Assignment 1 – Hello World

Reading:

- https://techterms.com/definition/interpreter
- http://pythoncentral.io/execute-python-script-file-shell/

Goals:

- Execute some commands in the Python interpreter
- Write a simple python script and run it

Interpreter:

1. Execute some commands in the Python interpreter
   1. Open the python interpreter and you should be greeted with some version information, followed by a prompt such as “>>> ”
   2. The interpreter is like the linux command line, but instead of running a shell like bash, it’s running python. The interpreter is a great way to experiment or test functionality while writing a script.
   3. Try some simple commands, for example:
      1. `2 + 2`
      2. `x = 3`
      3. `y = 5`
      4. `x * y`
2. Get the interpreter to say “`Hello World!`” as output (i.e. print a string of characters)

Scripts:

- hello.py
  - Input: none
  - Output: “`Hello World!`”
  - Extra Credit: print it in a different color

Assignment 2 – Copycat

Reading:

- https://www.learnpython.org/en/Modules_and_Packages
- https://www.computerhope.com/jargon/p/positional-parameter.htm

Goals:

- Use the `help()` and `dir()` functions
- Import a module and explore it
- Print the text passed on the command line

Interpreter:

1. Run “`import sys`”
    1. This imports the “`sys`” module (also called a package or library)
    2. Modules give you access to functions you might need to do more complicated tasks
    3. Just put “import sys” at the top of a python script to have access to it in that script
2. Run “`dir(sys)`”
    1. This lists all of the functions and variables available in the “`sys`” module
    2. For example, if you see “`argv`” in that list, you would access it with “`sys.argv`”
3. Run “`help(sys)`” and look through it, then press “Q” to leave
    1. This shows some documentation for the object you call it on, in this case the documentation for the “`sys`” module itself
    2. This is very useful for functions as well; try “`help(sys.exit)`”

Scripts:

- copycat.py
    - Input: some text on the command line
        - e.g. “`python copycat.py text to print`”
    - Output: the same text printed to the command line
    - References:
        - https://docs.python.org/3.10/library/sys.html#sys.argv
    - Extra Credit: if given an input including double quotes, preserve them

Assignment 3 – Number Translator

Reading:

- http://www.afterhoursprogramming.com/tutorial/Python/If-Statement/
- http://effbot.org/zone/python-list.htm
- http://www.pythonforbeginners.com/dictionary/how-to-use-dictionaries-in-python
- http://interactivepython.org/courselib/static/thinkcspy/Functions/mainfunction.html

Goals:

- Use if statements, lists, and dictionaries
- Get user input from the command line

Interpreter:

1. Write an if statement
   1. Enter the following "`if 3 < 5:`" and press enter
   2. You'll be greeted with "`...` " on the next line, which is how the interpreter allows you to enter multi-line statements
   3. Press space twice to indent, then type "`print('yes')`" and press enter twice
   4. The interpreter will print "`yes`"
   5. You can also modify this behavior and get more control with "`elif`" and "`else`"
2. Create a list
   1. Run "`mylist = [2, 5, 6]`" which creates a list with some numbers in it
   2. Run "`mylist[1]`" to get the second element in the list (the first index is 0)
   3. Run "`mylist[0] = 3`" to change the first element in the list (print the list to confirm)
   4. Run "`mylist.append('a')`" to add an element to the end of the list (print to confirm)
   5. Run "`mylist.index('a')`" to search the list and return the index of the character "`a`"
3. Create a dictionary
   1. Run "`mydict = {'tacos': 'good', 'mushrooms': 'bad'}`" to create a dictionary
   2. Run "`mydict['tacos']`" to get the value that corresponds to the "`tacos`" key
   3. Run "`mydict['dozen'] = 12`" to add a new entry (print the dict to confirm)
4. Get user input
   1. Run "`x = input('Enter something: ')`" and type something into the prompt
   2. Confirm that "`x`" now has the text you just typed by printing it

Scripts:

- num_translate.py
  - Input: user enters number names (e.g. “`one`”, “`two`”) when prompted
  - Output: those names converted to numerals (e.g. “`1`” ,“`2`”)
  - Functions:
    - `translate_if(text)` returns translated text by using if statements
    - `translate_dict(text)` returns translated text by using a dictionary
    - `translate_list(text)` returns translated text by using a list
    - `main()` prompts the user for input, then executes all three functions and prints the result from each
  - References:
    - [https://docs.python.org/3.10/library/functions.html#input](https://docs.python.org/3.10/library/functions.html#input)
  - Extra Credit: make the script run forever until the user enters “`exit`” at the prompt

Assignment 4 – Secret Code

Reading:

- http://pymbook.readthedocs.io/en/latest/file.html
- https://en.wikipedia.org/wiki/ROT13

Goals:

- Read and write files
- Manipulate strings

Interpreter:

1. First we'll write some text to a new file:

   ```
   >>> with open('temp.txt', 'w') as f:
   ...   f.write('blah blah blah')
   ```

2. Open that file in a GUI text editor and confirm your text is there
3. Back to the interpreter, let's print the contents of that file:

   ```
   >>> with open('temp.txt', 'r') as f:
   ...   print f.read()
   ```

4. Strings are lot like lists in how you can work with them
   1. You can index them, for example "`print('abcdefg'[4])`"
   2. You can combine them "`print('a' + 'b' + 'cde')`"
   3. You can iterate over them as well:

      ```
          >>> for x in 'abc':
      ...  print (x)
      ```

Scripts:

1. rot13.py
   1. Input: name of a file as a command line argument
   2. Output: run ROT13 on the text in the file and write it back to the original file
   3. Extra Credit: If the file doesn't exist, print an error message for the user

Assignment 5 – Send an E-mail

Reading:

- https://docs.python.org/3.10/library/email.examples.html

References:

- https://docs.python.org/3.10/library/email.html#module-email
- https://docs.python.org/3.10/library/smtplib.html#module-smtplib

Goals:

- Write a script that sends yourself an e-mail

Interpreter:

1. Make sure you can log in to an e-mail account
    1. You can use any account you'd like, although I have a test account you may use:
        1. Username: notify@jahschwa.com
        2. Password: located in this file on grandline `/home/share/notify.txt`
2. Running through the first example from the link in the "Reading" section in the interpreter is probably a good idea

Scripts:

- send_mail.py
    - Input: text to send via command line
    - Output: e-mail sent to yourself
    - Extra Credit: make this modular so other scripts can import your e-mail sending function or object and use it themselves

Assignment 6 – Write a Sibyl Chat Command

Reading:

- https://github.com/TheSchwa/sibyl/blob/master/README.md
- https://github.com/TheSchwa/sibyl/wiki
- https://github.com/TheSchwa/sibyl/wiki/Dev
- https://github.com/TheSchwa/sibyl/wiki/Dev-alarm

References:

- https://github.com/TheSchwa/sibyl/wiki/Dev-Plug-Ins
- https://github.com/TheSchwa/sibyl/wiki/Dev-Decorators

Goals:

- Write a chat command for the Sibyl bot

Suggestions (in vague order of increasing difficulty):

- Randomly print the name of someone in the current room
- Convert roman numerals into decimal numbers
- Send an e-mail using sibyl from a chatroom
- Get the title of a webpage
- Roll some dice (e.g. “4d6” or “3d8+5”)
- Anything you want!

Interpreter:

- You can test logic in the interpreter, for example write the chat command function before plugging it in to sibyl and make sure it works correctly

Script:

- Whatever filename you’d like
    - Input: the text the user entered will be passed to your function by sibyl
    - Output: sibyl will say in the chatroom whatever you return from the function
    - Extra Credit: make a config option in your plugin

Assignment 7 – Battleship

Reading:

- https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/
- https://www.python-course.eu/python3_magic_methods.php

References:

- https://grandline.jahschwa.com/files/battle_spec.py

Goals:

- Write a single player game of Battleship with some level of AI played via command line
- Edit the referenced file directly, completing all instructions marked with `[TODO]`
- Remove the `[TODO]` markers as you go so you can use your editor's find feature to locate things you still need to do

Interpreter:

- This will be useful for testing the behavior of the classes you define

Script:

- battleship.py
  - Input: locations on the board to hit
  - Output: ASCII representations of both boards and status messages
  - Classes:
    - Point
    - Ship
    - Board
    - AI
  - Functions:
    - `main()` plays the game forever
    - `play()` handles playing one game, alternating turns, and checking for winner
    - `take_turn(ai_board, ai, turn)` handles a single turn for player or AI
    - `print_boards(p1, ai, hide)` prints both boards, hiding the AI ships