Python Fundamentals

Juan Cabral - jbc.develop@gmail.com

Jan, 2018

What is Python?

- Multi-purpose (Web, GUI, Scripting, etc.)
- Object Oriented
- Interpreted
- Strongly typed and Dynamically typed
- Focus on readability and productivity

Features

- Batteries Included
- Everything is an Object
- Interactive Shell
- Strong Introspection
- Cross Platform
- CPython, Jython, IronPython, PyPy

Who Uses Python

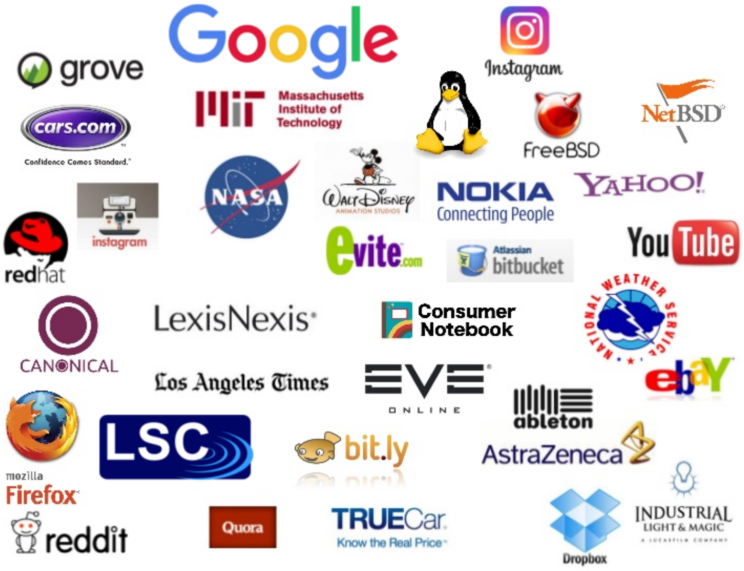


Figure 1: Who Uses Python

Releases

- Created in 1989 by Guido Van Rossum
- Python 1.0 released in 1994
- Python 2.0 released in 2000
- Python 3.0 released in 2008
- Python 2.7 is the deployed version (will be discontinued on 2020).
- 3.6 Is the current 3.x Python.
- oTree uses only Python 3.X.

History

- Invented in CERN, early 90s by Guido van Rossum
- Python was conceived in the late 1980s and its implementation was started in December 1989
- Named after 'Monty Python's Flying Circus'
- Open sourced from the beginning

Python is an experiment in how much freedom program-mers need. Too much freedom and nobody can read another's code; too little and expressive-ness is endangered.

Guido van Rossum

Why was python created?

My original motivation for creating Python was the perceived need for a higher level language in the Amoeba Operating Systems project. I realized that the development of system administration utilities in C was taking too long. Moreover, doing these things in the Bourne shell wouldn't work for a variety of reasons...

So, there was a need for a language that would bridge the gap between C and the shell

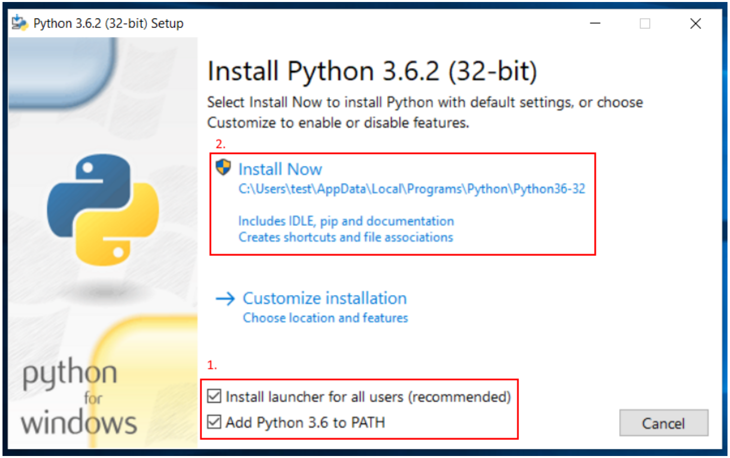
Guido van Rossum

Python Install

- oTree requires **Python 3.6** or higher.
- Python is pre-installed on most Unix systems, including Linux and MAC OS X.
- If you already have **Python 3.6** installed (check by entering `pip3 -V` at your command prompt), you can skip the below section. Or, uninstall your existing version of Python, and proceed with the below steps.

Python Install - Windows

- Download and install Python 3.6 - `https://www.python.org/downloads/release/python-360/`
- **IMPORTANT:** Check the box to add Python to `PATH`



Python 3.6.2 (32-bit) Setup

Install Python 3.6.2 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

2.

Install Now
C:\Users\test\AppData\Local\Programs\Python\Python36-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

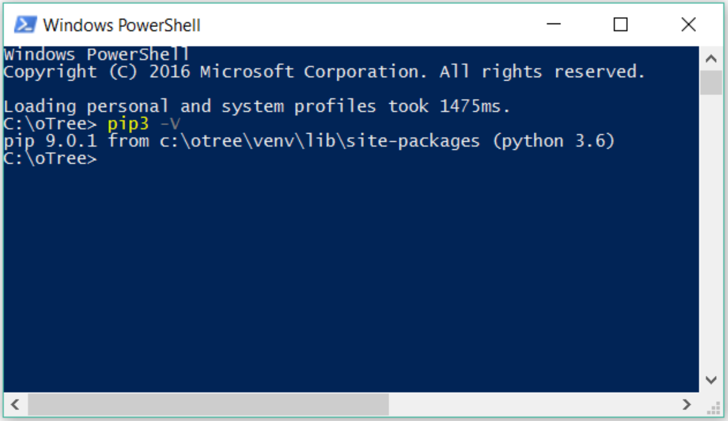
→ Customize installation
Choose location and features

1.

[✓] Install launcher for all users (recommended)
[✓] Add Python 3.6 to PATH

Python Install - Windows (part 2)

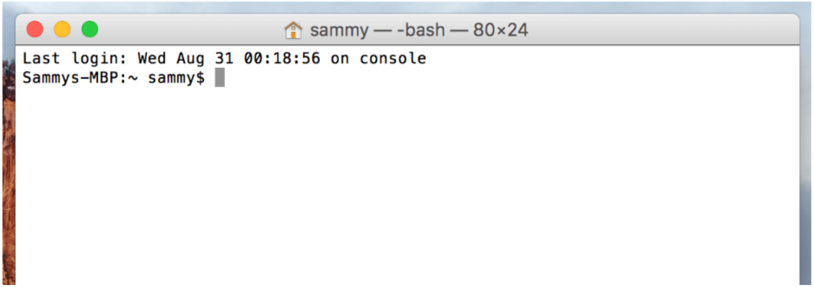
Once setup is done, search in your Windows Start Menu for the program **PowerShell**, open PowerShell, and enter `pip3 -V`



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

Loading personal and system profiles took 1475ms.
C:\oTree> pip3 -V
pip 9.0.1 from c:\otree\venv\lib\site-packages (python 3.6)
C:\oTree>
```

Python Install - OSX

- Download and install the latest Python 3 `https://www.python.org/downloads/mac-osx/`.
- In Finder, search for and open the `Terminal` app.



```
Last login: Wed Aug 31 00:18:56 on console
Sammys-MBP:~ sammy$ 
```

Python Install - OSX (part 2)

- Then type `pip3 -V` and hit enter. This should output something like:

```
pip N.N.N from /Library/Frameworks/Python.framework/[...] (
```

- If not, try closing and reopening the `Terminal` app. Run this command:

```
$ /Applications/Python\ 3.6/Install\ Certificates.command
```

Editor

- Do not use notepad.
- oTree author recommend using PyCharm

  We recommend using PyCharm. Professional Editon is better than Community Edition because it has Django support. PyCharm Professional is free if you are a student, teacher, or professor.

  oTree documentation

- Also a lot of good open-source editors are available:
  - Gedit (if you use ubuntu you already has this one)
  - Geany
  - Spyder
  - Atom

Python tutorial

- Full Document: `http://otree.readthedocs.io/en/latest/python.html`
- There are many other good python tutorials online (such as Codecademy: `https://www.codecademy.com/learn/python`), but note that some of the material covered in those tutorials is not necessary for oTree programming specifically.

Python Tutorial

```python
# Single line comments start with a number symbol.

""" Multiline strings can be written
    using three "s, and are often used
    as comments
"""
```

1. Primitive Datatypes and Operators

```
# You have numbers
3  # => 3

# Math is what you would expect
1 + 1  # => 2
8 - 1  # => 7
10 * 2  # => 20
35 / 5  # => 7

# Enforce precedence with parentheses
(1 + 3) * 2  # => 8
```

1. Primitive Datatypes and Operators (cont)

```
# Boolean Operators
# Note "and" and "or" are case-sensitive
True and False #=> False
False or True #=> True

# negate with not
not True  # => False
not False  # => True

# Equality is ==
1 == 1  # => True
2 == 1  # => False

# Inequality is !=
1 != 1  # => False
2 != 1  # => True
```

1. Primitive Datatypes and Operators (cont)

```python
# More comparisons
1 < 10  # => True
1 > 10  # => False
2 <= 2  # => True
2 >= 2  # => True

# Comparisons can be chained!
1 < 2 < 3  # => True
2 < 3 < 2  # => False

# Strings are created with " or '
"This is a string."
'This is also a string.'

# Strings can be added too!
"Hello " + "world!"  # => "Hello world!"
```

1. Primitive Datatypes and Operators (cont)

```
# A string can be treated like a list of characters
"This is a string"[0]  # => 'T'

# format strings with the format method.
"A triangle has {} sides".format(3)

# None is an object
None  # => None
```

1. Primitive Datatypes and Operators (cont)

```
# Any object can be used in a Boolean context.
# The following values are considered falsey:
#    - None
#    - zero of any numeric type (e.g., 0, 0L, 0.0, 0j)
#    - empty sequences (e.g., '', [])
#    - empty containers (e.g., {})
# All other values are truthy (using the bool()
# function on them returns True).
bool(0)  # => False
bool("")  # => False
```

2. Variables and Collections

```
# Python has a print statement
print("I'm Python. Nice to meet you!")

# No need to declare variables before assigning to them.
# Convention is to use lower_case_with_underscores
some_var = 5
some_var  # => 5

# incrementing and decrementing a variable
x = 0
x += 1  # Shorthand for x = x + 1
x -= 2  # Shorthand for x = x - 2
```

2. Variables and Collections (cont)

```python
# Lists store sequences
li = []
# You can start with a prefilled list
other_li = [4, 5, 6]

# Add stuff to the end of a list with append
li.append(1)    # li is now [1]
li.append(2)    # li is now [1, 2]
li.append(3)    # li is now [1, 2, 3]

# Access a list like you would any array
li[0]  # => 1
```

2. Variables and Collections (cont)

```
# Assign new values to indexes that have already been
# initialized with =
li[0] = 42
li[0]  # => 42

# Note: setting it back to the original value
li[0] = 1
# Look at the last element
li[-1]  # => 3

# You can look at ranges with slice syntax.
other_li[1:3]  # => [5, 6]
# Omit the beginning
other_li[1:]  # => [5, 6]
# Omit the end
other_li[:2]  # => [4, 5]
```

2. Variables and Collections (cont)

```python
# You can add lists
li + other_li    # => [1, 2, 3, 4, 5, 6]
# Note: values for li and for other_li are
# not modified.

# Check for existence in a list with "in"
1 in li    # => True

# Examine the length with "len()"
len(li)    # => 6
```

2. Variables and Collections (cont)

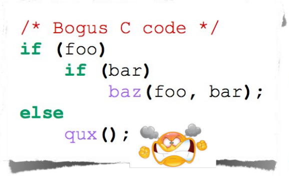
```
# Dictionaries store mappings
empty_dict = {}
# Here is a prefilled dictionary
filled_dict = {"one": 1, "two": 2, "three": 3}

# Look up values with []
filled_dict["one"]   # => 1

# Check for existence of keys  with "in"
"one" in filled_dict   # => True
1 in filled_dict   # => False

# Looking up a non-existing key is a KeyError
filled_dict["four"]   # raises KeyError!

# set the value of a key with is similar to lists
filled_dict["four"] = 4
```

3. Control Flow

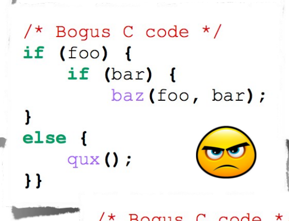
```python
# Let's just make a variable
some_var = 5

# Here is an if statement.
# prints "some_var is smaller than 10"
if some_var > 10:
    print("some_var is totally bigger than 10.")
elif some_var < 10:    # This elif clause is optional.
    print("some_var is smaller than 10.")
else:           # This is optional too.
    print("some_var is indeed 10.")
```

3. Control Flow (cont)

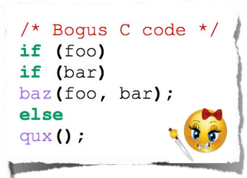
Special note about indenting



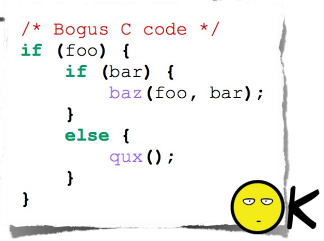
```c
/* Bogus C code */
if (foo)
    if (bar)
        baz(foo, bar);
else
    qux();
```



```c
/* Bogus C code */
if (foo) {
    if (bar) {
        baz(foo, bar);
}
else {
    qux();
}}
```



```c
/* Bogus C code */
if (foo)
if (bar)
baz(foo, bar);
else
qux();
```



```c
/* Bogus C code */
if (foo) {
    if (bar) {
        baz(foo, bar);
    }
    else {
        qux();
    }
}
```

3. Control Flow (cont)

Special note about indenting (cont)

In Python, you must indent your code correctly, or it will not work. (Python is different from some other languages in this regard) All lines in a block of code must be aligned along the left edge. When you're inside a code block (e.g. "if", "for", "def"; see below), you need to indent by 4 spaces.
Examples of wrong indentation:

```python
# error, this line needs to be indented by 4 spaces
if some_var > 10:
print("bigger than 10.")

# 'else' needs to be unindented by 1 space
if some_var > 10:
    print("bigger than 10.")
 else:
    print("less than 10")
```

3. Control Flow (cont)

for loops iterate over lists prints:

- dog is a mammal
- cat is a mammal
- mouse is a mammal

```
for animal in ["dog", "cat", "mouse"]:
    print("{} is a mammal".format(animal))
```

range(number) generates numbers from zero to the given number prints:

- 0
- 1
- 2
- 3

```
for i in range(4):
    print(i)
```

3. Control Flow (cont)

`range(lower, upper)` generates from the lower number to the upper number prints:

- 4
- 5
- 6
- 7

```python
for i in range(4, 8):
    print(i)
```

`while` loops iterate until the condition is False. Prints: 0, 1, 2, 3

```python
x = 0
while x < 4:
    print(x)
    x += 1
```

5. Modules

You can import modules

```
import random
import math
print(math.sqrt(16))
```

You can get specific functions from a module

```
from math import ceil, floor
print(ceil(3.7))  # => 4.0
print(floor(3.7))   # => 3.0
```

Python modules are just ordinary python files. You can write your own, and import them. The name of the module is the same as the name of the file.

6. Classes

Classes let you model complex real-world entities

```python
class Mammal(object):

    classification = "Mammalia"

    def set_age(self):
        self.age = 0

    def older_than_10(self):
        return self.age > 10

    def predict_age(self, years):
        return 'In {} years I will be {}'.format(
            years, self.age + years)
```

6. Classes (cont)

```python
class Dog(Mammal):
    classification = "Canis lupus"

    def bark(self):
        return "woof!"

Mammal.classification

# Instantiate a class
lassie = Dog()
lassie.classification # => "Canis lupus"
lassie.set_age()
lassie.older_than_10() # => False
lassie.age = 11
lassie.older_than_10() # => True
lassie.bark() # => "woof!"
```

Why do people use Python. . . ?

- **Python is object-oriented** Structure supports such concepts as polymorphism, operation overloading, and multiple inheritance.
- **Indentation** Indentation is one of the greatest future in Python.
- **It's free (open source)** Downloading and installing Python is free and easy Source code is easily accessible

Why do people use Python. . . ? (cont)

- **It's powerful**
  - Dynamic typing
  - Built-in types and tools
  - Library utilities
  - Third party utilities (e.g. Numeric, NumPy, SciPy)
  - Automatic memory management
- **It's portable**
  - Python runs virtually every major platform used today
  - As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform.
- **It's mixable**
  - Python can be linked to components written in other languages easily
  - Linking to fast, compiled code is useful to computationally intensive problems
  - Python/C integration is quite common

Why do people use Python. . . ? (cont)

- **It's easy to use**
  - No intermediate compile and link steps as in C/ C++
  - Python programs are compiled automatically to an intermediate form called bytecode, which the interpreter then reads
  - This gives Python the development speed of an interpreter without the performance loss inherent in purely interpreted languages
- **It's easy to learn**: Structure and syntax are pretty intuitive and easy to grasp

References

- https://es.slideshare.net/nowells/introduction-to-python-5182313
- https://es.slideshare.net/sujithkumar9212301/introduction-to-python-36647807
- http://otree.readthedocs.io/en/latest/