Python – (Weeks 3-9)

Module 2: Python Data Analytics

Kernel def

Delimiter = separation symbol

next(data) – to ignore column headers, or just go down 1 row

else vs elif

how to convert notebooks/JUPYTER LAB to `.py` files.

just go to `file -> Download As -> Python (.py)` while in the notebook

DOWNLOAD OR SAVE? DOESN’T LOOK GOOD THIS WAY

**HW skills**: how to import modules like csv;

to read and write files in various formats;

to store contents in variables, lists, and dictionaries;

to iterate through basic data structures;

and to debug along the way.

Using what we've learned, try to break down you tasks into discrete mini-objectives.

Both

Have variables in everything I am tracking (especially pypoll)

Pybank

2.7 2.8 2.9 reading the csv & writing a new text file (.txt)

Loop thru file

lists

Pypoll

Variables – each candidate, # of votes, total votes, winning candidate’s name, winning # of votes…wrestling ex

2.7 2.8 2.9 reading the csv & writing a new file

Dictionaries & loops

3.2

1.9 lists 1.11 loops

“jupyter lab” in **terminal** to start (Jupyter lab is newer iteration of jupyter notebook) <http://localhost:8888/lab>

running from Firefox

“control C” to get out

Files need to end in **.ipynb**

**Jupyter** is a coding environment to run diff. kinds of code

Julia python r (3 diff languages in data science/analysis)

Another way to input code and see result, similar to VSC

Used w Python moving forward

This is an interactive coding environment consisting of cells that let you run code and immediately see the result - its very useful for reproducible iterative code.

“Shift enter” is how I run cells

To create new cells

Esc b=below, a=above, dd=delete

‘shift/control/-‘ creates new cell box

**Common Commands**

`**mkdir** NAMEofFOLDER` will make a new directory/FOLDER on the desktop / in whatever folder I am in.

`**ls**` will show/LIST what in the current directory

`**touch** first\_file.py` will create a FILE

2^3 in python 2\*\*3

elif = elseif

**In terminal:** = command line = gitbash in Windows – everything is a folder…

‘Command t’ for a new tab in the terminal

Control C to get to David$

Cd ~ to go back to home directory

cd = change directory

`cd Desktop` will change to the desktop directory

`cd ..` to go back to previous folder

`cd WHATEVER.DIRECTORY/FOLDER I WANT TO GO TO` will move to the newly created folder

pwd = present working directory…how computer downloads

"**python** NAMEofFILE**.py**" will run file

open ---- opens python file in visual studio code

‘open .’ to show normal view of folder

(or `explorer .` in PC will open the current folder)

# Creates a **variable** with a **string** "Frankfurter"

title = "Frankfurter"

# Creates a **variable** with an **integer** 80

years = 80

# Creates a **variable** with the **boolean** value of True

expert\_status = True

# **Prints** a statement adding the variable

**print**("Nick is a professional " + title)

# **Convert** the integer years into a string and prints

print("He has been coding for " + str(years) + " years")

# Converts a boolean into a string and prints

print("Expert status: " + str(expert\_status))

# Collects the user's **input** for the prompt "What is your name?"

name = **input**("What is your name? ")

# Collects the user's **input** for the prompt and **converts** the string to an integer.

age = **int**(input("How old are you? "))

# Collects the user's **input** for the prompt and **converts** it to a **boolean**. Note that non-zero, non-empty objects are truth-y.

trueOrFalse = **bool**(input("Is the input truthy? "))

# Creates three **print** statements that to respond with the **output**.

print("My name is " + str(name))

print("I will be " + str(age + 1) + " next year.")

print("The input was converted to " + str(trueOrFalse))

# Take input of you and your neighbor

your\_first\_name = input("What is your name? ")

neighbor\_first\_name = input("What is your neighbors name? ")

# Take how long each of you have been coding

months\_you\_coded = input("How many months have you been coding? ")

months\_neighbor\_coded = input("How many months has your neighbor been coding? ")

# Add total month

total\_months\_coded = int(months\_you\_coded) + int(months\_neighbor\_coded)

# Print results

print("I am " + your\_first\_name + " and my neighbor is " + neighbor\_first\_name)

print("Together we have been coding for " + str(total\_months\_coded) + " months!")

x = 1

y = 10

# Checks if one value is equal to another

if(x == 1):

print("x is equal to 1")

# Checks if one value is NOT equal to another

if(y != 1):

print("y is not equal to 1")

# Checks if one value is less than another

if(x < y):

print("x is less than y")

# Checks if one value is greater than another

if(y > x):

print("y is greater than x")

# Checks if a value is greater than or equal to another

if(x >= 1):

print("x is greater than or equal to 1")

# Checks for two conditions to be met using "and"

if(x == 1 and y == 10):

print("Both values returned true")

# Checks if either of two conditions is met

if(x < 45 or y < 5):

print("One or more of the statements were true")

# Nested if statements

if(x < 10):

if(y < 5):

print("x is less than 10 and y is less than 5")

elif(y == 5):

print("x is less than 10 and y is equal to 5")

else:

print("x is less than 10 and y is greater than 5")

== IS USED WHEN?

!= is not =

# 1. oooo needs some work

x = 5

if (2 \* x > 10):

print("Question 1 works!")

else:

print("oooo needs some work")

# 2. Question 2 works!

x = 5

if (len("Dog") < x):

print("Question 2 works!")

else:

print("Still missing out")

# 3. GOT QUESTION 3!

x = 2

y = 5

if ((x\*\*3 >= y) and (y\*\*2 < 26)):

print("GOT QUESTION 3!")

else:

print("Oh good you can count")

# 4. Dan is in group three

name = "Dan"

group\_one = ["Greg", "Tony", "Susan"]

group\_two = ["Gerald", "Paul", "Ryder"]

group\_three = ["Carla", "Dan", "Jefferson"]

if (name in group\_one):

print(name + " is in the first group")

elif (name in group\_two):

print(name + " is in group two")

elif (name in group\_three):

print(name + " is in group three")

else:

print(name + " does not have a group")

# 5. Can ride bumper cars

height = 66

age = 16

adult\_permission = True

if ((height > 70) and (age >= 18)):

print("Can ride all the roller coasters")

elif ((height > 65) and (age >= 18)):

print("Can ride moderate roller coasters")

elif ((height > 60) and (age >= 18)):

print("Can ride light roller coasters")

elif (((height > 50) and (age >= 18)) or ((adult\_permission) and (height > 50))):

print("Can ride bumper cars")

else:

print("Stick to lazy river")

\* = “key activity”

\*1.9 Lists

# Create a variable and set it as a List

myList = ["Jacob", 25, "Ahmed", 80]

print(myList)

# Adds an element onto the end of a List

myList.append("Matt")

print(myList)

# Returns the index of the first object with a matching value

print(myList.index("Matt"))

# Changes a specified element within a List at the given index

myList[3] = 85

print(myList)

# Returns the length of the List

print(len(myList))

# Removes a specified object from an List

myList.remove("Matt")

print(myList)

# Removes the object at the index specified

myList.pop(0)

print(myList)

# Creates a tuple, a sequence of immutable Python objects that cannot be changed

myTuple = ('Python', 100, 'VBA', False)

print(myTuple)

\*1.10 Rock Paper Scissors

# Incorporate the random library

import random

# Print Title

print("Let's Play Rock Paper Scissors!")

# Specify the three options

options = ["r", "p", "s"]

# Computer Selection

computer\_choice = random.choice(options)

# User Selection

user\_choice = input("Make your Choice: (r)ock, (p)aper, (s)cissors? ")

# Run Conditionals

if (user\_choice == "r" and computer\_choice == "p"):

print("You chose rock. The computer chose paper.")

print("Sorry. You lose.")

elif (user\_choice == "r" and computer\_choice == "s"):

print("You chose rock. The computer chose scissors.")

print("Yay! You won.")

elif (user\_choice == "r" and computer\_choice == "r"):

print("You chose rock. The computer chose rock.")

print("A smashing tie!")

elif (user\_choice == "p" and computer\_choice == "p"):

print("You chose paper. The computer chose paper.")

print("A smashing tie!")

elif (user\_choice == "p" and computer\_choice == "s"):

print("You chose paper. The computer chose scissors.")

print("Sorry. You lose.")

elif (user\_choice == "p" and computer\_choice == "r"):

print("You chose paper. The computer chose rock.")

print("Yay! You won.")

elif (user\_choice == "s" and computer\_choice == "p"):

print("You chose scissors. The computer chose paper.")

print("Yay! You won.")

elif (user\_choice == "s" and computer\_choice == "s"):

print("You chose scissors. The computer chose scissors.")

print("A smashing tie!")

elif (user\_choice == "s" and computer\_choice == "r"):

print("You chose scissors. The computer chose rock.")

print("Sorry. You lose.")

else:

print("I don't understand that!")

print("Next time, choose from 'r', 'p', or 's'.")

\* Look into this [stackoverflow](<https://stackoverflow.com/questions/306400/how-to-randomly-select-an-item-from-a-list> ) question for usage.

1.11 LOOPS

# A For loop moves through a given range of numbers

# If only one number is provided it will loop from 0 to that number

for x in range(10):

print(x)

# If two numbers are provided then a For loop will loop from the first number up until it reaches the second number

for x in range(20, 30):

print(x)

# Iterate through letters in a string

word = "Peace"

for letters in word:

print(letters)

# If a list is provided, then the For loop will loop through each element within the list

words = ["Peanut", "Butter", "Jelly", "Time", "Is", "Now"]

for word in words:

print(word)

# Iterate through a list

zoo = ["cow", "dog", "bee", "zebra"]

for animal in zoo:

print(animal)

# A While Loop will continue to loop through the code contained within it until some condition is met

x = "Yes"

while x == "Yes":

print("Whee! Merry-Go-Rounds are great!")

x = input("Would you like to go on the Merry-Go-Round again? ")

# Loop while a condition is being met

run = "y"

while run == "y":

print("Hi!")

run = input("To run again. Enter 'y'")

\*1.12 Number Chain

# Initial variable to track game play

user\_play = "y"

# Set start and last number (BONUS…)

start\_number = 0

# While we are still playing...

while user\_play == "y":

# Ask the user how many numbers to loop through

user\_number = input("How many numbers? ")

# Loop through the numbers. (Be sure to cast the string into an integer.)

for x in range(start\_number, int(user\_number) + start\_number):

# Print each number in the range

print(x)

# Set the next start number as the last number of the loop (BONUS…)

start\_number = start\_number + int(user\_number)

# Once complete...

user\_play = input("Continue the chain: (y)es or (n)o? ")

**## BONUS ADDED 2 LINES ABOVE**

\* Rather than just displaying numbers constantly starting at 0, have the numbers start at the end of the previous chain.

\*2.3 Kid in candy store

# The list of candies to print to the screen

candyList = ["Snickers", "Kit Kat", "Sour Patch Kids", "Juicy Fruit",

"Swedish Fish", "Skittles", "Hershey Bar", "Skittles", "Starbursts", "M&Ms"]

# The amount of candy the user will be allowed to choose

allowance = 5

# The list used to store all of the candies selected inside of

candyCart = []

# Print all of the candies to the screen and their index in brackets

for candy in candyList:

print("[" + str(candyList.index(candy)) + "] " + candy)

# Run through a loop which allows the user to choose which candies to take home with them

for x in range(allowance):

selected = input("Which candy would you like to bring home? ")

# Add the candy at the index chosen to the candyCart list

candyCart.append(candyList[int(selected)])

# Loop through the candyCart to say what candies were brought home

print("I brought home with me...")

for candy in candyCart:

print(candy)

\* Create a version of the same code which allows a user to select as much candy as they want up until they say they do not want any more.

# The list of candies to print to the screen

candyList = ["Snickers", "Kit Kat", "Sour Patch Kids", "Juicy Fruit",

"Swedish Fish", "Skittles", "Hershey Bar", "Skittles", "Starbursts", "M&Ms"]

# The list used to store all of the candies selected inside of

candyCart = []

# Print all of the candies to the screen and their index in brackets

for candy in candyList:

print("[" + str(candyList.index(candy)) + "] " + candy)

# Set answer to "yes" for while loop

answer = "yes"

while answer == "yes":

# Ask which candy the user would like to bring ho

selected = input("Which candy would you like to bring home? ")

# Add the candy at the index chosen to the candyCart list

candyCart.append(candyList[int(selected)])

# ask the user if they want more candy

answer = input("Would you like to make another selection? ('yes' or 'no') ")

# Loop through the candyCart to say what candies were brought home

print("I brought home with me...")

for candy in candyCart:

print(candy)

\*2.4 House of Pies – END PART DOESN’T WORK…?

# Initial variable to track shopping status

shopping = 'y'

# List to track pie purchases

pie\_purchases = []

# Pie List

pie\_list = ["Pecan", "Apple Crisp", "Bean", "Banoffee", "Black Bun",

"Blueberry", "Buko", "Burek", "Tamale", "Steak"]

# Display initial message

print("Welcome to the House of Pies! Here are our pies:")

# While we are still shopping...

while shopping == "y":

# Show pie selection prompt

print("---------------------------------------------------------------------")

print("(1) Pecan, (2) Apple Crisp, (3) Bean, (4) Banoffee, " +

" (5) Black Bun, (6) Blueberry, (7) Buko, (8) Burek, " +

" (9) Tamale, (10) Steak ")

pie\_choice = input("Which would you like? ")

# Add pie to the pie list

pie\_purchases.append(pie\_choice)

print("------------------------------------------------------------------------")

# Inform the customer of the pie purchase

print("Great! We'll have that " + pie\_list[int(pie\_choice) - 1] + " right out for you.")

# Provide exit option

shopping = input("Would you like to make another purchase: (y)es or (n)o? ")

# Once the pie list is complete

print("------------------------------------------------------------------------")

print("You purchased a total of " + str(len(pie\_purchases)) + ".")

BONUS LOOKS SLOPPY…FIX?

# Initial variable to track shopping status

shopping = 'y'

# List to track pie purchases

pie\_purchases = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

# Pie List

pie\_list = ["Pecan", "Apple Crisp", "Bean", "Banoffee", "Black Bun",

"Blueberry", "Buko", "Burek", "Tamale", "Steak"]

# Display initial message

print("Welcome to the House of Pies! Here are our pies:")

# While we are still shopping...

while shopping == "y":

# Show pie selection prompt

print("---------------------------------------------------------------------")

print("(1) Pecan, (2) Apple Crisp, (3) Bean, (4) Banoffee, " +

" (5) Black Bun, (6) Blueberry, (7) Buko, (8) Burek, " +

" (9) Tamale, (10) Steak ")

pie\_choice = input("Which would you like? ")

# Get index of the pie from the selected number

choice\_index = int(pie\_choice) - 1

# Add pie to the pie list by finding the matching index and adding one to its value

pie\_purchases[choice\_index] += 1

print("------------------------------------------------------------------------")

# Inform the customer of the pie purchase

print("Great! We'll have that " + pie\_list[choice\_index] + " right out for you.")

# Provide exit option

shopping = input("Would you like to make another purchase: (y)es or (n)o? ")

# Once the pie list is complete

print("------------------------------------------------------------------------")

# Count instances of each pie

print("You purchased: ")

# Loop through the full pie list

for pie\_index in range(len(pie\_list)):

pie\_count = str(pie\_purchases[pie\_index])

pie\_name = str(pie\_list[pie\_index])

# Gather the count of each pie in the pie list and print them alongside the pies

print(pie\_count + " " + pie\_name)

2.5 Basic Read or connect to a file

# Store the file path associated with the file (note the backslash may be OS specific)

file = '../Resources/input.txt'

# Open the file in "read" mode ('r') and store the contents in the variable "text"

with open(file, 'r') as text:

print(text)

# Store all of the text inside a variable called "lines"

lines = text.read()

# Print the contents of the text file

print(lines)

2.6 Importing modules

# Import the random and string Module

import random

import string

# Utilize the string module's custom method: ".ascii\_letters"

print(string.ascii\_letters)

# Utilize the random module's custom method randint

for x in range(10):

print(random.randint(1, 10))

2.7 Read CSVs

# First we'll import the os module

# This will allow us to create file paths across operating systems

import os

# Module for reading CSV files

import csv

csvpath = os.path.join('..', 'Resources', 'accounting.csv')

# # Method 1: Plain Reading of CSV files

# with open(csvpath, 'r') as file\_handler:

# lines = file\_handler.read()

# print(lines)

# print(type(lines))

# Method 2: Improved Reading using CSV module

with open(csvpath, newline='') as csvfile:

# CSV reader specifies delimiter and variable that holds contents

csvreader = csv.reader(csvfile, delimiter=',')

print(csvreader)

# Read the header row first (skip this step if there is now header)

csv\_header = next(csvreader)

print(f"CSV Header: {csv\_header}")

# Read each row of data after the header

for row in csvreader:

print(row)

\*2.8 Read Netflix

# Modules

import os

import csv

# Prompt user for video lookup

video = input("What show or movie are you looking for? ")

# Set path for file

csvpath = os.path.join("..", "Resources", "netflix\_ratings.csv")

# Bonus

# ------------------------------------------

# Set variable to check if we found the video

found = False

# Open the CSV

with open(csvpath, newline="") as csvfile:

csvreader = csv.reader(csvfile, delimiter=",")

# Loop through looking for the video

for row in csvreader:

if row[0] == video:

print(row[0] + " is rated " + row[1] + " with a rating of " + row[5])

# BONUS: Set variable to confirm we have found the video

found = True

# BONUS: Stop at first results to avoid duplicates

break

# If the video is never found, alert the user

if found is False:

print("Sorry about this, we don't seem to have what you are looking for!")

2.9 Write CSV – REVIEW COMMANDS/TERMS W TA

# Dependencies

import os

import csv

# Specify the file to write to

output\_path = os.path.join("..", "output", "new.csv")

# Open the file using "write" mode. Specify the variable to hold the contents

with open(output\_path, 'w', newline='') as csvfile:

# Initialize csv.writer

csvwriter = csv.writer(csvfile, delimiter=',')

# Write the first row (column headers)

csvwriter.writerow(['First Name', 'Last Name', 'SSN'])

# Write the second row

csvwriter.writerow(['Caleb', 'Frost', '505-80-2901'])

2.10 Zip

import csv

import os

# Three Lists

indexes = [1, 2, 3, 4]

employees = ["Michael", "Dwight", "Meredith", "Kelly"]

department = ["Boss", "Sales", "Sales", "HR"]

# Zip all three lists together into tuples

roster = zip(indexes, employees, department)

# save the output file path

output\_file = os.path.join("output.csv")

# open the output file, create a header row, and then write the zipped object to the csv

with open(output\_file, "w", newline="") as datafile:

writer = csv.writer(datafile)

writer.writerow(["Index", "Employee", "Department"])

writer.writerows(roster)

# # to print out to terminal:

# #comment out above code and run the code below

# for employee in roster:

# print(employee)

\*2.11 Udemy Zip

import os

import csv

udemy\_csv = os.path.join("..", "Resources", "web\_starter.csv")

# Lists to store data

title = []

price = []

subscribers = []

reviews = []

review\_percent = []

length = []

# with open(udemy\_csv, newline="", encoding='utf-8') as csvfile:

with open(udemy\_csv, newline="") as csvfile:

csvreader = csv.reader(csvfile, delimiter=",")

for row in csvreader:

# Add title

title.append(row[1])

# Add price

price.append(row[4])

# Add number of subscribers

subscribers.append(row[5])

# Add amount of reviews

reviews.append(row[6])

# Determine percent of review left to 2 decimal places

percent = round(int(row[6]) / int(row[5]), 2)

review\_percent.append(percent)

# Get length of the course to just a number

new\_length = row[9].split(" ")

length.append(float(new\_length[0]))

# Zip lists together

cleaned\_csv = zip(title, price, subscribers, reviews, review\_percent, length)

# Set variable for output file

output\_file = os.path.join("web\_final.csv")

# Open the output file

with open(output\_file, "w", newline="") as datafile:

writer = csv.writer(datafile)

# Write the header row

writer.writerow(["Title", "Course Price", "Subscribers", "Reviews Left",

"Percent of Reviews", "Length of Course"])

# Write in zipped rows

writer.writerows(cleaned\_csv)

2.12 Defining Functions

# Define the function and tell it to print "Hello!" when called

def printHello():

print(f"Hello!")

# Call the function within the application to ensure the code is run

printHello()

# -------------#

# Functions that take in and use parameters can also be defined

def printName(name):

print("Hello " + name + "!")

# When calling a function with a parameter, a parameter must be passed into the function

printName("Bob Smith")

# -------------#

# The prime use case for functions is in being able to run the same code for different values

listOne = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

listTwo = [11, 12, 13, 14, 15]

def listInformation(simpleList):

print(f"The values within the list are...")

for value in simpleList:

print(value)

print("The length of this list is... " + str(len(simpleList)))

listInformation(listOne)

listInformation(listTwo)

**#### Day 3 Key Activities**

\* Students Do: Hobby-Book

\* Instructor Do: Keys and Values

\* Students Do: Store Manager

3.1 Cereal Cleaner

import os

import csv

cereal\_csv = os.path.join("..", "Resources", "cereal.csv")

# Open and read csv

with open(cereal\_csv, newline="") as csvfile:

csvreader = csv.reader(csvfile, delimiter=",")

# Read the header row first (skip this part if there is no header)

csv\_header = next(csvfile)

print(f"Header: {csv\_header}")

# OR SKIP the header row

next(csvreader, None)

# Read through each row of data after the header

for row in csvreader:

# Convert row to float and compare to grams of fiber

if float(row[7]) >= 5:

print(row)

3.2 Dictionaries

\* Another data type that is commonly used in Python is that of the dictionary.

\* Like lists and tuples, dictionaries can contain multiple values and types of data within them.

\* Unlike lists and tuples, however, dictionaries store data in key-value pairs. The key in a dictionary is a string referenced to collect the value it is associated with.

\* Open up [dictionaries.py](Solved/dictionaries.py) and explain the code contained within. Make sure to explain the following...

\* A pair of curly-braces are used to create a dictionary: `variable = {}`

\* Values are added to dictionaries at declaration by creating a key stored within a string, following it with a colon, and then placing the value desired afterwards.

\* Referencing a value within a dictionary is as simple as calling the dictionary and following it up with a pair of brackets containing the key for the value desired.

![basic dictionary](Images/02-Dictionary\_OneValue.png)

\* Dictionaries can hold multiple pieces of information by following up each key-value pairing with a comma and then placing another key-value pair afterwards.

\* Dictionaries can also store lists. They are accessed by first calling the key and then indexing the list. Assure students they just need a basic understanding of this for now and that when they get into APIs they will get a lot more practice.

\* Dictionaries can also contain other dictionaries. To access the values inside nested dictionaries, simply add another key to the reference.

![Advanced Dictionaries](Images/02-Dictionary\_MultiValue.png)

\* It is important to note how dictionaries in Python will always auto-sort alphabetically. This means that the order in which key-value pairs were declared may not be the order they would be when printed in the console.

# Unlike lists, dictionaries store information in pairs

# ---------------------------------------------------------------

# A list of actors

actors = ["Tom Cruise",

"Angelina Jolie",

"Kristen Stewart",

"Denzel Washington"]

# A dictionary of an actor

actor = {"name": "Tom Cruise"}

print(f'{actor["name"]}')

# ---------------------------------------------------------------

# A dictionary can contain multiple pairs of information

actress = {"name": "Angelina Jolie", "genre": "Action", "nationality": "United States"}

# ---------------------------------------------------------------

# A dictionary can contain multiple types of information

another\_actor = {

"name": "Sylvester Stallone",

"age": 62,

"married": True,

"best movies": [

"Rocky",

"Rocky 2",

"Rocky 3"]}

print(f'{another\_actor["name"]} was in {another\_actor["best movies"][0]}')

# ---------------------------------------------------------------

# A dictionary can even contain another dictionary

film = {"title": "Interstellar",

"revenues":

{"United States": 360, "China": 250, "United Kingdom": 73}}

print(f'{film["title"]} made {film["revenues"]["United States"]}'" in the US.")

# ---------------------------------------------------------------

\*3.3 Hobby Book w dictionaries

# Dictionary full of info

my\_info = {"name": "Rex",

"occupation": "dog",

"age": 21,

"hobbies": ["barking", "eating", "sleeping", "loving my owner"],

"wake-up": {"Mon": 5, "Friday": 5, "Saturday": 10, "Sunday": 9}}

# Print out results are stored in the dictionary

print(f'Hello I am {my\_info["name"]} and I am a {my\_info["occupation"]}')

print(f'I have {len(my\_info["hobbies"])} hobbies!')

print(f'On the weekend I get up at {my\_info["wake-up"]["Saturday"]}')

3.4 List comprehensions

fish = "halibut"

# Loop through each letter in the string

# and push to an array

letters = []

for letter in fish:

letters.append(letter)

print(letters)

# List comprehensions provide concise syntax for creating lists

letters = [letter for letter in fish]

print(letters)

# 2ND DOES THE SAME MORE QUICKLY

# We can manipulate each element as we go

capital\_letters = []

for letter in fish:

capital\_letters.append(letter.upper())

print(capital\_letters)

# List Comprehension for the above

capital\_letters = [letter.upper() for letter in fish]

print(capital\_letters)

# 2ND DOES THE SAME MORE QUICKLY

# We can also add conditional logic (if statements) to a list comprehension

july\_temperatures = [87, 85, 92, 79, 106]

hot\_days = []

for temperature in july\_temperatures:

if temperature > 90:

hot\_days.append(temperature)

print(hot\_days)

# List Comprehension with conditional

hot\_days = [temperature for temperature in july\_temperatures if temperature > 90]

print(hot\_days)

# 2ND DOES THE SAME MORE QUICKLY

3.5 List comprehensions

names = []

for \_ in range(5):

name = input("Please enter the name of someone you know. ")

names.append(name)

lowercased = [name.lower() for name in names]

titlecased = [name.title() for name in lowercased]

invitations = [

f"Dear {name}, please come to the wedding this Saturday!" for name in titlecased]

for invitation in invitations:

print(invitation)

See the documentation for the [title] (<https://docs.python.org/3/library/stdtypes.html#str.title> ) method.

3.6 Functions – WHAT IS THIS???

# Basic Definition

def name(parameters):

# code goes here

return

# Simple Function with no parameters

def show():

print(f"Hi!")

# You use parenthesis to run the code in a function

show()

# Simple function with one parameter

def show(message):

print(message)

# Think of the parameter `message` as a variable

# You assign the string "Hello, World!" when you call the function

# This is like saying `message = "Hello, World!"`

show("Hello, World!")

# Functions can have more than one parameter

def make\_quesadilla(protein, topping):

quesadilla = f"Here is a {protein} quesadilla with {topping}"

print(quesadilla)

# Supply the arguments (values) when calling the function

make\_quesadilla("beef", "guacamole")

make\_quesadilla("chicken", "salsa")

# @NOTE: Order is important when supplying arguments!

make\_quesadilla("sour cream", "beef")

# We can also specify default values for parameters

def make\_quesadilla(protein, topping="sour cream"):

quesadilla = f"Here is a {protein} quesadilla with {topping}"

print(quesadilla)

# Make a quesadilla using the default topping

make\_quesadilla("chicken")

# Make a quesadilla with a new topping

make\_quesadilla("beef", "guacamole")

# Functions can return a value

def square(number):

return number \* number

# You can save the value that is returned

squared = square(2)

print(squared)

# You can also just print the return value of a function

print(square(2))

print(square(3))

3.7 Functions

# Write a function that returns the arithmetic average for a list of numbers

def average(numbers):

length = len(numbers)

total = 0.0

for number in numbers:

total += number

return total / length

# Test your function with the following:

print(average([1, 5, 9]))

print(average(range(11)))

3.8 Wrestling w Functions

import os

import csv

# Path to collect data from the Resources folder

wrestlingCSV = os.path.join('..', 'Resources', 'WWE-Data-2016.csv')

# Define the function and have it accept the 'wrestlerData' as its sole parameter

def getPercentages(wrestlerData):

# Total matches can be found by adding wins, losses, and draws together

totalMatches = int(wrestlerData[1]) + int(wrestlerData[2]) + int(wrestlerData[3])

# Win % can be found by dividing the total wins by total matches and \* 100

winPercent = (int(wrestlerData[1]) / totalMatches) \* 100

# Loss % can be found by dividing the total losses by total matches and \* 100

lossPercent = (int(wrestlerData[2]) / totalMatches) \* 100

# Draw % can be found by dividing the total draws by total matches and \* 100

drawPercent = (int(wrestlerData[3]) / totalMatches) \* 100

# If the loss percentage is over 50, typeOfWrestler is "Jobber". Otherwise it is "Superstar".

if(lossPercent > 50):

typeOfWrestler = "Jobber"

else:

typeOfWrestler = "Superstar"

# Print out the wrestler's name and their percentage stats

print(f"Stats for {wrestlerData[0]}")

print(f"WIN PERCENT: {str(winPercent)}")

print(f"LOSS PERCENT: {str(lossPercent)}")

print(f"DRAW PERCENT: {str(drawPercent)}")

print(f"{wrestlerData[0]} is a {typeOfWrestler}")

# Read in the CSV file

with open(wrestlingCSV, 'r') as csvfile:

# Split the data on commas

csvreader = csv.reader(csvfile, delimiter=',')

header = next(csvreader)

# Prompt the user for what wrestler they would like to search for

nameToCheck = input("What wrestler do you want to look for? ")

# Loop through the data

for row in csvreader:

# If the wrestler's name in a row is equal to that which the user input, run the 'getPercentages()' function

if(nameToCheck == row[0]):

getPercentages(row)

Check out this site: <https://www.codecademy.com/courses/learn-python/lessons/python-syntax/exercises/print-statements?action=resume_content_item>

use something like <https://www.codewars.com> /, this lets you complete challenges in python by writing definitions. This is a great way to get exposed to lots of different coding concepts and problems, and allows you to see how others solved it after you solve a problem. The best way to get good at this is just to see a lot of different situations for how to use the syntax you learn.

Python is loosely type environment

Don't need to declare type of variable

Gain a strong foothold in one of the fundamental programming languages of today.

In the course of this module, you’ll gain deep proficiencies with core Python;

data analytic tools like NumPy, Pandas, and Matplotlib;

and specific libraries for interacting with web data like

Requests, BeautifulSoup, and Tweepy.

» APIs

» JSON

The purpose of this unit is to gain a firm understanding on how to use the core Python language to

store and manipulate data;

read and write to files;

work with the file system,

iteration using for loops,

iteration using list comprehension,

list methods,

"translate" logical solutions to problems to Python implementations of those solutions.

and additional miscellany where required to facilitate completion of the homework assignment.

**Objectives**

* Students will be able to navigate their desktop via the terminal
* Students will create Python scripts and run them in terminal
* Students will begin to understand programming concepts in Python
* Students will feel confident reading data into Python from CSV files
* Students will feel confident writing data from Python into CSV files
* Students will know how to zip two lists together and when this is helpful
* Students will have a firm understanding on how to create and use Python functions
* Students will be able to create and use Python dictionaries….the notion of dictionaries and methods on them
* Students will be able to read data in from a dictionary.
* Students will have a firm understanding of coding logic and reasoning.
* Students in the refresher group will feel far more comfortable in their understanding of Python.
* Students in the challenge group will see how powerful Python can be in solving tough problems.

**### Key Activities**

\* [Rock Paper Scissors](1/Activities/10-Stu\_RockPaperScissors)

\* [Number Chaining](1/Activities/12-Stu\_NumberChain)

\* [Read Netflix](2/Activities/08-Stu\_ReadNetFlix)

\* [Udemy Zip](2/Activities/11-Stu\_UdemyZip)

\* [HobbyBook](3/Activities/03-Stu\_HobbyBook)

<https://realpython.com/>

<https://jeffknupp.com/blog/2015/08/30/python-dictionaries/>

<https://geo-python.github.io/2017/_images/Function_anatomy.png>

all the different python packages(modules): <https://pypi.org/>

And, if you are a seasoned programmer, these books are great:

<https://automatetheboringstuff.com/>

<https://jakevdp.github.io/PythonDataScienceHandbook/>

**### Helpful Links**

\* [Python - Beginner](https://www.learnpython.org/)

\* [Python Scripting](https://automatetheboringstuff.com/)

\* [Python f-strings](https://www.python.org/dev/peps/pep-0498/)

\* [Python DataStructures](https://docs.python.org/3/tutorial/datastructures.html)

\* [Python CSV Module](https://docs.python.org/3/library/csv.html)

\* [Git/Github](https://github.com/Multishifties/No-Nonsense-Github-Project)

\* [Visual Git Guide](http://marklodato.github.io/visual-git-guide/index-en.html)

**### Additional Course Resources**

\* [Python CheatSheet](Supplemental/Python\_Reference\_Guide.pdf)

\* [Python-GettingStarted.md](Supplemental/Python-GettingStarted.md)

\* [Virtual Environments](Supplemental/conda\_pip.pdf)

\* [Conda Cheatsheet](Supplemental/conda-cheatsheet.pdf)