

Skriptsprachen: Python

Module

Jan Krüger, Alexander Sczyrba

Technische Fakultät
Universität Bielefeld



30. September 2016

Python

—

Module

Modularisierung

- Strukturisierung
- Namensräume
- Module ausführen

import

- Einbinden externer Programmbibliotheken

```
import math  
print math.sin(math.pi)
```

- mehrerer Module
- Änderung des Namensraums

```
import sys as system  
print system.argv
```

import(2)

- import in den globalen Namensraum

```
import math
import *
print sin(pi)
```

- import einzelner Fkt. in den globalen Namensraum

```
from math import sin,pi
```
- Änderung des Namensraum für einzelne Fkt

```
from math import sin as foo, cos as bar
```

ein kleines Modul

Haben wir schon geschrieben :-)

Datei `reverse.py`:

```
def reverse(a):  
    return a[::-1]
```

Modul benutzen:

```
import reverse  
reverse.reverse("Hello_World!")
```

Aufgabe

1. Erstelle ein Modul foo das eine Funktion bar enthält. Die Funktion bar soll 'Hello World' zurückgeben.
2. Erstelle ein Modul <login>.foo das eine Funktion bar enthält. Die Funktion soll 'Hallo Welt' zurückgeben.
3. Probiere jetzt folgendes aus :
 - import in eine interaktive Pythonshell (python oder ipython) im gleichen Verzeichnis ...
 - ... und in einem anderen Verzeichnis
 - import in ein kleines PythonSkript im gleichen Verzeichnis ...
 - ... und in einem anderen Verzeichnis

PythonPath

Python sucht per Default im globalen Python Library (systemabhängig) und im aktuellen Verzeichnis nach Python Modulen. Außerdem gibt es folgende Anpassungsoptionen:

- durch die Systemvariable: PYTHONPATH
- und zur Laufzeit: `sys.path`

```
import sys
```

```
sys.path.append("mein/modul/pfad")
```

Module ausführen

Wenn ein Modul `foo.py` durch `python` direkt ausgeführt wird
`python foo.py`, dann wird die interne Modul Variable `__name__` auf
den Namen `__main__` gesetzt.

```
...  
if __name__ == "__main__":  
    import sys  
    ...
```