Scientific Computing @ CLASSE

Developing python code

2025-06-11

Overview

- Managing python environments for development
- Introduction to jupyter
- Best practices: docstrings and style

Developing python code

Managing environments

System-wide default python

- No version options
- No permission to install packages like numpy, scipy, matplotlib, etc.
- Can’t keep track of the environment requirements for *your* project

Your own python environment

- Use any version of python
- Permission to install any packages your project needs
- Easy to keep track of the environment requirements for *your* project

Developing python code

Managing environments

`conda`

- Choose any python version
- Install packages, system libraries
- Install packages published for `conda` or `pip`
- Can be slow

`venv`

- Must use the system default python version
- Install only packages published for only `pip`
- Usually faster

Developing python code

Exercise 1

Install an instance of miniforge (which includes the conda CLI tool)

1. Open a terminal on the CLASSE Linux system using your preferred method

2. Change directories to an appropriate working directory (for CHESS students: /nfs/chess/user/<CLASSE-ID>)

3. Run: curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh"

4. Run: chmod u+x Miniforge3-Linux-x86_64.sh

5. Run: /Miniforge3-Linux-x86_64.sh

    a. Accept license agreement

    b. Specify new install directory (for CHESS students: /nfs/chess/user/<CLASSE-ID>/miniforge)

    c. DO NOT allow the installer to update your shell profile to automatically initialize conda

2. Run: source /nfs/chess/user/<CLASSE-ID>/miniforge/bin/activate

Developing python code

Exercise 2

Create and activate an environment that contains the latest version of python. In the same terminal used for exercise 1, run:

1. `which python`

2. `conda create -n <your CLASSE ID here> python`

   a. respond to “`Proceed ([y]/n)?`” prompt with “`y`”

3. `conda activate <your CLASSE ID here>`

4. `which python`

5. Note the different outputs from the 1st and 4th commands!

Developing python code

Exercise 3

Execute some python code in your new environment. In the same terminal used for the previous two exercises, run:

1. `echo "print('hello world')" > helloworld.py`

2. `python helloworld.py`

Developing python code

Exercise 4

Install a package in your new environment. In the same terminal used for the previous three exercises, run:

1. `python -c "import numpy"`

2. `conda install -y numpy`

3. `python -c "import numpy; print('import successful')"`

Developing python code

https://jupyterhub.classe.cornell.edu

- Another way to interact with the CLASSE filesystem
- Open a terminal, create / edit files, or use *jupyter notebooks*
- Notebooks can be a friendly option for developing python code if you're not comfortable using the terminal, but…
- There's a time and place for notebooks. Your project mentor can tell you if a jupyter notebook is an acceptable form for the final version of your code
- https://wiki.classe.cornell.edu/Computing/JupyterHub

Developing python code

https://jupyterhub.classe.cornell.edu

- `jupyterhub`'s file browser provides access to your `/home/<CLASSE-ID>` directory
- Recall: your work belongs somewhere in `/nfs/`…, NOT `/home/<CLASSE-ID>`
- Solution: use the symbolic link created in an exercise from the previous section to navigate to an appropriate directory for your project files
- To make a python environment available for use *in jupyter*, one must install an "ipykernel" for it.

Developing python code

Exercise 4

Install an ipykernel for the environment you created in exercise 2

1. `pip install ipykernel`

2. `python -m ipykernel install --user --name=my-python-env --display-name "My Python Env"`

3. In https://jupyterhub.classe.cornell.edu, open a new python notebook in select “My Python Env” for the kernel

Developing python code

Exercise 5

Write and run some code in your new python notebook. The code should print "`hello world`" when the cell is run.

Developing python code

Best practices — docstrings

- Docstring conventions — https://peps.python.org/pep-0257/
- Every module, class, and function in your project should have a docstring
- Docstrings should be written with python's built-in `help(object)` function and automatically-generated human-readable API documentation in mind
- Pick a canonical format for your docstrings and stick with it. If you don't already have one picked out, I recommend choosing the sphinx docstring format
- TIP: write docstrings for each module, class, and function BEFORE you write the actual code that goes inside
    - …but if you don't, ChatGPT usually does a good job at writing docstrings if you ask nicely

Developing python code

Docstrings & help(obj) example

```python
def func_no_docstring(x, y=None):
    # Describe what the function does.
    # describe what the argument `x` represents
    # describe what the argument `y` represents
    # describe what this function returns
    return None
```

```
> help(func_no_docstring)
Help on function func_no_docstring in module __main__:

func_no_docstring(x, y=None)
```

```python
def func_with_docstring(x, y=None):
    """Describe what the function does.

    :param x: describe what the argument `x` represents
    :type x: object
    :param y: describe what the argument `y` represents,
        defaults to None
    :type y: object, optional
    :return: describe what this function returns
    :rtype: object
    """
    return None
```

```
> help(func_with_docstring)
Help on function func_with_docstring in module __main__:

func_with_docstring(x, y=None)
    Describe what the function does.

    :param x: describe what the argument `x` represents
    :type x: object
    :param y: describe what the argument `y` represents,
        defaults to None
    :type y: object, optional
    :return: describe what this function returns
    :rtype: object
```

Developing python code

Best practices — style

- Style conventions — https://peps.python.org/pep-0008/
- Variable / function names: `snake_case`
- Class names: `CamelCase`
- Line widths: 79 or 99 characters (code) or 72 characters (comments & docstrings)
- …and many more guidelines that you are encouraged to follow
- You may break a guideline in PEP8 “when applying the guideline would make the code less readable, even for someone who is used to reading code that follows this PEP.”

Developing python code

Exercise 6

Copy the following code into a cell in your new jupyter notebook, then edit it so that it adheres to best practices:

```
# Prompt: problem 1 from Project Euler https://projecteuler.net/problem=1
def Sum_MultiplesofThreeor_five_below(n):
    Result=0
    for I in range(n):
        if I%3 == 0 or I % 5==0:
            Result +=I
    return Result
print(Sum_MultiplesofThreeor_five_below(1000))
```