Leiden University
Medical Center

Python Programming

Introduction

Mihai Lefter

Outline

About the course

- Aimed at PhD students, Postdocs, researchers, analysts, ...
- Focus on:
  - Basic understanding of Python.
  - Programming as a tool to do your research.
  - Slightly biased on bioinformatics.

Material

All course material is to be found here:
https://git.lumc.nl/courses/programming-course

Hands On!

Programming is fun!

- You only learn programming by doing it.
- Lecture format:
  - Blended teaching + exercising.
- Have your laptop open during the lessons.
- Repeat the code from the slides, play around with it.
- Do the session exercises.
- There will be a few assignments.

Teachers

- Mark Santcroos
  m.a.santcroos@lumc.nl
- Jonathan Vis
  j.k.vis@lumc.nl
- Ruben Vorderman
  r.h.p.vorderman@lumc.nl
- Redmar van den Berg
  r.r.van_den_berg@lumc.nl
- Mihai Lefter
  m.lefter@lumc.nl

Program

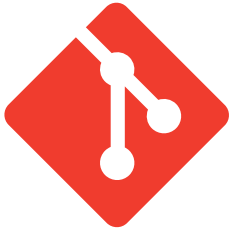
	Tuesday 26/11 J-1-83	Wednesday 27/11 P-5-34	Thursday 28/11 P-5-34	Friday 29/11 J-1-83
9:00 - 10:00	Welcome, introduction to Python Mihai	Assignments review Mihai/Mark	Assignments review Mihai/Jonathan	Assignments review Mihai/Ruben
10:00 - 11:00	Data types Mark	Functions Mihai	String methods, errors, and exceptions Ruben	Jupyter Notebook Ruben
11:00 - 12:00	Flow control Mihai	Object-oriented programming Jonathan	Standard library, reading, and writing files Mihai	Data mangling with pandas Mihai
12:00 - 13:00	Lunch break			
13:00 - 14:00	Practical session			
14:00 - 17:00				

Program

	Tuesday 26/11 J-1-83	Wednesday 27/11 P-5-34 Restricted area: we meet at **8:45** in **building 2 restaurant**.	Thursday 28/11 P-5-34	Friday 29/11 J-1-83
9:00 - 10:00	Welcome, introduction to Python Mihai	Assignments review Mihai/Mark	Assignments review Mihai/Jonathan	Assignments review Mihai/Ruben
10:00 - 11:00	Data types Mark	Functions Mihai	String methods, errors, and exceptions Ruben	Jupyter Notebook Ruben
11:00 - 12:00	Flow control Mihai	Object-oriented programming Jonathan	Standard library, reading, and writing files Mihai	Data mangling with pandas Mihai
12:00 - 13:00	Lunch break			
13:00 - 14:00	Practical session			
14:00 - 17:00				

Software requirements

- Anaconda:
  - Python 3.7.
  - Comes with all that's required:
    - Python interpreter.
    - Jupyter Notebook.
    - Libraries: NumPy, Panda, matplotlib, Bokeh, Biopython, ...
  - Installation instructions.
- Git:
  - Installation instructions.



Assignments

- We make use of GitHub Classroom.
  - GitHub account required.
  - Receive link with assignment repository.
- Own forked repository to work on:
  - Clone it.
  - Code it.
  - Push it.
- Direct file upload to repository is also possible.

Getting help

- Ask a teacher (we will be around in the afternoon).
- If it's private, mail one of the teachers.

History

- Created early 90's by Guido van Rossem at CWI.
  - Name: Monty Python.
- Design is driven by code readability.

CWI Centrum Wiskunde & Informatica

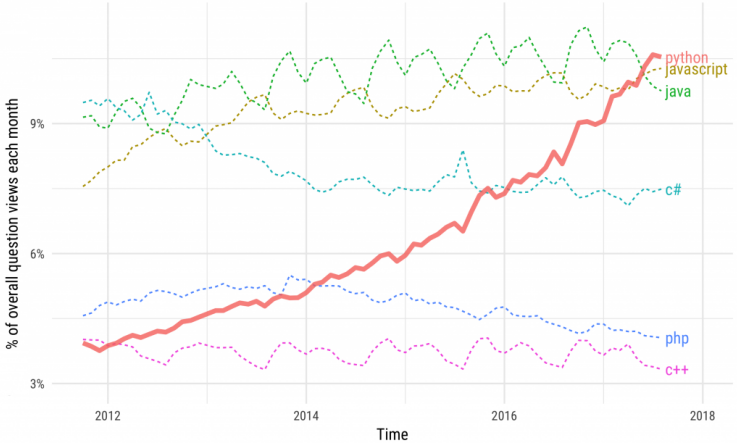
Features

- General purpose, high-level programming language.
- Interpreted, no separate compilation step needed.
- Imperative and object-oriented programming.
  - And some functional programming.
- Dynamic type system.
- Automatic memory management.

We'll come back to most of this.

Why Python?

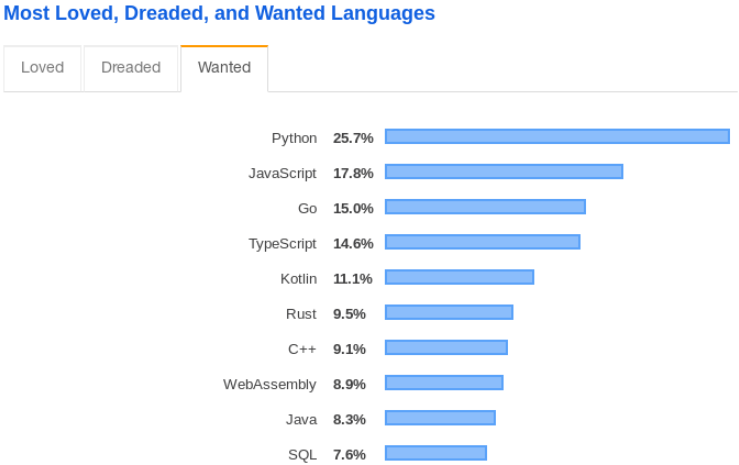
- Widely used with a large community.



https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Why Python?

- Widely used with a large community.



https://insights.stackoverflow.com/survey/2019/#most-loved-dreaded-and-wanted

Why Python?

- Widely used with a large community.

https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019

Why Python?

- Widely used with a large community.
- Rich scientific libraries.
- Many other libraries available.
- Readable and low barrier to entry.

Python 2 versus Python 3

- Python 2.7 is the last Python 2.
- Python 3 is backwards incompatible.
- Some libraries don't support it yet.
- Some Python 3 features are backported in Python 2.7.
- Python 2.7 will not be maintained past 2020.

We'll use Python 3.7 for this course.

Running Python Code

Interactively:

- Statement by statement, directly in the interpreter.

Running Python Code

Interactively:

- Statement by statement, directly in the interpreter.

terminal

```
$ python
Python 3.7 (default, Nov 26 2019, 10:23:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> print('Hello world')
Hello world
>>>
```

Running Python Code

Interactively:

- Statement by statement, directly in the interpreter.

terminal

```
$ python
Python 3.7 (default, Nov 26 2019, 10:23:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> print('Hello world')
Hello world
>>>
```

- Interpreters are great for prototyping.
- But not really suitable if you want to share or release code.

Running Python Code

Interactively:

- Statement by statement, directly in the interpreter.

terminal

```
$ python
Python 3.7 (default, Nov 26 2019, 10:23:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> print('Hello world')
Hello world
>>>
```

- Interpreters are great for prototyping.
- But not really suitable if you want to share or release code.

Non-interactively:

- By editing a file and running the code afterwards.

Running Python Code

Interactively:

- Statement by statement, directly in the interpreter.

terminal

```
$ python
Python 3.7 (default, Nov 26 2019, 10:23:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> print('Hello world')
Hello world
>>>
```

- Interpreters are great for prototyping.
- But not really suitable if you want to share or release code.

Non-interactively:

- By editing a file and running the code afterwards.

first_script.py

```
1 print("Hello world!")
```

Running Python Code

Interactively:

- Statement by statement, directly in the interpreter.

terminal

```
$ python
Python 3.7 (default, Nov 26 2019, 10:23:46)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> print('Hello world')
Hello world
>>>
```

- Interpreters are great for prototyping.
- But not really suitable if you want to share or release code.

Non-interactively:

- By editing a file and running the code afterwards.

first_script.py

```
1 print("Hello world!")
```

terminal

```
$ python first_script.py
Hello world
$
```

The IPython interpreter

Similar to the standard Python interpreter, but with:

- syntax highlighting;
- tab completion;
- cross-session history;
- etc.

terminal

```
$ ipython
Python 3.7 (default, Nov 12 2018, 13:43:14)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Integers

IPython

```
In [1]:  17
Out[1]:  17

In [2]:  (17 + 4) * 2 - 10
Out[2]:  32

In [3]:  14 // 4  # floor division - quotient
Out[3]:  3

In [4]:  14 % 4   # modulus - remainder
Out[4]:  2

In [5]:  2 ** 3   # exponentiation
Out[5]:  8
```

Floating point numbers

IPython

```
In [6]: 3.2 * 18 - 2.1
Out[6]: 55.5

In [7]: 36 / 5
Out[7]: 7.2
```

Floating point numbers

Scientific notation:

IPython

```
In [8]:  1.3e20 + 2
Out[8]:  1.3e+20

In [9]:  1.3 * 10**20
Out[9]:  1.3e+20
```

Variables

- Names (identifiers) used to reference values. Can consist of:
  - the uppercase and lowercase letters `A` through `Z`;
  - the underscore `_`;
  - the digits `0` through `9`, except for the first character.

Variables

- Names (identifiers) used to reference values. Can consist of:
  - the uppercase and lowercase letters `A` through `Z`;
  - the underscore `_`;
  - the digits `0` through `9`, except for the first character.
- Notes:
  - Python is case sensitive.
  - Reserved words (`if`, `then`, `for`, etc.) are not to be employed.

Variables

- Names (identifiers) used to reference values. Can consist of:
  - the uppercase and lowercase letters `A` through `Z`;
  - the underscore `_`;
  - the digits `0` through `9`, except for the first character.
- Notes:
  - Python is case sensitive.
  - Reserved words (`if`, `then`, `for`, etc.) are not to be employed.

IPython

```
In [16]: a = 17
In [17]: leiden_population = 123856
In [18]: f18_speed = 1.8
```

Variables

- No need to declare them first or define the type.

Variables

- No need to declare them first or define the type.
- However, you do need to give variables values before you use them.

IPython

```
In [21]: a = 17
In [22]: c = a + b

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-1674f6151070> in <module>
----> 1 c = a + b

NameError: name 'b' is not defined
```

Every value has a type

- View it using `type`.

IPython

```
In [23]: type(27)
Out[23]: int

In [24]: type(3 * 2)
Out[24]: int

In [25]: type(3 / 2)
Out[25]: float

In [26]: type(a)
Out[26]: int
```

Some operations are defined on more than one type

- Possibly with different meanings.

IPython

```
In [27]: type(3 * 2.0)
Out[27]: float

In [28]: drinks = 'beer' * 5 + 'whiskey'
In [29]: drinks
Out[29]: 'beerbeerbeerbeerbeerwhiskey'

In [30]: type(drinks)
Out[30]: str
```

Dynamic typing

- At runtime variables can be assigned values of different types.

IPython

```
In [31]: a
Out[31]: 17

In [32]: type(a)
Out[32]: int

In [33]: a = 'spezi'
In [34]: type(a)
Out[34]: str
```

Strongly typed

- Operations on values with incompatible types are forbidden.

IPython

```
In [35]: 'beer' + 34

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-ec918fbfdf41> in <module>()
----> 1 'beer' + 34

TypeError: Can't convert 'int' object to str implicitly
```

Ipython Magics

IPython

```
In [36]: %save introduction 1-25
The following commands were written to file 'introduction.py':
...
In [37]: %run introduction.py

In [38]: exit
```

Documentation of all the magics is to be found here.

Hands On!

1. We've seen that `b = 2` is legal.
   a. What about `2 = b`?
   b. How about `a = b = 1`?
2. In math notation you can multiply x and y like this: xy. What happens if you try that in Python?
3. How many seconds are there in 42 minutes and 42 seconds?
4. How many miles are there in 16 kilometers? (1 mile = 1.61 km)
5. Let's assume that you run a 42 km race in 4 hours 42 minutes and 42 seconds.
   a. What is your average pace (time per mile in minutes and seconds)?
   b. What is your average speed in miles per hour?
6. Use string operations to reference `'tra la la la'` in a variable named `song`.
7. If an article costs 249 Euros including the 19% Value Added Tax (VAT), what is the actual VAT amount in Euros for the corresponding article?

Leiden University
Medical Center

Acknowledgements

Martijn Vermaat
Jeroen Laros
Jonathan Vis