Script language: Python
Module

Cédric Saule

Universität Bielefeld

12. Februar 2016

Python

–

Module

Modularization

- Structuring
- Namespace
- Performing module

import

- Using external libraries

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

- Several modules.

```
import sys,random,math
```

- Modification of the namespace

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

import(2)

- Import into the global namespace.

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

- Import individual function in the global namespace.
  `from math import sin,pi`
- Modification of the namespace for an invidual function.
  `from math import sin as foo, cos as bar`

A small module.

Did we write a lot :-) ?

File `reverse.py`:

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

Module use:

```
import reverse
reverse.reverse("Hello␣World!")
```

Exercise

1. Create a module foo which contains a function "bar". The bar function has to return 'Hello World'.
2. Create a module <login>.foo which contains a function "bar". The function has to return 'Hallo Welt'.
3. Try now the following :
   - import an interactive Python shell (python or ipython) in the same directory ...
   - ... and in another directory.
   - import in a small Python script in the same directory ...
   - ... and in another directory.

Python path

Python looks for the default one in the global Python librairy (system dependent) and in the current directory of the Python module. In addition, there are the following customization options:

- Through the system variable: `PYTHONPATH`
- and at runtime: `sys.path`

```
import sys

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

Module list

When a module foo.py is directly executed by python `python foo.py`, then the internal module variable `__name__` is set on the name `__main__`.

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