OMIS 30: Intro to Programming (with Python)

Week 2, Class 2

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

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

Primitives Review

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

Review 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"]
- Python is 'zero-based' so indexing for the first item:
  - >> mylist[0]
  - Returns "the"

Review Indexing - exercise

- mylist = [1, True, “in”, 7.0, “hat”]
  - mylist[0] -> ?
  - mylist[1] -> ?
  - mylist[-1] -> ?
  - mylist[-3] -> ?

Review 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)

Review Slicing - exercise

- Example: mystr = ‘SantaClara’
  - mystr[0:2] = ?
  - mystr[4:6].upper() = ?
  - mystr[1:5:3] = ?
  - mystr[::-1] = ?

Review 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, |)

List

- Unlimited length, known order, mixed data types, mutable
  - y = [1,2,3]
  - mylist = [“the”, “cat”, “in”, “the”, “hat”]
  - another_list = [1, “the”, 3.45, True]

List Methods and Operations

- x = [1,2,3]
  - Creates a list
  - To create a blank list is: x = list() or x = []
- mylist = [4,5,6]
- x + mylist -> [1,2,3,4,5,6] (adds both lists together but doesn’t assign it to anything)
- x * 3 -> [1, 2, 3, 1, 2, 3, 1, 2, 3] (makes 3 copies of list but doesn’t assign it to anything)

More List Methods and Operations

- To start: x = [1,2,3]
- .append() - adds an item to the end of the list
  - x.append(4) -> [1,2,3,4]
- .remove() - removes an item from the list (from the end)
  - x.remove(4) -> [1,2,3]
- .pop() - pops an item of the end of the list and returns it
  - x.pop() -> 3
  - x.pop(0) -> 1 (pop(0) = front of the list)
- .extend() - extends the first list by adding the 2nd list to it
  - y = [9,10]
  - x.extend(y) -> [2,9,10]

Advanced String and List Methods

- **str.join()**
  - Used to concatenate a sequence of strings into one string
  - .join() takes a list as argument
  - separator = “-” # a string
  - sequence = [“join”, “me”, “together”] # a list of strings
  - separator.join(sequence) = “join-me-together”
  - “ ”.join(sequence) = “join me together”

IF Statements

- Creates a condition, which if True, executes the block of code. If False, it skips over it.
- Format:

```
if boolean_condition:
    #code to execute
```

- The block of code is indented with a tab (or 4 spaces)
- The IF block ends when the indentation en

IF Statements

- Creates a condition, which if True, executes the block of code. If False, it skips over it.

```
if my_animal == “mammal”:
    print(“It’s a mammal”)
```

- The block of code is indented with a tab (or 4 spaces)
- The IF block ends when the indentation en

Nested IF Statements

- IF statements can be nested

```
If my_animal == “mammal”:
    if my_animal_species == “dog”:
                    print(“It’s a dog!”)
print(“It’s a mammal”)
```

IF-ELSE

- ELSE blocks execute if the IF condition is False

```
if my_animal == “mammal”:
    if my_animal _species == “dog:
                print(“It’s a dog!”)
    else:
        print(“It’s a mammal, but not a dog”)
```

ELIF (else if)

- ELIF executes a second IF condition, if the first condition is False

```
if my_animal == “mammal”:
    if my_animal _species == “dog:
                print(“It’s a dog!”)
    elif my_animal _species == “cat”:
                print(“It’s a cat!”)
    Elif my_animal _species == “elephant”:
                print(“It’s an elephant!!”)
    else:
          print(“It’s a mammal, but not a dog or a cat…or an elephant”)
```

Dictionary (dict)

- A mutable unordered set of key:value pairs, with unique keys
  - Syntax: sound_dict = {“cat”: “meow”, “duck”: “quack”}
  - To access a given value, call the key:
    - >> sound_dict[“duck”]
    - Returns: “quack”
  - To assign a new key:value pair or reassign an existing key:
    - >> sound_dict[“cow”] = “moo”
    - >> print(sound_dict)
    - Returns: {“cat”: “meow”, “duck”: “quack”, “cow”: “moo”}

Dictionary Methods and Operations

- sound_dict = {“cat”: “meow”, “duck”: “quack”, “cow”: “moo”}
- .keys() - returns a list of the keys of the dictionary
  - sound_dict.keys() = dict_keys(['cat', 'duck', 'cow'])
- .values() - returns a list of the values of the dictionary
  - sound_dict.values() = dict_values(['meow', 'quack', 'moo'])
- .items() - returns a list of tuples of (key,value)
  - sound_dict.items() = dict_items([('cat', 'meow'), ('duck', 'quack'), ('cow', 'moo')])

Advanced Methods

- **zip()**
  - Used to pair up the elements of two lists (or other iterable) based on shared index
    - odd = (1,3,5), even = (2,4,6)
    - >> print(list(zip(odd, even)))

      [(1,2),(3,4),(5,6)]
  - Can also be used with dictionaries:
    - students = [“Matt”, “Jane”, “Bob”], grades = [82, 97, 70]
    - >> print(dict(zip(names, grades)))

      {“Matt”:82, “Jane”:97, “Bob”:70}

Tuple

- An immutable ordered list with a known number of elements.
  - Syntax: x = (1,4,6)
  - Immutability refers to the inability to be changed after the original assignment.
  - Tuples, like all the primitive data types, are immutable.

Set

- An unordered collection of UNIQUE items.
  - Syntax: x = {4,1,6} or x = set((4,1,6))
    - If y = {4,4,6,1} -> y = {4,6,1} (the extra 4 is removed because its not unique item)
  - Cannot update an item only add or remove
    - set.add() -> adds that item to the set
      - x.add(7) -> {1, 4, 6, 7}
    - remove() -> removes that item from the set
      - x.remove(1) -> {6, 4, 7}

Basic Built-ins

- Introducing a few useful/common built in functions:
  - len() <- returns the length of that object
  - type() <- returns the type of that object

Homework 2.2

Submit this homework on Camino.

- Repeat the HW that was due today.
- You will get the higher grade of your 2 submissions on both.
- Start your project 1; it’s due in 2 weeks!

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
  - You should be familiar with the basic commands for navigating aroudn the file structure and modifying/creating files/folders. You should be aware of the harder to remember ones (like grep and vi) so you know what to Google when you need them!