python

Maher Khan

Recitation 5
Date: 5th October, 2018
slides and codes: https://github.com/maher460/Pitt-CS1520-Recitations
Office hours today from 6:00 PM to 8:00 PM

Today...

- Installing Python, PIP and virtualenv
  - for Windows, mac OSX and Unix/Linux
- Structuring your Python project
- Some cool things you can do in python
  - returning multiple values
  - zipping
  - functional programming: map, filter and reduce

installing Python, PIP and virtualenv

You want all three of these to be accessible as commands in your Terminal/bash/shell/CMD/PowerShell/etc.

install Python

Download and install from here:
https://www.python.org/downloads/
(Make sure PATH option is selected if available!)

setting Path for Windows

- In case the “python” command does not work on your CMD/ PowerShell, then you have to fix your PATH
  How-to:
  https://www.pythoncentral.io/add-python-to-path-python-is-not-recognized-as-an-internal-or-external-command/

- Access "System Settings" from your Control Panel.
- Click on the "Advanced" tab.
- Click on the "Environmental Variables" button on the bottom of the screen.
- Click the "New" button under the "System Variables" section.
- Type "PYTHONPATH" in the "Variable" field. Type the path for Python modules in the value field. Click "OK" when you are finished setting the PYTHONPATH environmental variable

setting PATH for mac OSX

- Open the shell script that runs every time you access your terminal in a text editor.
- In Mac OS X environments, the file is called “.profile.”
- Type:
  PYTHONPATH = "$ {PYTHONPATH} : /path/where/python/package/is/ located/ export PYTHONPATH
- For Mac OS X a typical path is “/Library/Frameworks/Python.framework/ Versions/2.7/lib/python2.7/site-packages".
- Save the file. Changes to your path will take effect when you start a new shell.

setting PATH for Unix/Linux

To add the Python directory to the path for a particular session in Unix –

- **In the csh shell** – type setenv PATH "$PATH:/usr/local/bin/python" and press Enter.

- **In the bash shell (Linux)** – type export ATH="$PATH:/usr/local/bin/python" and press Enter.

- **In the sh or ksh shell** – type PATH="$PATH:/usr/local/bin/python" and press Enter.

- **Note** – /usr/local/bin/python is the path of the Python directory in this example

Why Pip?

- It allows you to install packages for python very easily!
- Just: “pip install <whatever>”
- Done!

install PIP

- pip is already installed if you are using Python 2 >=2.7.9 or Python 3 >=3.4 downloaded from python.org

- If it is not installed, then follow the instructions here: https://pip.pypa.io/en/stable/installing/

When you are done, you should have both “**python**” and “**pip**” available as commands in your terminal/CMD/PowerShell/bash/etc.

Why virtualenv?

- Multiple projects using the same package, but different versions
- Updating the package will update it for all, so it might break your projects
- virtualenv creates an isolated python environment for each project
- You can save and load the “state” (dependencies, package versions, etc.) for each virtualenv project.

virtualenv on Windows

In your Command Prompt enter:
pip install virtualenv

In your Command Prompt navigate to your project:
cd your_project

Within your project:
virtualenv env

On Windows, virtualenv creates a batch file: \env\Scripts\activate.bat

To activate virtualenv on Windows, activate script is in the Scripts folder :
\path\to\env\Scripts\activate

Example:
C:\Users\’Username'\venv\Scripts\activate.bat

More help: https://programwithus.com/learn-to-code/Pip-and-virtualenv-on-Windows/

virtualenv on mac OSX

- Install virtualenv via pip:
  pip install virtualenv

- Test your installation
  virtualenv --version

- Create a virtual environment for a project:
  cd my_project_folder
  virtualenv my_project

- You can also use the Python interpreter of your choice (like Python 2.7)
  virtualenv -p /usr/bin/python2.7 my_project

- To begin using the virtual environment, it needs to be activated:
  source my_project/bin/activate

- If you are done working in the virtual environment for the moment, you can deactivate it:
  deactivate

saving state of packages

- In order to keep your environment consistent, it's a good idea to "freeze" the current state of the environment packages. To do this, run:

  pip freeze > requirements.txt

- Later it will be easier for a different developer (or you, if you need to re-create the environment) to install the same packages using the same versions:

  pip install -r requirements.txt

More help on python, pip, virtualenv and virtualenvwrapper: https://docs.python-guide.org/dev/virtualenvs/

structuring your Python project

structuring your Python project

- my_project/
  - .gitignore
  - requirements.txt
  - Readme.md
  - docs/
  - src/
    - my_project_venv/ //should be omitted in .gitignore
    - my_project/
      - main.py
      - code_folder_1/
        - __init__.py
        - code_file_1.py
        - code_file_2.py
      - code_folder_2/
        - __init__.py
        - code_file_3.py

Some cool stuff in python

return multiple values from function call

```
def some_func():
        return 1, 2, 3, 4, 5

a, b, c, d, e = some_func()
print a  #1
print b  #2
print c  #3
print d  #4
print e  #5
```

store all values of the list in new variables

```
cool_list = [1, 2, 3, 4]
a, b, c, d = cool_list

print a    # 1
print b    # 2
print c    # 3
print d    # 4
```

zipping two lists

```
list_1 = [“a”, “b”, “c”, “d”]
list_2 = [“p”, “q”, “r”, “s”]
list_3 = zip(list1, list2)
# list_3 = [(“a”, “p”), (“b”, “q”), (“c”, “r”), (“d”, “s”)]
```

map

- blueprint:
  map(function_to_apply, list_of_inputs)

- traditional example:

```
def square_func(x):
        return x ** 2

items = [1, 2, 3, 4, 5]
squared = []
for i in items:
    new_val = square_func(i)
    squared.append(new_val)
# squared = [1, 4, 9, 16, 25]
```

- better example:

```
items = [1, 2, 3, 4, 5]
squared = list(map(lambda x: square_func(x), items))
# squared = [1, 4, 9, 16, 25]
```

filter

- blueprint:
  filter(function_to_apply, list_of_inputs)

- example:

```
number_list = range(-5, 5)
# number_list = [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]

less_than_zero = list(filter(lambda x: x < 0, number_list))
print(less_than_zero)
# Output: [-5, -4, -3, -2, -1]
```

reduce

- blueprint:
  reduce(function_to_apply, list_of_inputs)

- traditional example:

```
product = 1
list = [1, 2, 3, 4]
for num in list:
    product = product * num
# product = 24
```

- better example:

```
from functools import reduce
product = reduce((lambda x, y: x * y), [1, 2, 3, 4])
# Output: 24
```

Questions?