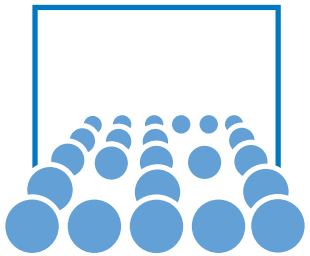
Scripting with Python

Compact Course @ Max-Planck

Tobias Neckel

May 6 - 10, 2019



neckel@in.tum.de

Part I

Basics

What Python is all about

Advantages

- Fast to learn
- Easy to read
- Open source
- Available on most computers

What Python is all about (2)

Suitable for

- OS scripting
- Web programming
- Scientific computing
- Parallelisation
- GUI
- Visualisation
- Rapid prototype development
- . . . and much, much more!

What Python is all about (3)

Note

- We'll be using Python 3.x
- You need to have installed *Anaconda*: `https://www.anaconda.com/distribution/#download-section`

Structure of the Course

Day 1
- Part 1: Basics
- Part 2: Control flow
- Part 3: Functions, modules & packages

Day 2
- Part 3: Functions, modules & packages
- Part 4: Object-oriented programming

Day 3
- Part 5: File IO
- Part 6: Scientific Computing with Python

Day 4
- Part 7: Operating system functions

Day 5
- Part 8: Selected topics

Let's get started...

- Start the Python interpreter
- At prompt >>> type command, press <Enter> to confirm

```
$ python3
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license"
for more information.
>>> help
Type help() for interactive help, or help(object) for
help about object.
>>> help()

Welcome to Python 3.5's help utility!

[...]
```

Getting help

- Vast online documentation at `http://docs.python.org/3/`
- Interactive using
  - `help()` for interactive help,
  - `help(object|'command')` for help on an object or command
- google / forums
- Literature
  - David M. Beasley: *Python - Essential Reference*, Addison-Wesley Professional, 4th edition, 2009
  - Hans Petter Langtangen: *A Primer on Scientific Programming with Python*, Springer, 2009

A first “Hello World!”

Python – simplify your Code

- Just type

```
>>> print("Hello␣World!")
Hello World!
```

- Compare with Java, e.g.:

```
public class Hello{
    public static void main(String argv[]){
        System.out.println("Hello␣World!");
    }
}

$ javac Hello.java
$ java Hello
Hello World!
```

Statements and assignments

```
>>> 3+4**2/8+1
>>> (6*3.5)*2-5
>>> 10e-4
>>> a=17
>>> a
>>> a-1.5
>>> b = "Hello World!"
>>> print(b)
>>> print(a, b, (4+5**0.5))
```

- assignment: variableName = expression (different to math. “=”)
- Interactive session shows return values
- `print` prints variables and expressions as string

3 Ways to Python

1. Start the Python interpreter with `python`
2. Use IPython: `ipython`
3. Run Python on a file (e.g. using Spyder)

2. IPython

```
$ ipython
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
Type "copyright", "credits" or "license" for more
information.

IPython 2.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's
             features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??'
             for extra details.
In [1]: print("Hello World!")
Hello World!
```

2. IPython

Highlights

- Enhanced interactive shell
- Additional shell syntax
  - “magic” functions, starting with “%” such as `%psearch` to search for function in namespace
  - Run scripts with `run filename.py`
  - Log session, define macros, . . .
  - Pretty printing, toggle with `%Pprint`
- Syntax highlighting
- Tab completion, string completion
- History
  - Access previous command blocks
- Automatic indentation
- Pro Tip: Use the `jupyter notebook`

3. Run Python on a File

- File `square_me.py`:

```
print(6**2)
```

- Then run

```
$ python square_me.py
36
```

- Use a nice environment to write and run your programs! We recommend `spyder`.

Built-in Python Types

Numeric Types

- `bool` – boolean type

```
print(True, False)
True == True
True != True
```

- `int` – integer

```
1+5-12
```

- range of integers only limited by memory ⇒ no overflow

```
>>> b = 123**56
>>> print(b)
10831441054800221700495871957152692828292650624493
02603600066970681484857420741062641453509422921381
29480771505554256 1
```

Numeric Types (cont.)

- `float` – floating point (double precision)

```
3.14159265*5.6
1.5e-10
```

- `complex` – complex numbers

```
1+4j
c = complex(2,3)
print(c)
d = 2 * (c**3 - 3j)
print(d)
d.real
d.imag
```

None type

- `None` – special type (“undefined”)

General Remarks

- Everything is an object (int, float, tuple, classes, ...)
- Thus everything can have attributes and methods

```
c = complex(2,3)
print(c.real, c.imag)
```

- Dynamic typing: Type of a variable determined during assignment
- Automatic conversion where necessary and possible

```
a = 1234
type(a)
b = 100
type(b)
a = 1234**100
type(a)
b = b*1.0       # equals b = float(b)*1.0
type(b)
```

Built-in Sequences

Strings

- `string` type
- Sequence of characters
- Single, double or triple double (multiline) quotes

```
"...␣there␣lived␣a␣hobbit."
"a␣'hobbit'"
'a␣"hobbit"'
'a␣\'hobbit\''
"""In a hole in the ground
there "lived" a 'hobbit'"""
```

- Concatenation

```
s1 = "In␣a␣hole␣in␣the␣ground"
s2 = "there␣lived␣a␣hobbit"
s = s1 + '␣' + s2
print(s)
```

Built-in Sequences – Strings (2)

- Replication

```
"Hi!␣"*3
```

- Indexing and Slicing (immutable sequence type)

```
s = "Hello␣Hobbit"
s[3]
s[-2]
s[2:4]
s[6:]
s[:6]
s[0:-1:2] # stride 2
s[0::2]
s[2] = 4  # error
len(s)
```

Built-in Sequences – Strings (3)

- String formatting using %

```
"Bilbo's␣%d.␣birthday" % (111)
"Frodo,␣%s,␣and␣Bilbo␣ate␣%f␣apples" %("Sam",1.5)
```

- Some format modifiers

```
'%05d' % (13) # fill 0
'%-5d' % (13) # left justified
'%+5d' % (13) # sign
'%d' % (13)   # signed integer
'%e' % (13)   # floating point exponential
'%g␣%g,␣%f' % (13, 1e-6, 1e-6)
'%s' % ('hi') # string
'%%'          # escape %
```

Built-in Sequences – Strings (4)

Special string methods (excerpt)

```
s = "␣␣Frodo␣and␣␣Sam␣and␣␣␣Bilbo"
s.islower()
s.isupper()
s.startswith("Frodo") # try s.startswith("Frodo", 2)
s.endswith("Bilbo")
s = s.strip()
s.upper()
s = s.lower()
s.capitalize()          # capitalize first character
s = s.center(len(s)+4)# center (padding default: space)
s.lstrip()
s.rstrip("␣")
s = s.strip()
s.find("sam")
s.rfind("and")
```

Built-in Sequences – Strings (5)

Searching, splitting and joining (excerpt)

```
s.find("sam")
s.rfind("and")
s.replace("and", "or")

s.split()  # or s.split(None); arbitrary numb. whitesp.
s.split("␣")
s.split("and")
s.split("and", 1)

s = """Line
by
Line"""
s.splitlines()

",␣".join(["sequence", "of", "strings"])
```

Built-in Sequences – Tuples

- Sequences of arbitrary objects `(,)`
- Immutable
- Indexing and slicing as before

```
tup = (1,)
tup = (1, "two", (3,4,5))
tup[0]
tup[-1]
tup[1] = 2   # error
len(tup)
min(tup)     # error: unorderable types
tup = ("Hello", "Hobbit")
min(tup)
max(tup)
tup2 = tup + (6,7,8)
```

Built-in Sequences – Lists

- Sequences of arbitrary objects `[,]`
- Mutable

```
lis = [1, "two", (3,4,5), 5]
lis[0]
lis[-1]
lis[1] = 2   # works!
len(lis)
```

- Operations on lists

```
lis = [0,1,2,3,4,5,6,7,8,9]
lis[2:4] = [10, 11]
lis[1:7:2] = [-1, -2, -3]
del lis[::2]
```

Built-in Sequences – Lists (2)

- List methods

```
l = [0,1,2,3]
l.append("four")
l.extend([5,6,7,8,9])
l.insert(8, "four")
l.count("four")
l.sort()
l.reverse()
l.remove("four")
l.pop()
```

- Integer ranges: `range` function

```
range(10)
range(5, 10)
range(-10, 20, 3)
```

Built-in Sequences – Lists (3)

Special integer lists (for loops etc.)

- function `range` creates object of immutable data type
- lazy evaluation: dynamic creation of corresponding values only at access
- real lists: via `list(range(...))`

```
>>> range(5)
range(0, 5)
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> range(-10, 5, 3)
range(-10, 5, 3)
>>> list(range(-10, 5, 3))
[-10, -7, -4, -1, 2]
```

Built-in Sequences

Dictionaries

- Map keys to values, arbitrary types
- Only immutable objects as keys
- Can serve as database

```
d = {}
d["Bilbo"] = "Hobbit"
d["Elrond"] = "Elf"
d[42] = 1234
d.has_key("Bilbo")
d.keys()                    # unordered
d.items()
d.values()
del d[42]
d.clear()
```