Introduction to Python

A cookbook

Marcelo Garcia

KAUST Library

Origins

- created by Guido Van Rossum in late 1980s for the experimental Amoeba Operating System at “Vrije Universiteit Amsterdam.”
    - Interesting note: Guido was working with Andrew S. Tanenbaum, who co-authored the book “Operating systems: Design and Implementation,” that would inspire Linus Torvalds to create the Linux kernel, as he wrote in his initial post in 1991: “Hello everybody out there using minix - I’m doing a (free) operating system (just a hobby, won’t be big and professional like gnu) for 386(486) AT clones.”
- Version 1.0 released in 1994, version 2.0 released in 2000, and version 3.0 released 2008.
- The name is from the BBC TV show “Monty Python’s Flying Circus.”

Installing

- For *nix and MacOS most probably will be already installed.
    - On Ubuntu systems, it's a good idea to install the package `python3-venv`, and if you don't plan to use Python 2, then install the package `python-is-python3`, so the command "python" starts `python3` instead of an error that Python 2 is missing.
- For Windows, it's possible to install via the binary provided by Python org, but, it's better to install one of the Python binaries available on Microsoft Store. There are several versions available.

The Zen of Python

Writing *pythonic* scripts

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Basic Elements

Python has all the expected elements of a modern programming language: numbers, strings, functions, objects, etc.

```
>>> c = 300_000_000_00 # speed of light in vacum in SI units (m/s).
>>> ff = 5.678 # float number
>>> zz = 3 + 4j # complex number
>>> hello = "hello world!"
>>> hello.upper() # Basic string functions
'HELLO WORLD!'
>>> hello[2:4] # String slicing.
'll'
>>>
>>> if vv > 10:
...     print('more than 10')
... else:
...     print('not more than 10')
...
not more than 10
>>>
```

Basic Elements (cont.)

```
>>> for ii in range(5):
...     print(f"{ii}", end=",")
...
0,1,2,3,4,>>>
>>>
>>> numbers = range(1, 16) # Create a list from 1 to 15
>>> list(numbers)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>>
>>> [ nn*nn for nn in numbers] # List comprehension
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225]
```

Configuration Files

Create a configuration file to control the behaviour of your script

```
[ACCESSION]
accession_id = 000_000_0000

[BAGGER]
source_dir = C:\Users\joe\Work\boat_trip_pictures
dest_dir = C:\Users\joe\Work\${ACCESSION:accession_id}

[CLAMAV]
(...)
quarantine_days = 30
run_it = no
(...)
```

Reading the Configuration Files

Reading configuration file with `ConfigParser`

```
import configparser
   (...)
   # Creating a configparser object
    config = configparser.ConfigParser(
        interpolation=configparser.ExtendedInterpolation()
    )
    config.read(config_file)
   (...)

    # Reading values
    acc_number = config['ACCESSION']['accession_id']

    # Using a convenience function to read a boolean value
    if config['CLAMAV'].getboolean('run_it'):
        (...)
```

Dot Env File

- Reading environment variables, like data base passwords, without putting them on the code.
- Good for GitHub, but don't forget to add to the `.gitignore`.
- On Linux or MacOS environment variables are easy to use, but on Windows not so much.
- The `.env` file offers a uniform way for all platforms

Consider the file with some credentials

```
me@myserver:~/Work/repo$ cat .env
USERNAME="joe.doe@example.com"
PASSWORD="abc123"
```

Reading the Dot Env File

Load the `.env`, and read the variable from the environment variable:

```
import os
from dotenv import load_dotenv
(...)
    load_dotenv()

    api_passwd = os.environ['MY_API_PW']
```