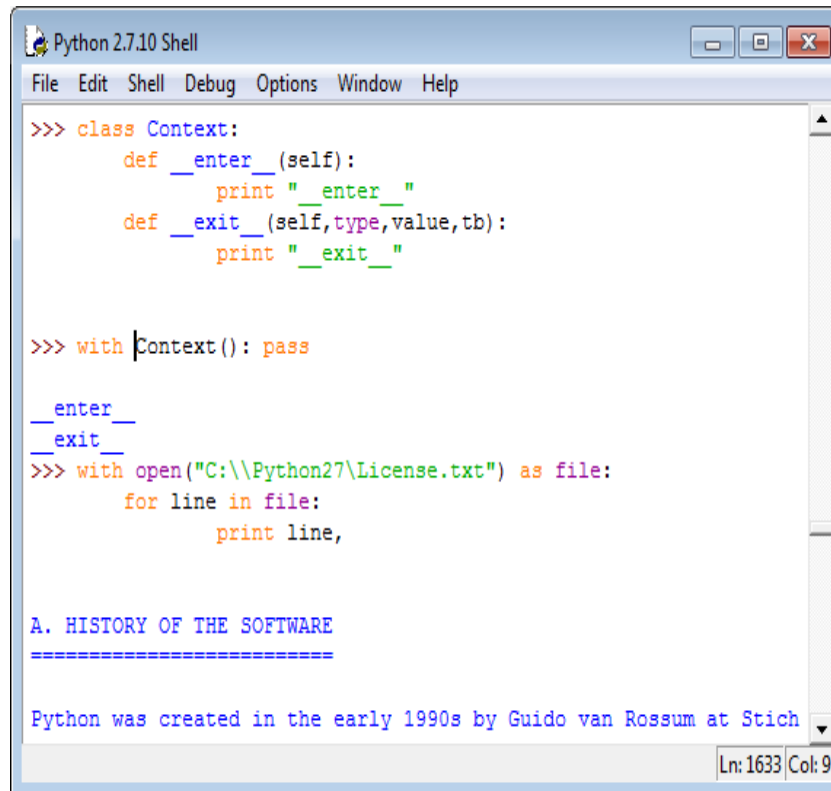
Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

Kontext-Manager

`__enter__` und `__exit__` werden beim Ein- und Austritt in Kontexte automatisch aufgerufen.



```
Python 2.7.10 Shell
File Edit Shell Debug Options Window Help

>>> class Context:
    def __enter__(self):
        print "__enter__"
    def __exit__(self, type, value, tb):
        print "__exit__"

>>> with Context(): pass

__enter__
__exit__
>>> with open("C:\\Python27\\License.txt") as file:
    for line in file:
        print line,

A. HISTORY OF THE SOFTWARE
=====

Python was created in the early 1990s by Guido van Rossum at Stichting

Ln: 1633 Col: 9
```

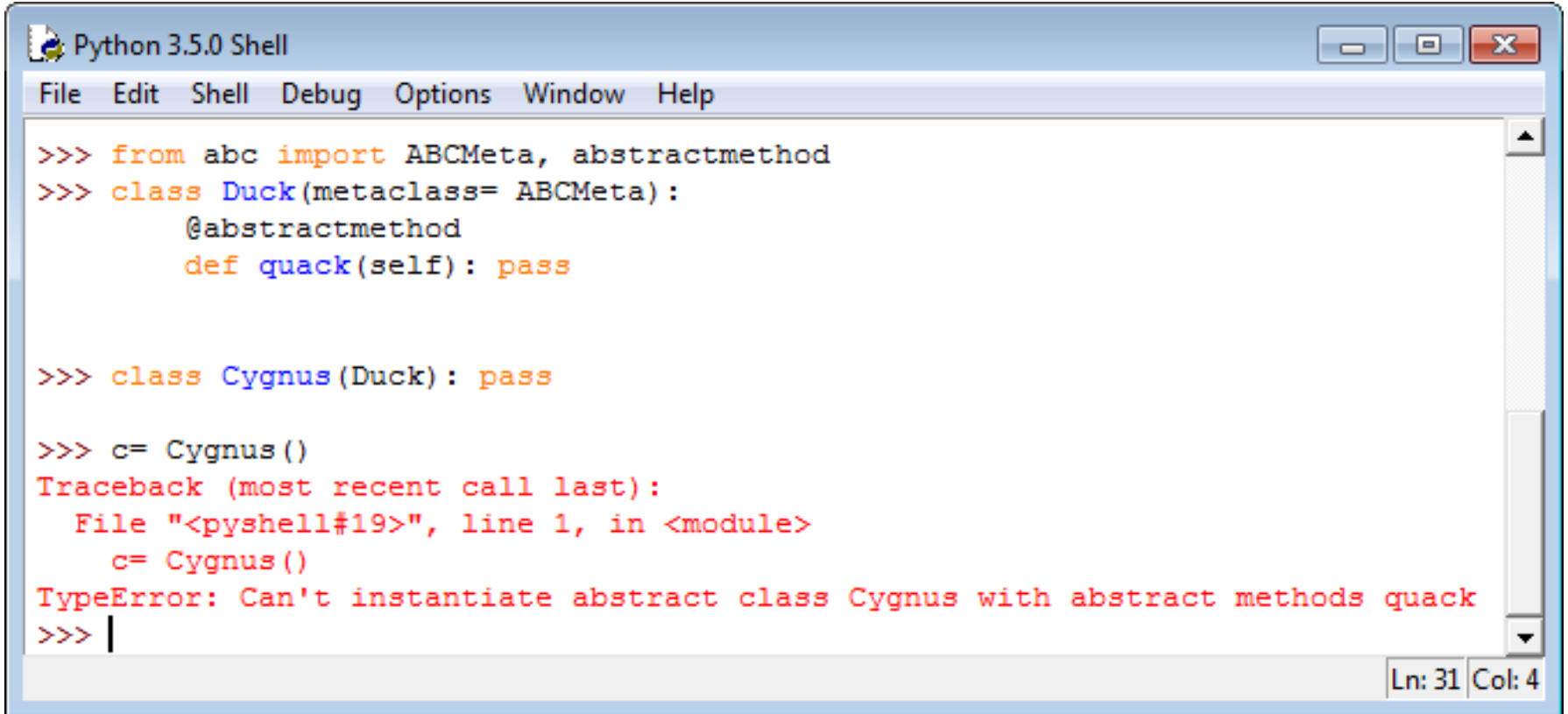
Abstrakte Basisklassen

Python 3 erlaubt die Deklaration von abstrakten Methoden.

➔ Dadurch werden die Klassen zu abstrakten Klassen und können nicht instanziiert werden.

- Eine Methode einer Klasse wird zu abstrakten Methode, indem die
 - die Klasse von der Metaklasse `ABCMeta` abgeleitet ist und
 - die Methode den Dekorator `abstractmethod` besitzt.

Abstrakte Basisklassen



```
Python 3.5.0 Shell
File Edit Shell Debug Options Window Help
>>> from abc import ABCMeta, abstractmethod
>>> class Duck(metaclass=ABCMeta):
    @abstractmethod
    def quack(self): pass

>>> class Cygnus(Duck): pass

>>> c= Cygnus()
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    c= Cygnus()
TypeError: Can't instantiate abstract class Cygnus with abstract methods quack
>>> |
```

Ln: 31 Col: 4

Multiprocessing Bibliothek

Die neue `multiprocessing` Bibliothek in Python

- besitzt das gleiche Interface wie die `threading` Bibliothek.

```
import multiprocessing as threading
```

- startet plattformunabhängig einen Prozess statt einem Thread.
- hilft das Global Interpreter Lock (GIL) in Python zu lösen.



Verwenden Sie die `multiprocessing` Bibliothek.

Migration auf Python 3

Neue Feature in Python 3

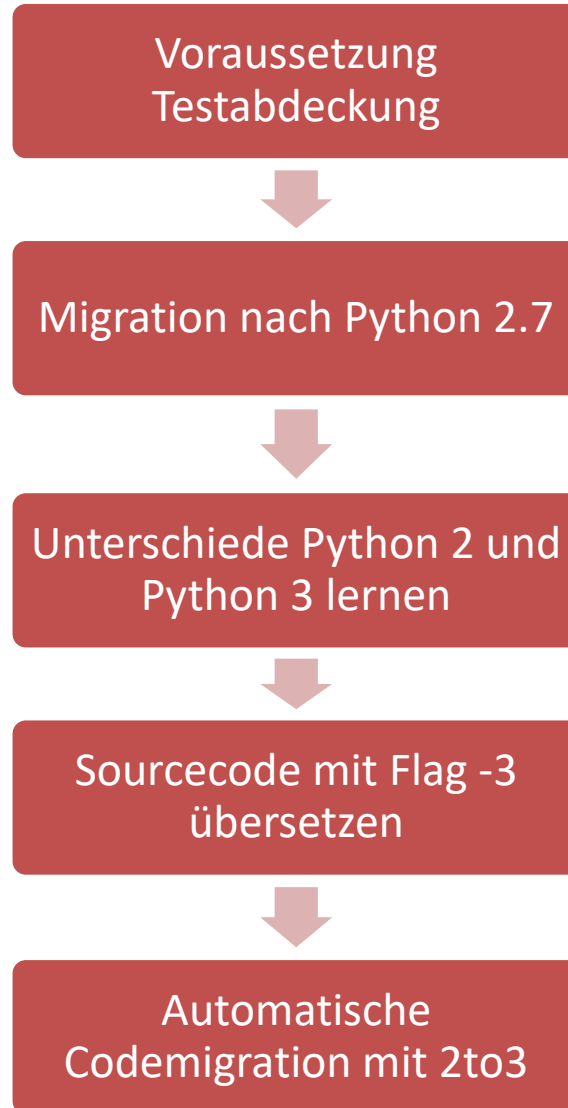
Aufräumarbeiten in Python 3

Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

Vollkommener Umstieg auf Python 3



Voraussetzung Testabdeckung

- [doctest](#) – Interaktives Testen von Programmen
 - Tests werden in den docstrings formuliert
 - Ist bekannt als *literate testing* oder *executable documentation* bekannt

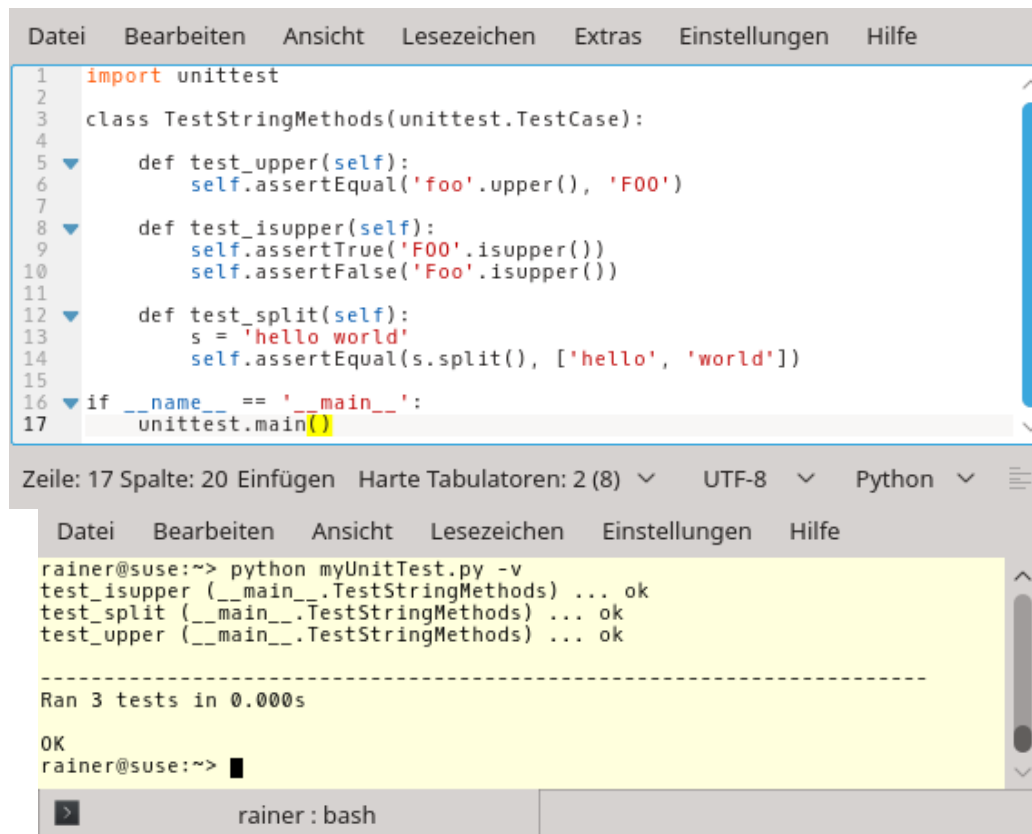


```
1 def factorial(n):
2     """
3     >>> [factorial(n) for n in range(6)]
4     [1, 1, 2, 6, 24, 120]
5     """
6     if ( n==0 ): return 1
7     else: return n*factorial(n-1)
8
9 if __name__ == "__main__":
10 import doctest
11 doctest.testmod()
```

```
rainer@suse:~> python2 myDoc.py -v
Trying:
  [factorial(n) for n in range(6)]
Expecting:
  [1, 1, 2, 6, 24, 120]
ok
1 items had no tests:
  __main__
1 items passed all tests:
  1 tests in __main__.factorial
1 tests in 2 items.
1 passed and 0 failed.
Test passed.
rainer@suse:~>
```


Voraussetzung Testabdeckung

- [unittest](#) – Das Unit Test Framework in Python
 - Wird gerne PyUnit genannt
 - Basiert auf dem [JUnit](#) Framework von Kent Beck und Eric Gamma



```
1 import unittest
2
3 class TestStringMethods(unittest.TestCase):
4
5     def test_upper(self):
6         self.assertEqual('foo'.upper(), 'FOO')
7
8     def test_isupper(self):
9         self.assertTrue('FOO'.isupper())
10        self.assertFalse('Foo'.isupper())
11
12    def test_split(self):
13        s = 'hello world'
14        self.assertEqual(s.split(), ['hello', 'world'])
15
16 if __name__ == '__main__':
17     unittest.main()
```

```
rainer@suse:~> python myUnitTest.py -v
test_isupper (__main__.TestStringMethods) ... ok
test_split (__main__.TestStringMethods) ... ok
test_upper (__main__.TestStringMethods) ... ok

-----
Ran 3 tests in 0.000s

OK
rainer@suse:~>
```

rainer : bash

Migration nach Python 2.7

Python 2.5 und älter:

- Starke Code Modifikationen sind eventuell notwendig.

Python 2.6:

- Wird nicht mehr offiziell unterstützt.
- Werkzeuge wie `pylint` stehen nicht zur Verfügung.

Python 2.7:

- Wird offiziell unterstützt.
- Migration ist am einfachsten.

Unterschiede Python 2 und Python 3 lernen

Veränderte Semantik

- Bedarfsauswertung
- True Division
- Unicode
- `nonlocal`
- Annotationen an Funktionen

Aufräumarbeiten

- old-style Klassen entfernt
- `input` und `raw_input`
- Vergleiche
- Methodensuchpfad
- `print`

Rückportierung

- Kontext-Manager
- Abstrakte Basisklassen
- Multiprocessing Bibliothek

Sourcecode mit Flag -3 übersetzen

```
python -3
```

- Steht seit Python 2.6 zur Verfügung
- Warnt vor Python 3.x Inkompatibilitäten mit einer Ausnahme vom Typ `DeprecationWarning`
- Schlägt bei entfernten und stark veränderten Features in 3.x an

Automatische Codemigration mit 2to3

- [2to3](#) – Automatische Python Sourcecodetransformation von 2 nach 3
- Features
 - Besitzt viele *Fixers* für die automatische Transformation.
 - Kann mit Dateien und Verzeichnissen (rekursive) umgehen.
 - Kann jede Fixture einzeln ansprechen.
- Anwendung
 - `2to3 example.py:`
 - `diff` Ausgabe wird ausgegeben.
 - `2to3 -w example.py:`
 - Die Datei wird überschrieben.
 - Eine Backupdatei wird erzeugt.
 - `2to3 -w -n example.py:`
 - Die Datei wird überschrieben.
 - Keine Backupdatei wird erzeugt.
 - `2to3 -o python3Code -w -n python2Code:`
 - Schreibt alle Dateien aus dem Verzeichnis `python2Code` nach `python3Code`.

Automatische Codemigration mit 2to3

- Installation

- Ist Umfang jeder Python Installation
- Im Verzeichnis `Tools/scripts`
- Neuere Suse Distributionen ≥ 13.2 besitzen es nur noch im Python3 Dokumentations Verzeichnis:
`doc/packages/Python/Tools/scripts`

- 2to3

```
#!/usr/bin/env python
import sys
from lib2to3.main import main

sys.exit(main("lib2to3.fixes"))
```

Migration auf Python 3

Neue Feature in Python 3

Aufräumarbeiten in Python 3

Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

Automatische Unterstützung von Python 2 und 3

- Automatische Migration des Codes für
 - Python 2.7
 - [futurize](#)
 - [modernize](#) (konservativer als `futurize`)
 - Python 2.6
 - [six](#)

Migration auf Python 3

Neue Feature in Python 3

Aufräumarbeiten in Python 3

Rückportierung von Python 3 Features

Migration auf Python 3

Python 2 und Python 3 unterstützen

Nützliche Werkzeuge

- Was ist neu in Python 3: [Python 3 Q & A](#)
- Testframework: [unittest](#) und [doctest](#)
- Testabdeckung: [coverage](#)

- Automatische Codemigration auf Python 3: [2to3](#)

- Migration auf Python 3: [Porting to Python 3](#)

- Unterstützung von Python 2.* und Python 3:
 - Python 2.6 und Python 3: [six](#)
 - Python 2.7 und Python 3: [futurize](#) und [modernize](#)
 - [Cheat Sheet: Writing Python 2-3 compatible code](#)