Environments for Python

Instructor: Jorge Mendez
Logistics

• Homework 3 is out
  • Due Friday

• No homework this week
Modules
Scripts

- .py files
- May contain function and class definitions
- Are executed by running `python3 script_name.py`
Modules

• Also `.py` files
• Contain function and class definitions
• Can be imported to other modules or scripts
• Main module: top level
• Access the name of the current module with variable `__name__`
More on modules

• May contain executable statements
  • Are run the first time they are encountered upon importing the module
  • Also run if executing the module as a script (e.g., python3 module_name.py)

• Modules define a namespace
  • Global variables live within the module
  • Access them from outside via module_name.var
Importing modules

- Import statements usually at the top
  - `import module_name`
  - Access functions via `module_name.function`
  - `from module_name import function`
  - Access functions directly via `function`
  - `from module_name import *`
  - Import all names that don’t begin with `_`
  - Not recommended!
  - `import module_name as mn`
  - Access functions via `mn.function`
  - `from module_name import function as f`
  - Access functions via `module_name.f`
Modules as scripts

• *if* __name__ == '__main__'*

• Executable statements only run if it is main module
  • Not if imported as module!

• File can serve as module *and* script

• Pass in arguments with *sys.argv*
  • `python3 script_name.py arg1 arg2 ... argN`
  • `sys.argv[0]` — script_name
  • `sys.argv[1:N]` — arg1:N (as strings)

• Must import *sys*
Packages

• Collection of modules organized in directory structure
• Each directory must contain a `__init__.py` file for Python to treat them as packages
  • May be an empty file or contain initialization code for the package
Examples

- import sound.effects.echo
- from sound.effects import echo
- from sound.effects.echo import function
Package managers and virtual environments
pip

• Install packages from the Python Package Index and other indices
• Comes pre-installed with Python >= 3.4
  • `pip install package_name` — latest version
  • `pip install package_name==1.0.4` — specific version
  • `pip install package_name>=1.0.3` — minimum version
  • `pip install -r requirements.txt` — from requirements file
• Packages installed this way are available to import in Python
• Can also install local packages with `pip`
conda

- **pip** only installs Python packages
- **conda** can install non-Python packages as well
- Can be installed with Anaconda or Miniconda
  - Anaconda includes a large variety of scientific packages
  - Miniconda only includes conda and its dependencies
- **conda install package_name**
- **conda install package_name=1.0.4’**
- Packages installed this way are also available to import in Python
pip and conda

• For complex projects with many dependencies, you might need to use a combination of both

• Miniconda and Anaconda both include pip
  • Should be mostly compatible with conda
Virtual environments

• What if different application require conflicting versions of a package?
• Can’t install everything to `/usr/lib/python3.6/site-packages`
• Virtual environments have their own installation directories
• Work on different projects or use different applications within their own virtual environments
venv

• Installed with python

• `python3 -m venv path/env_name` — create environment

• `source path/env_name/bin/activate` — activate environment (Unix)

• `pip` is installed into the environment
virtualenv

- `pip install virtualenv`
- `virtualenv path/env_name` — create environment
- `source path/env_name/bin/activate` — activate environment (Unix)
- `pip` is installed into the environment
conda

- Installed with Miniconda or Anaconda
- conda create --name env_name python=3.7
- conda activate env_name or source /anaconda3/envs/env_name/bin/activate
- Install pip into environment via conda install pip
Recommended for the course (and in general)

- Use conda
- Install Anaconda if you are not terminal-savvy
- Create a new environment for each homework to follow
- Great package management and tends to work well with pip
- Environment management is easy
- Certain packages (e.g., PyTorch) just work better with conda
Python development environments
Why use and IDE?

- Using text editors (Notepad++, Sublime Text, Vim) gives you limited functionality
- It gets complicated to
  - Navigate large codebases
  - Debug faulty programs
- IDEs also give you nice features like code completion, highlighting...
Python interactive shell

• Simplest way to run Python code
• Runs instructions immediately
• Good for testing out simple sets of instructions
IPython

• Interactive Python shell (beyond Python’s default)
• Can use as a drop-in replacement for Python
• For our purposes
  • Used as the kernel for debugging in Spyder
  • Used as the kernel of Jupyter notebooks
Jupyter

• Comes pre-installed with Anaconda
• Can run within virtual environment
  • conda activate env; conda install jupyter; jupyter notebook
• Very common in scientific communities (e.g., data science)
• To execute: jupyter notebook
• Create notebooks directly on browser
• Code is separated in cells
  • Raw
  • Markdown
  • Code
PyCharm

• Lightweight version is free
• Support for web development tools
  • JavaScript, HTML/CSS, Angular JS, Node.js
• Powerful debugger with graphical interface
• Plenty of support for virtual environments
PyDev on Eclipse

• Pros: it runs on Eclipse. Cons: it runs on Eclipse
• Very bulky and heavy, as anything on Eclipse
• Probably the most powerful IDE for traditional programming
• Support for Django projects, hotkeys, and configurations
Spyder

• Pre-installed with Anaconda
• Designed for scientific computing (data science, specifically)
• May run on IPython
• Debugger interacts very well with NumPy
• Support only for conda environments through Anaconda
Debugging without an IDE

• Python comes with a built-in debugger pdb
• Supports
  • Breakpoints
  • Line-by-line stepping
  • Inspecting stack frames
• It is messy to use, it is undocumented, it is not graphical
• There are other similar debuggers (e.g., pydb)
Takeaways

• Files can act as scripts and/or modules
• Bundle modules together into packages
• Install packages with conda or pip
• Create virtual environments (preferably with conda)
• Different IDEs with different functionality