pep8 = a style guide for Python code: https://www.python.org/dev/peps/pep-0008/
pep8 linter = checks your code against style guide standards: http://peponline.com/

Make sure you're directories don't have spaces in the name (i.e. SI_507 vs. SI 507)
$ atom = opens current directory (folder) in atom, nice!

Navigating directories (i.e. Paths):
$ cd = change directory
$ cd ~ = home directory (shortcut)
$ cd ~/Desktop = change directory to desktop
$ cd .. = go back one directory
$ cd ../.. = go back 2 directories, and so on
$ pwd = present working directory = what directory am I in now?

Once you're in a directory:
$ ls = lists all FILES and FOLDERS within that are in a directory
$ ls ~Desktop or $ ls ~Directory1/Directory2/AndSoOn = lists all files and folders WITHOUT CHANGING directory you are currently on

Unix commands for utility (i.e. More than just navigating):
$ mkdir = **make directory** (can either do in the pwd OR route to where u want to make this new file)
$ mkdir ~/Desktop/507/folder/**sub_folder** = makes **sub_folder** in that location without having to navigate to it via the command prompt

$ cp = **copy (can only copy FILES, not FOLDERS i.e. directories)**
$ cp <path of what you want to copy> <path of location you want that file copied to>
$ cp ~/Desktop/507/folder/test.txt ~/Desktop = copies test.txt from internal folder to Desktop
$ cp ~/Desktop/507/folder/test.txt **.** = the **period (.).** at the end means copy HERE (to pwd)
$ cp ~/Desktop/507/folder/test.txt **..** = copy ONE LEVEL UP from HERE (pwd)
$ cp ~/Desktop/507/folder/***** ~/Desktop = copy ALL FILES from 507 internal folder to new location

Using unix to see inside FILES:
$ cat sample.txt = once you're in a directory, you can see the contents of that file
$ less sample.txt = say that file has a lot of stuff in it, 'less' lets you move forward and back using the ':' Use the arrow buttons or space bar to see more
$ q = quits special text viewer in terminal

$ cat sample1.txt sample2.txt = concatenates or combines the contents of two files together

Other important things to remember:
*If a path you specify does not begin with a ~ or /, then it is a **relative path** (i.e. Sub directory of current directory OR **ONE STEP** and one step only removed from where are you are now)
Example: I'm in Documents folder and want to navigate one level down
$ cd Documents or $ cd Documents**/**
$ cd Courses or $ cd Courses/

VIRTUAL ENVIRONMENTS:

 # pip is a program tied to Python interpreter >> install from PyPi to my Python! >> like downloading movies from Amazon. Except instead you're downloading code you got for free

pip install <> Amazon store for programmers :)
$ pip3 freeze = shows all the Python libraries you've installed in global environment
$ pip3 freeze = " for python 3
$ pip3 install XXX = command for installing python libraries that are a part of the **Python Package Index**
$ pip3 install requests

Leverage pip install in a clean way using Virtual Environments for projects!
$ pip3 install virtualenv = you now have the library that allows you to make and use VE's

Step 1. Make a Directory for a Project:
$ mkdir ~/Desktop/507/test_project
Step 2. Create a Virtual Environment in that directory:
$ virtualenv --python=python3.6 name_of_virtualenv
^ when creating a VE, you're making a mini-Py interpreter, so tell it which version of Py you want it to use!
Step 3. Activate that Virtual Env. WITHIN that project directory:
$ source ~/Desktop/507/test_project/name_of_virtualenv/bin/activate
command prompt should now look like this:
(name_of_virtualenv) m-########:VirtualEnvs jucruz$
^ this is the command prompt telling you your virtual env is ready to be used/filled!
Step 4. Install and necessary project libraries:
(name_of_virtualenv)....$ pip install library_name_here
or
(name_of_virutalenv)....$ pip install -r requirements.txt
Step 4a. Can see which libraries you have installed in a VE using **pip freeze**
Step 5. Once you're done, Deactivate the VE:
(name_of_virtualenv)...$ deactivate

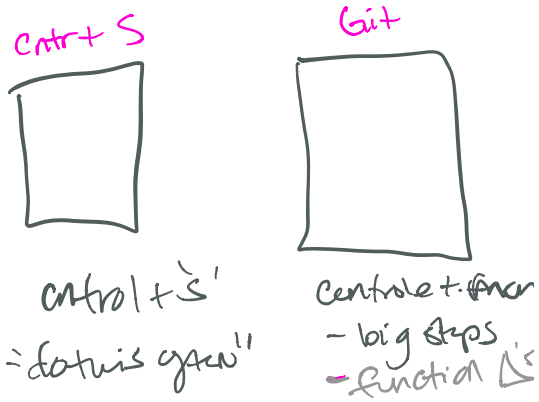
#see installed packages while virtualenv is activated:
pip freeze

save installed packages in requirements.txt
pip freeze > requirements.txt

#install packages from requirements.txt into your current environment
pip install -r requirements.txt

NOTE: when using git commit to changes, DON"T commit your VE because a VE is specific to YOUR operating system, and the folder would look very different for someone else.

So put your venv_folder in .gitignore



GIT & GITHUB:

- version control system
- software that acts as layer on top of files
- github = website

Committing your changes on YOUR device

- **mkdir new_repo** (start a new project directory)
- **cd new_repo** (navigate to that directory)
- **git init** (makes the current directory into git repository)
  - creates invisible .git directory within project folder
  - all other files are the user's
- now say you add files/make changes to the files in new_repo
- **git add file1.txt file2.txt folderX** (add the files and folders such that git can index them)
  - GIT saves these files in its invisible directory .git/index
  - Any files you want to ignore can go under .gitignore (file)
  - or use **git add .** to add all recent files in THAT project folder
- **git status** (shows you what files/folders were modified and which were created but have yet to be added via the git add command)
- **git commit -m "type your commit message here"** (now git has committed or taken a screenshot of the current state of your new_repo)
- **git log** (shows you a log of ALL your commits with your corresponding message)
- **git diff** (shows one level further: all + and - changes made to FILES in new_repo)
- **ls -a** (shows all the VISIBLE and INVISIBLE files you have in that repo, including .git repos)

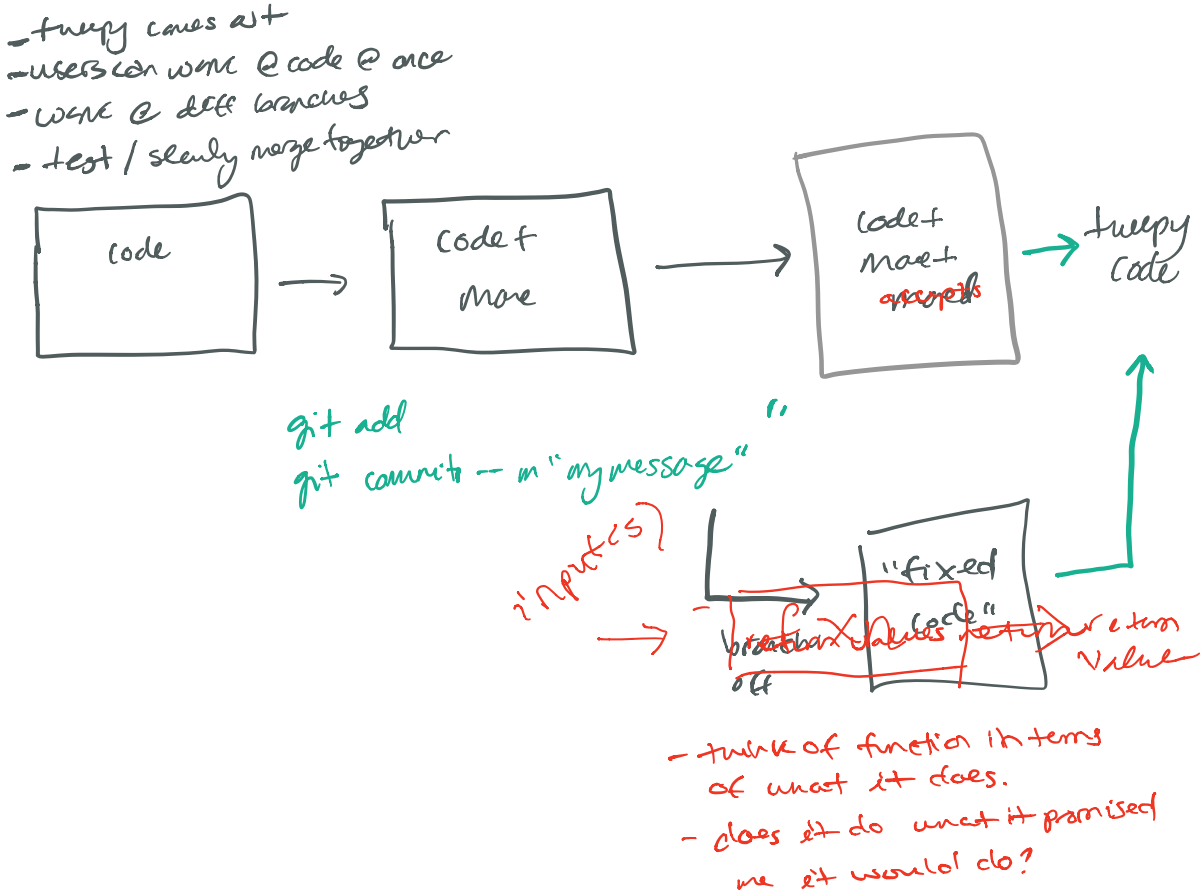
Pushing personal GIT repo to/from GITHUB:

- **git clone + URL** transfers github repo TO your computer
- to transfer a git repo FROM your computer, go to github.com create new repository
- Push an existing respository from the command line
- **git remote add origin + link**
  - origin = the link between MY git respository and its "remote place" on the internet (i.e. GitHub)
- **git push -u origin master**
  - push all of the data in this git repo to remote place (origin)
  - you know its successful when you see:

- **Branch master set up to track remote branch master from origin**

Cloning and Forking from GitHub to YOUR personal computer
- github.com and find repo you want to work on/add to/edit, etc.
- If its YOUR OWN: clone it
- If its NOT YOUR OWN: fork it
- copy cloning URL
- **git clone + URL** in command prompt @ cd project_folder
  - now can make changes on local device and continue PUSHING them (if its your own github repo) to GitHub

CLONING from GIT HUB to YOUR COMPUTER:
- go to repo on github.com
- Fork
- Clone repo onto your computer
- Work on those files



UNIT TESTING

- **unittest =** Python has a built-in module for writing and running test cases
- **import unittest**
- **class Name_Test_Case_Section**
    - **def test_name_of_specific_test(self):**
        - **self.assertEqual(condition_you_are_testing, expected_result, "text here")**
    - **ect.**
    - **ect.**
    - **def tearDown(self):**
        - close any files you opened in setup or in tests!
- **unittest.main(verbosity=2)**
-
- Return value test = testing that function in code returns a certain value
- Side effect test = a function doesn't return, but rather a mutable object, list or dict, is modified

Def teardown(self):

- close any files that you opened in setup
- Test environment same as when left

Steps:
#function takes in integer, returns int - 1

- import unittest
- Class FunctionTests(unittest.TestCase):
    - def setUp(self):

Process:
Describe in English what the function/class methods are supposed to do:
What are these methods supposed to return in a Class or Function?