FAIM Python Workshop
General Introduction & OOP

benjamin.titze@fmi.ch

A brief history…

Created by Dutch software developer **Guido van Rossum** at CWI, Amsterdam. Released in 1991.

Van Rossum was Python's BDFL (Benevolent Dictator for Life) until July 2018.

Started out as a "hobby programming project" to bridge the gap between shell scripts and C.

Named after the British comedy group 'Monty Python'.

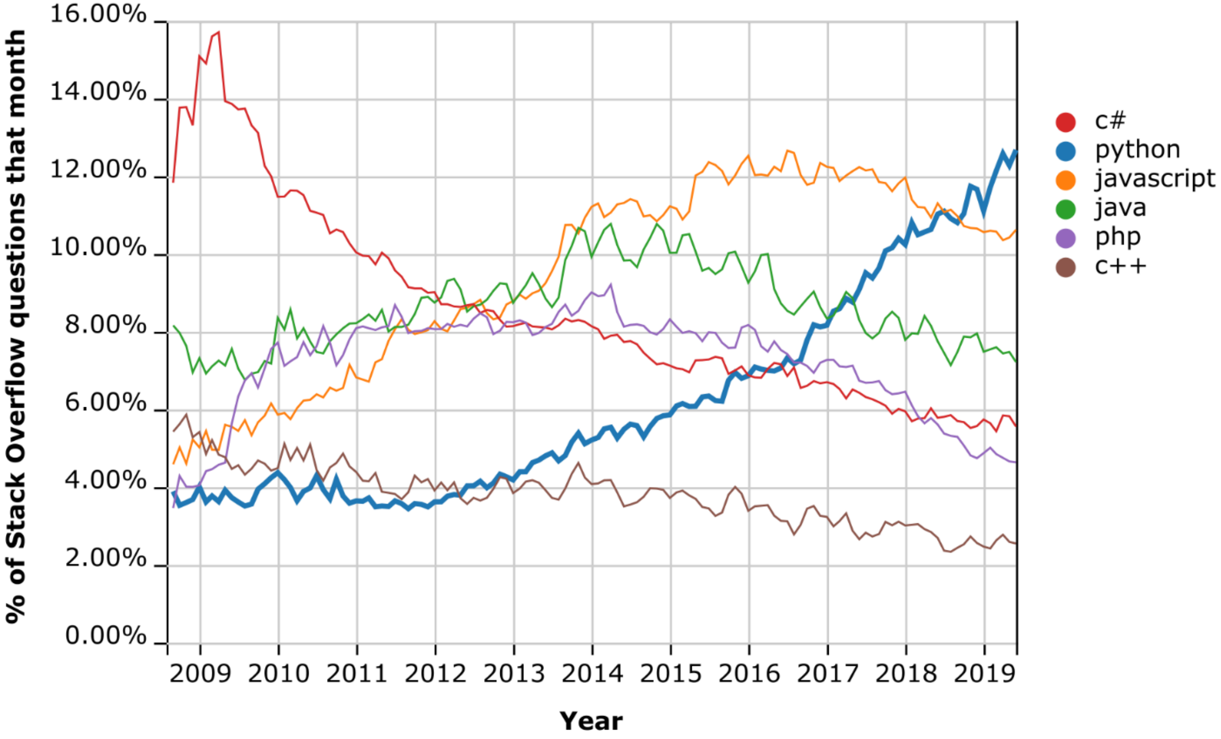
Python 1.0 – 1994
→ 1.x versions are obsolete

Python 2.0 – 2000
→ Latest and final: 2.7 in 2010

Python 3.0 – 2008
→ Currently 3.7 (June 2018)

Strong growth over the last decade.

Python is now a mainstream programming language alongside Java, C/C++/C#, JavaScript, …



Why Python?

Python is a

general-purpose

interpreted

high-level

object-oriented

programming language.

Increasing use by large IT companies

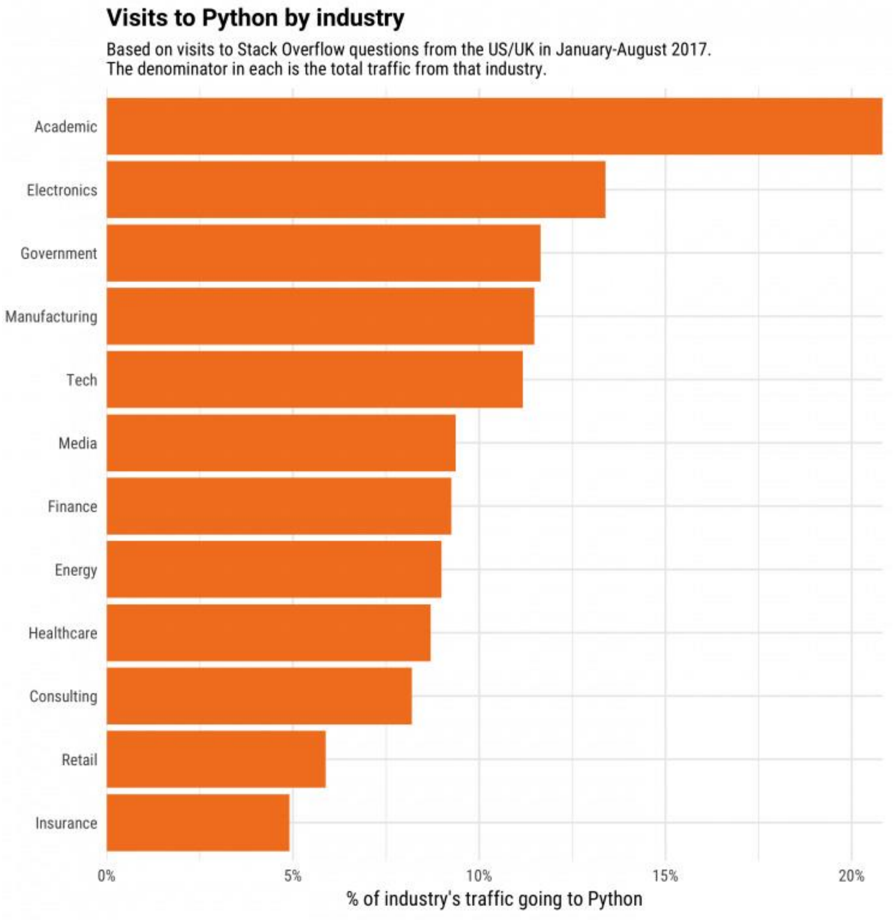
Prominent examples:

Google, Amazon, Instagram, Dropbox, Spotify, Netflix…

Why Python?

- Easy to learn
- Fast development
- Platform-independent
- Great ecosystem: lots of third-party libraries, vibrant community.

But:
Slow compared to compiled languages, high memory consumption, runtime errors…



Source: https://www.peerbits.com/blog/factors-will-drive-python-growth-in-2018.html

Python 2 vs 3

- Python 2 will become obsolete in the long run (won’t be maintained past 2020);
  Python 3 is the future.

  → https://python3statement.org and https://pythonclock.org !
- If you start a project now, don’t use Python 2 unless you really have to.
- Examples of differences:
  - Division operator: 5/2 yields 2 in Python 2, but 2.5 in Python 3.
  - print() is a function in 3.x; but in 2.x print is a statement.
    - `print "Hello"` in Python 2.x; `print("Hello")` in Python 3.x
    - `print "Hello",` to stay on the same line. In Python 3.x: `print("Hello", end="")`
  - Implicit str type in Python 2 is ASCII, in Python 3 it’s utf-8 (Unicode) by default:
    - `print("こんにちは、世界！")`
    - `Ω = math.sin(θ) * Ŵ`
  - Changed behaviour/naming of some functions:
    - `xrange()` vs `range()`; `raw_input()` vs `input()`
- ‘Future’ functionality can be made available to earlier versions:
  - For example: `from __future__ import division`

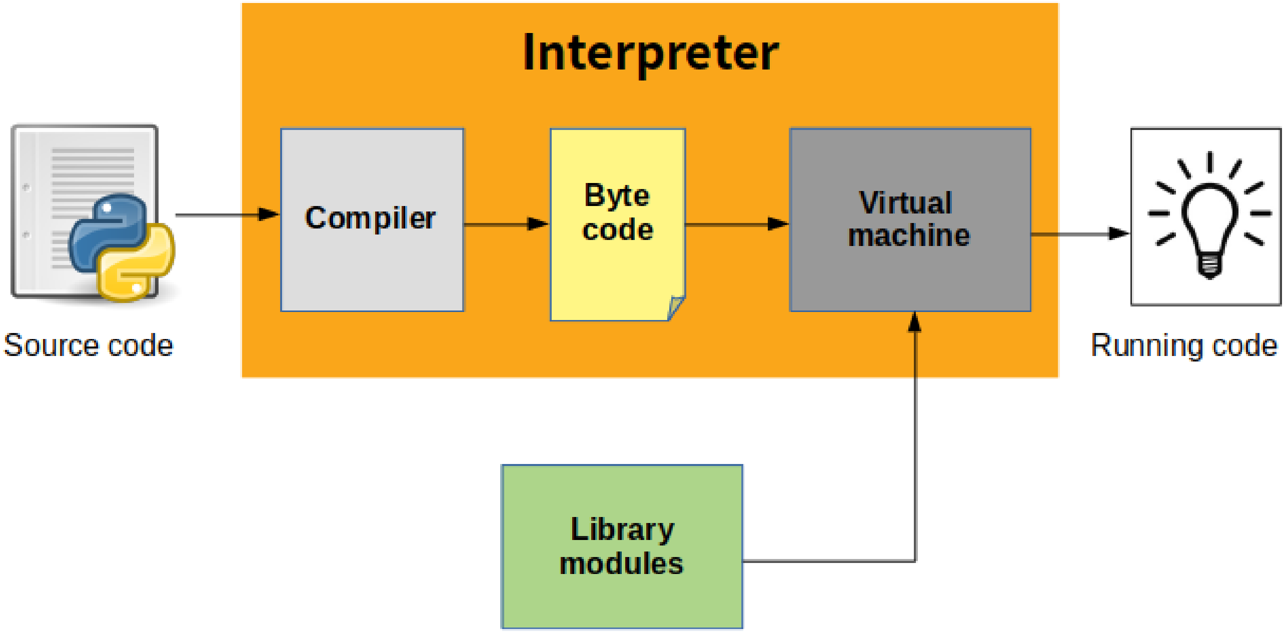
From source code to execution

The *source code* (*.py files) is not interpreted directly, but first compiled into *byte code* (*.pyc files), an intermediate lower-level language for faster execution.

Byte-code is interpreted on the Python Virtual Machine.

CPython:
The Python reference implementation in C, github.com/python/cpython)

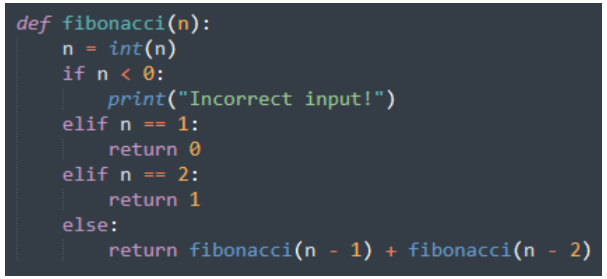
Alternatives: Jython, Iron Python, PyPy…



Source: https://indianpythonista.wordpress.com

Byte code example

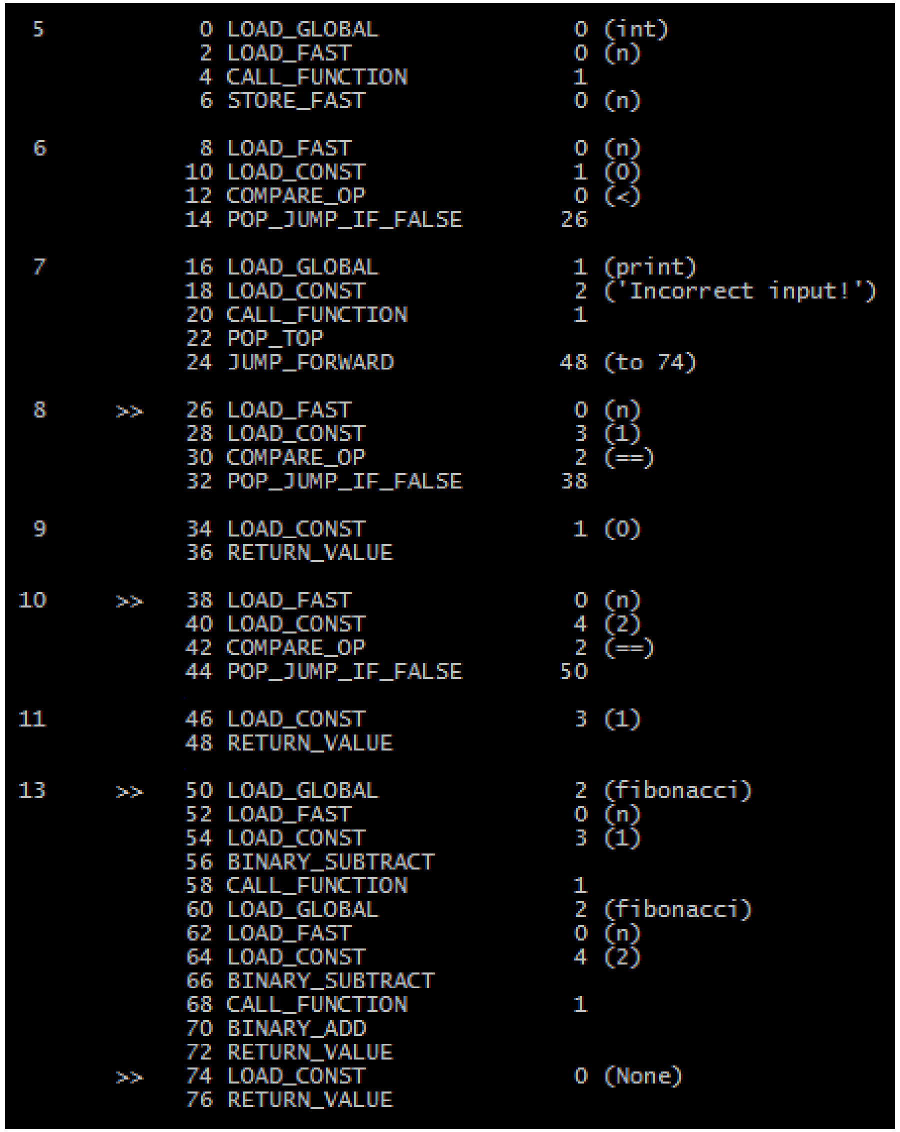
Python source code



```python
def fibonacci(n):
    n = int(n)
    if n < 0:
        print("Incorrect input!")
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

This function returns the nth Fibonacci number.

Bytecode in human-readable form:



```
  5           0 LOAD_GLOBAL              0 (int)
              2 LOAD_FAST                0 (n)
              4 CALL_FUNCTION            1
              6 STORE_FAST               0 (n)

  6           8 LOAD_FAST                0 (n)
             10 LOAD_CONST               1 (0)
             12 COMPARE_OP               0 (<)
             14 POP_JUMP_IF_FALSE       26

  7          16 LOAD_GLOBAL              1 (print)
             18 LOAD_CONST               2 ('Incorrect input!')
             20 CALL_FUNCTION            1
             22 POP_TOP
             24 JUMP_FORWARD            48 (to 74)

  8     >>   26 LOAD_FAST                0 (n)
             28 LOAD_CONST               3 (1)
             30 COMPARE_OP               2 (==)
             32 POP_JUMP_IF_FALSE       38

  9          34 LOAD_CONST               1 (0)
             36 RETURN_VALUE

 10     >>   38 LOAD_FAST                0 (n)
             40 LOAD_CONST               4 (2)
             42 COMPARE_OP               2 (==)
             44 POP_JUMP_IF_FALSE       50

 11          46 LOAD_CONST               3 (1)
             48 RETURN_VALUE

 13     >>   50 LOAD_GLOBAL              2 (fibonacci)
             52 LOAD_FAST                0 (n)
             54 LOAD_CONST               3 (1)
             56 BINARY_SUBTRACT
             58 CALL_FUNCTION            1
             60 LOAD_GLOBAL              2 (fibonacci)
             62 LOAD_FAST                0 (n)
             64 LOAD_CONST               4 (2)
             66 BINARY_SUBTRACT
             68 CALL_FUNCTION            1
             70 BINARY_ADD
             72 RETURN_VALUE
        >>   74 LOAD_CONST               0 (None)
             76 RETURN_VALUE
```

Disassembled with **dis** package:
https://docs.python.org/3/library/dis.html

Read more about this: https://opensource.com/article/18/4/introduction-python-bytecode

Installing and running Python / Editors and IDEs

- Different options:
  - Installer from www.python.org (Python Software Foundation)
    - Choose version 3.x for your operating system. Let installer add Python to the system path.
  - Distribution:
    - Anaconda: www.anaconda.com – comes with lots of useful packages and tools preinstalled.
    - Or the minimalist Miniconda: just Python and the conda package manager
  - … or try it out online:
    - https://www.python.org/shell
    - https://www.pythonanywhere.com/
- Test installation on the command line interface:
  - Typing `python` should start Python shell.
  - Try: `import this` and `import antigravity`
- You may need to set environment variables manually.
- What else do you need?
  - Good text editor with syntax highlighting: Notepad++, VIM, Emacs, Sublime Text, Atom, many others…
  - IDE – integrated development environment (optional): Spyder (comes with Anaconda), PyCharm, Ecplise + PyDev…
  - Jupyter Notebook

Jupyter notebook

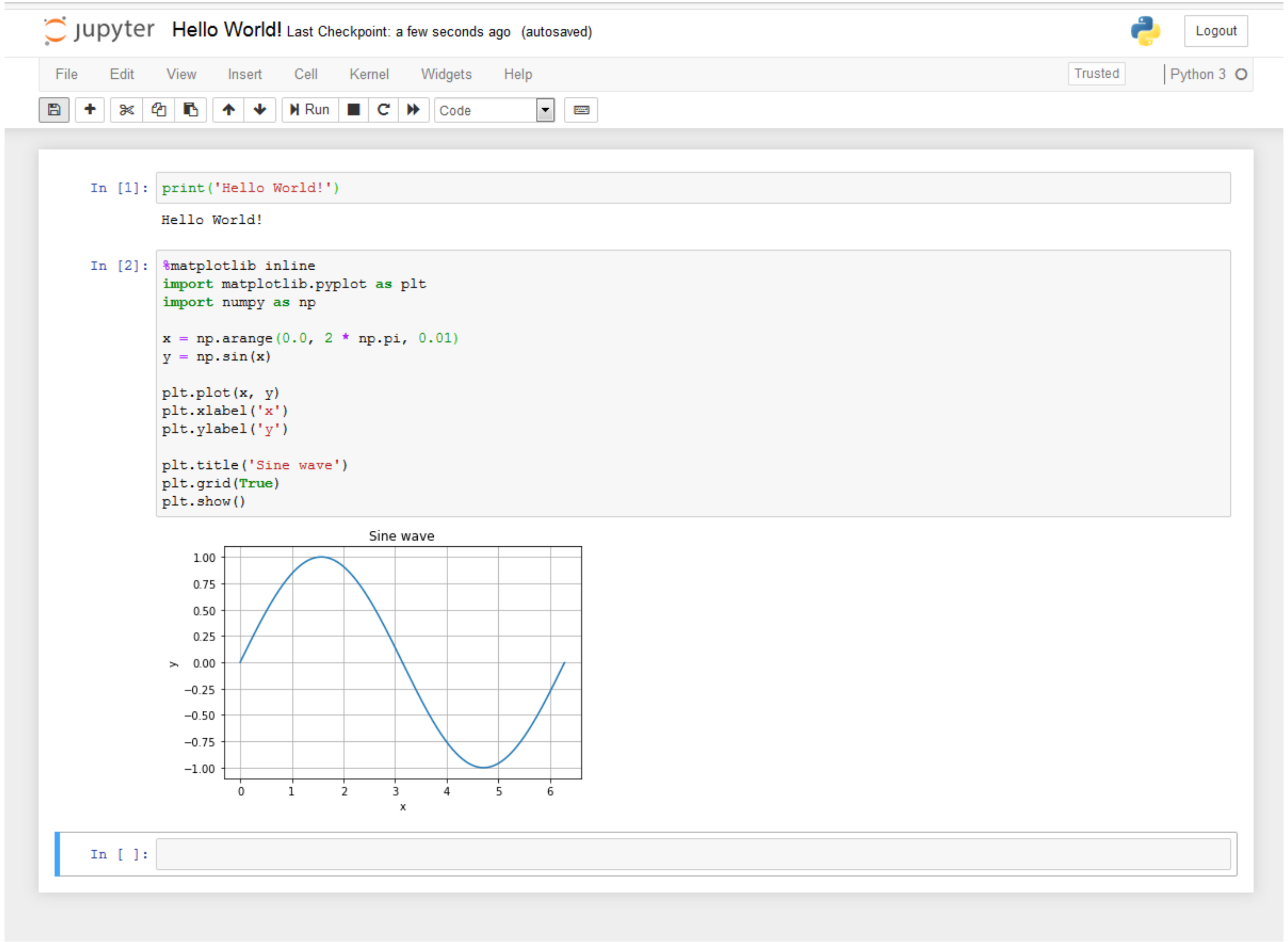
Browser-based interactive programming environment (previously IPython Notebook)

Named after the core languages supported: **Ju**lia, **Py**thon, and **R**.

A common tool for data science / scientific computing in both academia and industry.

Run from command line: `jupyter notebook`
If it’s not installed yet:
Install with: `pip install jupyter`

Notebook files: *.ipynb



```
In [1]: print('Hello World!')

Hello World!

In [2]: %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np

        x = np.arange(0.0, 2 * np.pi, 0.01)
        y = np.sin(x)

        plt.plot(x, y)
        plt.xlabel('x')
        plt.ylabel('y')

        plt.title('Sine wave')
        plt.grid(True)
        plt.show()

In [ ]:
```

Hands-on exercise I:
Use Jupyter notebook and review basic syntax and language elements

Open a new Jupyter Notebook (Python 3)
Get familiar with the Jupyter Notebook

Write code that can do the following:

- Ask the user for his/her name and age
- Greet the user with his/her name
- Check if the user's name is a palindrome.
- Check if the user's age is a prime number.

Write functions is_prime() and is_palindrome() for the checks. Do not use any imports.

You can work with others or try it alone. We'll then go through the code together (Review_Basics.ipynb). ☺

Virtual environments

Virtual environments allow you to use **specific versions** of Python and Python packages **in isolation**.

Different options for environments:

- `virtualenv`
- `venv` (new in Python 3.3)
- `conda`

conda (from Anaconda) is both a package manager and an environment manager.

Let’s try it on the command line:

```
pip install virtualenv
```
(if not installed yet)

```
virtualenv <..\path\to\venv>
virtualenv --python=python2.5 <..\path\to\venv>
```
(if python2.5 installed at system level, otherwise point to path: --python=\path\to\python2.5)

Alternatives:

```
python -m venv <..\path\to\venv>
conda create --name myenv [python=3.4]
conda create -n myenv python=3.4 scipy=0.15.0 [other packages]
```

Go to `..\path\to\venv\Scripts`

```
activate
```
(Prompt will show the new environment!)

```
deactivate
```

With conda:

```
conda activate myenv
conda deactivate
```

Package managers

Package managers install packages and their dependencies:

- `pip` – Python's default
- `conda` – Anaconda's package manager

If you activate virtual environment X and install packages, they will be only available in X.

Let's try it on the command line:

```
pip install <package_name>
pip uninstall <package_name>
```

Upgrade to latest version:

```
pip install --upgrade <package_name>
```

Specific version:

```
pip install <package_name==1.7>
```

Install from requirements file (list of packages):

```
pip install –r requirements.txt
```

Similar with conda:

```
conda install [--name myenv] <package_name>
conda remove <package_name>
conda install <package_name=1.7>
conda install --file requirements.txt
```

(single equal sign!)

This page may be helpful (under Windows): https://www.lfd.uci.edu/~gohlke/pythonlibs/
Provides binaries (wheel files) for installation with pip.

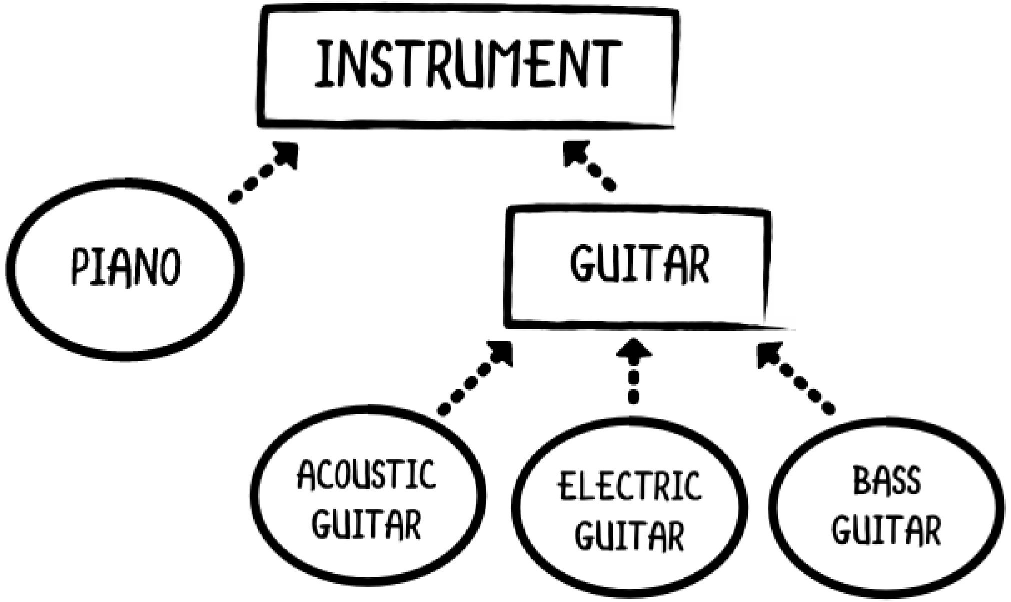
Object-oriented programming (OOP)

Bind together data and functions in logical units – for better organization and maintainability!

Terminology:
Classes and objects, instances of classes
In Python, everything is an object!

Four basic principles:

- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**



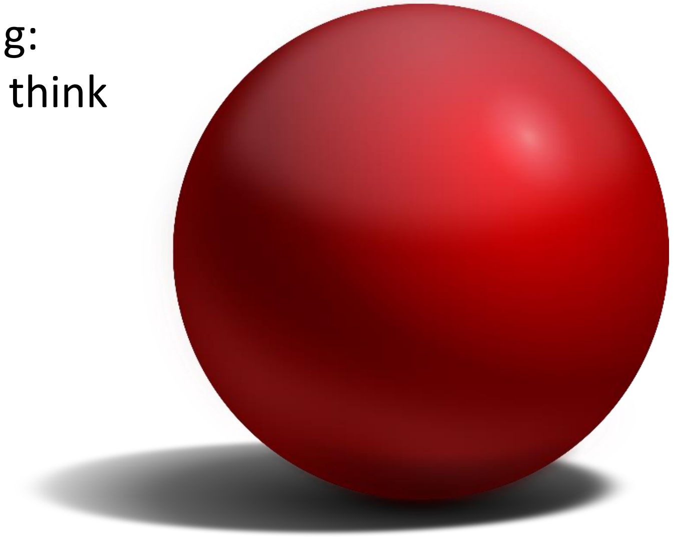
https://www.raywenderlich.com/599-object-oriented-programming-in-swift

Hands-on exercise II:
Learn basics of OOP with classes for 3D shapes

We will work in an editor for this exercise and develop classes for 3D shapes. I will create the base class and the cube class and explain the basic concepts.
You will then try to create a sphere class.

Our 3D shape module should be able to do the following:

- Manage 3D objects (cubes and spheres, but you can think of other shapes)
- Calculate their surface areas and volumes
- Calculate the distance between two objects
- (Pretend to) draw them



We'll go over the entire code together:
shapes.py, shapes-main.py
Optional: Use of 'properties' (advanced):
shapes_properties.py, shapes-main_properties.py

Hands-on exercise III:
Trying out packages for text analysis, computer algebra, and networks

We will use Jupyter Notebook to try out the following packages:

TextBlob/NLTK

Simplified text processing (builds upon NLTK)

https://textblob.readthedocs.io

```
pip install –U textblob
python -m textblob.download_corpora
```

SymPy

Lightweight library for symbolic mathematics

https://www.sympy.org

```
pip install sympy
```

NetworkX

Creating, manipulating and analysing networks

https://networkx.github.io/

```
pip install networkx
```

→ *See corresponding Jupyter notebooks!*