Python Introduction

Summary

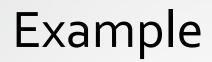
- 1. What is Python?
- 2. Python Distributions
- 3. Development Environments
- 4. Python Project Structure
- Best Practices
- 6. Logging
- 7. Unit-Tests

What is Python?

- Interpreted Language
- Used in various domains: Tooling, Web, AI, Glue language, ...
- Multiple Versions:
 - Python 2
 - Python 3
- Multiple Implementations
 - CPython: Reference
 - Pypy
 - On top of other "VM": IronPython (.net), Jython(Java), Cython (C)
 - •

What is Python?

- Pros of Python
 - Fast Prototyping
 - Tons of Open-Source libraries/API
 - Easy to learn
 - Great Documentation (most of the time)
 - Active Community
- Cons of Python
 - "Slow"
 - Easy to produce quick and dirty code
 - Too many ways to do it => heterogenous code



Main Python Distributions

CPython

- Reference Implementation
- Minimal Footprint
- Package manager: Pypi
- Virtualenv: venv or virtualenv

Anaconda

- Interpreter is Cpython
- Big footprint (≈1GB)
- Package manager: Conda (also Pypi but not recommended)
- Virtualenv: Conda
- Includes most of the Data Science/ Math packages

Development Environment – Package Manager

Pypi

- Tons of libraries
- Comes with most Python distributions
- Relatively easy to use
- Installing Libraries with C bindings can be hard to install

Conda

- Has most of the scientific libraries
- Makes it easy to install Libraries with C bindings
- Relatively easy to use
- Only available with Ananconda or Miniconda

Development Environment – Virtual Environment

- Python Virtual Environments Help:
 - Isolating Project Dependencies
 - Handling Same Package with different versions across different projects
- CPython:
 - Python 2: Virtualenv
 - Python 3: Virtualenv or venv (Recommended venv which comes with Python 3)
- Anaconda:
 - Conda

Development Environment - IDEs

- Application Development
 - Pycharm Community
 - Visual Studio Code with python extension
 - <u>Eclipse</u> with pydev
- Data Science/ Machine Learning
 - Jupyter
 - Spyder
 - <u>Pycharm Professional</u> (Expensive)

Python Project Structure

```
---project
 setup.py — File describing the project and its dependencies
 readme.md — Detailed description of the project
  requirements.txt — Optional: Contains all the dependencies
 test_requirements.txt→ Optional: Contains all the dependencies necessary for testing
+---project_name — This folder will have the name of the library/application
   entry_point.py — Optional: Entry point (not needed for libraries)
   module_name.py — Package module
   __init__.py — Needed for python to understand that project_name is a package
 ---tests
   test_function.py —— Unit-Tests for the code in functions
   __init__.py
```

- Starting Projects
 - Create the project structure
 - Create the virtualenv
 - Fill readme.md and setup.py
- Create Virtual Environment
 - Pycharm: creates one at the creation of the project
 - Virtualenv/venv: <u>Tutorial</u>/<u>Tutorial</u>
 - Conda: <u>Tutorial</u>

Create setup.py (<u>Documentation</u>)

- Setup.py useful commands:
 - python setup.py install install the package
 - python setup.py develop install the package in development mode
 - python setup.py test install required test packages and run the tests
- Pypi useful commands:
 - pip install package_name install the package
 - pip uninstall package_name uninstall the package
 - pip install –r requirements.txt install of the requirements defined in requirements.txt
 - pip freeze > requirements.txt write all the requirements with their versions in requirements.txt

- Style Guide Python <u>PEP8</u>:
 - Naming:
 - Classes: CamelCase
 - Functions/Variables/Modules/Packages: lower_underscore
 - Constants: CAP_UNDERSCORE
 - Protected Attribute: _start_underscore
 - Private Attribute: __double_underscore (to avoid)

Logging

- The Logging Module
 - Pros:
 - Level of logging
 - Timestamping fairly easy
 - Encourages debug logging => facilitates debugging
 - Cons:
 - Proper set-up is hard
 - Configuration is required to have logging

Logging

- The Logging Setup
 - In libraries:
 - In your root __init__.py, add the following to avoid error message is no logging is configured:

```
logging.getLogger(__name___).addHandler(logging.NullHandler())
```

• Every time, you create a logger, it should follow the template:

```
logging.getLogger(__name__ + "XXX")
```

- In applications:
 - Choose a configuration option (only one):
 - File: ini or json (ini recommended):
 - Limited Configuration
 - End-user can modify it (Especially logging level)
 - Code configuration:
 - More options
 - Inaccessible for end-user

Unit-Tests

- Main Modules
 - <u>Unittest</u>: Part of Python standard library
 - Nose: Extend the capabilities of Unittest (Mainly: Python 2)
 - Pytest: Extend the capabilities of Unittest (Mainly Python 3)

Example and Questions