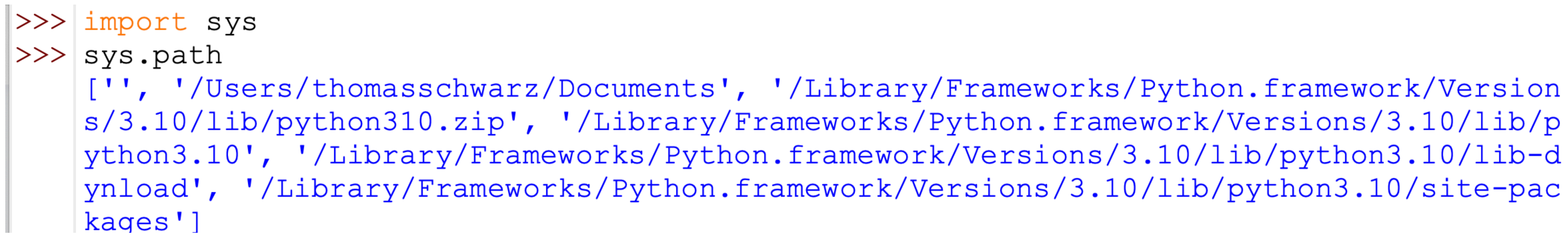
Modules

Thomas Schwarz, SJ

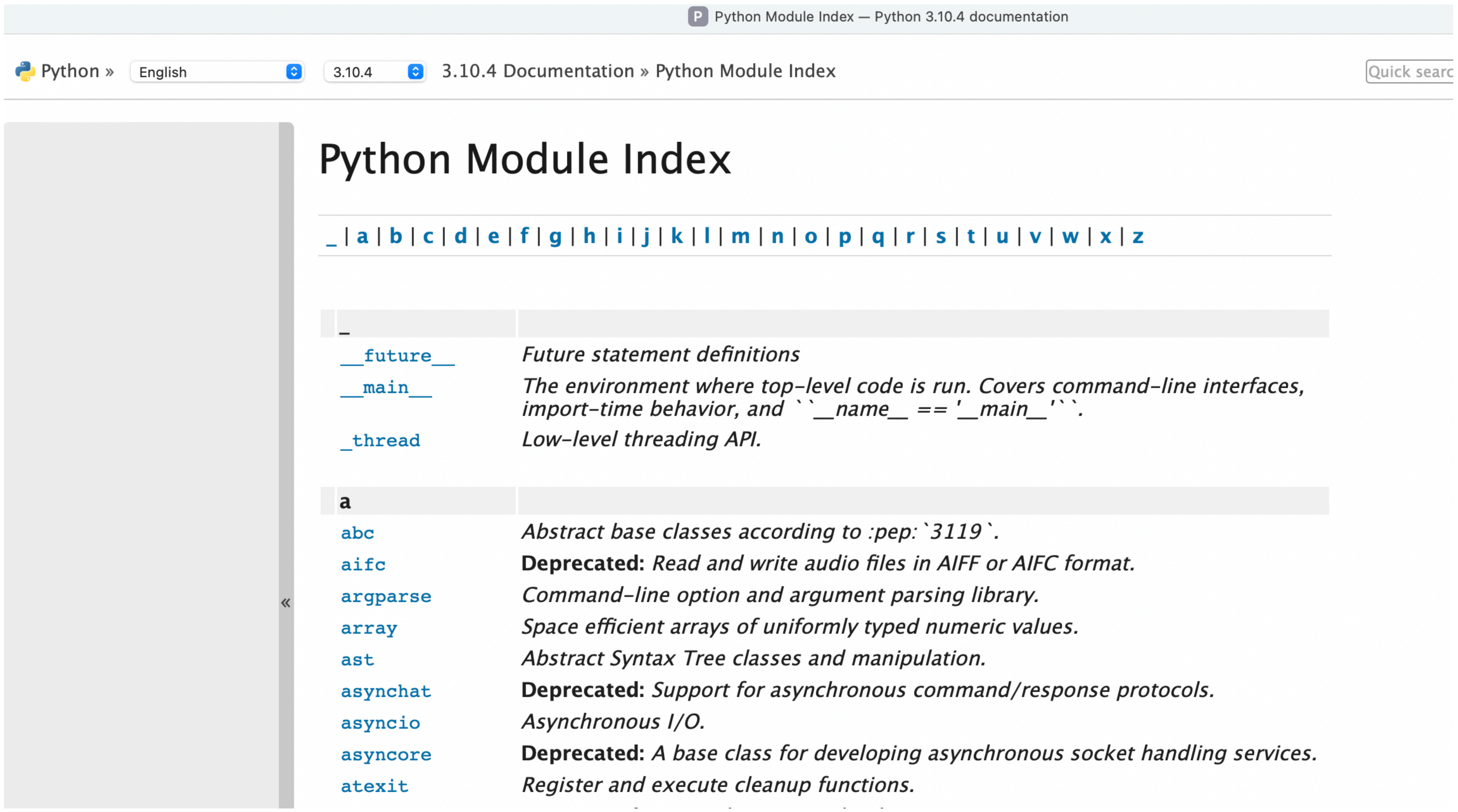
Modules

- A module is just a file with a .py extension
  - It contains definitions for functions, constants, classes, *et cet*.
  - It should be located on the Python path
    - And here is how you find your python path:
      - Import the module sys (the system module)
      - Say sys.path



```
>>> import sys
>>> sys.path
    ['', '/Users/thomasschwarz/Documents', '/Library/Frameworks/Python.framework/Version
    s/3.10/lib/python310.zip', '/Library/Frameworks/Python.framework/Versions/3.10/lib/p
    ython3.10', '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/lib-d
    ynload', '/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-pac
    kages']
```

Interlude: Python Modules

- Predefined modules
  - Python comes with many modules (see the docs)
- 

Python Module Index — Python 3.10.4 documentation

Python » English 3.10.4 3.10.4 Documentation » Python Module Index Quick searc

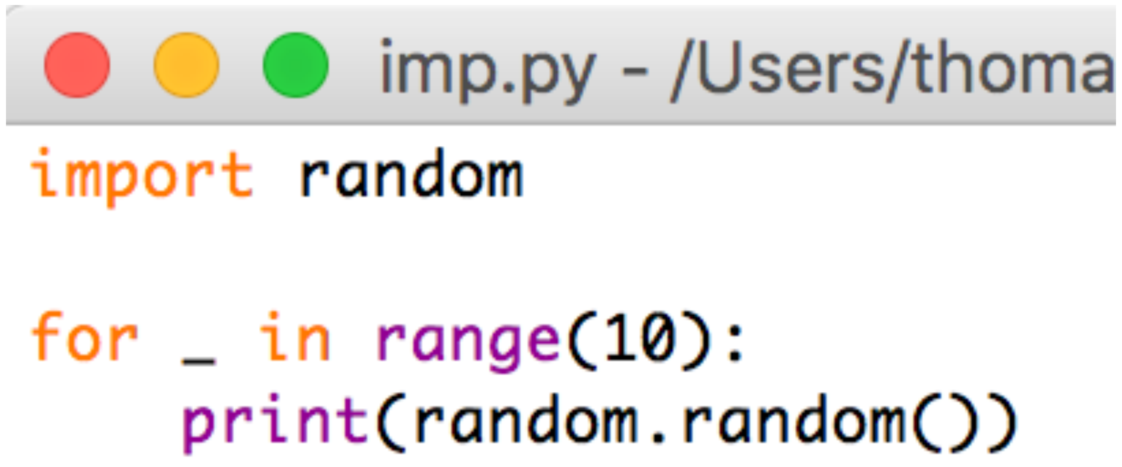
Python Module Index

_ | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | z

_	
`__future__`	*Future statement definitions*
`__main__`	*The environment where top-level code is run. Covers command-line interfaces, import-time behavior, and ``__name__ == '__main__'``.*
`_thread`	*Low-level threading API.*
a	
`abc`	*Abstract base classes according to :pep:`3119`.*
`aifc`	**Deprecated:** *Read and write audio files in AIFF or AIFC format.*
`argparse`	*Command-line option and argument parsing library.*
`array`	*Space efficient arrays of uniformly typed numeric values.*
`ast`	*Abstract Syntax Tree classes and manipulation.*
`asynchat`	**Deprecated:** *Support for asynchronous command/response protocols.*
`asyncio`	*Asynchronous I/O.*
`asyncore`	**Deprecated:** *A base class for developing asynchronous socket handling services.*
`atexit`	*Register and execute cleanup functions.*

Python Modules

- If I just import the module random, then I can use its functions by prefixing "random."
  -



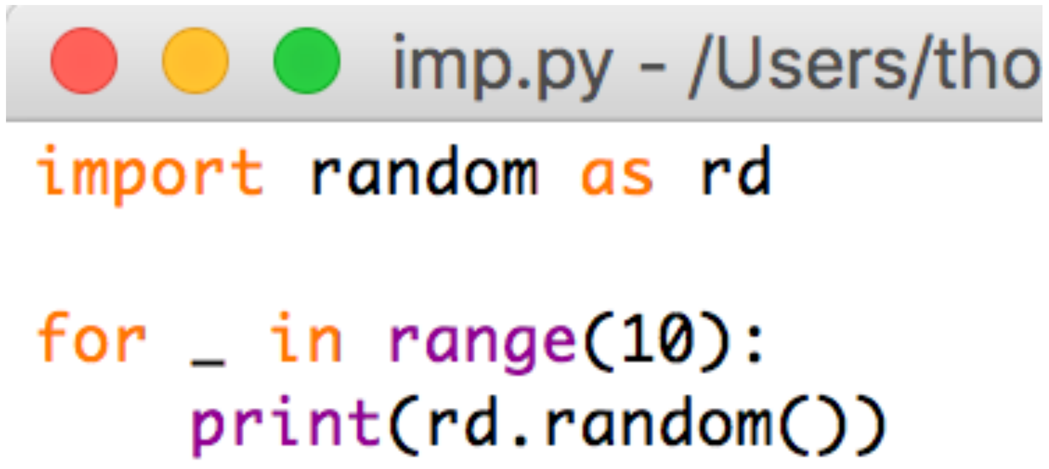
```python
import random

for _ in range(10):
    print(random.random())
```

Using the function random inside the module random

Python Modules

- If I want to avoid writing the module name I can use an "as" clause that redefines the name of the module within the script



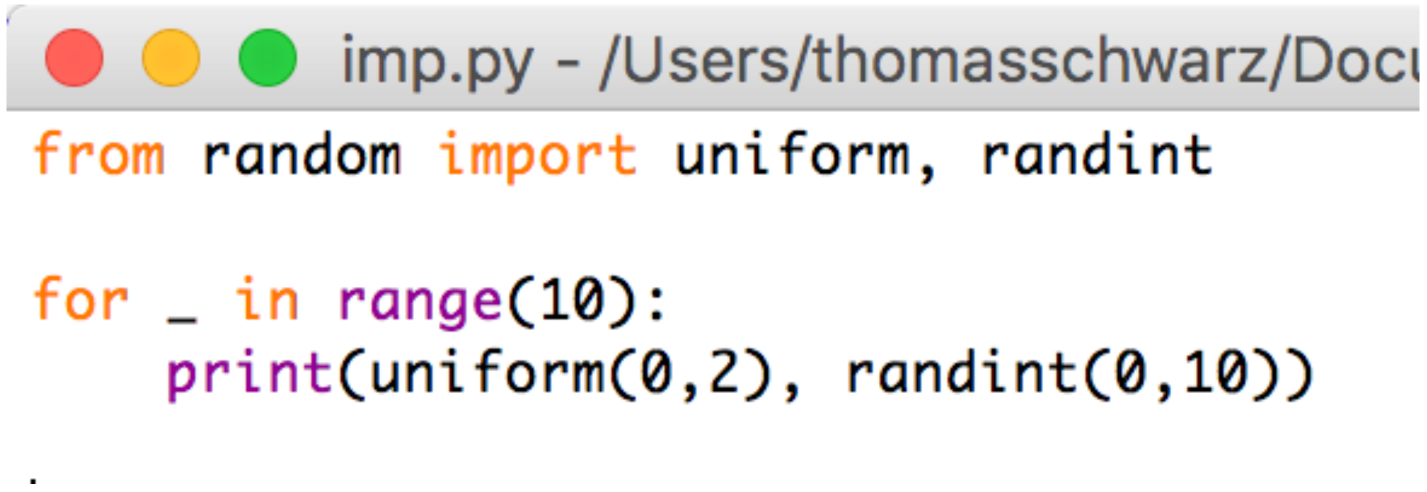
```
import random as rd

for _ in range(10):
    print(rd.random())
```

Using the same function in the same module,
but now after internally renaming the module

Python Modules

- By using the “from — import” clause, I can use variables and functions without repeating the module name



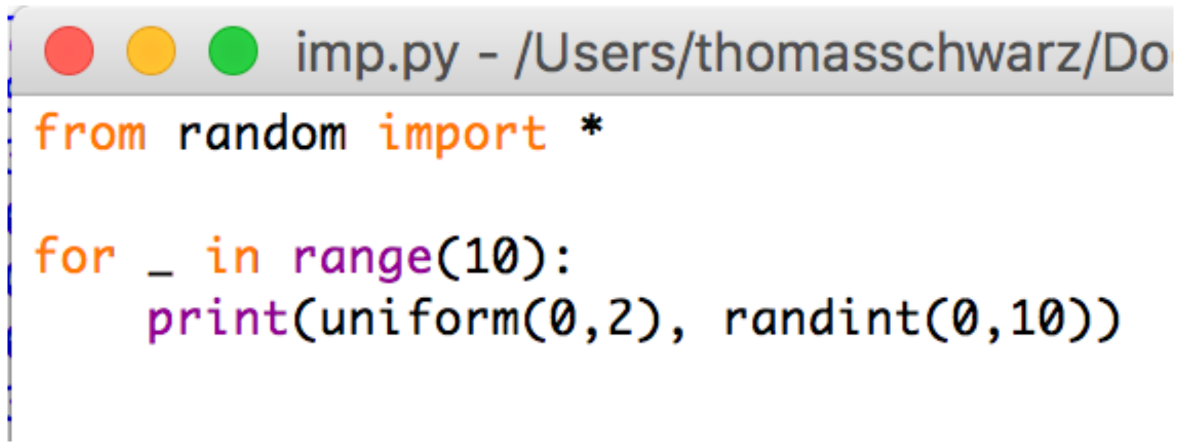
```
from random import uniform, randint

for _ in range(10):
    print(uniform(0,2), randint(0,10))
```

Importing the two functions uniform and randint from the random module.

Python Modules

- I could even import everything from a module
  - But this can create havoc if I defined a function with the same name as a function in the module



```
from random import *

for _ in range(10):
    print(uniform(0,2), randint(0,10))
```

A dangerous practice: Importing all functions from a module

Python Modules

- External modules:
  - Everyone can build and publish python modules
    - And publish it using pip or github
  - This is where a lot of the power of python resides:
    - Can easily find good modules to solve my problems
  - Often, modules are implemented in C and made usable in Python

Python Modules

- To import from pip:
  - go to terminal or command window
    - Type: pip3.10 install the_module_name
      - pip uses your current main python installation, which could be Python 2.7 on MacOS
      - pip3 uses Python 3
      - pip3.10 uses the Python 3.10 installed
    - Installation is usually automatic, but can (rarely) fail
      - E.g.: the module has not been maintained and does not match your architecture