Introduction to Python

6 April 2016

Giacomo Tartari
Engineer/PhD student, UiT

Python

Brief History

- In February 1991, Guido Van Rossum published the code (version 0.9.0)
- Python 1.0 - January 1994
- Python 2.0 - October 16, 2000
- Python 3.0 - December 3, 2008
- Python 3.5 - September 13, 2015

Python 2 or 3?

The older the better?

Not so much!

Features

- interpreted
- multi-paradigm: object-oriented, imperative, functional, procedural, reflective
- dynamic and strongly typed (duck)
- garbage collected
- semantic indentation

Hello World

```
print("Hello, World!")
```

Run

Just code it!

Open your favourite editor

Type:

```
print("Hello, World!")
```

Save as hello.py

On the terminal type:

```
python3 hello.py
```

Execution

```
python3 script.py
```

Or add **#!/usr/bin/env python3** as first line

```
chmod +x script.py

./script.py
```

The interpreter

When you run python code

you run a program that

reads your script (text and human readable)

and executes it on the computer.

Interactive shell (REPL)

Read Evaluate Print Loop

Open your terminal and type python3

Now type some python statements

Exit with CTRL+D

pydoc and help()

The built-in function help() invokes the online help system

Which uses pydoc to generate its documentation as text on the console

```
$ pydoc os
Help on module os:

NAME
    os - OS routines for NT or Posix depending on what system we're on.

FILE
    /usr/local/Cellar/python/2.7.11/Frameworks/Python.framework/Versions/2.7/lib/python2.7/os.py

MODULE DOCS
    http://docs.python.org/library/os

DESCRIPTION
    This exports:
      - all functions from posix, nt, os2, or ce, e.g. unlink, stat, etc.
      - os.path is one of the modules posixpath, or ntpath
      - os.name is 'posix', 'nt', 'os2', 'ce' or 'riscos'
      - os.curdir is a string representing the current directory ('.' or ':')
      - os.pardir is a string representing the parent directory ('..' or '::')
      ...
```

pydoc and help()

```
$ pydoc math.cos
Help on built-in function cos in math:

math.cos = cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
```

```
$ >>> help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

Built in Types

- int, float, boolean etc...
- string
- list
- dictionary
- sets
- tuples

Operators

```
+ Addition
- Subtraction
* Multiplication
/ Division
% Modulus
** Exponent

and  And
or  Or

==    Equality
!=    Inequality
>    Greater then
<    Lesser then
>=    Greater or equal
<=  Lesser or equal

+=  Increment
-=  Decrement
etc...
```

Variables

Declaring a variable is easy in python

```
i = 42

print(i)

π = 22/7

print(π)

i = i + 1

print(i)
```

Run

More Variables

Python is a dynamically typed language

```
x = 42

print(x, type(x))

x = "The answer to the great question"

print(x, type(x))

π = 22/7

print(π, type(π))
```

Run

Try the type() built in function on your terminal

Strings and Lists

Strings

```
text = "Lists and Strings can be accessed via indices!"

print(text[0], text[10], text[-1])
```

Run

Lists

```
lst = ["Vienna", "London", "Paris", "Berlin", "Zurich", "Hamburg"]

print(lst[0])

print(lst[2])

print(lst[-1])
```

Run

More strings

Strings can be defined in different ways

```
str1 = "Double quotes"

str2 = 'Single quotes'

str3 = "Double quotes can have single 'quotes' inside"

str4 = 'And "vice versa"'

str5 = """Also spanning
more than one line"""

print(str1)

print(str2)

print(str3)

print(str4)

print(str5)
```

Run

String functionalities

String objects are packed with useful functionalities

```
text = "Interesting text"

print(text.split())

print(text.endswith("xt"))

print(text.swapcase())

print(text.index("rest"))
```

Run

Slicing

```
str = "Python is great"

first_six = str[0:6]

print(first_six)

starting_at_five = str[5:]

print(starting_at_five)

a_copy = str[:]

print(a_copy)

without_last_five = str[0:-5]

print(without_last_five)
```

Run

Concatenation

```
text = "Interesting text!"

lst = ["Vienna", "London", "Paris", "Berlin", "Zurich", "Hamburg"]
```

String

```
print(text + text)
```

Run

Lists

```
print(lst[:2]+lst[3:])

print(lst[3]+lst[-1])

print()
```

Run

Dictionaries

```
city = {"Vienna": 1.741, "London": 8.539, "Paris": 2.244, "Berlin": 3.502, "Zurich": 0.378}


print(city["London"])


print(city)
```

Run

Insert and remove

```
city = {"Vienna": 1.741, "London": 8.539, "Paris": 2.244, "Berlin": 3.502, "Zurich": 0.378}

city["Hamburg"] = 1.734

print(city["Hamburg"])

del city["Paris"]

print(city)
```

Run

Sets

```
langs = {"Perl", "Python", "Java", "Go", "C++", "Rust"}

bad_langs = {"Perl", "C++"}

print(langs.difference(bad_langs))

new_langs = {"Rust", "Go", "Dart"}

print(langs.intersection(new_langs))

cool_langs ={"Go", "Python"}

print(cool_langs.issubset(langs))
```

Run

Tuples

Tuples are immutable lists

```
tup = (1,2,3,4)

print(tup[0], tup[2], tup[-1])
```

Run

Is present?

There is an easy way to check is an element is present

```
text = "Lists and Strings can be accessed via indices!"

lst = ["Vienna", "London", "Paris", "Berlin", "Zurich", "Hamburg"]

city = {"Vienna": 1.741, "London": 8.539, "Paris": 2.244, "Berlin": 3.502, "Zurich": 0.378}

tup = (1,2,3,4)

print('t' in text)

print("Hamburg" not in lst)

print("Hamburg" in city)

print(3 in tup)
```

Run

Conditional Statements

if *this* then *that*

If checks on some boolean conditions, e.g.:
- is this number greater than 10?
- is this file present in the working directory?
- is this item in the list?

While *this* and *that* are block of code

If

```
city = {"Vienna": 1.741, "London": 8.539, "Paris": 2.244, "Berlin": 3.502, "Zurich": 0.378}

if city["Vienna"] > city["London"]:
    print("There are more people in Vienna")
else:
    print("There are more people in London")
```

Run

Code Blocks

```
city = {"Vienna": 1.741, "London": 8.539, "Paris": 2.244, "Berlin": 3.502, "Zurich": 0.378}

if city["Vienna"] < city["London"]:
    print("There are")
    print("less people in Vienna")
else:
    print("There are")
    print("less people in London")
```

Run

More conditions

```
city = {"Vienna": 1.741, "London": 8.539, "Paris": 2.244, "Berlin": 3.502, "Zurich": 0.378}

if city["Vienna"] < city["London"]:
    print("There are")
    print("less people in Vienna")

elif city["Vienna"] == city["London"]:
    print("There are")
    print("the same amount of people in London and in Vienna")

elif city["Vienna"] > city["London"]:
    print("There are")
    print("less people in London")
```

Run

Loops

Loops are used to do *something* a number of times

Or until a condition is satisfied

The number or the condition doesn't have to be known when you write the code

There are typically two kind of loops:

- for
- while

While

While loops a block of code *while* a condition is satisfied

```
num = 0

while num < 5:
    print(num)
    num = num +1

print("more code here...")
```

Run

For

For loops look like this:

```
for <variable> in <sequence>:
    <statements>
```

```
langs = {"Perl", "Python", "Java", "Go", "C++", "Rust"}

for l in langs:
    print(l)
```

Run

Continue and Break

Loops iterations can be skipped with *continue*

Or loops can be interrupted with *break*

```
langs = {"Perl", "Python", "Java", "Go", "C++", "Rust"}

bad_langs = {"Perl", "C++"}

for l in langs:
    if l in bad_langs:
        continue
    print(l)
```

Run

Continue and Break

```
langs = ["Perl", "Python", "Java", "Go", "Perl", "Rust", "C++", "Python", "Perl", "Go"]

bad_langs = {"Perl", "C++"}

bad_counter = 0

for l in langs:
    if l in bad_langs:
        bad_counter += 1
        if bad_counter > 2:
            print("I quit!!")
            break
        continue
    print(l)
```

Run

Functions

Not really mathematical functions but close enough

```
def function_name(parameters, go, here):
    statements, i.e. the function body
```

```
def do_something(todo_list):
    for thing in todo_list:
        print(thing)



result = do_something(my_list)
```

Functions

```
langs = ["Perl", "Python", "Java", "Go", "Perl", "Rust", "C++", "Python", "Perl", "Go"]

bad_langs = {"Perl", "C++"}

def count_set(list, set):

    print(locals())
    count = 0
    for elem in list:
        if elem in set:
            count += 1
    return count

result = count_set(langs, bad_langs)

print("Level of badness: ", result/len(langs))
```

Run

Import and modules

How do I reuse useful code?

Python libraries are made up of in modules

```
import os

print(os.listdir())
```

Run

And the batteries are included

```
import os

tot_bytes = 0
lst = os.listdir()

for f in lst:
    stat = os.stat(f)
    tot_bytes += stat.st_size

result = "The directory contains {1} files for a total of {0} bytes".format(len(lst), tot_bytes)

print(result)
```

Run

Making your own module

A module is just a Python file.

Put this in *simplemath.py*

```
def add(a, b):
    return a + b

def mul(a, b):
    return a * b
```

You can now use it like this:

```
import simplemath
from simplemath import mul, ALMOST_PI

print(simplemath.add(1, 2))

print(mul(2, ALMOST_PI))
```

Run

Always put import statements at the top of the file.

Operation on files

Common use case:
- open a file
- read the content
- *do something useful* ™
- print it to stdout

Example: open a file and print all the words longer than *x*

Operation on files

```
#!/usr/bin/env python3

import os


with open('loremipsum', encoding='utf-8') as fil:
    words = fil.read().split(".")

    for word in words:
        if len(word) > 4:
            print(word)
```

Run

Operation on files

```
#!/usr/bin/env python3

import os

result = []

"""kjzshdkljshdf
ks;adjhfks;djhf
;ksdjhf;ks"""

with open('loremipsum', encoding='utf-8') as fil:
    words = fil.read().split(".")  # commet
    for word in words:
        if len(word) > 4:
            if word.endswith(".") or word.endswith(","):
                result.append(word[:-1])
            else:
                result.append(word)



for word in result:
    print(word)
```

Run

Thank you

Giacomo Tartari
Engineer/PhD student, UiT
giacomo.tartari@uit.no (mailto:giacomo.tartari@uit.no)