Advanced Python

Description

Advanced Python is a course internded for people who are working with Python as a primary programming language, and not just as an integration langauge. The course dwells on such topics are the correct use of the data structures, standard libraries, functional featuers and on such topics such as performance, multi-threading and multi-processing.

The course also puts en emphasis on writing Idiomatic Python, which means Python as it should be written and not Python as translated from other programming languags.

Another emphasis of the course is about how to manage large scale python projects, which includes managing virtual environments and dependencies.

The syllabus for this course is intended so that it will be customized to specific needs, and for that there is a time marking on each chapter.

Duration

56 hours / 7 days

Intended Audience

- Software developers who are already working with Python for a year.

Prerequisites

- Students should have working knowledge of Python prior to the course.

Objectives

- Improve the understanding of how Python works
- Improve the knowledge of the standard libraries.
- Learn how to write more idiomatic Python.
- Learn to handle large scale Python projects.
- Write more idiomatic Python code
- Write more efficient Python code
- Maintain larger scale Python projects
- Package and deliver your Python code to your users

Outline

- Python refersher (2 hours)
  - Python basic types
  - Python iterators

  - type() vs isinstance()
- Idiomatic Python (2 hours)
  - Idiomatic data structures
    - frozen structures
  - Correct use of basic features
    - Correct iteration
    - `enumerate`, `zip`, ...
  - Correct use of basic libraries
    - `collections`
- Memory and garbage collector (2 hours)
  - What is the size of my Python data structure?
  - How many Python objects can I have anyway?
  - Reducing memory footprint techniqeus
    - Using specialized libraries
    - Copy on write
    - Flyweight design pattern
    - Using C extensions
- Advanced error handling in python (2 hours)
  - Correct exception handling
  - Writing your own exceptions
  - `raise from` syntax
  - Assertions in your code
- Object Oriented Python toolkit (4 hours)
  - Reminder: inheritence in Python
  - Functions
  - Classes
    - Abstract Classes
    - Properties
    - Multiple inheritance and the diamond problem
    - Mixins
    - Static methods
    - Class methods
    - Overwriting methods in instances
    - Attaching and detaching methods at runtime
  - Modules
- Functional Python (4 hours)
  - What's the idea of functional programming?
  - Review of Python's support for functional programming: `map`, `reduce`
  - More functional support from external modules
  - Understanding closures: `lambda` and inner function.
  - Writing correct closure code in practice
  - Deep closure magic - changing closures
  - The problem of mutable default arguments and how to overcome it
  - Interators and generators
- Python environments (4 hours)
  - Using pip

  - Virtual environments
    - What are they?
  - The basic issues of dependency managements
    - using "requirements.txt" files
    - using more than one requirements file
    - locking dependencies
    - dividing your project dependencies
  - Various tools for environment creation and dependency management
    - The os itself
    - venv
    - virtualenv
    - pipenv
    - poetry
    - pip-tools
    - Hatch
  - Setting version numbers, should you do it and when?
- Packaging and delivering your Python modules (4 hours)
  - distutils
  - setuptools
  - The various output formats
    - source format
    - wheels
  - Uploading to pypi or other repositories
  - Creating your own Python repository
  - Creating a binary distribution containing Python and your code
  - Maintaining a stable API to your module
  - Documenting your module
- Debugging Linting and Testing your Python code (4 hours)
  - How to use the various Python debuggers
  - Using Python assertions correctly
  - Using type hints correctly
  - Using pylint, flake8 and more
  - Using unittest, pytest
  - Doing coverage analysis
- Async Python (4 hours)
  - What are generators?
  - Writing your own generators
  - What are co-routines?
  - Writing your own co-routines
  - What is asynchronous programming?
  - The Python `asyncio` module
  - The Python `twisted` framework.
- Python's `with` statement (2 hours)
  - How to use it
  - How to write your own resource
  - When should you use it

- Python Meta classes (2 hours)
  - What are meta classes?
  - Your first meta class use
  - Creating DSLs with meta classes
  - Examples from third party libraries
- Logging in Python (2 hours)
  - The Python "logging" module
  - Using logging correctly on large systems
  - Configuring logging correctly
  - redirecting logs to syslogd
  - Writing your own formatters and log routers
  - collecting logs
  - flushing when logging on your own
- Python function decorators (2 hours)
  - What are function decorators?
  - Writing your own decorators
  - Examples from the standard libraries
  - Closures and decorators
  - Some notes on efficient caching with decorators
- C and Python integration (only for C programmers) (2 hours)
  - Basic C/Python integration
  - Modules for C/Python integration
- Memory mappings and large data structures in Python (2 hours)
  - the Python mmap function
  - lmdb
  - numpy.memmap
- Porting from Python2 to Python3 (2 hours)
  - Most notable changes
  - the `__future__` module
  - Using the `six` module
  - Doing it the agile way
- Python descriptors (2 hours)
  - What are descriptors?
  - Using descriptors
  - Writing your first descriptor
  - Examples from third party modules
  - Further issues (using from instance, class, super)
- Python and the web (4 hours)
  - Web servers in python with Flask
  - Web clients using the `requests` module
  - Scraping using various libraries:
    - `BeautifulSoup`
    - `scrapy`
    - `urllib3`
  - Scrapiong related libraries:
    - `lxml`

    * `urllib`
  - Testing web apps using `Selenium`
- Python multi-threading and Multi-processing (4 hours)
  - Using the `subprocess` module for launching sub processes
  - Using the `multiprocess` module for multi-processing
  - Python and the GIL
  - Using the `thread` module
  - Synchronizing threads
  - Passing data between threads
  - Locking

Installations

Each student should have:

- An Ubuntu 22:04 machine, real or virtual
- 4 GB RAM for each machine is enough. This is not much.
- Free, wide band, access to the internet from all machines with no weird corporate firewalls that might stop us from installing software and python packages via pip.
- Username and password of a user that has `sudo` priveleges on the machine.
- https://www.linuxvmimages.com/images/ubuntu-2204/
- Users who want to exercise on Windows or on MacOS are welcome to do so and I will help them to fix issues but a minority of the more advanced exercises may not work on those machine.