OMIS 30: Intro to Programming (with Python)

Week 2, Class 1

Introduction to Programming
Instructor: Denis Vrdoljak

Learn programming with Python

Goals for this week

This week, we'll introduce the basics of Python. We'll cover the basic format, basic variable types, how to print to standard output and take user inputs, and iterable types (like strings, lists, and dictionaries). Finally, we'll introduce conditional statements, or IF statements, and loops.

By the end of this week, you should:

- Understand what dynamic typing means and how it works
- Understand primitives and iterables in Python
- Be able to index and slice strings and lists, and work with dictionaries
- Know some common methods for strings, lists, and dictionaries
- Know how to print to standard out
- Know how to take in a user input
- Control a program's flow with IF statements and WHILE loops

Office Hours

Instructor	Days available
Yuan Wang (our TA)	M 2:30-3:30p, W 9-10a
Mike Davis (other section's instructor)	Tu 9:30-10:20a, Th 12-1p
Denis Vrdoljak	Tu 3:40-4:40p, W 5-6p

Course Topics

- ~~Computer setup~~
- ~~Shell - ls, mv (and rename), cp, pwd,cd, '..',mkdir, rm, touch, echo~~
- ~~cat, pipe, output redirect (> and >>), 'python --version' (introduce args)~~
- **Python Basics - print, input**, math.
- Pseudo-code, algorithm design, comments
- **iterables (lists, sets, dicts, strings)**
- Loops, Nested Loops, Recursion
- Flow Control (If, else, elif, try, except)
- Functions
- **Strings, upper(), lower()**
- **indexing and slicing iterables**
- **lists, extending, appending**
- **mutability**
- Jupyter Notebooks

What is Python?

- High Level,
- Interpreted,
- Dynamically Typed,
- Object Oriented,
- programming language

Command Line Review

- Writing to a file
  - **> <file_name>** = push the output of that command to that file_name (and overwrites the file!!)
    - echo Hi > hi.txt
  - **>> <file_name>** = appends the output of that command to that file_name
    - echo How are you? >> hi.txt
- See contents of a file:
  - **cat <file_name>** = view the contents of that file_name

Command Line Review

- See contents of a file:
  - **less <file_name>** = view the full contents of that file_name, with scrolling for large files, press ‘q’ to exit (Linux/Mac tool, some versions of Windows CMD)
  - **more <file_name>** = view the contents of that file_name with scrolling for large files (Windows CMD/Dos tool, also ported to all Linux/Mac)
    - up/down arrows to scroll
    - **f** to page down, **b** to page up
    - **/<string>** to search for next occurence of “string”
    - **q** to exit
    - Example to show running processes: **ps -ef | more**
      - *(“|” is the vertical pipe above \)

Command Line: PC vs Mac/Linux

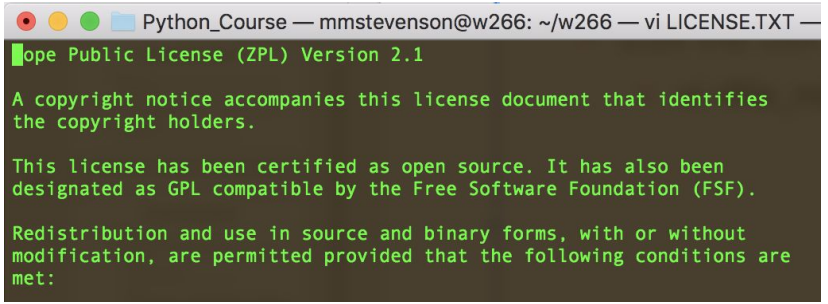
- echo
  - **Linux/Mac: echo “text goes here (parentheses need to be in quotes!)”**
  - **PC: echo text goes here (and parentheses are ok)**
- **| (pipe): pipe output of one command (eg, cat logfile.txt) to another command or script (eg, grep “error”)**
  - **E.g.: cat logfile.txt | grep “error”**

Command Line: Making an Executable

- At top of file:
  - #!/bin/sh
  - (This is called a “shebang”)
  - **Make executable (chmod +x script.sh)**
  - **Can now run as ./script.sh**
- **For more details/answers:**
  - **https://stackoverflow.com/questions/8779951/how-do-i-run-a-shell-script-without-using-sh-or-bash-commands**

POP QUIZ

- How do you exit out of vi or vim?



Vim text editor

- Edit the contents of a file:
  - **vi <file_name>** = enter Vim for a given file/directory
  - While cursor is blinking, you can edit the document directly
    - **Careful, edits are permanent once saved!
  - Hit escape (**esc**) or colon (**:**) to get out of edit mode
  - **:?<string>** to search for next occurence of “string”
  - **:help** for a full list of available commands
  - **:wq** to save edits (write) and quit
  - **:q** to exit Vim without saving changes

Primitive (Atomic) Variables in Python

- Integer (int)
- Float
- String (str)
- Boolean (bool)

Primitive Variables - OLD SLIDE

- Four basic data types (aka “primitives”):
  - Integer (int)
    - **2, 4, 6, …** whole numbers
  - Float
    - **2.5, 3.33, 7.67** … numeric values that include decimals
  - String
    - **“Cat”, “Dog”, “h7jb67”** … strings are often words
  - Boolean
    - **True, False, [0,1] , ==** ...tests of equality or existence

Integer

Positive or negative whole numbers with no decimal point

Examples: -1001, 0, 6

Python does not have a ‘limit’ on how small or large the number can be (min/max is only limited by the memory size of your machine)

Float

- Represent real numbers and are written with a decimal point dividing the integer and fractional parts.
- Floats may also be in scientific notation

Examples: -2.5, 3.1415, 7.67, 1e6

Bool (boolean type)

True or False

- **Equivalent to [1,0] (respectively):**
- **False == 0**
  - #evaluates to True; they are logically equal
- **True == 1**
  - # also evaluates to True; they are also logically equal
- **Can also add: 2 + True = 3**

Mutable vs Immutable

- Mutable = the item can be changed after created
- Immutable = the item cannot be changed after created
- All primitives are immutable

Iterable Data Types in Python

- String
- Ways to hold and group the primitive data types include:
  - List
  - Dictionary
  - Tuple
  - Set

String

- A string in Python is a sequence of characters
- It is a derived data type
- Can use “” or ‘’ to designate
- Strings are immutable

Examples: “w”, “Dog”, “h7jb67”, ‘four’, ‘4’,

“The cat in the hat.”

String Methods and Operations

- str(), " " or ' ' - creates an empty string variable
- There is no char type, just a string with length 1
- Concatenation
  - “mystring” + “anotherstring” → “mystringanotherstring”
- “mystring” * 3 → 'mystringmystringmystring'
- """ """ - triple quotes, used for block comments on multiple lines
  - Example: """This is a long string that is
    split over two lines """

More String Methods and Operations

- .upper(), .lower(), .capitalize(), .title()
  - mystring = ‘Abc dEf’
  - mystring.**upper()** = ‘ABC DEF’
    - Uppercase all letters
  - mystring.**lower()** = ‘abc def’
    - Lowercase all letters
  - mystring.**capitalize()** = ‘Abc def’
    - Capitalizes first letter of first word
  - mystring.**title()** = ‘Abc Def’
    - Capitalizes first letter of each word

Advanced String Methods and Operations

- **str.strip()**
  - Used to remove ’unwanted’ characters from the start & end of a string:
    - mystr = "----Our string we want----"
    - mystr.strip(‘-’) = ‘Our string we want’
- **str.split()**
  - Used to split a string into multiple items on a character - returns a list of those strings
    - mystr = “eat, drink, sleep”
    - mystr.split(‘,’) = [‘eat’, ‘drink’, ‘sleep’]

Indexing

- An iterable is any data type that can be used in a sequential fashion to find the next item, which includes string, list, tuple, dictionary, etc.
- We use the iterable property when searching through the various items to find a specific item, which is called indexing:
- >> mylist = [“the”, “cat”, “in”, “the”, “hat”]
- Pythno is ‘zero-based’ so indexing for the first item:
  - >> mylist[0]
  - Returns “the”

More Indexing

- mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[1] -> “cat”
  - mylist[-1] -> “hat”
  - mylist[-4] -> “cat”
- mystr = ‘python’
  - mystr[0] = ?
  - mystr[-1] = ?

Slicing

- To call up a subset/part of a list, we use a slice
- Slice syntax = [# to start with, # to end on (does not include): step]:
  - If either of the first two numbers are left blank - defaults to the start or end of the iterable
  - If the step is left blank - defaults to a step of 1
- Examples: mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[0:2] returns [“the”, “cat”] (includes items 0 and 1, but not 2)
  - mylist[2:3] returns [“in”] (only include item 2, equivalent to indexing mylist[2])
  - mylist[2:] returns [“in”, “the”, “hat”] (the remainder of the list)
  - mylist[:-1] returns [“the”, “cat”, “in”, “the”] (everything up to the last item)

More Slicing fun

- Examples: mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - mylist[0:4:2] returns ['the', 'in'] (first item then step of 2)
  - mylist[::-1] returns ['hat', 'the', 'in', 'cat', 'the'] (reverses!)
  - mylist[4:8] returns [‘hat’]
- Example: mystr = ‘Python’
  - mystr[0:2] = ?
  - mystr[4:6].upper() = ?
  - mystr[1:5:3] = ?
  - mystr[::-1] = ?

Homework 2.1

Submit on Camino:

- Complete problems in the jupyter notebook posted here:

  - https://github.com/denisvrdoljak/OMIS30_Fall2018/tree/master/wk2
  - (details in wk2hw1.md file)

Appendix: Variable Names

- Case matters (mystr is a different variable than MyStr)
- Cannot start with a number
- Usually variable names are in all lowercase
  - Can use underscores to make them more readable
    - E.g.: word_dict or my_list
- Keep variable names short (you might have to write them a lot!)
- ‘Counter’ variables are often a single letter like: i,j,k
- Try to name variables something that is easy to read for you and other programmers (i.e., avoid lowercase “L” as it looks like uppercase “I” and pipe: l, I, |)

Appendix

- Reserved keywords in Python:
  - https://pentangle.net/python/handbook/node52.html
  - DO NOT USE THESE as VARIABLE NAMES!
- Dos/Windows vs Unix/Linux:
  - https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Step_by_Step_Guide/ap-doslinux.html
  - Built in functions in Python (full list):
  - https://docs.python.org/3/library/functions.html