**Python**

**03-Python/1/Activites/03\_Ins\_Variables/variables.py**

# Creates a variable with a string "Frankfurter"

title = "Frankfurter"

# Creates a variable with an integer 80

years = 80

# Creates a variable with the boolean of False

trueOrFalse = False

# Prints a statemeent adding the variable

print("Ahmed is a professional " + title)

# Convert the integer years into a string and prints

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

# Converts a boolean into a string and prints

print("Pretty much all of that was " + str(trueOrFalse))

**03-Python/1/Activities/04-Stu\_HelloVariable/HelloVariableWorld.py**

# Create a variable called 'name' that holds a string

name = "Jacob Deming"

# Create a variable called 'country' that holds a string

country = "United States"

# Create a variable called 'age' that holds an integer

age = 25

# Create a variable called 'hourly\_wage' that holds an integer

hourly\_wage = 15

# Calculate the daily wage for the user

daily\_wage = hourly\_wage \* 8

# Create a variable called 'satisfied' that holds a boolean

satisfied = True

# Print out "Hello <name>!"

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

# Print out what country the user entered

print("You live in " + country)

# Print out the user's age

print("You are " + str(age) + " years old")

# Print out the daily wage that was calculated

print("You make " + str(daily\_wage) + " per day")

# Print out whether the users were satisfied

print("Are you satisfied with your current wage? " + str(satisfied))

**03-Python/1/Activities/05\_Ins\_Prompts/inputs.py**

# 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 "How old are you?" and converts the string to an integer

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

# Collects the user's input for the prompt "Is this statement true?" and converts it to a boolean

trueOrFalse = bool(input("Is this statement true? "))

# Creates three print statements that to respond with the output

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

print("I am " + str(age) + " years old.")

print("The statement was " + str(trueOrFalse))

**03-Python/1/Activities/06-Stu\_DownToInput/Solved/DownToInput.py**

# 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!")

**03-Python/1/Activities/07\_Ins\_Conditionals/conditionals.py**

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 less 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 the other 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")

**03-Python/1/Activities/08-Stu\_ConditionalConundrum/Solved/conditionals\_solved.py**

# 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("Sitck to lazy river")

**03-Python/1/Activities/09\_lists/lists.py**

# Create a variable and set it as an List

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

print(myList)

# Adds an element onto the end of a List

myList.append("Matt")

print(myList)

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

myList[3] = 85

print(myList)

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

print(myList.index("Matt"))

# 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)

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)

**03-Python/1/Activities/10-Stu\_RockPaperScissors/Solved/RPS\_Solved.py**

# 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!")

else:

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

print("Sorry. You lose.")

**03-Python/1/Activities/11-INS\_Loops/loopdeeloop.py**

# Loop through a range of numbers (0 through 4)

for x in range(5):

print(x)

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

# Loop through a range of numbers (2 through 6)

for x in range(2, 7):

print(x)

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

# Iterate through letters in a string

word = "Peace"

for letters in word:

print(letters)

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

# Iterate through a list

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

for animal in zoo:

print(animal)

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

# Loop while a condition is being met

run = "y"

while run == "y":

print("Hi!")

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

**03-Python/1/Activities/12-Stu\_NumberChain/numberchain.py**

# Number Chain

#Chain up the numbers

## Instructions

#\* Using a while loop ask the user "How many numbers?" then print out a chain of descending numbers.

#\* After the results have printed ask the user if they would like to continue.

#\* If "y" keep the chain running.

#\* If "n" exit the chain.

## Bonus

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

# Loop while a condition is being met

x = "y"

while x == "y":

number = int(input("How many numbers? " ))

'print(number)

# Loop through a range of numbers (0 through 4)

for numberlist in range(number):

print(numberlist)

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

x = input("Would you like to continue? enter y for yes n for no ")

**03-Python/2/Activities/01-Stu\_QuickCheckup/Solved/QuickCheckUp\_Solved.py**

# Print Hello User!

print("Hello User!")

# Take in User Input

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

# Respond Back with User Input

print("Hi " + name + "!")

# Take in the User Age

age = input("What is your age? ")

# Respond Back with a statement based on age

if (int(age) < 8):

print("Aww! You are just a baby!")

if (int(age) >= 8):

print("Ah... A well traveled soul are ye!")

**03-Python/2/Activities/02-Ins\_SimpleLoops.py**

# 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)

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

for x in ["Peanut","Butter","Jelly","Time","Is","Now"]:

print(x)

# 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? ")

**03-Python/2/Activities/03-Stu\_KidInCandyStore/Solved/KidInCandyStore\_Solved.py**

# The list of candies to print to the screen

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

**03-Python/2/Activities/04-Stu\_HouseOfPies/HouseOfPies\_Solved.py**

# 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)) + "." )

**03-Python/2/Activities/04-Stu\_HouseOfPies/HouseOfPies\_Solved\_Bonus.py**

# 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? ")

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

pie\_purchases[int(pie\_choice)-1] = pie\_purchases[int(pie\_choice)-1] + 1

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

# Count instances of each pie

print("You purchased: ")

# Loop through the full pie list

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

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

print(str(pie\_purchases[pie\_index]) + " " + str(pie\_list[pie\_index]))

**03-Python/2/Activities/05-Ins\_BasicRead/ReadFIle.py**

# 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)

**03-Python/2/Activities/06-Ins\_Modules/imports.py**

# Import the String Module

import string

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

print(string.ascii\_letters)

# Import the Random Module

import random

# Utilize the random module's custom method randint

for x in range(10):

print(random.randint(1,10))

**03-Python/2/Activities/07-Ins\_ReadCSV/ReadCSV.py**

# First we'll import the os module

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

import os

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

# # Method 1: Plain Reading of CSVs

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

# lines = file\_handler.read()

# print(lines)

# print(type(lines))

# Method 2: Improved Reading using CSV module

import csv

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

# CSV reader specifies delimiter and variable that holds contents

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

print(csvreader)

# Each row is read as a row

for row in csvreader:

print(row)

**03-Python/2/Activities/08-Stu\_ReadNetFlix/Netflix\_ratings.py**

import os

import csv

#Get file

file\_name = os.path.join('Resources', 'Netflix\_ratings.py')

found = False

#Open file newline ='' tells it the file end is

with open(file\_name, newline='') as csvfile:

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

selected = input('Which show would you like to see? ')

for row in csvreader:

if row[0] == selected:

print(row[0]+ " is rated "+ row[1] + " with a rating of " + row[6])

# Set variable to confirm we have found the video

found = True

if found == False:

print("Sorry we were unable to find your show")

**03-Python/2/Activities/08-Stu\_ReadNetFlix/netflix.py**

# 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[6])

# Set variable to confirm we have found the video

found = True

# If the video is never found, alert the user

if found == False:

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

**03-Python/2/Activities/09-Ins\_WriteCSV/Write.py**

import csv

import os

# 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'])

**03-Python/2/Activities/10-Ins\_Zip/zipper.py**

# 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)

# Print the contents of each row

for employee in roster:

print(employee)

**03-Python/2/Activities/11-Stu\_UdemyZip/Solved/web\_solved.py**

import os

import csv

udemy\_csv = os.path.join("../Resources", "WebDevelopment.csv")

# Lists to store data

title = []

price = []

subscribers = []

reviews = []

review\_percent = []

length = []

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(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)

**03-Python/2/Activities/12-Ins\_Functions/functions.py**

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

def printHello():

print("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("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)

**Print First Row of a CSV file.py**

import csv

with open('netflix\_ratings.csv','r', newline='')

csv\_data = csv.reader(csv\_filex, delimiter=',')

print(csv\_data)

#one way using .index

for row in csv\_data:

print(row)

if row.index > 1:

break

#another way to do it using variables

x = 0

for row in csv\_data:

print(row)

x += 1

if x > 1:

break

**03-Python/2/Activities/13-Stu\_WrestlingWithFunctions/Solved/WrestlingFunctions.py**

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 percent can be found by dividing the the total wins by the total matches and multiplying by 100

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

# Loss percent can be found by dividing the total losses by the total matches and multiplying by 100

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

# Draw percent can be found by dividing the total draws by the total matches and multiplying by 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("Stats for " + wrestlerData[0])

print("WIN PERCENT: " + str(winPercent))

print("LOSS PERCENT: " + str(lossPercent))

print("DRAW PERCENT: " + str(drawPercent))

print(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=',')

# 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)

**03-Python/3/Activities/01-Stu\_CerealCleaner/Solved/cereal\_solved.py**

# Dependencies

import os

import csv

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

# Open and read csv

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

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

# Iterate throught each row

for row in csvreader:

# Convert row to float and compare to grams of fiber

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

print(row)

**03-Python/3/Activities/01-Stu\_CerealCleaner/Solved/cereal\_bonus\_solved.py**

import os

import csv

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

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

csv\_reader = csv.reader(csvfile, delimiter=",")

# Skip the first row of the csv

next(csv\_reader, None)

# Loop through rows

for row in csv\_reader:

# Convert row to float and compare to grams of fiber

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

print(row)

**03-Python/3/Activities/02-Ins\_Dicts/dictionaries.py**

# 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(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(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(film["title"] + " made " + str(film["revenues"]["United States"]) + " in the US.")

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

**03-Python/3/Activities/03-Stu\_HobbyBook/Solved/HobbyBook\_Solved.py**

# 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('Hello I am ' + my\_info["name"] + " and I am a " + my\_info["occupation"])

print("I have " + str(len(my\_info["hobbies"])) + " hobbies!")

print("On the weekend I get up at " + str(my\_info["wake-up"]["Saturday"]))

**03-Python/3/Activities/04-Ins\_keys\_values/keys\_values.py**

# Two Dictionaries

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

actor = {"name": "Adam Sandler", "genre": "comedy", "nationality": "United States"}

# Access all the keys

print(actor.keys())

# Access all values

print(actress.values())

# We can iterate through dictionaries with a for-loop

for key in actress.keys():

print("This is a key:", key)

# Looping through values

for value in actress.values():

print("This is a value:", value)

# Use items() to loop through both keys and values

for key, value in actress.items():

print("This is a key:", key)

print("This is a value:", value)

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

film = {"title": "Interstellar", "revenues": {"United States":360, "China":250, "United Kingdom":73}}

#cannot delete when you are in a loop

# Use del to delete a key-value pair from a dictionary

del film["revenues"]

print(film)

#delete only united states

del film["revenues"][United states]

print(film)

#can change value

file(revenues) = ''

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

# Add a key-value pair to a dictionary

actor["hair"] = "brown"

actor["genre"] = "drama"

print(actor)

**03-Python/3/Activities/05-Stu\_StoreManager/Solved/StoreManager.py**

store\_items = {

"footballs": 20,

"basketballs": 2,

"shorts": 80,

"bats": 2

}

manage\_system = "y"

# Run the code so long as the user wants to manage the systm

while manage\_system == "y":

# Take users input

action\_item = input("What would like to do today? (A)dd new item, (R)emove an item, or (S)how all items ")

if action\_item == "A":

# Input item to add

item\_added = input("What would you like to add? ")

amount\_added = input("How many would you like to add? ")

# Add item to dictionary

store\_items[item\_added] = amount\_added

# Print confirmation

print(item\_added + " " + amount\_added + " added!")

elif action\_item == "R":

# Input item to be removed

item\_removed = input("Which item would you like to be removed? ")

# Check if item exists

if item\_removed in store\_items:

# Remove item

del store\_items[item\_removed]

# Print confirmation

print(item\_removed + " has been removed! ")

# Print error if item is does not exist

else:

print("That item does not exist")

elif action\_item == "S":

print("Store inventory")

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

# Iterate through all keys and values

for key, value in store\_items.items():

# Print all items

print(key + ": " + str(value))

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

else:

print("Sorry the action is not available")

# Check if the user wants to continue working within the system

manage\_system = input("Would you like to continue working? (y)es or (n)o ")

**03-Python/3/Activities/05-Stu\_StoreManager/StoreManager.py**

## Store Manager

### Instructions

'''

\* Create a dictionary that has a few items already stored.

\* Prompt the user is they would like to add a new item, remove an existing one, or to display all the items currently in stock.

\* If the user would like to add a new item, ask what they would like to add and how many. Add the results to your dictionary.

\* If the user wants to remove and item, first check if that item is available then remove that item from the dictionary.

\* If the user want to display the result print out an easy to read list of all the items and amount to the console.

'''

my\_dictionary={

"toast": 2,

"jelly": 4,

"peanut butter": 5

}

keep\_going = "y"

while keep\_going == "y":

choice = input("What would you like to do (A)dd, (R)emove or (D)iplay items in stock?")

if choice == "A":

add\_item = input("What would you like to add?")

how\_many = input("How many would you like to add?")

my\_dictionary[add\_item] = how\_many

elif choice == "R":

remove\_item = input("What would you like to remove?")

if remove\_item in my\_dictionary.keys():

del my\_dictionary[remove\_item]

else:

print("This item is not available")

elif choice == "D":

#for item, quantity in my\_dictionary

print(my\_dictionary)

keep\_going = input("Do you want to continue? Enter y for yes")

**03-Python/3/Activities/06-Stu\_conditionalConundrum/Solved/conditionals\_solved.py**

x = 5

y = 10

# 1. oooo needs some work

if 2 \* x > 10:

print("Question 1 works!")

else:

print("oooo needs some work")

# 2. Question 2 we got it!

if len("Dog") < x:

print("Question 2 we got it!")

else:

print("Still missing out")

# 3. That's two thus for shooter!

x = 2

y = 5

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

print("That's two thus for shooter!")

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("Sitck to lazy river")

**03-Python/3/Activities/07-Stu\_RockPaperScissors/Solved/RPS\_Solved.py**

# 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!")

else:

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

print("Sorry. You lose.")

**03-Python/3/Activities/08-Stu\_NumberChain/Solved/NumberChain\_Solved.py**

# Initial variable to track game play

user\_play = "y"

# 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(int(user\_number) + 1):

# Print each number in the range

print(x)

# Once complete...

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

**03-Python/3/Activities/08-Stu\_NumberChain/Solved/NumberChainBonus\_Solved.py**

# Initial variable to track game play

user\_play = "y"

# Set start and last number

start\_number = 1

# 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

start\_number = x

# Once complete...

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

**03-Python/3/Activities/09-Stu\_ReadNetFlix/netflix.py**

# 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[6])

# Set variable to confirm we have found the video

found = True

# If the video is never found, alert the user

if found == False:

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

**03-Python/3/Activities/10-Stu\_UdemyZip/Solved/web\_solved.py**

import os

import csv

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

# Lists to store data

title = []

price = []

subscribers = []

reviews = []

review\_percent = []

length = []

with open(netflix\_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

# 4 hours in time

# new\_length = ["4", "hours", "in", "time"]

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

length.append(new\_length[0])

# Zip lists together

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

print(cleaned\_csv)

# 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)

**03-Python/3/Activities/11-Stu\_Shifty/Solved/shifty\_solved\_Regular.py**

import string

# List of letters in the alphabet

regularAlphabet = ['a', 'b', 'c', 'd', 'e', 'f',

'g', 'h', 'i', 'j', 'k', 'l',

'm', 'n', 'o', 'p', 'q', 'r',

's', 't', 'u', 'v', 'w', 'x',

'y', 'z']

# Empty list for the shifted letters of the cipher

cipherAlphabet = []

# Sentence to encode

word = input("Word to be ciphered: ")

# Number of letters we will be shifting to create our cipher

shiftNumber = int(input("Shift number? "))

# Variable to hold our encoded sentence

new\_word = ""

# Build the Encoder (Cipher) List

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

# Loop through the alphabet

for i in range(0, len(regularAlphabet)-shiftNumber):

# Position each letter shifted

cipherAlphabet.append(regularAlphabet[i+shiftNumber])

print(cipherAlphabet)

# Loop through the beginning and add to cipher list

for i in range(0, shiftNumber):

# Position each letter shifted from its original position in the alphabet

cipherAlphabet.append(regularAlphabet[i])

print(cipherAlphabet)

# Encode the Original Sentence

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

# Loops through each character in the sentence string

for i in word:

# Determine the index location of the letter in the alphabet

letterPosition = regularAlphabet.index(i)

# Add the encoded letter to the new sentence

new\_word = new\_word + cipherAlphabet[letterPosition]

# Print the sentence to the screen

print("Your ciphered word it " + new\_word)

**03-Python/3/Activities/11-Stu\_Shifty/Solved/shifty\_solved\_Bonus.py**

# List of letters in the alphabet

regularAlphabet = ['a', 'b', 'c', 'd', 'e', 'f',

'g', 'h', 'i', 'j', 'k', 'l',

'm', 'n', 'o', 'p', 'q', 'r',

's', 't', 'u', 'v', 'w', 'x',

'y', 'z']

# Empty list for the shifted letters of the cipher

cipherAlphabet = []

# Sentence to encode

sentence = input("Enter a sentence to be shifted: ")

# Number of letters we will be shifting to create our cipher

shiftNumber = int(input("Enter a shift number: "))

# Variable to hold our encoded sentence

newSentence = ""

# Loop through the alphabet (26 minus the shiftNumber times to account for overage)

for i in range(0, len(regularAlphabet)-shiftNumber):

# Position each letter shifted from its original position in the alphabet

cipherAlphabet.append(regularAlphabet[i+shiftNumber])

# Loop through the beginning part of the alphabet and separately add it to the cipher list

for i in range(0, shiftNumber):

# Position each letter shifted from its original position in the alphabet

cipherAlphabet.append(regularAlphabet[i])

# Loops through each character in the sentence string

for i in sentence:

# Handle the space by skipping it

if i != " ":

# Determine the index location of the letter in the alphabet

letterPosition = regularAlphabet.index(i)

# Add the encoded letter to the new sentence

newSentence = newSentence + cipherAlphabet[letterPosition]

# If the character is a space, immediately incorporate it.

else:

newSentence = newSentence+" "

# Print the sentence to the screen

print("Your ciphened sentence is: " + newSentence)

**03-Python/3/Activities/12-Stu\_WrestlingWrite/Solved/wrestle\_write\_solved.py**

# Part 1

import os

import csv

# List of year

wrestlingYears = ['2017', '2016', '2015', '2014']

# Loop through years

for yearToCheck in wrestlingYears:

# Grab wrestling CSV

wrestlingCSV = os.path.join('..', 'raw\_data', 'WWE-Data-' + yearToCheck + '.csv')

# Create new CSV

newWrestlingCSV = os.path.join('..', 'output', 'WWE-Data-' + yearToCheck + '.csv')

# Set empty list variables

wrestlers = []

wins = []

winPercent = []

losses = []

lossPercent = []

draws = []

drawPercent = []

# Open current wrestling CSV

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

csvReader = csv.reader(csvFile, delimiter=',')

# Skipp headers

next(csvReader, None)

for row in csvReader:

# Append data from the row

wrestlers.append(row[0])

wins.append(row[1])

losses.append(row[2])

draws.append(row[3])

# Calculate percentages and append to the list

winPercent.append(int(row[1])/(int(row[1]) + int(row[2]) + int(row[3]))\*100)

lossPercent.append(int(row[2])/(int(row[1]) + int(row[2]) + int(row[3]))\*100)

drawPercent.append(int(row[3])/(int(row[1]) + int(row[2]) + int(row[3]))\*100)

# Zip lists together

cleanCSV = zip(wrestlers, wins, losses, draws, winPercent, lossPercent, drawPercent)

with open(newWrestlingCSV, 'w', newline="") as csvFile:

csvWriter = csv.writer(csvFile, delimiter=',')

# Write Headers into file

csvWriter.writerow(["Wrestler","Wins","Losses","Draws","Win Percent","Loss Percent","Draw Percent"])

# Write the zipped lists to a csv

csvWriter.writerows(cleanCSV)

**PANDAS**

**04-Pandas/01/Activities/01-Ins\_JupyterIntro/JupyterIntro.ipynb#**

# coding: utf-8

# In[2]:

# Running the basic "Hello World" code

hello = "Hello World"

print(hello)

# In[3]:

# Doing simple math

4 + 4

# In[4]:

# Storing results in variables

a = 5

# In[5]:

# Using those variables elsewhere in the code

a

# In[6]:

# Variables will hold the value most recently run

# This means that, if we run the code above, it will now print 2

a = 2

**04-Pandas/01/Activities/02-Stu\_NetflixRemix/Solved/NetflixRemix.ipynb#**

# coding: utf-8

# In[12]:

# Modules

import os

import csv

# In[13]:

# Prompt user for video lookup

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

# In[16]:

# Set path for file

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

print(csvpath)

# Set variable to check if we found the video

found = False

# In[15]:

# 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[6])

# Set variable to confirm we have found the video

found = True

# If the video is never found, alert the user

if found == False:

print("We don't seem to have what you are looking for!")

**04-Pandas/01/Activities/03-Ins\_IntroToPandas/creating\_data\_frames.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import pandas as pd

import numpy as np

# In[2]:

# We can create a Pandas Series from a raw list

data\_series = pd.Series(["UCLA", "UC Berkeley", "UC Irvine", "University of Central Florida", "Rutgers University"])

data\_series

# In[16]:

# Convert a list of dictionarys into a dataframe

states\_dicts = [{"STATE":"New Jersey","ABBREVIATION":"NJ"},{"STATE":"New York","ABBREVIATION":"NY"}]

df\_states = pd.DataFrame(states\_dicts)

df\_states

# In[15]:

# Convert a single dictionary containing lists into a dataframe

df = pd.DataFrame(

{"Dynasty": ["Early Dynastic Period","Old Kingdom"],

"Pharoh": ["Thinis","Memphis"]

}

)

df

**04-Pandas/01/Activities/04-Stu\_DataFrameShop/Solved/DataFrameShop.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import pandas as pd

# In[6]:

# Create a DataFrame of frames using a dictionary of lists

frame\_df = pd.DataFrame({

"Frame":["Ornate","Classical","Modern","Wood","Cardboard"],

"Price":[15.00, 12.50, 10.00, 5.00, 1.00],

"Sales":[100, 200, 150, 300, "N/A"]

})

frame\_df

# In[4]:

# Create a DataFrame of paintings using a list of dictionaries

painting\_df = pd.DataFrame([

{"Painting":"Mona Lisa (Knockoff)", "Price":25, "Popularity":"Very Popular"},

{"Painting":"Van Gogh (Knockoff)", "Price":20, "Popularity":"Popular"},

{"Painting":"Starving Artist", "Price":10, "Popularity":"Average"},

{"Painting":"Toddler Drawing", "Price":1, "Popularity":"Not Popular"}

])

painting\_df

**04-Pandas/01/Activities/05-Ins\_DataFunctions/data\_functions.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import pandas as pd

# In[2]:

# Save path to data set in a variable

data\_file = "Resources/dataSet.csv"

# In[3]:

# Use Pandas to read data .head() only prints the first 5 rows of data

data\_file\_pd = pd.read\_csv(data\_file)#putting the dataframe into a variable makes is a class which has methods associated with it

data\_file\_pd.head()

# In[4]:

# Display a statistical overview of the DataFrame this gives stats on the files

data\_file\_pd.describe()

# In[19]:

# Reference a single column within a DataFrame

#dtype: show the data type

#string gets a dtype: object one string in your column will change the data type of your entire column

data\_file\_pd["Amount"].head()

# In[18]:

# Reference multiple columns within a DataFrame

#can change the order of the columns

data\_file\_pd[["Amount","Gender"]].head()

# In[5]:

# The mean method averages the series

average = data\_file\_pd["Amount"].mean()

average

# In[7]:

# The sum method adds every entry in the series

total = data\_file\_pd["Amount"].sum()

total

# In[8]:

# The unique method shows every element of the series that appears only once

unique = data\_file\_pd["Last Name"].unique()

unique

# In[9]:

# The value\_counts method counts unique values in a column

count = data\_file\_pd["Gender"].value\_counts()

count

# In[22]:

# Calculations can also be performed on Series and added into DataFrames as new columns

thousands\_of\_dollars = data\_file\_pd["Amount"]/1000

data\_file\_pd["Thousands of Dollars"] = thousands\_of\_dollars

data\_file\_pd.head()

**04-Pandas/01/Activities/06-Stu\_TrainingGrounds/Solved/TrainingGrounds.ipynb**

# coding: utf-8

# In[9]:

# Import Dependencies

import pandas as pd

import random

# In[36]:

# A seriously gigantic DataFrame of individuals' names, their trainers, their weight, and their days as gym members

training\_data = pd.DataFrame({

"Name":["Gino Walker","Hiedi Wasser","Kerrie Wetzel","Elizabeth Sackett","Jack Mitten","Madalene Wayman","Jamee Horvath","Arlena Reddin","Tula Levan","Teisha Dreier","Leslie Carrier","Arlette Hartson","Romana Merkle","Heath Viviani","Andres Zimmer","Allyson Osman","Yadira Caggiano","Jeanmarie Friedrichs","Leann Ussery","Bee Mom","Pandora Charland","Karena Wooten","Elizabet Albanese","Augusta Borjas","Erma Yadon","Belia Lenser","Karmen Sancho","Edison Mannion","Sonja Hornsby","Morgan Frei","Florencio Murphy","Christoper Hertel","Thalia Stepney","Tarah Argento","Nicol Canfield","Pok Moretti","Barbera Stallings","Muoi Kelso","Cicely Ritz","Sid Demelo","Eura Langan","Vanita An","Frieda Fuhr","Ernest Fitzhenry","Ashlyn Tash","Melodi Mclendon","Rochell Leblanc","Jacqui Reasons","Freeda Mccroy","Vanna Runk","Florinda Milot","Cierra Lecompte","Nancey Kysar","Latasha Dalton","Charlyn Rinaldi","Erline Averett","Mariko Hillary","Rosalyn Trigg","Sherwood Brauer","Hortencia Olesen","Delana Kohut","Geoffrey Mcdade","Iona Delancey","Donnie Read","Cesar Bhatia","Evia Slate","Kaye Hugo","Denise Vento","Lang Kittle","Sherry Whittenberg","Jodi Bracero","Tamera Linneman","Katheryn Koelling","Tonia Shorty","Misha Baxley","Lisbeth Goering","Merle Ladwig","Tammie Omar","Jesusa Avilla","Alda Zabala","Junita Dogan","Jessia Anglin","Peggie Scranton","Dania Clodfelter","Janis Mccarthy","Edmund Galusha","Tonisha Posey","Arvilla Medley","Briana Barbour","Delfina Kiger","Nia Lenig","Ricarda Bulow","Odell Carson","Nydia Clonts","Andree Resendez","Daniela Puma","Sherill Paavola","Gilbert Bloomquist","Shanon Mach","Justin Bangert","Arden Hokanson","Evelyne Bridge","Hee Simek","Ward Deangelis","Jodie Childs","Janis Boehme","Beaulah Glowacki","Denver Stoneham","Tarra Vinton","Deborah Hummell","Ulysses Neil","Kathryn Marques","Rosanna Dake","Gavin Wheat","Tameka Stoke","Janella Clear","Kaye Ciriaco","Suk Bloxham","Gracia Whaley","Philomena Hemingway","Claudette Vaillancourt","Olevia Piche","Trey Chiles","Idalia Scardina","Jenine Tremble","Herbert Krider","Alycia Schrock","Miss Weibel","Pearlene Neidert","Kina Callender","Charlotte Skelley","Theodora Harrigan","Sydney Shreffler","Annamae Trinidad","Tobi Mumme","Rosia Elliot","Debbra Putt","Rena Delosantos","Genna Grennan","Nieves Huf","Berry Lugo","Ayana Verdugo","Joaquin Mazzei","Doris Harmon","Patience Poss","Magaret Zabel","Marylynn Hinojos","Earlene Marcantel","Yuki Evensen","Rema Gay","Delana Haak","Patricia Fetters","Vinnie Elrod","Octavia Bellew","Burma Revard","Lakenya Kato","Vinita Buchner","Sierra Margulies","Shae Funderburg","Jenae Groleau","Louetta Howie","Astrid Duffer","Caron Altizer","Kymberly Amavisca","Mohammad Diedrich","Thora Wrinkle","Bethel Wiemann","Patria Millet","Eldridge Burbach","Alyson Eddie","Zula Hanna","Devin Goodwin","Felipa Kirkwood","Kurtis Kempf","Kasey Lenart","Deena Blankenship","Kandra Wargo","Sherrie Cieslak","Ron Atha","Reggie Barreiro","Daria Saulter","Tandra Eastman","Donnell Lucious","Talisha Rosner","Emiko Bergh","Terresa Launius","Margy Hoobler","Marylou Stelling","Lavonne Justice","Kala Langstaff","China Truett","Louanne Dussault","Thomasena Samaniego","Charlesetta Tarbell","Fatimah Lade","Malisa Cantero","Florencia Litten","Francina Fraise","Patsy London","Deloris Mclaughlin"],

"Trainer":['Bettyann Savory','Mariah Barberio','Gordon Perrine','Pa Dargan','Blanch Victoria','Aldo Byler','Aldo Byler','Williams Camire','Junie Ritenour','Gordon Perrine','Bettyann Savory','Mariah Barberio','Aldo Byler','Barton Stecklein','Bettyann Savory','Barton Stecklein','Gordon Perrine','Pa Dargan','Aldo Byler','Brittani Brin','Bettyann Savory','Phyliss Houk','Bettyann Savory','Junie Ritenour','Aldo Byler','Calvin North','Brittani Brin','Junie Ritenour','Blanch Victoria','Brittani Brin','Bettyann Savory','Blanch Victoria','Mariah Barberio','Bettyann Savory','Blanch Victoria','Brittani Brin','Junie Ritenour','Pa Dargan','Gordon Perrine','Phyliss Houk','Pa Dargan','Mariah Barberio','Phyliss Houk','Phyliss Houk','Calvin North','Williams Camire','Brittani Brin','Gordon Perrine','Bettyann Savory','Bettyann Savory','Pa Dargan','Phyliss Houk','Barton Stecklein','Blanch Victoria','Coleman Dunmire','Phyliss Houk','Blanch Victoria','Pa Dargan','Harland Coolidge','Calvin North','Bettyann Savory','Phyliss Houk','Bettyann Savory','Harland Coolidge','Gordon Perrine','Junie Ritenour','Harland Coolidge','Blanch Victoria','Mariah Barberio','Coleman Dunmire','Aldo Byler','Bettyann Savory','Gordon Perrine','Bettyann Savory','Barton Stecklein','Harland Coolidge','Aldo Byler','Aldo Byler','Pa Dargan','Junie Ritenour','Brittani Brin','Junie Ritenour','Gordon Perrine','Mariah Barberio','Mariah Barberio','Mariah Barberio','Bettyann Savory','Brittani Brin','Aldo Byler','Phyliss Houk','Blanch Victoria','Pa Dargan','Phyliss Houk','Brittani Brin','Barton Stecklein','Coleman Dunmire','Bettyann Savory','Bettyann Savory','Gordon Perrine','Blanch Victoria','Junie Ritenour','Phyliss Houk','Coleman Dunmire','Williams Camire','Harland Coolidge','Williams Camire','Aldo Byler','Harland Coolidge','Gordon Perrine','Brittani Brin','Coleman Dunmire','Calvin North','Phyliss Houk','Brittani Brin','Aldo Byler','Bettyann Savory','Brittani Brin','Gordon Perrine','Calvin North','Harland Coolidge','Coleman Dunmire','Harland Coolidge','Aldo Byler','Junie Ritenour','Blanch Victoria','Harland Coolidge','Blanch Victoria','Junie Ritenour','Harland Coolidge','Junie Ritenour','Gordon Perrine','Brittani Brin','Coleman Dunmire','Williams Camire','Junie Ritenour','Brittani Brin','Calvin North','Barton Stecklein','Barton Stecklein','Mariah Barberio','Coleman Dunmire','Bettyann Savory','Mariah Barberio','Pa Dargan','Barton Stecklein','Coleman Dunmire','Brittani Brin','Barton Stecklein','Pa Dargan','Barton Stecklein','Junie Ritenour','Bettyann Savory','Williams Camire','Pa Dargan','Calvin North','Williams Camire','Coleman Dunmire','Aldo Byler','Barton Stecklein','Coleman Dunmire','Blanch Victoria','Mariah Barberio','Mariah Barberio','Harland Coolidge','Barton Stecklein','Phyliss Houk','Pa Dargan','Bettyann Savory','Barton Stecklein','Harland Coolidge','Junie Ritenour','Pa Dargan','Mariah Barberio','Blanch Victoria','Williams Camire','Phyliss Houk','Phyliss Houk','Coleman Dunmire','Mariah Barberio','Gordon Perrine','Coleman Dunmire','Brittani Brin','Pa Dargan','Coleman Dunmire','Brittani Brin','Blanch Victoria','Coleman Dunmire','Gordon Perrine','Coleman Dunmire','Aldo Byler','Aldo Byler','Mariah Barberio','Williams Camire','Phyliss Houk','Aldo Byler','Williams Camire','Aldo Byler','Williams Camire','Coleman Dunmire','Phyliss Houk'],

"Weight":[128,180,193,177,237,166,224,208,177,241,114,161,162,151,220,142,193,193,124,130,132,141,190,239,213,131,172,127,184,157,215,122,181,240,218,205,239,217,234,158,180,131,194,171,177,110,117,114,217,123,248,189,198,127,182,121,224,111,151,170,188,150,137,231,222,186,139,175,178,246,150,154,129,216,144,198,228,183,173,129,157,199,186,232,172,157,246,239,214,161,132,208,187,224,164,177,175,224,219,235,112,241,243,179,208,196,131,207,182,233,191,162,173,197,190,182,231,196,196,143,250,174,138,135,164,204,235,192,114,179,215,127,185,213,250,213,153,217,176,190,119,167,118,208,113,206,200,236,159,218,168,159,156,183,121,203,215,209,179,219,174,220,129,188,217,250,166,157,112,236,182,144,189,243,238,147,165,115,160,134,245,174,238,157,150,184,174,134,134,248,199,165,117,119,162,112,170,224,247,217],

"Membership (Days)":[52,70,148,124,186,157,127,155,37,185,158,129,93,69,124,13,76,153,164,161,48,121,167,69,39,163,7,34,176,169,108,162,195,86,155,77,197,200,80,142,179,67,58,145,188,147,125,15,13,173,125,4,61,29,132,110,62,137,197,135,162,174,32,151,149,65,18,42,63,62,104,200,189,40,38,199,1,12,8,2,195,30,7,72,130,144,2,34,200,143,43,196,22,115,171,54,143,59,14,52,109,115,187,185,26,19,178,18,120,169,45,52,130,69,168,178,96,22,78,152,39,51,118,130,60,156,108,69,103,158,165,142,86,91,117,77,57,169,86,188,97,111,22,83,81,177,163,35,12,164,21,181,171,138,22,107,58,51,38,128,19,193,157,13,104,89,13,10,26,190,179,101,7,159,100,49,120,109,56,199,51,108,47,171,69,162,74,119,148,88,32,159,65,146,140,171,88,18,59,13]

})

training\_data.head()

# In[37]:

# Collecting a summary of all numeric data

training\_data.describe()

# In[38]:

# Finding how the names of the trainers

training\_data["Trainer"].unique()

# In[39]:

# Finding how many students each trainer has

training\_data["Trainer"].value\_counts()

# In[40]:

# Finding the average weight of all students

training\_data["Weight"].mean()

# In[44]:

# Finding the combined weight of all students

training\_data["Weight"].sum()

# In[42]:

# Converting the membership days into weeks and then adding a column to the DataFrame

weeks = training\_data["Membership (Days)"]/7

training\_data["Membership (Weeks)"] = weeks

training\_data.head()

**04-Pandas/01/Activities/07-Ins\_ColumnManipulation/ColumnManipulation.ipynb**

# coding: utf-8

# In[2]:

# Import Dependencies

import pandas as pd

# In[3]:

# A gigantic DataFrame of individuals' names, their trainers, their weight, and their days as gym members

training\_data = pd.DataFrame({

"Name":["Gino Walker","Hiedi Wasser","Kerrie Wetzel","Elizabeth Sackett","Jack Mitten","Madalene Wayman","Jamee Horvath","Arlena Reddin","Tula Levan","Teisha Dreier","Leslie Carrier","Arlette Hartson","Romana Merkle","Heath Viviani","Andres Zimmer","Allyson Osman","Yadira Caggiano","Jeanmarie Friedrichs","Leann Ussery","Bee Mom","Pandora Charland","Karena Wooten","Elizabet Albanese","Augusta Borjas","Erma Yadon","Belia Lenser","Karmen Sancho","Edison Mannion","Sonja Hornsby","Morgan Frei","Florencio Murphy","Christoper Hertel","Thalia Stepney","Tarah Argento","Nicol Canfield","Pok Moretti","Barbera Stallings","Muoi Kelso","Cicely Ritz","Sid Demelo","Eura Langan","Vanita An","Frieda Fuhr","Ernest Fitzhenry","Ashlyn Tash","Melodi Mclendon","Rochell Leblanc","Jacqui Reasons","Freeda Mccroy","Vanna Runk","Florinda Milot","Cierra Lecompte","Nancey Kysar","Latasha Dalton","Charlyn Rinaldi","Erline Averett","Mariko Hillary","Rosalyn Trigg","Sherwood Brauer","Hortencia Olesen","Delana Kohut","Geoffrey Mcdade","Iona Delancey","Donnie Read","Cesar Bhatia","Evia Slate","Kaye Hugo","Denise Vento","Lang Kittle","Sherry Whittenberg","Jodi Bracero","Tamera Linneman","Katheryn Koelling","Tonia Shorty","Misha Baxley","Lisbeth Goering","Merle Ladwig","Tammie Omar","Jesusa Avilla","Alda Zabala","Junita Dogan","Jessia Anglin","Peggie Scranton","Dania Clodfelter","Janis Mccarthy","Edmund Galusha","Tonisha Posey","Arvilla Medley","Briana Barbour","Delfina Kiger","Nia Lenig","Ricarda Bulow","Odell Carson","Nydia Clonts","Andree Resendez","Daniela Puma","Sherill Paavola","Gilbert Bloomquist","Shanon Mach","Justin Bangert","Arden Hokanson","Evelyne Bridge","Hee Simek","Ward Deangelis","Jodie Childs","Janis Boehme","Beaulah Glowacki","Denver Stoneham","Tarra Vinton","Deborah Hummell","Ulysses Neil","Kathryn Marques","Rosanna Dake","Gavin Wheat","Tameka Stoke","Janella Clear","Kaye Ciriaco","Suk Bloxham","Gracia Whaley","Philomena Hemingway","Claudette Vaillancourt","Olevia Piche","Trey Chiles","Idalia Scardina","Jenine Tremble","Herbert Krider","Alycia Schrock","Miss Weibel","Pearlene Neidert","Kina Callender","Charlotte Skelley","Theodora Harrigan","Sydney Shreffler","Annamae Trinidad","Tobi Mumme","Rosia Elliot","Debbra Putt","Rena Delosantos","Genna Grennan","Nieves Huf","Berry Lugo","Ayana Verdugo","Joaquin Mazzei","Doris Harmon","Patience Poss","Magaret Zabel","Marylynn Hinojos","Earlene Marcantel","Yuki Evensen","Rema Gay","Delana Haak","Patricia Fetters","Vinnie Elrod","Octavia Bellew","Burma Revard","Lakenya Kato","Vinita Buchner","Sierra Margulies","Shae Funderburg","Jenae Groleau","Louetta Howie","Astrid Duffer","Caron Altizer","Kymberly Amavisca","Mohammad Diedrich","Thora Wrinkle","Bethel Wiemann","Patria Millet","Eldridge Burbach","Alyson Eddie","Zula Hanna","Devin Goodwin","Felipa Kirkwood","Kurtis Kempf","Kasey Lenart","Deena Blankenship","Kandra Wargo","Sherrie Cieslak","Ron Atha","Reggie Barreiro","Daria Saulter","Tandra Eastman","Donnell Lucious","Talisha Rosner","Emiko Bergh","Terresa Launius","Margy Hoobler","Marylou Stelling","Lavonne Justice","Kala Langstaff","China Truett","Louanne Dussault","Thomasena Samaniego","Charlesetta Tarbell","Fatimah Lade","Malisa Cantero","Florencia Litten","Francina Fraise","Patsy London","Deloris Mclaughlin"],

"Trainer":['Bettyann Savory','Mariah Barberio','Gordon Perrine','Pa Dargan','Blanch Victoria','Aldo Byler','Aldo Byler','Williams Camire','Junie Ritenour','Gordon Perrine','Bettyann Savory','Mariah Barberio','Aldo Byler','Barton Stecklein','Bettyann Savory','Barton Stecklein','Gordon Perrine','Pa Dargan','Aldo Byler','Brittani Brin','Bettyann Savory','Phyliss Houk','Bettyann Savory','Junie Ritenour','Aldo Byler','Calvin North','Brittani Brin','Junie Ritenour','Blanch Victoria','Brittani Brin','Bettyann Savory','Blanch Victoria','Mariah Barberio','Bettyann Savory','Blanch Victoria','Brittani Brin','Junie Ritenour','Pa Dargan','Gordon Perrine','Phyliss Houk','Pa Dargan','Mariah Barberio','Phyliss Houk','Phyliss Houk','Calvin North','Williams Camire','Brittani Brin','Gordon Perrine','Bettyann Savory','Bettyann Savory','Pa Dargan','Phyliss Houk','Barton Stecklein','Blanch Victoria','Coleman Dunmire','Phyliss Houk','Blanch Victoria','Pa Dargan','Harland Coolidge','Calvin North','Bettyann Savory','Phyliss Houk','Bettyann Savory','Harland Coolidge','Gordon Perrine','Junie Ritenour','Harland Coolidge','Blanch Victoria','Mariah Barberio','Coleman Dunmire','Aldo Byler','Bettyann Savory','Gordon Perrine','Bettyann Savory','Barton Stecklein','Harland Coolidge','Aldo Byler','Aldo Byler','Pa Dargan','Junie Ritenour','Brittani Brin','Junie Ritenour','Gordon Perrine','Mariah Barberio','Mariah Barberio','Mariah Barberio','Bettyann Savory','Brittani Brin','Aldo Byler','Phyliss Houk','Blanch Victoria','Pa Dargan','Phyliss Houk','Brittani Brin','Barton Stecklein','Coleman Dunmire','Bettyann Savory','Bettyann Savory','Gordon Perrine','Blanch Victoria','Junie Ritenour','Phyliss Houk','Coleman Dunmire','Williams Camire','Harland Coolidge','Williams Camire','Aldo Byler','Harland Coolidge','Gordon Perrine','Brittani Brin','Coleman Dunmire','Calvin North','Phyliss Houk','Brittani Brin','Aldo Byler','Bettyann Savory','Brittani Brin','Gordon Perrine','Calvin North','Harland Coolidge','Coleman Dunmire','Harland Coolidge','Aldo Byler','Junie Ritenour','Blanch Victoria','Harland Coolidge','Blanch Victoria','Junie Ritenour','Harland Coolidge','Junie Ritenour','Gordon Perrine','Brittani Brin','Coleman Dunmire','Williams Camire','Junie Ritenour','Brittani Brin','Calvin North','Barton Stecklein','Barton Stecklein','Mariah Barberio','Coleman Dunmire','Bettyann Savory','Mariah Barberio','Pa Dargan','Barton Stecklein','Coleman Dunmire','Brittani Brin','Barton Stecklein','Pa Dargan','Barton Stecklein','Junie Ritenour','Bettyann Savory','Williams Camire','Pa Dargan','Calvin North','Williams Camire','Coleman Dunmire','Aldo Byler','Barton Stecklein','Coleman Dunmire','Blanch Victoria','Mariah Barberio','Mariah Barberio','Harland Coolidge','Barton Stecklein','Phyliss Houk','Pa Dargan','Bettyann Savory','Barton Stecklein','Harland Coolidge','Junie Ritenour','Pa Dargan','Mariah Barberio','Blanch Victoria','Williams Camire','Phyliss Houk','Phyliss Houk','Coleman Dunmire','Mariah Barberio','Gordon Perrine','Coleman Dunmire','Brittani Brin','Pa Dargan','Coleman Dunmire','Brittani Brin','Blanch Victoria','Coleman Dunmire','Gordon Perrine','Coleman Dunmire','Aldo Byler','Aldo Byler','Mariah Barberio','Williams Camire','Phyliss Houk','Aldo Byler','Williams Camire','Aldo Byler','Williams Camire','Coleman Dunmire','Phyliss Houk'],

"Weight":[128,180,193,177,237,166,224,208,177,241,114,161,162,151,220,142,193,193,124,130,132,141,190,239,213,131,172,127,184,157,215,122,181,240,218,205,239,217,234,158,180,131,194,171,177,110,117,114,217,123,248,189,198,127,182,121,224,111,151,170,188,150,137,231,222,186,139,175,178,246,150,154,129,216,144,198,228,183,173,129,157,199,186,232,172,157,246,239,214,161,132,208,187,224,164,177,175,224,219,235,112,241,243,179,208,196,131,207,182,233,191,162,173,197,190,182,231,196,196,143,250,174,138,135,164,204,235,192,114,179,215,127,185,213,250,213,153,217,176,190,119,167,118,208,113,206,200,236,159,218,168,159,156,183,121,203,215,209,179,219,174,220,129,188,217,250,166,157,112,236,182,144,189,243,238,147,165,115,160,134,245,174,238,157,150,184,174,134,134,248,199,165,117,119,162,112,170,224,247,217],

"Membership(Days)":[52,70,148,124,186,157,127,155,37,185,158,129,93,69,124,13,76,153,164,161,48,121,167,69,39,163,7,34,176,169,108,162,195,86,155,77,197,200,80,142,179,67,58,145,188,147,125,15,13,173,125,4,61,29,132,110,62,137,197,135,162,174,32,151,149,65,18,42,63,62,104,200,189,40,38,199,1,12,8,2,195,30,7,72,130,144,2,34,200,143,43,196,22,115,171,54,143,59,14,52,109,115,187,185,26,19,178,18,120,169,45,52,130,69,168,178,96,22,78,152,39,51,118,130,60,156,108,69,103,158,165,142,86,91,117,77,57,169,86,188,97,111,22,83,81,177,163,35,12,164,21,181,171,138,22,107,58,51,38,128,19,193,157,13,104,89,13,10,26,190,179,101,7,159,100,49,120,109,56,199,51,108,47,171,69,162,74,119,148,88,32,159,65,146,140,171,88,18,59,13]

})

training\_data.head(10)

# In[4]:

# Collecting a list of all columns within the DataFrame

training\_data.columns #references the metadata of the dataframe

training\_Data.columns.tolist #this is a function of the variable

# In[6]:

# Reorganizing the columns using double brackets

organized\_df = training\_data[["Name","Trainer","Weight","Membership(Days)"]]#always use [] this is passing a list[] to the dataframe

organized\_df.head()

# In[8]:

# Using .rename(columns={}) in order to rename columns

renamed\_df = organized\_df.rename(columns={"Membership(Days)":"Membership in Days", "Weight":"Weight in Pounds"})

renamed\_df.head()

**04-Pandas/01/Activities/08-Stu\_Hey\_Arnold/Solved/hey\_arnold.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import pandas as pd

# In[2]:

# Create a DataFrame with given columns and value

hey\_arnold = pd.DataFrame(

{"Character\_in\_show": ["Arnold", "Gerald", "Helga", "Phoebe", "Harold", "Eugene"],

"color\_of\_hair": ["blonde", "black","blonde", "black", "unknown", "red"],

"Height": ["average", "tallish", "tallish", "short", "tall", "short"],

"Football\_Shaped\_Head": [True, False, False, False, False, False]

})

hey\_arnold

# In[3]:

# Rename columns for readability

hey\_arnold\_renamed = hey\_arnold.rename(columns={"Character\_in\_show": "Character",

"color\_of\_hair": "Hair Color",

"Height": "Height",

"Football\_Shaped\_Head": "Football Head"

})

hey\_arnold\_renamed

# In[4]:

# Organize the columns so they are in a more logical order

hey\_arnold\_alphabetical = hey\_arnold\_renamed[["Character","Football Head","Hair Color","Height"]]

hey\_arnold\_alphabetical

**04-Pandas/01/Activities/09-Ins\_ReadingWritingCSV/pandas\_reading\_files.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import pandas as pd

# In[2]:

# Store filepath in a variable

-file\_one = "Resources/DataOne.csv"

# In[3]:

# Read our Data file with the pandas library

# Not every CSV requires an encoding, but be aware this can come up

file\_one\_df = pd.read\_csv(file\_one, encoding = "ISO-8859-1")

# In[10]:

# Show just the header

file\_one\_df.head()

# In[6]:

# Show a single column

file\_one\_df["first\_name"].head()

# In[7]:

# Show mulitple specific columns--note the extra brackets

file\_one\_df[["first\_name", "email"]].head()

# In[8]:

# Head does not change the DataFrame--it only displays it

file\_one\_df.head()

# In[12]:

# Export file as a CSV, without the Pandas index, but with the header

file\_one\_df.to\_csv("Output/fileOne.csv", index=False, header=True)

**04-Pandas/01/Activities/10-Stu\_GoodReads/Solved/GoodReads.ipynb**

# coding: utf-8

# In[25]:

# Import Dependencies

import pandas as pd

# In[26]:

# Make a reference to the books.csv file path

csv\_path = "../Resources/books.csv"

# Import the books.csv file as a DataFrame

books\_df = pd.read\_csv(csv\_path, encoding="utf-8")

books\_df.head()

# In[27]:

# Remove unecessary columns from the DataFrame and save the new DataFrame

reduced\_df = books\_df[["isbn","original\_publication\_year","original\_title","authors",

"ratings\_1","ratings\_2","ratings\_3","ratings\_4","ratings\_5"]]

reduced\_df.head()

# In[28]:

# Rename the headers to be more explanatory

renamed\_df = reduced\_df.rename(columns={"isbn":"ISBN",

"original\_title":"Original Title",

"original\_publication\_year":"Publication Year",

"authors":"Authors",

"ratings\_1":"One Star Reviews",

"ratings\_2":"Two Star Reviews",

"ratings\_3":"Three Star Reviews",

"ratings\_4":"Four Star Reviews",

"ratings\_5":"Five Star Reviews",})

renamed\_df.head()

# In[29]:

# Push the remade DataFrame to a new CSV file

renamed\_df.to\_csv("Output/books\_clean.csv", encoding="utf-8", index=False, header=True)

**04-Pandas/01/Activities/11-Stu\_GoodReadsSummary/Solved/GoodReadsSummary.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import pandas as pd

# In[4]:

# File to Load

goodreads\_path = "../Resources/books\_clean.csv"

# Read the modified GoodReads csv and store into Pandas DataFrame

goodreads\_df = pd.read\_csv(goodreads\_path, encoding="utf-8")

goodreads\_df.head()

# In[11]:

# Calculate the number of unique authors in the DataFrame

author\_count = len(goodreads\_df["Authors"].unique())

# Calculate the earliest/latest year a book was published

earliest\_year = goodreads\_df["Publication Year"].min()

latest\_year = goodreads\_df["Publication Year"].max()

# Calculate the total reviews for the entire dataset

total\_reviews = goodreads\_df["One Star Reviews"].sum() + goodreads\_df["Two Star Reviews"].sum() + goodreads\_df["Three Star Reviews"].sum() + goodreads\_df["Four Star Reviews"].sum() +goodreads\_df["Five Star Reviews"].sum()

# In[12]:

# Place all of the data found into a summary DataFrame

summary\_table = pd.DataFrame({"Total Unique Authors":author\_count,

"Earliest Year":[earliest\_year],

"Latest Year":[latest\_year],

"Total Reviews":[total\_reviews]})

summary\_table

**04-Pandas/01/Activities/12-Ins\_LocAndIloc/LocAndIloc.ipynb**

# coding: utf-8

# In[2]:

import pandas as pd

# In[3]:

file = "Resources/sampleData.csv"

# In[4]:

df\_original = pd.read\_csv(file)

df\_original.head()

# In[20]:

# Set new index to last\_name

df = df\_original.set\_index("last\_name")

df.head()

# In[8]:

# Grab the data contained within the "Berry" row and the "Phone Number" column

berry\_phone = df.loc["Berry", "Phone Number"]

print("Using Loc: " + berry\_phone)

#iloc is zero indexed integer location

#iloc to get indivual items

also\_berry\_phone = df.iloc[1,2]

print("Using Iloc: " + also\_berry\_phone)

# In[17]:

# Grab the first five rows of data and the columns from "id" to "Phone Number"

# The problem with using "last\_name" as the index is that the values are not unique so duplicates are returned

# If there are duplicates and loc[] is being used, Pandas will return an error

richardson\_to\_morales = df.loc[["Richardson", "Berry", "Hudson", "Mcdonald", "Morales"],["id", "first\_name", "Phone Number"]]

print(richardson\_to\_morales)

print()

# Using iloc[] will not find duplicates since a numeric index is always unique

#index slicing iloc[row, column] not inclusive of the last number zero based

also\_richardson\_to\_morales = df.iloc[0:4,0:3]

print(also\_richardson\_to\_morales)

# In[18]:

# The following will select all rows for columns `first\_name` and `Phone Number`

df.loc[:, ["first\_name", "Phone Number"]].head()

# In[27]:

# Loc and Iloc also allow for conditional statments to filter rows of data

only\_billys = df.loc[df["first\_name"] == "Billy",:]

print(only\_billys)

print()

# Multiple conditions can be set to narrow down or widen the filter

only\_billy\_and\_peter = df.loc[(df["first\_name"] == "Billy") | (df["first\_name"] == "Peter"), :]

print(only\_billy\_and\_peter)

**04-Pandas/01/Activities/13-Stu\_GoodMovies/Solved/good\_movies.ipynb**

# coding: utf-8

# In[1]:

# Dependencie

import pandas as pd

# In[2]:

# Load in file

movie\_file = "../Resources/movie\_scores.csv"

# In[3]:

# Read and display the CSV with Pandas

movie\_file\_pd = pd.read\_csv(movie\_file)

movie\_file\_pd.head()

# In[4]:

# List all the columns in the table

movie\_file\_pd.columns

# In[5]:

# We only want IMDb data, so create a new table that takes the Film and all the columns relating to IMDB

imdb\_table = movie\_file\_pd[["FILM", "IMDB", "IMDB\_norm", "IMDB\_norm\_round", "IMDB\_user\_vote\_count"]]

imdb\_table.head()

# In[6]:

# We only like good movies, so find those that scored over 7, and ignore the norm rating

good\_movies = movie\_file\_pd.loc[movie\_file\_pd["IMDB"] > 7, ["FILM", "IMDB", "IMDB\_user\_vote\_count"]]

good\_movies.head()

# In[7]:

# Find less popular movies--i.e., those with fewer than 20K votes

unknown\_movies = good\_movies.loc[good\_movies["IMDB\_user\_vote\_count"] < 20000, ["FILM", "IMDB", "IMDB\_user\_vote\_count"]]

unknown\_movies.head()

# In[8]:

# Finally, export this file to a spread so we can keep track of out new future watch list without the index

unknown\_movies.to\_excel("../output/movieWatchlist.xlsx", index=False)

**04-Pandas/01/Activities/14-Ins\_CleaningData/CleaningData.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import pandas as pd

import numpy as np

# In[2]:

# Name of the CSV file

file = 'donors2008.csv'

# In[3]:

# The correct encoding must be used to read the CSV in pandas

df = pd.read\_csv(file, encoding = "ISO-8859-1")

# In[4]:

# Preview of the DataFrame

# Note that FIELD8 is likely a meaningless column

df.head()

# In[5]:

# Delete extraneous column

del df['FIELD8']

df.head()

# In[7]:

# Identify incomplete rows

df.count()

# In[8]:

# Drop all rows with missing information

df = df.dropna(how='any')

# In[9]:

# Verify dropped rows

df.count()

# In[10]:

# The Amount column is the wrong data type. It should be numeric.

df.dtypes

# In[71]:

# Use pd.to\_numeric() method to convert the datatype of the Amount column

df['Amount'] = pd.to\_numeric(df['Amount'])

# In[13]:

# Verify that the Amount column datatype has been made numeric

df['Amount'].dtype

# In[14]:

# Display an overview of the Employers column

df['Employer'].value\_counts()

# In[15]:

# Clean up Employer category. Replace 'Self Employed' and 'Self' with 'Self-Employed'

df['Employer'] = df['Employer'].replace({'Self Employed':'Self-Employed', 'Self': 'Self-Employed'})

# In[16]:

# Verify clean-up.

df['Employer'].value\_counts()

# In[17]:

df['Employer'] = df['Employer'].replace({'Not Employed': 'Unemployed'})

df['Employer'].value\_counts()

# In[89]:

# Display a statistical overview

# We can infer the maximum allowable individual contribution from 'max'

df.describe()

**04-Pandas/01/Activities/15-Par\_PortlandCrime/Solved/PortlandCrime.ipynb**

# coding: utf-8

# In[24]:

# Import Dependencies

import pandas as pd

# In[25]:

# Reference the file where the CSV is located

crime\_csv\_path = "../Resources/crime\_incident\_data2017.csv"

# Import the data into a Pandas DataFrame

crime\_df = pd.read\_csv(crime\_csv\_path)

crime\_df.head()

# In[27]:

# In[28]:

# Check to see if there are any values with mispelled or similar values in "Offense Type"

no\_null\_crime\_df["Offense Type"].value\_counts()

# In[35]:

# Combining similar offenses together

no\_null\_crime\_df["Offense Type"] = no\_null\_crime\_df["Offense Type"].replace({"Commercial Sex Acts":"Prostitution","Assisting or Promoting Prostitution":"Prostitution"})

no\_null\_crime\_df

# In[40]:

# Create a new DataFrame that looks into a specific neighborhood

vernon\_crime\_df = no\_null\_crime\_df.loc[no\_null\_crime\_df["Neighborhood"] == "Vernon"]

vernon\_crime\_df

**04-Pandas/02/Activities/01-Evr\_PandasRecap/Solved/PandasRecap.ipynb**

# coding: utf-8

# In[1]:

# Import the Pandas library

import pandas as pd

# In[3]:

# Create a reference the CSV file desired

csv\_path = "../Resources/ufoSightings.csv"

# Read the CSV into a Pandas DataFrame

ufo\_df = pd.read\_csv(csv\_path)

# Print the first five rows of data to the screen

ufo\_df.head()

# In[4]:

# Check to see if there are any rows with missing data

ufo\_df.count()

# In[6]:

# Remove the rows with missing data

clean\_ufo\_df = ufo\_df.dropna(how="any")

clean\_ufo\_df.count()

# In[7]:

# Filter the data so that only those sightings in the US are in a DataFrame

usa\_ufo\_df = clean\_ufo\_df.loc[clean\_ufo\_df["country"] == "us",:]

#dataframe.loc[dataframe["column"]]=="what we are filtering for",:]

usa\_ufo\_df

# In[28]:

# Count how many sightings have occured within each state

state\_counts = usa\_ufo\_df["state"].value\_counts()

state\_counts

# In[30]:

# Convert the state\_counts Series into a DataFrame

state\_ufo\_counts\_df = pd.DataFrame(state\_counts)

state\_ufo\_counts\_df.head()

# In[33]:

# Convert the column name into "Sum of Sightings"

state\_ufo\_counts\_df = state\_ufo\_counts\_df.rename(columns={"state":"Sum of Sightings"})

state\_ufo\_counts\_df.head()

# In[34]:

# Want to add up the seconds UFOs are seen? There is a problem

# Problem can be seen by examining datatypes within the DataFrame

usa\_ufo\_df.dtypes

# In[36]:

# Using to\_numeric() to convert a column's data into floats

usa\_ufo\_df["duration (seconds)"] = pd.to\_numeric(usa\_ufo\_df["duration (seconds)"])

usa\_ufo\_df.dtypes

# In[37]:

# Now it is possible to find the sum of seconds

usa\_ufo\_df["duration (seconds)"].sum()

**04-Pandas/02/Activities/02-Ins\_GroupBy/GroupBy.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import pandas as pd

# In[2]:

# Create a reference the CSV file desired

csv\_path = "Resources/ufoSightings.csv"

# Read the CSV into a Pandas DataFrame

ufo\_df = pd.read\_csv(csv\_path)

# Print the first five rows of data to the screen

ufo\_df.head()

# In[3]:

# Remove the rows with missing data

clean\_ufo\_df = ufo\_df.dropna(how="any")

clean\_ufo\_df.count()

# In[4]:

# Converting the "duration (seconds)" column's values to numeric

clean\_ufo\_df["duration (seconds)"] = pd.to\_numeric(clean\_ufo\_df["duration (seconds)"])

# In[6]:

# Filter the data so that only those sightings in the US are in a DataFrame

usa\_ufo\_df = clean\_ufo\_df.loc[clean\_ufo\_df["country"] == "us",:]

usa\_ufo\_df.head()

# In[11]:

# Count how many sightings have occured within each state

state\_counts = usa\_ufo\_df["state"].value\_counts()

state\_counts.head()

# In[8]:

# Using GroupBy in order to separate the data into fields according to "state" values

grouped\_usa\_df = usa\_ufo\_df.groupby(['state'])

# The object returned is a "GroupBy" object and cannot be viewed normally...

print(grouped\_usa\_df)

# In order to be visualized, a data function must be used...

grouped\_usa\_df.count().head(10)

# In[12]:

# Since "duration (seconds)" was converted to a numeric time, it can now be summed up per state

state\_duration = grouped\_usa\_df["duration (seconds)"].sum()

state\_duration.head()

# In[15]:

# Creating a new DataFrame using both duration and count

state\_summary\_table = pd.DataFrame({"Number of Sightings":state\_counts,

"Total Visit Time":state\_duration})

state\_summary\_table.head()

# In[21]:

# It is also possible to group a DataFrame by multiple columns

# This returns an object with multiple indexes, however, which can be harder to deal with

grouped\_international\_data = clean\_ufo\_df.groupby(['country','state'])

# Converting a GroupBy object into a DataFrame

international\_duration = pd.DataFrame(grouped\_international\_data["duration (seconds)"].sum())

international\_duration.head(10)

**04-Pandas/02/Activities/03-Par\_Pokemon/Solved/pokemon.ipynb**

# coding: utf-8

# In[16]:

# Dependencies

import pandas as pd

import numpy as np

# In[17]:

# Save file path to variable

pokemon = "../Resources/Pokemon.csv"

# In[18]:

# Read with Pandas

pokemon\_pd = pd.read\_csv(pokemon)

pokemon\_pd.head()

# In[19]:

# Create new table [[]] first [] means data frame is getting an index second [] data frame you are getting a list of columns

pokemon\_type = pokemon\_pd[["Type 1", "HP", "Attack", "Defense", "Sp. Atk", "Sp. Def", "Speed"]]

pokemon\_type.head()

# In[25]:

# Create the GroupBy object based on the "Type 1" column

pokemon\_group = pokemon\_type.groupby(["Type 1"])

# Calculate averages for combat stats using the .mean() method

pokemon\_group.mean()

# In[26]:

# Converting the averages for each Pokemon Type into a DataFrame

pokemon\_comparison = pd.DataFrame(pokemon\_group.mean())

pokemon\_comparison

# In[29]:

# Total number of points

pokemon\_comparison["Total"] = pokemon\_comparison["HP"] + pokemon\_comparison["Attack"] + pokemon\_comparison["Defense"] + pokemon\_comparison["Sp. Atk"] + pokemon\_comparison["Sp. Def"] + pokemon\_comparison["Speed"]

pokemon\_comparison["Total"]

# In[31]:

# Sort by strongest Pokemon, and reset index

strongest\_pokemon = pokemon\_comparison.sort\_values(["Total"], ascending=False)

strongest\_pokemon.reset\_index(inplace=True)

strongest\_pokemon

# In[32]:

# Save output to Excel

pokemon\_comparison.to\_excel("../output/pokemonRankings.xlsx", index=False)

**04-Pandas/02/Activities/04-Ins\_Sorting/Sorting.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import pandas as pd

# In[2]:

csv\_path = "Resources/Happiness\_2017.csv"

happiness\_df = pd.read\_csv(csv\_path)

happiness\_df.head()

# In[5]:

# Sorting the DataFrame based on "Freedom" column

# Will sort from lowest to highest if no other parameter is passed

freedom\_df = happiness\_df.sort\_values("Freedom")

freedom\_df.head()

# In[6]:

# To sort from highest to lowest, ascending=False must be passed in

freedom\_df = happiness\_df.sort\_values("Freedom", ascending=False)

freedom\_df.head()

# In[8]:

# It is possible to sort based upon multiple columns

family\_and\_generosity = happiness\_df.sort\_values(["Family","Generosity"], ascending=False)

family\_and\_generosity.head()

**04-Pandas/02/Activities/05-Stu\_SearchForTheWorst/Solved/SearchForTheWorst.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import pandas as pd

import numpy as np

# In[2]:

# Create reference to CSV file

csv\_path = "../Resources/Soccer2018Data.csv"

# Import the CSV into a pandas DataFrame

soccer\_2018\_df = pd.read\_csv(csv\_path, low\_memory=False)

soccer\_2018\_df

# In[7]:

# Collect a list of all the unique values in "Preferred Position"

soccer\_2018\_df["Preferred Position"].unique()

# In[3]:

# Looking only at strikers (ST) to start

strikers\_2018\_df = soccer\_2018\_df.loc[soccer\_2018\_df["Preferred Position"] == "ST",:]

strikers\_2018\_df.head()

# In[4]:

# Sort the DataFrame by the values in the "ST" column to find the worst

strikers\_2018\_df = strikers\_2018\_df.sort\_values("ST")

# Reset the index so that the index is now based on the sorting locations

strikers\_2018\_df = strikers\_2018\_df.reset\_index(drop=True)

strikers\_2018\_df.head()

# In[6]:

# Save all of the information collected on the worst striker

worst\_striker = strikers\_2018\_df.loc[0,:]

worst\_striker

**04-Pandas/02/Activities/06-Ins\_Merging/Merging.ipynb**

# coding: utf-8

# In[2]:

# Dependencies

import pandas as pd

# In[8]:

# Create DataFrames

raw\_data\_items = {

"customer\_id": [403, 112, 543, 999, 654],

"item": ["soda", "chips", "TV", "Laptop", "Cooler"],

"cost": [3.00, 4.50, 600, 900, 150]

}

items\_pd = pd.DataFrame(raw\_data\_items, columns=["customer\_id", "item", "cost"])

items\_pd

# In[11]:

raw\_data\_info = {

"customer\_id": [112, 403, 999, 543, 123],

"name": ["John", "Kelly", "Sam", "April", "Bobbo"],

"email": ["jman@gmail", "kelly@aol.com", "sports@school.edu", "April@yahoo.com", "HeyImBobbo@msn.com"]

}

info\_pd = pd.DataFrame(raw\_data\_info, columns=["customer\_id", "name", "email"])

info\_pd

# In[12]:

# Merge two dataframes using an inner join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id")

merge\_table

# In[13]:

# Merge two dataframes using an outer join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id", how="outer")

merge\_table

# In[14]:

# Merge two dataframes using a left join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id", how="left")

merge\_table

# In[15]:

# Merge two dataframes using a right join

merge\_table = pd.merge(info\_pd, items\_pd, on="customer\_id", how="right")

merge\_table

**04-Pandas/02/Activities/07-Stu\_Cryptocurrency/Solved/Cryptocurrency.ipynb**

# coding: utf-8

# In[27]:

# Import Dependencies

import pandas as pd

# In[28]:

bitcoin\_csv = "../Resources/bitcoin\_cash\_price.csv"

dash\_csv = "../Resources/dash\_price.csv"

# In[29]:

bitcoin\_df = pd.read\_csv(bitcoin\_csv)

dash\_df = pd.read\_csv(dash\_csv)

# In[30]:

bitcoin\_df.head()

# In[31]:

dash\_df.head()

# In[32]:

# Merge the two DataFrames together based on the Dates they share

crypto\_df = pd.merge(bitcoin\_df, dash\_df, on="Date")

crypto\_df.head()

# In[33]:

# Rename columns so that they are differentiated

crypto\_df = crypto\_df.rename(columns={"Open\_x":"Bitcoin Open","High\_x":"Bitcoin High","Low\_x":"Bitcoin Low",

"Close\_x":"Bitcoin Close","Volume\_x":"Bitcoin Volume","Market Cap\_x":"Bitcoin Market Cap"})

crypto\_df = crypto\_df.rename(columns={"Open\_y":"Dash Open","High\_y":"Dash High","Low\_y":"Dash Low",

"Close\_y":"Dash Close","Volume\_y":"Dash Volume","Market Cap\_y":"Dash Market Cap"})

crypto\_df.head()

# In[39]:

# Collecting best open for Bitcoin and Dash

bitcoin\_open = crypto\_df["Bitcoin Open"].max()

dash\_open = crypto\_df["Dash Open"].max()

# Collecting best close for Bitcoin and Dash

bitcoin\_close = crypto\_df["Bitcoin Close"].max()

dash\_close = crypto\_df["Dash Close"].max()

# Collecting the total volume for Bitcoin and Dash

bitcoin\_volume = round(crypto\_df["Bitcoin Volume"].sum()/1000000, 2)

dash\_volume = round(crypto\_df["Dash Volume"].sum()/1000000, 2)

# In[40]:

# Creating a summary DataFrame using above values

summary\_df = pd.DataFrame({"Best Bitcoin Open":[bitcoin\_open],

"Best Bitcoin Close":[bitcoin\_close],

"Total Bitcoin Volume":str(bitcoin\_volume)+" million",

"Best Dash Open":[dash\_open],

"Best Dash Close":[dash\_close],

"Total Dash Volume":str(dash\_volume)+" million"})

summary\_df

**04-Pandas/02/Activities/08-Ins\_Binning/Binning.ipynb**

# coding: utf-8

# In[17]:

# Import Dependencies

import pandas as pd

# In[22]:

raw\_data = {'Regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts', 'Scouts', 'Scouts', 'Scouts'],

'Company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd','1st', '1st', '2nd', '2nd'],

'Name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'],

'Test Score': [25, 94, 57, 62, 70, 30, 94, 57, 62, 70, 62, 70]}

regiment\_df = pd.DataFrame(raw\_data)

regiment\_df

# In[23]:

# Create the bins in which Data will be held

# Bins are 0 to 25, 25 to 50, 50 to 75, 75 to 100

bins = [0, 25, 50, 75, 100]

# Create the names for the four bins

group\_names = ['Low', 'Okay', 'Good', 'Great']

# In[24]:

# Cut postTestScore and place the scores into bins

pd.cut(regiment\_df["Test Score"], bins, labels=group\_names)

# In[25]:

regiment\_df["Test Score Summary"] = pd.cut(regiment\_df["Test Score"], bins, labels=group\_names)

regiment\_df

# In[29]:

# Creating a group based off of the bins

regiment\_groups = regiment\_df.groupby("Test Score Summary")

regiment\_groups.max()

**04-Pandas/02/Activities/09-Stu\_TedTalks/Solved/BinningTed.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import pandas as pd

# In[4]:

# Create a path to the csv and read it into a Pandas DataFrame

csv\_path = "../Resources/ted\_talks.csv"

ted\_df = pd.read\_csv(csv\_path)

ted\_df.head()

# In[7]:

# Figure out the minimum and maximum views for a TED Talk

print(ted\_df["views"].max())

print(ted\_df["views"].min())

# In[51]:

# Create bins in which to place values based upon TED Talk views

bins = [0,200000,400000,600000,800000,1000000,2000000,3000000,4000000,5000000,50000000]

# Create labels for these bins

group\_labels = ["0 to 200k","200k to 400k","400k to 600k","600k to 800k","800k to 1mil","1mil to 2mil",

"2mil to 3mil","3mil to 4mil","4mil to 5mil","5mil to 50mil"]

# In[52]:

# Slice the data and place it into bins

pd.cut(ted\_df["views"],bins,labels=group\_labels).head()

# In[53]:

# Place the data series into a new column inside of the DataFrame

ted\_df["View Group"] = pd.cut(ted\_df["views"],bins,labels=group\_labels)

ted\_df.head()

# In[54]:

# Create a GroupBy object based upon "View Group"

ted\_group = ted\_df.groupby("View Group")

# Find how many rows fall into each bin

print(ted\_group["comments"].count())

# Get the average of each column within the GroupBy object

ted\_group.mean()

**04-Pandas/02/Activities/10-Ins\_Mapping/Mapping.ipynb**

# coding: utf-8

# In[2]:

import pandas as pd

# In[7]:

# Mapping lets you format an entire DataFrame

file = "sample\_data.csv"

file\_df = pd.read\_csv(file)

file\_df.head()

# In[8]:

# Use Map to format all the columns

file\_df["avg\_cost"] = file\_df["avg\_cost"].map("${:.2f}".format)

file\_df["population"] = file\_df["population"].map("{:,}".format)

file\_df["other"] = file\_df["other"].map("{:.2f}".format)

file\_df.head()

# In[9]:

# Mapping has changed the datatypes of the columns to strings

file\_df.dtypes

**04-Pandas/02/Activities/11-Stu\_CleaningKickstarter/Solved/KickstarterClean.ipynb**

# coding: utf-8

# In[1]:

import pandas as pd

# In[2]:

# The path to our CSV file

file = "Resources/KickstarterData.csv"

# Read our Kickstarter data into pandas

df = pd.read\_csv(file)

df.head()

# In[3]:

# Get a list of all of our columns for easy reference

df.columns

# In[4]:

# Extract "name", "goal", "pledged", "state", "country", "staff\_pick",

# "backers\_count", and "spotlight"

reduced\_kickstarter\_df = df.loc[:,["name", "goal", "pledged", "state", "country", "staff\_pick","backers\_count", "spotlight"]]

reduced\_kickstarter\_df

# In[5]:

# Sort out any projects that made no money at all

reduced\_kickstarter\_df = reduced\_kickstarter\_df.loc[(reduced\_kickstarter\_df["pledged"] > 0)]

reduced\_kickstarter\_df.head()

# In[6]:

# Collect only those projects that were hosted in the US

hosted\_in\_us = reduced\_kickstarter\_df.loc[reduced\_kickstarter\_df["country"] == "US"]

hosted\_in\_us.head()

# In[7]:

# Create a new column that finds the average amount pledged to a project

hosted\_in\_us["average\_donation"] = hosted\_in\_us['pledged']/hosted\_in\_us['backers\_count']

hosted\_in\_us.head()

# In[8]:

# Format our "average\_donation", "goal", and "pledged" columns

# to go to two decimal places, include a dollar sign, and use comma notation

hosted\_in\_us["average\_donation"] = hosted\_in\_us["average\_donation"].map("$ {:,.2f}".format)

hosted\_in\_us["goal"] = hosted\_in\_us["goal"].map("$ {:,.2f}".format)

hosted\_in\_us["pledged"] = hosted\_in\_us["pledged"].map("$ {:,.2f}".format)

hosted\_in\_us.head()

# In[9]:

# Calculate the total number of backers for all US projects

hosted\_in\_us["backers\_count"].sum()

# In[10]:

# Calculate the average number of backers for all US projects

hosted\_in\_us["backers\_count"].mean()

# In[11]:

# Collect only those US campaigns that have been picked as a "Staff Pick"

picked\_by\_staff = hosted\_in\_us.loc[hosted\_in\_us["staff\_pick"] == True]

picked\_by\_staff

# In[12]:

# Group by the state of the campaigns and see if staff picks matter (Seems to matter quite a bit)

state\_groups = picked\_by\_staff.groupby("state")

state\_groups["name"].count()

**04-Pandas/03/Activities/01-Ins\_IntroToBugfixing/IntroToBugfixing\_Solved.ipynb**

# coding: utf-8

# In[21]:

# Import dependencies

import pandas as pd

# In[22]:

# Reference to CSV and reading CSV into Pandas DataFrame

csv\_path = "Resources/flavors\_of\_cacao.csv"

chocolate\_ratings\_df = pd.read\_csv(csv\_path)

chocolate\_ratings\_df.head(10)

# In[23]:

chocolate\_ratings\_df.columns

# In[24]:

# Converting the "Cocoa Percent" column to floats

chocolate\_ratings\_df["Cocoa Percent"] = chocolate\_ratings\_df["Cocoa Percent"].replace('%','',regex=True).astype('float')

# Finding the average cocoa percent

chocolate\_ratings\_df["Cocoa Percent"].mean()

**04-Pandas/03/Activities/02-Stu\_BugfixingBonanza/Solved/BugfixBonanza.ipynb**

# coding: utf-8

# In[156]:

# Import Dependencies

import pandas as pd

# In[157]:

# Create a reference to the CSV and import it into a Pandas DataFrame

csv\_path = "../Resources/EclipseBugs.csv"

eclipse\_df = pd.read\_csv(csv\_path)

eclipse\_df.head()

# In[158]:

# Get a reference to the column names

eclipse\_df.columns

# In[159]:

# Removing the newlines from column headers

eclipse\_df = eclipse\_df.rename(columns={"Bug\nID":"Bug ID",

"Assignee\nReal\nName":"Assignee Real Name",

"Number of\nComments":"Number of Comments",

"Reporter\nReal\nName":"Reporter Real Name",

"Target\nMilestone":"Target Milestone"})

eclipse\_df.columns

# In[160]:

# Finding the average number of comments per bug

average\_comments = eclipse\_df["Number of Comments"].mean()

average\_comments

# In[161]:

# Grouping the DataFrame by "Assignee"

assignee\_group = eclipse\_df.groupby("Assignee")

# Count how many of each component Assignees worked on and create DataFrame of the data

assignee\_work = pd.DataFrame(assignee\_group["Component"].value\_counts())

assignee\_work.head()

# In[162]:

# Rename the "Component" column to "Component Bug Count"

assignee\_work = assignee\_work.rename(columns={"Component":"Component Bug Count"})

assignee\_work.head()

# In[167]:

# Find the percentage of bugs overall fixed by each Assignee

total\_bugs = eclipse\_df["Assignee"].count()

bugs\_per\_user = assignee\_group["Assignee"].count()

user\_bug\_percent = pd.DataFrame((bugs\_per\_user/total\_bugs)\*100)

user\_bug\_percent.head()

# In[164]:

# Rename the "Assignee" column to "Percent of Total Bugs Assigned"

user\_bug\_percent = user\_bug\_percent.rename(columns={"Assignee":"Percent of Total Bugs Assigned"})

# Reset the index for this DataFrame so "Assignee" is a column

user\_bug\_percent = user\_bug\_percent.reset\_index()

user\_bug\_percent.head()

# In[165]:

# Reset the index of "assignee\_group" so that "Assignee" and "Component" are columns

assignee\_work = assignee\_work.reset\_index()

assignee\_work.head()

# Merge the "Percent of Total Bugs Assigned" into the DataFrame

assignee\_work = assignee\_work.merge(user\_bug\_percent,on="Assignee")

# Remove the extra columns

assignee\_work = assignee\_work[["Assignee","Percent of Total Bugs Assigned",

"Component","Component Bug Count"]]

assignee\_work.head()

**04-Pandas/03/Activities/03-Stu\_Project\_Part\_1/Student\_Project\_Part1.ipynb**

# coding: utf-8

# In[2]:

import os

import pandas as pd

import numpy as np

# In[9]:

file\_path = '2016-FCC-New-Coders-Survey-Data.csv'

new\_coder\_survey\_df = pd.read\_csv(file\_path)

new\_coder\_survey\_df.head()

# In[19]:

new\_coder\_survey\_df.columns[57]

# In[18]:

new\_coder\_survey\_df.dtypes

#new\_coder\_survey\_df.iloc[:,21] #= mydata.iloc[:,columns].astype(str)

**04-Pandas/03/Activities/04-Stu\_Project\_Part\_2/Solved/project\_part\_two\_solved.ipynb**

# coding: utf-8

# In[4]:

# Dependencies

import pandas as pd

# In[5]:

# load CSV

new\_coders = "../Resources/2016-FCC-New-Coders-Survey-Data.csv"

# In[6]:

# Read with pandas--low\_memory required to suppress errors about mixed data types

new\_coders\_pd = pd.read\_csv(new\_coders, encoding='iso-8859-1', low\_memory=False)

new\_coders\_pd.head()

# In[7]:

# Take only columns 0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 29, 30, 32, 36, 37, 45, 48, 56, 110, 111

reduced\_coders\_pd = new\_coders\_pd.iloc[:, [0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11, 29, 30, 32, 36, 37, 45, 48, 56, 110, 111]]

reduced\_coders\_pd.head()

# In[8]:

# Change reading 0 and 1 to No and Yes, respectively

reduced\_coders\_pd= reduced\_coders\_pd.replace({0.0: "No", 1.0:"Yes"})

reduced\_coders\_pd.head()

# In[9]:

# Extract rows for only those who attended a bootcamp

attended\_bootcamp = reduced\_coders\_pd.loc[reduced\_coders\_pd["AttendedBootcamp"] == "Yes"]

attended\_bootcamp.head()

# In[10]:

# Create DataFrame of the different boot camps that had a significant number of attendees

bootcamp\_name = pd.DataFrame(reduced\_coders\_pd["BootcampName"].value\_counts())

bootcamp\_name.reset\_index(inplace=True)

bootcamp\_name.columns = ["BootcampName", "Count"]

msg = 'Free Code Camp is not a bootcamp - please scroll up and change answer to "no"'

# Extract only schools with a sufficient number of responses

bootcamp\_name = bootcamp\_name.loc[(bootcamp\_name["Count"] > 9) &

(bootcamp\_name["BootcampName"] != msg)]

bootcamp\_name.head()

# In[11]:

# Count how many attendees of each bootcamp would recommend the bootcamp

recommend\_bootcamp = attended\_bootcamp.replace({"Yes": 1, "No": 0})

recommend\_bootcamp = pd.DataFrame(recommend\_bootcamp.groupby("BootcampName")["BootcampRecommend"].sum())

recommend\_bootcamp.reset\_index(inplace=True)

recommend\_bootcamp.columns=["BootcampName", "Recommenders"]

recommend\_bootcamp.head()

# In[12]:

# Merge the two created data frames on the name of tbe bootcamp

merged\_df = pd.merge(bootcamp\_name, recommend\_bootcamp, on="BootcampName")

merged\_df.head()

# In[35]:

# Calculate percentage of eac bootcamp's students who are recommenders

merged\_df["% Recommend"] = merged\_df["Recommenders"] / merged\_df["Count"] \* 100

# Sort results in descending order

merged\_df = merged\_df.sort\_values(["% Recommend"], ascending=False).round(2)

# Format for percentages

merged\_df["% Recommend"] = merged\_df["% Recommend"].map("{0:,.2f}%".format)

merged\_df.head()

# In[ ]:

# Export to excel and remove index

merged\_df.to\_excel("output/BootcampOutputPart2.xlsx", index=False)

**Matplotlib**

**05-Matplotlib/01/Activities/01-Ins\_BasicLineGraphs/ExponentialChart.ipynb**

# coding: utf-8

# In[26]:

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

# In[27]:

# Creates a list from 0 to 10 with each step being 0.1 higher than the last

x\_axis = np.arange(0, 5, 0.1)

x\_axis

# In[28]:

# Creates an exponential series of values which we can then chart

e\_x = [np.exp(x) for x in x\_axis]

e\_x

# In[29]:

# Create a graph based upon the two lists we have created

plt.plot(x\_axis, e\_x)

# In[30]:

# Show the graph that we have created

plt.show()

# In[31]:

# Give our graph axis labels

plt.xlabel("Time With MatPlotLib")

plt.ylabel("How Cool MatPlotLib Seems")

# Have to plot our chart once again as it doesn't stick after being shown

plt.plot(x\_axis, e\_x)

plt.show()

**05-Matplotlib/01/Activities/01-Ins\_BasicLineGraphs/SinCos.ipynb**

# coding: utf-8

# In[20]:

# Import Numpy for calculations and matplotlib for charting

import numpy as np

import matplotlib.pyplot as plt

# In[21]:

# Create our x\_axis list

x\_axis = np.arange(0, 6, 0.1)

# In[22]:

# Creates a list based on the sin of our x\_axis values

sin = np.sin(x\_axis)

# In[23]:

# Creates a list based on the cos of our x\_axis values

cos = np.cos(x\_axis)

# In[24]:

# Plot both of these lines so that they will appear on our final chart

plt.plot(x\_axis, sin)

plt.plot(x\_axis, cos)

plt.show()

**05-Matplotlib/01/Activities/02-Stu\_NJTemp/Solved/NJ\_temp.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import numpy as np

import matplotlib.pyplot as plt

# In[3]:

# Set x axis to numerical value for month

x\_axis = np.arange(1,13,1)

x\_axis

# In[4]:

# Average weather temp

points = [39, 42, 51, 62, 72, 82, 86, 84, 77, 65, 55, 44]

# In[5]:

# Plot the line

plt.plot(x\_axis, points)

plt.show()

# In[8]:

# Convert to Celsius C = (F-39) \* 0.56

points\_C = [(x-39) \* 0.56 for x in points]

points\_C

# In[9]:

# Plot using Celsius

plt.plot(x\_axis, points\_C)

plt.show()

# In[10]:

# Plot both on the same chart

plt.plot(x\_axis, points)

plt.plot(x\_axis, points\_C)

Plt.show()

**05-Matplotlib/01/Activities/03-Ins\_ConfiguringLinePlots/line\_config.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# In[2]:

# Set x axis and variables

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# In[3]:

# Draw a horizontal line with 0.25 transparency

plt.hlines(0, 0, 10, alpha=0.25)

# In[4]:

# Assign plots to tuples that stores result of plot

# Each point on the sine chart is marked by a blue circle

sine\_handle, = plt.plot(x\_axis, sin, marker ='o', color='blue', label="Sine")

# Each point on the cosine chart is marked by a red triangle

cosine\_handle, = plt.plot(x\_axis, cos, marker='^', color='red', label="Cosine")

# In[5]:

# Adds a legend and sets its location to the lower right

plt.legend(handles=[sine\_handle, cosine\_handle], loc="lower right")

# In[6]:

# Saves an image of our chart so that we can view it in a folder

plt.savefig("lineConfig.png")

plt.show()

**05-Matplotlib/01/Activities/04-Stu\_LegendaryTemperature/Solved/LegendaryTemp.ipynb**

# coding: utf-8

# In[2]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# In[3]:

# Set x axis to numerical value for month

x\_axis = np.arange(1,13,1)

x\_axis

# In[4]:

# Avearge weather temp

points\_F = [39, 42, 51, 62, 72, 82, 86, 84, 77, 65, 55, 44]

# In[11]:

# Convert to Celsius C = (F-39) \* 0.56

points\_C = [(x-39) \* 0.56 for x in points\_F]

# In[18]:

# Create a handle for each plot

fahrenheit, = plt.plot(x\_axis, points\_F, marker="+",color="blue", linewidth=1, label="Fahreneit")

celcius, = plt.plot(x\_axis, points\_C, marker="s", color="Red", linewidth=1, label="Celcius")

# In[19]:

# Set our legend to where the chart thinks is best

plt.legend(handles=[fahrenheit, celcius], loc="best")

# In[14]:

# Create labels for the X and Y axis

plt.xlabel("Months")

plt.ylabel("Degrees")

# In[15]:

# Save and display the chart

plt.savefig("avg\_temp.png")

plt.show()

**05-Matplotlib/01/Activities/05-Ins\_Aesthetics/Aesthetics.ipynb**

# coding: utf-8

# In[12]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# In[13]:

# Generate the x values from 0 to 10 using a step of 0.1

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# In[14]:

# Add a semi-transparent horizontal line at y = 0

plt.hlines(0, 0, 10, alpha=0.25)

# In[15]:

# Use dots or other markers for your plots, and change their colors

plt.plot(x\_axis, sin, linewidth=0, marker="o", color="blue")

plt.plot(x\_axis, cos, linewidth=0, marker="^", color="red")

# In[16]:

# Add labels to the x and y axes

plt.title("Juxtaposed Sine and Cosine Curves")

plt.xlabel("Input (Sampled Real Numbers from 0 to 10)")

plt.ylabel("Value of Sine (blue) and Cosine (red)")

# In[17]:

# Set your x and y limits

plt.xlim(0, 10)

plt.ylim(-1, 1)

# In[18]:

# Set a grid on the plot

plt.grid()

# In[19]:

# Save the plot and display it

plt.savefig("sin\_cos\_with\_markers.png")

plt.show()

**05-Matplotlib/01/Activities/05-Ins\_Aesthetics/sin\_cos.py**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

# Generate the x values from 0 to 10 using a step of 0.1

x\_axis = np.arange(0, 10, 0.1)

sin = np.sin(x\_axis)

cos = np.cos(x\_axis)

# Add a semi-transparent horizontal line at y = 0

plt.hlines(0, 0, 10, alpha=0.25)

# Use dots or other markers for your plots, and change their colors

plt.plot(x\_axis, sin, linewidth=0, marker="o", color="blue")

plt.plot(x\_axis, cos, linewidth=0, marker="^", color="red")

# Add labels to the x and y axes

plt.title("Juxtaposed Sine and Cosine Curves")

plt.xlabel("Input (Sampled Real Numbers from 0 to 10)")

plt.ylabel("Value of Sine (blue) and Cosine (red)")

# Set your x and y limits

plt.xlim(0, 2 \* np.pi)

plt.ylim(-1, 1)

# Set a grid on the plot

plt.grid()

# Save the plot and display it

plt.savefig("sin\_cos\_with\_markers.png")

plt.show()

**05-Matplotlib/01/Activities/06-Stu\_RollerCoaster/Solved/CoasterSpeed.ipynb**

# coding: utf-8

# In[39]:

# Import Dependencies

import matplotlib.pyplot as plt

import numpy as np

# In[82]:

# Create the X and Y axis lists

time = np.arange(0,130,10)

speed\_chain = [9, 8, 90, 85, 80, 70, 70, 65, 55, 60, 70, 65, 50]

speed\_launch = [75, 70, 60, 65, 60, 45, 55, 50, 40, 40, 35, 35, 30]

# In[83]:

# Plot the charts and apply some styling

danger\_drop, = plt.plot(time, speed\_chain, color="red", label="Danger Drop")

railgun, = plt.plot(time, speed\_launch, color="green", label="RailGun")

# In[84]:

# Add labels to X and Y axes :: Add title

plt.title("Coaster Speed Over Time")

plt.xlabel("Coaster Runtime")

plt.ylabel("Speed (MPH)")

# In[85]:

# Set the limits for the X and Y axes

plt.xlim(0,120)

plt.ylim(5,95)

# In[86]:

# Create a legend for the chart

plt.legend(handles=[danger\_drop, railgun], loc="best")

# In[87]:

# Add in a grid for the chart

plt.grid()

# In[88]:

plt.show()

**05-Matplotlib/01/Activities/07-Ins\_BarCharts/BarChart.ipynb**

# coding: utf-8

# In[82]:

import matplotlib.pyplot as plt

import numpy as np

# In[83]:

# Create an array that contains the number of users each language has

users = [13000, 26000, 52000, 30000, 9000]

x\_axis = np.arange(len(users))

# In[84]:

# Tell matplotlib that we will be making a bar chart

# Users is our y axis and x\_axis is, of course, our x axis

# We apply align="edge" to ensure our bars line up with our tick marks

plt.bar(x\_axis, users, color='r', alpha=0.5, align="edge")

# In[85]:

# Tell matplotlib where we would like to place each of our x axis headers

tick\_locations = [value+0.4 for value in x\_axis]

plt.xticks(tick\_locations, ["Java", "C++", "Python", "Ruby", "Clojure"])

# In[86]:

# Sets the x limits of the current chart

plt.xlim(-0.25, len(x\_axis))

# In[87]:

# Sets the y limits of the current chart

plt.ylim(0, max(users)+5000)

# In[88]:

# Give our chart some labels and a tile

plt.title("Popularity of Programming Languages")

plt.xlabel("Programming Language")

plt.ylabel("Number of People Using Programming Languages")

# In[89]:

# Print our chart to the screen

plt.show()

**05-Matplotlib/01/Activities/08-Stu\_PyBars/Solved/PyBars.ipynb**

# coding: utf-8

# In[2]:

import matplotlib.pyplot as plt

import numpy as np

# In[3]:

cities = ["New Orleans", "Milwaukee", "Omaha", "Pittsburgh", "Toledo"]

bars\_in\_cities = [8.6, 8.5, 8.3, 7.9, 7.2]

x\_axis = np.arange(len(bars\_in\_cities))

# In[4]:

# Create a bar chart based upon the above data

plt.bar(x\_axis, bars\_in\_cities, color="b", align="edge")

# In[5]:

# Create the ticks for our bar chart's x axis

tick\_locations = [value+0.4 for value in x\_axis]

plt.xticks(tick\_locations, cities)

# In[6]:

# Set the limits of the x axis

plt.xlim(-0.25, len(x\_axis))

# In[7]:

# Set the limits of the y axis

plt.ylim(0, max(bars\_in\_cities)+0.4)

# In[8]:

# Give the chart a title, x label, and y label

plt.title("Density of Bars in Cities")

plt.xlabel("Cities")

plt.ylabel("Bars Per 10,000 Households")

# In[9]:

# Save an image of the chart and print it to the screen

plt.savefig("BarDensity.png")

plt.show()

**05-Matplotlib/01/Activities/09-Ins\_PieCharts/PieChart.ipynb**

# coding: utf-8

# In[1]:

# Import our dependencies

import matplotlib.pyplot as plt

import numpy as np

# In[2]:

# Labels for the sections of our pie chart

labels = ["Python", "C++", "Ruby", "Java"]

# The values of each section of the pie chart

sizes = [185, 172, 100, 110]

# The colors of each section of the pie chart

colors = ["yellowgreen", "red", "lightcoral", "lightskyblue"]

# Tells matplotlib to seperate the "Python" section from the others

explode = (0.1, 0, 0, 0)

# In[3]:

# Creates the pie chart based upon the values above

# Automatically finds the percentages of each part of the pie chart

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct="%1.1f%%", shadow=True, startangle=140)

# In[4]:

# Tells matplotlib that we want a pie chart with equal axes

plt.axis("equal")

# In[5]:

# Prints our pie chart to the screen

plt.show()

**05-Matplotlib/01/Activities/10-Stu\_PyPies/Solved/PyPie.ipynb**

# coding: utf-8

# In[1]:

import matplotlib.pyplot as plt

import numpy as np

# In[2]:

pies = ["Apple", "Pumpkin", "Chocolate Creme", "Cherry", "Apple Crumb", "Pecan", "Lemon Meringue", "Blueberry", "Key Lime", "Peach"]

pie\_votes = [47,37,32,27,25,24,24,21,18,16]

colors = ["yellow","green","lightblue","orange","red","purple","pink","yellowgreen","lightskyblue","lightcoral"]

explode = (0.1,0,0,0,0,0,0,0,0,0)

# In[3]:

# Tell matplotlib to create a bar chart based upon the above data

plt.pie(pie\_votes, explode=explode, labels=pies, colors=colors,

autopct="%1.1f%%", shadow=True, startangle=140)

# In[4]:

# Create axes which are equal so we have a perfect circle

plt.axis("equal")

# In[5]:

# Save an image of our chart and print the final product to the screen

plt.savefig("PyPies.png")

plt.show()

**05-Matplotlib/01/Activities/11-Ins\_ScatterPlots/ScatterPlot.ipynb**

# coding: utf-8

# In[8]:

# Import Dependencies

import random

import matplotlib.pyplot as plt

import numpy as np

# In[15]:

# The maximum x value for our chart will be 100

x\_limit = 100

# List of values from 0 to 100 each value being 1 greater than the last

x\_axis = np.arange(0, x\_limit, 1)

# Create a random array of data that we will use for our y values

data = [random.random() for value in x\_axis]

# In[10]:

# Tells matplotlib that we want to make a scatter plot

# The size of each point on our plot is determined by their x value

plt.scatter(x\_axis, data, marker="o", facecolors="red", edgecolors="black",

s=x\_axis, alpha=0.75)

# In[11]:

# The y limits of our scatter plot is 0 to 1

plt.ylim(0, 1)

# In[12]:

# The x limits of our scatter plot is 0 to 100

plt.xlim(0, x\_limit)

# In[13]:

# Prints the scatter plot to the screen

plt.show()

**05-Matplotlib/01/Activities/12-Stu\_ScatterPy/Solved/IceCreamSales.ipynb**

# coding: utf-8

# In[27]:

import matplotlib.pyplot as plt

import numpy as np

# In[28]:

temp = [14.2, 16.4, 11.9, 15.2, 18.5, 22.1, 19.4, 25.1, 23.4, 18.1, 22.6, 17.2]

sales = [215, 325, 185, 332, 406, 522, 412, 614, 544, 421, 445, 408]

# In[29]:

# Tell matplotlib to create a scatter plot based upon the above data

plt.scatter(temp, sales, marker="o", facecolors="red", edgecolors="black")

# In[30]:

# Set the upper and lower limits of our y axis

plt.ylim(200,625)

# In[31]:

# Set the upper and lower limits of our x axis

plt.xlim(13,25)

# In[32]:

# Create a title, x label, and y label for our chart

plt.title("Ice Cream Sales v Temperature")

plt.xlabel("Temperature (Celsius)")

plt.ylabel("Sales (Dollars)")

# In[33]:

# Save an image of the chart and print to screen

plt.savefig("IceCreamSales.png")

plt.show()

**05-Matplotlib/01/Activities/13-Stu\_AvgRain/Solved/avg\_state\_rain.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# In[2]:

# Load in csv

rain\_df = pd.read\_csv("../Resources/avg\_rain\_state.csv")

rain\_df.head()

# In[3]:

# Set x axis and tick locations

x\_axis = np.arange(len(rain\_df))

tick\_locations = [value+0.4 for value in x\_axis]

# In[4]:

# Create a list indicating where to write x labels and set figure size to adjust for space

plt.figure(figsize=(20,3))

plt.bar(x\_axis, rain\_df["Inches"], color='r', alpha=0.5, align="edge")

plt.xticks(tick\_locations, rain\_df["State"], rotation="vertical")

# In[5]:

# Set x and y limits

plt.xlim(-0.25, len(x\_axis))

plt.ylim(0, max(rain\_df["Inches"])+10)

# In[6]:

# Set a Title and labels

plt.title("Average Rain per State")

plt.xlabel("State")

plt.ylabel("Average Amount of Rainfall in Inches")

# In[7]:

# Save our graph and show the grap

plt.tight\_layout()

plt.savefig("avg\_state\_rain.png")

plt.show()

**05-Matplotlib/2/Activities/01-Stu\_PlotsReview/Solved/PlotDrills.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import numpy as np

import matplotlib.pyplot as plt

# In[2]:

# DATASET 1

gyms = ["Crunch", "Planet Fitness", "NY Sports Club", "Rickie's Gym"]

members = [49, 92, 84, 53]

# In[3]:

x\_axis = np.arange(0, len(gyms))

tick\_locations = []

for x in x\_axis:

tick\_locations.append(x + 0.4)

plt.title("NYC Gym Popularity")

plt.xlabel("Gym Name")

plt.ylabel("Number of Members")

plt.xlim(-0.25, len(gyms))

plt.ylim(0, max(members) + 5)

plt.bar(x\_axis, members, facecolor="red", alpha=0.75, align="edge")

plt.xticks(tick\_locations, gyms)

plt.show()

# In[4]:

# DATASET 2

x\_lim = 2 \* np.pi

x\_axis = np.arange(0, x\_lim, 0.1)

sin = np.sin(x\_axis)

# In[5]:

plt.title("Sin from 0 to 2$\pi$")

plt.xlabel("Real Numbers from 0 to 2$\pi$")

plt.ylabel("sin(x)")

plt.hlines(0, 0, x\_lim, alpha=0.2)

plt.xlim(0, x\_lim)

plt.ylim(-1.25, 1.25)

plt.plot(x\_axis, sin, marker="o", color="red", linewidth=1)

plt.show()

# In[6]:

# DATASET 3

gyms = ["Crunch", "Planet Fitness", "NY Sports Club", "Rickie's Gym"]

members = [49, 92, 84, 53]

colors = ["yellowgreen", "red", "lightcoral", "lightskyblue"]

explode = (0, 0.05, 0, 0)

# In[7]:

plt.title("NYC Gym Popularity")

plt.pie(members, explode=explode, labels=gyms, colors=colors,

autopct="%1.1f%%", shadow=True, startangle=90)

plt.axis("equal")

plt.show()

# In[8]:

# DATASET 4

x\_axis = np.arange(0, 10, 0.1)

times = []

for x in x\_axis:

times.append(x \* x + np.random.randint(0, np.ceil(max(x\_axis))))

# In[9]:

plt.title("Running Time of FakeSort for Sample Input Sizes")

plt.xlabel("Length of Input Array")

plt.ylabel("Time to Sort (s)")

plt.plot(x\_axis, times, marker="o", color="red")

plt.show()

**05-Matplotlib/2/Activities/02-Ins\_PandasPlot/avg\_state\_rain.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# ### Using MatplotLib to Chart a DataFrame

# In[2]:

# Load in csv

rain\_df = pd.read\_csv("Resources/avg\_rain\_state.csv")

rain\_df.head()

# In[3]:

# Set x axis and tick locations

x\_axis = np.arange(len(rain\_df))

tick\_locations = [value+0.4 for value in x\_axis]

# In[13]:

# Create a list indicating where to write x labels and set figure size to adjust for space

plt.figure(figsize=(20,3))

plt.bar(x\_axis, rain\_df["Inches"], color='r', alpha=0.5, align="edge")

plt.xticks(tick\_locations, rain\_df["State"], rotation="vertical")

# In[12]:

# Set x and y limits

plt.xlim(-0.25, len(x\_axis))

plt.ylim(0, max(rain\_df["Inches"])+10)

# In[10]:

# Set a Title and labels

plt.title("Average Rain per State")

plt.xlabel("State")

plt.ylabel("Average Amount of Rainfall in Inches")

# In[11]:

# Save our graph and show the grap

plt.tight\_layout()

plt.savefig("avg\_state\_rain.png")

plt.show()

# ### Using Pandas to Chart a DataFrame

# In[16]:

# Filter the DataFrame down only to those columns to chart

state\_and\_inches = rain\_df[["State","Inches"]]

# Set the index to be "State" so they will be used as labels

state\_and\_inches = state\_and\_inches.set\_index("State")

state\_and\_inches.head()

# In[15]:

# Use DataFrame.plot() in order to create a bar chart of the data

state\_and\_inches.plot(kind="bar", figsize=(20,3))

# Set a title for the chart

plt.title("Average Rain Per State")

plt.show()

# In[17]:

# Pandas can also plot multiple columns if the DataFrame includes them

multi\_plot = rain\_df.plot(kind="bar", figsize=(20,5))

# PandasPlot.set\_xticklabels() can be used to set the tick labels as well

multi\_plot.set\_xticklabels(rain\_df["State"], rotation=45)

plt.show()

**05-Matplotlib/2/Activities/03-Stu\_BattlingKings/Solved/BattlingKings.ipynb**

# coding: utf-8

# In[2]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# In[3]:

# Read CSV

got\_data = pd.read\_csv("Resources/got.csv")

got\_data

# In[4]:

# Get attacker and defender data

attacker\_data = got\_data["attacker\_king"].value\_counts()

defender\_data = got\_data["defender\_king"].value\_counts()

# In[13]:

# Get total battle data

battle\_data = attacker\_data.add(defender\_data, fill\_value=0)

# In[18]:

# Configure plot and ticks

battle\_data.plot(kind="bar", facecolor="red")

# In[19]:

# Set textual properties

plt.title("The Bloodthirst of Kings")

plt.ylabel("Number of Battles Participated In")

plt.xlabel("King")

# In[20]:

# Show plot

plt.show()

**05-Matplotlib/2/Activities/04-Ins\_GroupPlots/PlottingGroups.ipynb**

# coding: utf-8

# In[2]:

# Import Dependencies

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# In[3]:

# Import our data into pandas from CSV

used\_string = 'Data/UsedCars.csv'

used\_car\_df = pd.read\_csv(used\_string)

used\_car\_df

# In[4]:

# Create a group based on the values in the 'maker' column

maker\_group = used\_car\_df.groupby('maker')

# Count how many times each maker appears in our group

count\_makers = maker\_group['manufacture\_year'].count()

count\_makers

# In[38]:

# Create a bar chart based off of the group series from before

count\_chart = count\_makers.plot(kind='pie', figsize=(9,9))

# Set the xlabel and ylabel using class methods

count\_chart.set\_xlabel("Car Manufacturer")

count\_chart.set\_ylabel("Number of Cars")

plt.show()

**05-Matplotlib/2/Activities/05-Stu\_BikeTrippin/Solved/BikeTrippin.ipynb**

# coding: utf-8

# In[1]:

# Import Dependencies

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# In[2]:

# Import our data into pandas from CSV

string\_thing = '../Data/trip.csv'

bike\_trips\_df = pd.read\_csv(string\_thing)

bike\_trips\_df

# In[3]:

# Split up our data into groups based upon 'gender'

gender\_groups = bike\_trips\_df.groupby('gender')

# Find out how many of each gender took bike trips

gender\_trips = gender\_groups['tripduration'].count()

# Drop the 'stoptime' row that is contained within our group

gender\_trips = gender\_trips.drop(gender\_trips.index[3])

# Chart our data, give it a title, and label the axes

gender\_chart = gender\_trips.plot(kind="bar", title="Bike Trips by Gender")

gender\_chart.set\_xlabel("Gender")

gender\_chart.set\_ylabel("Number of Trips Taken")

plt.show()

# In[1]:

# Split up our data into groups based upon 'bikeid' and 'gender'

bike\_groups = bike\_trips\_df.groupby(['bikeid','gender'])

# Create a new variable that holds the sum of our groups

sum\_it\_up = bike\_groups.sum()

sum\_it\_up.head(12)

# In[6]:

# Make a variable called bike\_id and store a 'bikeid' in it

bike\_id = "SEA00001"

# Collect the trips of the 'bikeid' above

just\_one\_bike = sum\_it\_up.loc[bike\_id]

# Place the gender keys for that single bike into a list

gender\_list = just\_one\_bike.keys()

# Create a pie chart based upon the trip duration of that single bike

bike\_pie = just\_one\_bike.plot(kind="pie", y=gender\_list, title=("Trips of " + bike\_id))

bike\_pie.set\_ylabel("Trip Duration")

plt.show()

**05-Matplotlib/2/Activities/06-Stu\_MilesPerGallon/Solved/MilesPerGallon.ipynb**

# coding: utf-8

# In[41]:

# Dependencies and Setup

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# In[42]:

car\_data = pd.read\_csv('../Data/MPG.csv')

car\_data.head()

# In[43]:

# Remove the rows with missing values in horsepower

car\_data = car\_data.loc[car\_data['horsepower'] != "?"]

car\_data.head()

# In[44]:

# Set the 'car name' as our index

car\_data = car\_data.set\_index('car name')

# Remove the 'origin' column

del car\_data['origin']

car\_data.head()

# In[48]:

# Convert the "horsepower" column to numeric so the data can be used

car\_data['horsepower'] = pd.to\_numeric(car\_data['horsepower'])

# In[53]:

# Create a scatter plot which compares MPG to horsepower iwth the size being model year

car\_data.plot(kind="scatter", x="horsepower", y="mpg", grid=True, figsize=(20,10),

title="Horsepower Vs. MPG")

plt.show()

**05-Matplotlib/2/Activities/07-Ins\_PandasMultiLine/UnemployChart.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# In[2]:

# Read CSV

unemployed\_data\_one = pd.read\_csv("Data/Unemployment\_2010-2011.csv")

unemployed\_data\_two = pd.read\_csv("Data/Unemployment\_2012-2014.csv")

# Merge our two data frames together

combined\_unemployed\_data = pd.merge(unemployed\_data\_one, unemployed\_data\_two, on="Country Name")

combined\_unemployed\_data.head()

# In[8]:

# Delete the duplicate 'Country Code' column and rename the first one back to 'Country Code'

del combined\_unemployed\_data['Country Code\_y']

combined\_unemployed\_data = combined\_unemployed\_data.rename(columns={"Country Code\_x":"Country Code"})

combined\_unemployed\_data.head()

# In[6]:

# Set the 'Country Code' to be our index for easy referencing of rows

combined\_unemployed\_data = combined\_unemployed\_data.set\_index("Country Code")

# In[10]:

# Collect the mean unemployment rates for the world

average\_unemployment = combined\_unemployed\_data.mean()

# Collect the years where data was collected

years = average\_unemployment.keys()

# In[11]:

# Plot the world average as a line chart

world\_avg, = plt.plot(years, average\_unemployment, color="red", label="World Average" )

# Plot the unemployment values for a single country

country\_one, = plt.plot(years, combined\_unemployed\_data.loc['USA',["2010","2011","2012","2013","2014"]],

color="green",label=combined\_unemployed\_data.loc['USA',"Country Name"])

# Create a legend for our chart

plt.legend(handles=[world\_avg, country\_one], loc="best")

# Show the chart

plt.show()

**05-Matplotlib/2/Activities/08-Stu\_WinnerWrestling-Part1/Solved/WinningWrestlers.ipynb**

# coding: utf-8

# # Winnining Wrestlers Entertainment

#

# In this activity you will be taking four seperate csvs that were scraped down from a wrestling database, merging them together, and then creating charts to visualize a wrestler's wins and losses over the course of four years.

#

# ### Part 1 - Macho Merging

#

# \* You will likely need to perform three different merges over the course of this activity, changing the names of your columns as you go along.

# In[2]:

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# In[3]:

# Take in all of our wrestling data and read it into pandas

wrestling\_2013 = "../Data/WWE-Data-2013.csv"

wrestling\_2014 = "../Data/WWE-Data-2014.csv"

wrestling\_2015 = "../Data/WWE-Data-2015.csv"

wrestling\_2016 = "../Data/WWE-Data-2016.csv"

wrestlers\_2013\_df = pd.read\_csv(wrestling\_2013)

wrestlers\_2014\_df = pd.read\_csv(wrestling\_2014)

wrestlers\_2015\_df = pd.read\_csv(wrestling\_2015)

wrestlers\_2016\_df = pd.read\_csv(wrestling\_2016)

# In[10]:

# Merge the first two datasets on "Wrestler" so that no data is lost (should be 182 rows)

combined\_wrestlers\_df = pd.merge(wrestlers\_2013\_df, wrestlers\_2014\_df,

how='outer', on='Wrestler')

combined\_wrestlers\_df.head()

# In[5]:

# Rename our \_x columns to "2013 Wins", "2013 Losses", and "2013 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_x":"2013 Wins",

"Losses\_x":"2013 Losses",

"Draws\_x":"2013 Draws"})

# Rename our \_y columns to "2014 Wins", "2014 Losses", and "2014 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_y":"2014 Wins",

"Losses\_y":"2014 Losses",

"Draws\_y":"2014 Draws"})

combined\_wrestlers\_df.head()

# In[6]:

# Merge our newly combined dataframe with the 2015 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2015\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# In[7]:

# Rename "wins", "losses", and "draws" to "2015 Wins", "2015 Losses", and "2015 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2015 Wins","Losses":"2015 Losses","Draws":"2015 Draws"})

combined\_wrestlers\_df.head()

# In[8]:

# Merge our newly combined dataframe with the 2016 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2016\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# In[9]:

# Rename "wins", "losses", and "draws" to "2016 Wins", "2016 Losses", and "2016 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2016 Wins","Losses":"2016 Losses","Draws":"2016 Draws"})

combined\_wrestlers\_df.head(10)

**05-Matplotlib/2/Activities/09-Stu\_WinnerWrestling-Part2/Solved/WinningWrestlers.ipynb**

# coding: utf-8

# # Winnining Wrestlers Entertainment

#

# In this activity you will be taking four seperate csvs that were scraped down from a wrestling database, merging them together, and then creating charts to visualize a wrestler's wins and losses over the course of four years.

#

# ### Part 1 - Macho Merging

#

# \* You will likely need to perform three different merges over the course of this activity, changing the names of your columns as you go along.

# In[1]:

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# In[2]:

# Take in all of our wrestling data and read it into pandas

wrestling\_2013 = "../Data/WWE-Data-2013.csv"

wrestling\_2014 = "../Data/WWE-Data-2014.csv"

wrestling\_2015 = "../Data/WWE-Data-2015.csv"

wrestling\_2016 = "../Data/WWE-Data-2016.csv"

wrestlers\_2013\_df = pd.read\_csv(wrestling\_2013)

wrestlers\_2014\_df = pd.read\_csv(wrestling\_2014)

wrestlers\_2015\_df = pd.read\_csv(wrestling\_2015)

wrestlers\_2016\_df = pd.read\_csv(wrestling\_2016)

# In[3]:

# Merge the first two datasets on "Wrestler" so that no data is lost (should be 182 rows)

combined\_wrestlers\_df = pd.merge(wrestlers\_2013\_df, wrestlers\_2014\_df, how='outer', on='Wrestler')

combined\_wrestlers\_df

# In[4]:

# Rename our \_x columns to "2013 Wins", "2013 Losses", and "2013 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_x":"2013 Wins", "Losses\_x":"2013 Losses", "Draws\_x":"2013 Draws"})

# Rename our \_y columns to "2014 Wins", "2014 Losses", and "2014 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_y":"2014 Wins","Losses\_y":"2014 Losses","Draws\_y":"2014 Draws"})

combined\_wrestlers\_df.head()

# In[5]:

# Merge our newly combined dataframe with the 2015 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2015\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# In[6]:

# Rename "wins", "losses", and "draws" to "2015 Wins", "2015 Losses", and "2015 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2015 Wins","Losses":"2015 Losses","Draws":"2015 Draws"})

combined\_wrestlers\_df.head()

# In[7]:

# Merge our newly combined dataframe with the 2016 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2016\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# In[8]:

# Rename "wins", "losses", and "draws" to "2016 Wins", "2016 Losses", and "2016 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2016 Wins","Losses":"2016 Losses","Draws":"2016 Draws"})

combined\_wrestlers\_df.head()

# ### Part 2 - Time to Calculate!

#

# \* When your tables have been merged together into one data frame, calculate the total number of wins, losses, and draws a wrestler has had over the course of their career. Also create a new column that will hold the total matches a wrestler has been in over the course of their career.

# \* You will need to convert all NaN values to a number so that you can perform these calculations

#

# \* We are only interested in those wrestlers who have been with the WWE from 2013 to 2016. You will need to come up with some way of filtering out rows that do not meet these conditions.

# \* Also set the 'Wrestler' column as your key for easier referencing later on.

# In[9]:

# Replace all NaN values with 0

combined\_wrestlers\_df = combined\_wrestlers\_df.fillna(0)

# Create a new column called "Total Wins" and add up each wrestler's wins per year to fill in the values

combined\_wrestlers\_df["Total Wins"] = combined\_wrestlers\_df["2013 Wins"] + combined\_wrestlers\_df["2014 Wins"] + combined\_wrestlers\_df["2015 Wins"] + combined\_wrestlers\_df["2016 Wins"]

# Create a new column called "Total Losses" and add up each wrestler's losses per year to fill in the values

combined\_wrestlers\_df["Total Losses"] = combined\_wrestlers\_df["2013 Losses"] + combined\_wrestlers\_df["2014 Losses"] + combined\_wrestlers\_df["2015 Losses"] + combined\_wrestlers\_df["2016 Losses"]

# Create a new column called "Total Draws" and add up each wrestler's draws per year to fill in the values

combined\_wrestlers\_df["Total Draws"] = combined\_wrestlers\_df["2013 Draws"] + combined\_wrestlers\_df["2014 Draws"] + combined\_wrestlers\_df["2015 Draws"] + combined\_wrestlers\_df["2016 Draws"]

# Create a new column called "Total Matches" and add up the total wins, losses, and draws for each wrestler to fill in the values

combined\_wrestlers\_df["Total Matches"] = combined\_wrestlers\_df["Total Wins"] + combined\_wrestlers\_df["Total Losses"] + combined\_wrestlers\_df["Total Draws"]

combined\_wrestlers\_df

# In[10]:

# Create a new dataframe for those wrestlers who have wrestled at least 100 matches,

# have at least one win in 2013,

# and have at least one win in 2016

wrestled\_over\_hundred = combined\_wrestlers\_df.loc[(combined\_wrestlers\_df["Total Matches"] >= 100) &

(combined\_wrestlers\_df["2013 Wins"] > 0) &

(combined\_wrestlers\_df["2016 Wins"] > 0)]

# Set the index of this new dataframe to be the wrestlers names

wrestled\_over\_hundred = wrestled\_over\_hundred.set\_index("Wrestler")

wrestled\_over\_hundred.head()

**05-Matplotlib/2/Activities/10-Stu\_WinnerWrestling-Part3/Solved/WinningWrestlers.ipynb**

# coding: utf-8

# # Winnining Wrestlers Entertainment

#

# In this activity you will be taking four seperate csvs that were scraped down from a wrestling database, merging them together, and then creating charts to visualize a wrestler's wins and losses over the course of four years.

#

# ### Part 1 - Macho Merging

#

# \* You will likely need to perform three different merges over the course of this activity, changing the names of your columns as you go along.

# In[2]:

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

# In[3]:

# Take in all of our wrestling data and read it into pandas

wrestling\_2013 = "../Data/WWE-Data-2013.csv"

wrestling\_2014 = "../Data/WWE-Data-2014.csv"

wrestling\_2015 = "../Data/WWE-Data-2015.csv"

wrestling\_2016 = "../Data/WWE-Data-2016.csv"

wrestlers\_2013\_df = pd.read\_csv(wrestling\_2013)

wrestlers\_2014\_df = pd.read\_csv(wrestling\_2014)

wrestlers\_2015\_df = pd.read\_csv(wrestling\_2015)

wrestlers\_2016\_df = pd.read\_csv(wrestling\_2016)

# In[4]:

# Merge the first two datasets on "Wrestler" so that no data is lost (should be 182 rows)

combined\_wrestlers\_df = pd.merge(wrestlers\_2013\_df, wrestlers\_2014\_df, how='outer', on='Wrestler')

combined\_wrestlers\_df

# In[5]:

# Rename our \_x columns to "2013 Wins", "2013 Losses", and "2013 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_x":"2013 Wins", "Losses\_x":"2013 Losses", "Draws\_x":"2013 Draws"})

# Rename our \_y columns to "2014 Wins", "2014 Losses", and "2014 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins\_y":"2014 Wins","Losses\_y":"2014 Losses","Draws\_y":"2014 Draws"})

combined\_wrestlers\_df.head()

# In[6]:

# Merge our newly combined dataframe with the 2015 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2015\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# In[7]:

# Rename "wins", "losses", and "draws" to "2015 Wins", "2015 Losses", and "2015 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2015 Wins","Losses":"2015 Losses","Draws":"2015 Draws"})

combined\_wrestlers\_df.head()

# In[8]:

# Merge our newly combined dataframe with the 2016 dataframe

combined\_wrestlers\_df = pd.merge(combined\_wrestlers\_df, wrestlers\_2016\_df, how="outer", on="Wrestler")

combined\_wrestlers\_df

# In[9]:

# Rename "wins", "losses", and "draws" to "2016 Wins", "2016 Losses", and "2016 Draws"

combined\_wrestlers\_df = combined\_wrestlers\_df.rename(columns={"Wins":"2016 Wins","Losses":"2016 Losses","Draws":"2016 Draws"})

combined\_wrestlers\_df.head()

# In[10]:

# Replace all NaN values with 0

combined\_wrestlers\_df = combined\_wrestlers\_df.fillna(0)

# Create a new column called "Total Wins" and add up each wrestler's wins per year to fill in the values

combined\_wrestlers\_df["Total Wins"] = combined\_wrestlers\_df["2013 Wins"] + combined\_wrestlers\_df["2014 Wins"] + combined\_wrestlers\_df["2015 Wins"] + combined\_wrestlers\_df["2016 Wins"]

# Create a new column called "Total Losses" and add up each wrestler's losses per year to fill in the values

combined\_wrestlers\_df["Total Losses"] = combined\_wrestlers\_df["2013 Losses"] + combined\_wrestlers\_df["2014 Losses"] + combined\_wrestlers\_df["2015 Losses"] + combined\_wrestlers\_df["2016 Losses"]

# Create a new column called "Total Draws" and add up each wrestler's draws per year to fill in the values

combined\_wrestlers\_df["Total Draws"] = combined\_wrestlers\_df["2013 Draws"] + combined\_wrestlers\_df["2014 Draws"] + combined\_wrestlers\_df["2015 Draws"] + combined\_wrestlers\_df["2016 Draws"]

# Create a new column called "Total Matches" and add up the total wins, losses, and draws for each wrestler to fill in the values

combined\_wrestlers\_df["Total Matches"] = combined\_wrestlers\_df["Total Wins"] + combined\_wrestlers\_df["Total Losses"] + combined\_wrestlers\_df["Total Draws"]

combined\_wrestlers\_df

# In[11]:

# Create a new dataframe for those wrestlers who have wrestled at least 100 matches,

# have at least one win in 2013,

# and have at least one win in 2016

wrestled\_over\_hundred = combined\_wrestlers\_df.loc[(combined\_wrestlers\_df["Total Matches"] >= 100) &

(combined\_wrestlers\_df["2013 Wins"] > 0) &

(combined\_wrestlers\_df["2016 Wins"] > 0)]

# Set the index of this new dataframe to be the wrestlers names

wrestled\_over\_hundred = wrestled\_over\_hundred.set\_index("Wrestler")

wrestled\_over\_hundred.head()

# ### Part 3 - Charting Careers

#

# \* Create a variable and an input that asks the user what wrestler they would like to look for.

#

# \* Store an individual wrestler's wins over time in a variable

#

# \* Store that same wrestler's losses over time in a variable as well

#

# \* Create a line chart that will plot this wrestler's wins and losses from 2013 to 2016

# In[12]:

# Collect the user's input to search through our data frame

wrestler\_name = input("What wrestler's career would you like to look at?")

# In[13]:

# Create a series that looks for a wrestler by name and then traces their wins from 2013 to 2016

wins\_over\_time = wrestled\_over\_hundred.loc[wrestler\_name,["2013 Wins","2014 Wins", "2015 Wins", "2016 Wins"]]

# Create a series that looks for a wrestler by name and then traces their losses from 2013 to 2016

losses\_over\_time = wrestled\_over\_hundred.loc[wrestler\_name,["2013 Losses","2014 Losses",

"2015 Losses", "2016 Losses"]]

# In[15]:

# Create a list of the years that we will use as our x axis

years = [2013,2014,2015,2016]

# Plot our line that will be used to track a wrestler's wins over the years

plt.plot(years, wins\_over\_time, color="green", label="Wins")

# Plot our line that will be used to track a wrestler's losses over the years

plt.plot(years, losses\_over\_time, color="red", label="Losses")

# Place a legend on the chart in what matplotlib believes to be the "best" location

plt.legend(loc="best")

plt.title(wrestler\_name + "'s Recent Career")

plt.xlabel("Years")

plt.ylabel("Number of Wins/Losses")

# Print our chart to the screen

plt.show()

**05-Matplotlib/03/Activities/01-Ins\_Mean\_Median\_Mode/Solved/samples.py**

# Dependencies

from statistics import mean, median, mode, multi\_mode

# Prices of random electronics at Best Buy

prices = [4, 425, 984, 2932, 49]

print("Median Price: {}".format(median(prices)))

# Ages of students in bootcamp

bootcamp\_classroom\_ages = [27, 35, 42, 52, 36, 28]

print("Mean Bootcamp Age: {}".format(mean(bootcamp\_classroom\_ages)))

print("Median Bootcamp Age: {}".format(median(bootcamp\_classroom\_ages)))

# Ages of children and parents at child's party

birthday\_party\_ages = [6, 5, 6, 6, 35, 42, 34]

print("Mode of Birthday Party Ages: {}".format(mean(birthday\_party\_ages)))

# Test score from a 2nd grade geography test

geo\_grades = [87, 89, 91, 93, 95]

print("Mean of Geography Test Scores: {}".format(mean(geo\_grades)))

# Test scores from a graduate quantum mechanics midterm

quantum\_grades = [63, 63, 98, 13, 58, 13, 8]

print("Median of QM Grades: {}".format(median(quantum\_grades)))

print("Modes of QM Grades: {}".format(multi\_mode(quantum\_grades)))

print(mean(quantum\_grades))

**05-Matplotlib/03/Activities/01-Ins\_Mean\_Median\_Mode/Solved/statistics.py**

# Dependencies

from collections import Counter

def mean(arr):

"""

Calculates the mean of a list of numbers.

Usage: mean([3, 4, 5, 6, 7]) # 5

"""

return sum(arr) / len(arr)

def median(arr):

"""

Calculates the median of a list of numbers.

Usage: median([3, 4, 5, 6, 7]) # 5

"""

arr.sort()

return arr[len(arr) // 2]

def mode(arr):

"""

Calculates the mode of a list of numbers as a tuple containing the

most frequently occurring element in the first slot, and the number

of times it occurs in the second. If there are multiple modes,

returns only one.

Usage: mode([1, 1, 2, 3]) # (1, 2)

"""

return Counter(arr).most\_common(1)[0]

def multi\_mode(arr):

"""

Calculates the mode of a list of numbers as an array of tuples

containing the most frequently occurring elements. Each tuple

contains the item itself in the first slot, and the number of times

it occurs in the second. If there are multiple modes, returns all.

Usage: mode([1, 1, 2, 2, 3]) # [(1, 2), (2, 2)]

"""

# Count items

counter = Counter(arr)

# Get number of times most frequently occurring item appears

freq = counter.most\_common(1)[0][1]

# Return list of tuples containing most frequently occurring elements

return [(key, val) for key, val in counter.items() if val == freq]

**05-Matplotlib/03/Activities/02-Ins\_Variance\_and\_Z\_Score/Solved/samples.py**

# Dependencies

from spread import variance, standard\_deviation, zipped\_z\_scores

def summarize(title, arr):

print("Summarizing {}".format(title))

print("Variance: {}".format(variance(arr)))

print("Standard Deviation: {}".format(standard\_deviation(arr)))

print("Z-Scores: {}".format(zipped\_z\_scores(arr)))

print("======")

# Prices of random electronics at Best Buy

prices = [4, 425, 984, 2932, 49]

summarize("Prices", prices)

# Ages of students in bootcamp

bootcamp\_classroom\_ages = [27, 35, 42, 52, 36, 28]

summarize("Bootcamp Ages", bootcamp\_classroom\_ages)

# Ages of children and parents at child's party

birthday\_party\_ages = [6, 5, 6, 6, 35, 34, 42]

summarize("Birthday Party Ages", birthday\_party\_ages)

# Test score from a 2nd grade geography test

geo\_grades = [87, 89, 91, 93, 95]

summarize("Geograph Grades", geo\_grades)

# Test scores from a graduate quantum mechanics midterm

quantum\_grades = [63, 63, 98, 13, 58, 13, 8]

summarize("Quantum Mechanics Grades", quantum\_grades)

# Prices

summarize("Prices", [30, 31, 31, 32, 32, 40, 41, 41, 1000])

**05-Matplotlib/03/Activities/02-Ins\_Variance\_and\_Z\_Score/Solved/spread.py**

# Dependencies

from math import sqrt

from statistics import mean

def variance(arr):

\_mean = mean(arr)

normalized = [x - \_mean for x in arr]

squares = [x \*\* 2 for x in normalized]

return sum(squares) / len(arr)

def standard\_deviation(arr):

return sqrt(variance(arr))

def z\_score(arr, index):

return (arr[index] - mean(arr)) / standard\_deviation(arr)

def z\_scores(arr):

return [z\_score(arr, ind) for ind in range(0, len(arr))]

def zipped\_z\_scores(arr):

return list(zip(arr, z\_scores(arr)))

sample = [-2, -1, 0, 1, 2]

print(variance(sample))

print(standard\_deviation(sample))

print(zipped\_z\_scores(sample))

**05-Matplotlib/03/Activities/02-Ins\_Variance\_and\_Z\_Score/Solved/statistics.py**

# Dependencies

from collections import Counter

def mean(ar

"""

Calculates the mean of a list of numbers.

Usage: mean([3, 4, 5, 6, 7]) # 5

"""

return sum(arr) / len(arr)

def median(arr):

"""

Calculates the median of a list of numbers.

Usage: median([3, 4, 5, 6, 7]) # 5

"""

arr.sort()

return arr[len(arr) // 2]

def mode(arr):

"""

Calculates the mode of a list of numbers as a tuple containing the

most frequently occurring element in the first slot, and the number

of times it occurs in the second. If there are multiple modes,

returns only one.

Usage: mode([1, 1, 2, 3]) # (1, 2)

"""

return Counter(arr).most\_common(1)[0]

def multi\_mode(arr):

"""

Calculates the mode of a list of numbers as an array of tuples

containing the most frequently occurring elements. Each tuple

contains the item itself in the first slot, and the number of times

it occurs in the second. If there are multiple modes, returns all.

Usage: mode([1, 1, 2, 2, 3]) # [(1, 2), (2, 2)]

"""

# Count items

counter = Counter(arr)

# Get number of times most frequently occurring item appears

freq = counter.most\_common(1)[0][1]

# Return list of tuples containing most frequently occurring elements

return [(key, val) for key, val in counter.items() if val == freq]

**05-Matplotlib/03/Activities/03-Ins\_Quartiles\_and\_Outliers/samples.py**

# Dependencies

import numpy as np

numbers = [3, 3, 4, 5, 5, 6, 7, 7, 8, 8, 9]

median = 6

lower\_quartile = 4

upper\_quartile = 8

**05-Matplotlib/03/Activities/04-Stu\_Quartiles\_and\_Outliers/Solved/quartiles.py**

#!usr/bin/python

def lower\_quartile(arr):

index = len(arr) // 4

return (index, arr[index])

def upper\_quartile(arr):

index = 3 \* len(arr) // 4

return (index, arr[index])

def iqr(arr):

(\_, lower) = lower\_quartile(arr)

(\_, upper) = upper\_quartile(arr)

return upper - lower

**05-Matplotlib/03/Activities/04-Stu\_Quartiles\_and\_Outliers/Solved/samples.py**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

arr = [8, 8, 12, 24, 54, 54, 75, 78, 98, 102, 132]

x\_axis = np.arange(0, len(arr), 1)

# Calculate the indices for the lower and upper quartiles

lower\_quartile\_index = (len(arr) + 1) // 4

upper\_quartile\_index = 3 \* len(arr) // 4

# Retrieve the lower and upper quartiles

lower\_quartile = arr[lower\_quartile\_index]

upper\_quartile = arr[upper\_quartile\_index]

# Calculate the interquartile range

iqr = upper\_quartile - lower\_quartile

# Create axes for the included and excluded data

included = arr[lower\_quartile\_index:upper\_quartile\_index]

included\_axis = np.arange(lower\_quartile\_index, upper\_quartile\_index, 1)

excluded\_low = arr[0:lower\_quartile\_index]

low\_axis = np.arange(0, len(excluded\_low), 1)

excluded\_high = arr[upper\_quartile\_index:len(arr)]

high\_axis = np.arange(len(included) + len(excluded\_high), len(arr), 1)

# Create a plot displaying included and excluded data

fig, ax = plt.subplots()

fig.suptitle("Interquartile Range Example", fontsize=16, fontweight="bold")

ax.plot(included\_axis, included, marker='o', color='b', label="IQR")

ax.scatter(

low\_axis,

excluded\_low,

marker='o',

color='r',

label="Excluded (Low)")

ax.scatter(

high\_axis,

excluded\_high,

marker='o',

color='r',

label="Excluded (High)")

plt.legend(loc="upper left", fancybox=True)

plt.show()

# Report descriptions of the data

print("The lowr quartile of the data is {}.".format(lower\_quartile))

print("The upper quartile of the data is {}.".format(upper\_quartile))

print("The interquartile range of the data is {}.".format(iqr))

**05-Matplotlib/03/Activities/05-Ins\_Standard\_Error/samples.py**

# Dependencies

from random import random

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem

# "Will you vote for a republican in this election?"

sample\_size = 100

samples = [[True if random() < 0.5 else False for x in range(0, sample\_size)]

for y in range(0, 10)]

x\_axis = np.arange(0, len(samples), 1)

means = [np.mean(s) for s in samples]

standard\_errors = [sem(s) for s in samples]

# Setting up the plot

fig, ax = plt.subplots()

ax.errorbar(x\_axis, means, standard\_errors, fmt="o")

ax.set\_xlim(-1, len(samples) + 1)

ax.set\_xlabel("Sample Number")

ax.set\_ylabel("Proportion of People Voting Republican")

plt.show()

**05-Matplotlib/03/Activities/06-Stu\_Standard\_Error/Solved/samples.py**

# Dependencies

from matplotlib import pyplot as plt

import numpy as np

import pandas as pd

# Read data

housing\_data = pd.read\_csv("../Resources/housing\_data.csv")

housing\_data = housing\_data.sample(frac=1).reset\_index(drop=True)

# Create a bunch of samples, each with div items

div = 20

lim = len(housing\_data) // div

samples = [housing\_data.iloc[(i \* div):(i \* div + div), 13]

for i in range(0, lim)]

# Calculate means

means = [s.mean() for s in samples]

# Calculate standard error on means

sem = [s.sem() for s in samples]

# Plot sample means with error bars

fig, ax = plt.subplots()

ax.errorbar(np.arange(0, len(means)), means, yerr=sem, fmt="o", color="b",

alpha=0.5, label="Mean of House Prices")

ax.set\_xlim(-0.5, len(means))

ax.set\_xlabel("Sample Number")

ax.set\_ylabel("Mean of Median House Prices")

plt.legend(loc="best", fontsize="small", fancybox=True)

plt.show()

**05-Matplotlib/03/Activities/07-Ins\_Students\_t\_test/silly\_samples.py**

# Dependencies

from random import randint

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem

high\_prices = [randint(1, 5) \* 1000 for x in range(1, 10)]

high\_means = np.mean(high\_prices)

high\_sem = sem(high\_prices)

medium\_prices = [randint(1, 5) \* 500 for x in range(1, 10)]

medium\_means = np.mean(medium\_prices)

medium\_sem = sem(medium\_prices)

low\_prices = [randint(1, 5) \* 200 for x in range(1, 10)]

low\_means = np.mean(low\_prices)

low\_sem = sem(low\_prices)

means = [high\_means, medium\_means, low\_means]

sems = [high\_sem, medium\_sem, low\_sem]

labels = ["High Prices", "Medium Prices", "Low Prices"]

fig, ax = plt.subplots()

ax.errorbar(np.arange(0, len(means)), means, yerr=sems, fmt="o")

ax.set\_xlim(-0.5, 2.5)

ax.set\_xticklabels(labels)

ax.set\_xticks([0, 1, 2])

ax.set\_ylabel("Mean House Price")

plt.show()

**05-Matplotlib/03/Activities/07-Ins\_Students\_t\_test/silly\_samples\_ttest.py**

# Dependencies

from random import randint

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem, ttest\_ind

# Generate

high\_prices = [randint(1, 5) \* 1000 for x in range(1, 10)]

high\_means = np.mean(high\_prices)

high\_sem = sem(high\_prices)

medium\_prices = [randint(1, 5) \* 500 for x in range(1, 10)]

medium\_means = np.mean(medium\_prices)

medium\_sem = sem(medium\_prices)

low\_prices = [randint(1, 5) \* 200 for x in range(1, 10)]

low\_means = np.mean(low\_prices)

low\_sem = sem(low\_prices)

means = [high\_means, medium\_means, low\_means]

sems = [high\_sem, medium\_sem, low\_sem]

labels = ["High Prices", "Medium Prices", "Low Prices"]

# Plot

fig, ax = plt.subplots()

ax.errorbar(np.arange(0, len(means)), means, yerr=sems, fmt="o")

ax.set\_xlim(-0.5, 2.5)

ax.set\_xticklabels(labels)

ax.set\_xticks([0, 1, 2])

ax.set\_ylabel("Mean House Price")

plt.show()

# t-test

(t\_stat, p) = ttest\_ind(high\_prices, low\_prices, equal\_var=False)

#p value is what we care about. tells us how these two things correlate

#.05 is the classic example of a p value

#you need to determine the value of the p before you begin

if p < 0.05:

print("The differences between the high and low prices are significant.")

else:

print("The differences between high and low prices are due to chance.")

**05-Matplotlib/03/Activities/07-Ins\_Students\_t\_test/ttest.py**

# Dependencies

import numpy as np

import pandas as pd

from scipy import stats

# Read data

housing\_data = pd.read\_csv("./Resources/housing\_data.csv")

housing\_data = housing\_data.sample(frac=1).reset\_index(drop=True)

# Create two samples

s1 = housing\_data.iloc[0:19, 13]

s2 = housing\_data.iloc[20:40, 13]

(t\_stat, p) = stats.ttest\_ind(s1, s2, equal\_var=False)

print("t-statistics is {}.".format(t\_stat))

print("p-value is {}.".format(p))

**05-Matplotlib/03/Activities/08-Stu\_Students\_t\_test/Solved/ttest.py**

# Dependencies

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from scipy import stats

general\_heights = pd.read\_csv("../Resources/general\_heights.csv")

wba\_data = pd.read\_csv("../Resources/wba\_data.csv")

wba\_heights = wba\_data.iloc[:, -1]

(t\_stat, p) = stats.ttest\_ind(general\_heights, wba\_heights, equal\_var=False)

# Report

print("The mean height of WBA players is {}.".format(wba\_heights.mean()))

print("The mean height of women sampled is {}.".format(

general\_heights.values.mean()))

print("p is {}.".format(p[0]))

if p < 0.05:

print("The difference in sample means is significant.")

else:

print("The difference in sample means is not significant.")

# Plot sample means with error bars

tick\_labels = ["General Public", "WBA Players"]

means = [general\_heights.mean().values[0], wba\_heights.mean()]

x\_axis = np.arange(0, len(means))

sem = [general\_heights.sem().values[0], wba\_heights.sem()]

# Plot

fig, ax = plt.subplots()

fig.suptitle("Mean Height of Women in General Population and WBA Players",

fontsize=12, fontweight="bold")

ax.errorbar(x\_axis, means, yerr=sem, fmt="o")

ax.set\_xlim(-0.5, 1.5)

ax.set\_ylim(64, 73)

ax.set\_xticklabels(tick\_labels)

ax.set\_xticks([0, 1])

ax.set\_ylabel("Height (Inches)")

plt.show()

**05-Matplotlib/03/Activities/09-Ins\_Fits\_and\_Regression/regression.py**

# Dependencies

from matplotlib import pyplot as plt

from scipy.stats import linregress

import numpy as np

x\_axis = np.arange(0, 10, 1)

fake = [1, 2.5, 2.75, 4.25, 5.5, 6, 7.25, 8, 8.75, 9.8]

(slope, intercept, \_, \_, \_) = linregress(x\_axis, fake)

fit = slope \* x\_axis + intercept

fig, ax = plt.subplots()

fig.suptitle("Fake Banana Data!", fontsize=16, fontweight="bold")

ax.set\_xlim(0, 10)

ax.set\_ylim(0, 10)

ax.set\_xlabel("Fake Banana Ages (in days)")

ax.set\_ylabel("Fake Banana Weights (in Hundres of Kilograms)")

ax.plot(x\_axis, fake, linewidth=0, marker='o')

ax.plot(x\_axis, fit, 'b--')

plt.show()

**05-Matplotlib/03/Activities/10-Stu\_Fits\_and\_Regression/Solved/crime.py**

# Dependencies

from matplotlib import pyplot as plt

from scipy import stats

import numpy as np

import pandas as pd

crime\_data = pd.read\_csv("../Resources/crime\_data.csv")

year = crime\_data.iloc[:, 0]

violent\_crime\_rate = crime\_data.iloc[:, 3]

vc\_slope, vc\_int, vc\_r, vc\_p, vc\_std\_err = stats.linregress(

year, violent\_crime\_rate)

vc\_fit = vc\_slope \* year + vc\_int

murder\_rate = crime\_data.iloc[:, 5]

m\_slope, m\_int, m\_r, m\_p, m\_std\_err = stats.linregress(year, murder\_rate)

m\_fit = m\_slope \* year + m\_int

aggravated\_assault\_rate = crime\_data.iloc[:, 9]

aa\_slope, aa\_int, aa\_r, aa\_p, aa\_std\_err = stats.linregress(

year, aggravated\_assault\_rate)

aa\_fit = aa\_slope \* year + aa\_int

fig, (ax1, ax2, ax3) = plt.subplots(3, sharex=True)

fig.suptitle("Crime Rates Over Time", fontsize=16, fontweight="bold")

ax1.set\_xlim(min(year), max(year))

ax1.plot(year, violent\_crime\_rate, linewidth=1, marker="o")

ax1.plot(year, vc\_fit, "b--", linewidth=1)

ax1.set\_ylabel("Violent Crime Rate")

ax2.plot(year, murder\_rate, linewidth=1, marker="o", color="r")

ax2.plot(year, m\_fit, "r--", linewidth=1)

ax2.set\_ylabel("Murder Rate")

ax3.plot(year, aggravated\_assault\_rate, linewidth=1, marker="o", color="g")

ax3.plot(year, aa\_fit, "g--", linewidth=1)

ax3.set\_ylabel("Aggravated Assault Rate")

ax3.set\_xlabel("Year")

year = 2019

print("The violent crime rate in 2019 will be " +

str(vc\_slope \* year + vc\_int) + ".")

print("The murder rate in 2019 will be " + str(m\_slope \* year + m\_int) + ".")

print("The aggravated assault rate in 2019 will be " +

str(aa\_slope \* year + aa\_int) + ".")

plt.savefig("../../../Images/18-final-plot.png")

plt.show()

**Python APIs**

**06-PythonAPIs/01/Activities/00-Ins\_Standard\_Error/sem.py**

# Dependencies

from random import random

import matplotlib.pyplot as plt

import numpy as np

# <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.sem.html>

from scipy.stats import sem

# "Will you vote for a republican in this election?"

sample\_size = 100

# Like a nested for-loop, there is a list comprehension nested inside the outer list comprehension

# The inner list comprehension generates a list of True and False values.

# The outer list comprehension generates ten such lists

samples = [[True if random() < 0.5 else False for x in range(0, sample\_size)]

for y in range(0, 10)]

x\_axis = np.arange(0, len(samples), 1)

# Evaluate the mean of each list in samples

means = [np.mean(s) for s in samples]

# Evaluate the standard error measurement for each list in samples

standard\_errors = [sem(s) for s in samples]

# Generate an errorbar plot

# The 'yerr' argument refers to the error bar for each sample, oriented to the y-axis

plt.errorbar(x\_axis, means, yerr=standard\_errors, color='r', marker='o', markersize=5, linestyle='dashed', linewidth=0.5)

# Display the plot

plt.show()

**06-PythonAPIs/01/Activities/01-Ins\_RequestsIntro/requests\_demo.py**

# Dependencies

import requests as req

import json

# URL for GET requests to retrieve vehicle data

url = "<https://api.spacexdata.com/v1/launchpads/>"

# Print the response object to the console

# print(req.get(url))

# Retrieving data and converting it into JSON

# print(req.get(url).json())

# Pretty Print the output of the JSON

response = req.get(url).json()

print(json.dumps(response, indent=4, sort\_keys=True))

**06-PythonAPIs/01/Activities/02-Stu\_SpaceX/Solved/spaceX.py**

# Dependencies

import requests as req

import json

# URL for GET requests to retrieve vehicle data

url = "<https://api.spacexdata.com/v1/launchpads/>"

# Pretty print JSON for all launchpads

# response = req.get(url).json()

# print(json.dumps(response, indent=4, sort\_keys=True))

# Pretty print JSON for a specific launchpad

response = req.get(url + "vafb\_slc\_4w").json()

print(json.dumps(response, indent=4, sort\_keys=True))

**06-PythonAPIs/01/Activities/03-Ins\_ManipulatingResponses/ManipulatingJson.py**

# Dependencies

import requests as req

# Performing a GET Request and saving the API's response within a variable

url = "<https://api.spacexdata.com/v1/vehicles/falcon9>"

response = req.get(url)

response\_json = response.json()

# It is possible to grab a specific value from within the JSON object

print(response\_json["cost\_per\_launch"])

# It is also possible to perform some analyses on values stored within the JSON object

print("There are " + str((len(response\_json["payload\_weights"]))) + " payloads.")

# Finally, it is possible to reference the values stored within sub-dictionaries and sub-lists

print("The first payload weighed " + str(response\_json["payload\_weights"][0]["kg"]) + " Kilograms")

**06-PythonAPIs/01/Activities/04-Stu\_FarFarAway/Solved/FarFarAway.py**

# Dependencies

import requests as req

import json

# URL for GET requests to retrieve Star Wars character data

url = "<https://swapi.co/api/people/>"

# Storing the JSON response within a variable

response = req.get(url + '4').json()

# Collecting the name of the character collected

character\_name = response["name"]

# Counting how many films the character was in

film\_number = len(response["films"])

# Figure out what their first starship was

first\_ship\_url = response["starships"][0]

ship\_response = req.get(first\_ship\_url).json()

first\_ship = ship\_response["name"]

# Print character name and how many films they were in

print(character\_name + " was in " + str(film\_number) + " films")

# Print what their first ship was

print("Their first ship: " + first\_ship)

**06-PythonAPIs/01/Activities/05-Par\_NumberFacts/Solved/NumberFacts.py**

# Dependencies

import requests as req

import json

# Base URL for GET requests to retrieve number/date facts

url = "<http://numbersapi.com/>"

# Ask the user what kind of data they would like to search for

kind\_of\_search = input("What type of data would you like to search for? [Trivia, Math, Date, or Year] ")

# If the kind of search is "date" take in two numbers

if(kind\_of\_search.lower() == "date"):

# Collect the month to search for

month = input("What month would you like to search for? ")

# Collect the day to search for

day = input("What day would you like to search for? ")

# Make an API call to the "date" API and convert response object to JSON

response = req.get(url + month + "/" + day + "/" + kind\_of\_search.lower()+ "?json").json()

# Print the fact stored within the response

print(response["text"])

# If the kind of search is anything but "date" then take one number

else:

# Collect the number to search for

number = input("What number would you like to search for? ")

# Make an API call to the API and convert response object to JSON

response = req.get(url + number + "/" + kind\_of\_search.lower()+ "?json").json()

# Print the fact stored within the response

print(response["text"])

**06-PythonAPIs/01/Activities/06-Ins\_OMDbRequests/OmdbRequests.py**

import requests

import json

# Note that the ?t= is a query param for the t-itle of the

# movie we want to search for.

url = "<http://www.omdbapi.com/?t=>"

api\_key = "&apikey=40e9cece"

# Performing a GET request similar to the one we executed

# earlier

response = requests.get(url + "Aliens" + api\_key)

# Converting the response to JSON, and printing the result.

json = response.json()

print(json)

# Print a few keys from the response JSON.

print("Movie was directed by " + json["Director"])

print("Movie was released in " + json["Country"])

**06-PythonAPIs/01/Activities/09-Stu\_MovieQuestions/Solved/MovieQuestions.py**

# Dependencies

import requests as req

url = "<http://www.omdbapi.com/?apikey=40e9cece&t=>"

# Who was the director of the movie Aliens?

movie = req.get(url + "Aliens").json()

print("The director of Aliens was " + movie["Director"] + ".")

# What was the movie Gladiator rated?

movie = req.get(url + "Gladiator").json()

print("The rating of Gladiator was " + movie["Rated"] + ".")

# What year was 50 First Dates released?

movie = req.get(url + "50 First Dates").json()

print("The movie 50 First Dates was released in " + movie["Year"] + ".")

# Who wrote Moana?

movie = req.get(url + "Moana").json()

print("Moana was written by " + movie["Writer"] + ".")

# What was the plot of the movie Sing?

movie = req.get(url + "Sing").json()

print("The plot of Sing was: '" + movie["Plot"] + "'.")

# BONUS: Complete this activity with a loop.

**06-PythonAPIs/01/Activities/10-Ins\_IterativeRequests/IterativeRequests.py**

# Dependencies

import random

import json

import requests

# Let's get the JSON for 100 posts sequentially.

url = "<http://jsonplaceholder.typicode.com/posts/>"

response\_json = []

# Create random indices representing, for instance, a user's choice of posts

indices = random.sample(list(range(1, 100)), 10)

for i in range(0, len(indices)):

print("Making request number " + str(i) +

" for ID " + str(indices[i]) + ".")

# Get post #i

post\_response = requests.get(url + str(indices[i]))

# Save post's JSON

response\_json.append(post\_response.json())

# Now we have 10 post objects, which we got by making 100 requests to the API.

print("We have " + str(len(response\_json)) + " posts!")

**06-PythonAPIs/01/Activities/11-Stu\_MovieLoop/Solved/MovieLoop.py**

# Dependencies

import requests as req

url = "<http://www.omdbapi.com/?apikey=40e9cece&t=>"

\_movies = []

movies = ["Aliens", "Sing", "Moana"]

for movie in movies:

\_movie = req.get(url + movie).json()

print("The director of " + movie + " is '" + \_movie["Director"] + "'.")

**06-PythonAPIs/01/Activities/12-Ins\_NYTAPI/NytApi.py**

# Dependencies

import requests as req

url = "<https://api.nytimes.com/svc/search/v2/articlesearch.json?>"

api\_key = "164b73c522a8420c8e05343ef1da0a7e"

# Search for articles that mention granola

q = "granola"

# Build query URL

query = url + "api-key=" + api\_key + "&q=" + q

# Populate articles

articles = req.get(query).json()

# The "response" property in articles contains the actual articles

# list comprehension.

\_articles = [article for article in articles["response"]["docs"]]

# Proof articles stored

print("Your Reading List\n")

for article in \_articles:

print(article["web\_url"])

**06-PythonAPIs/01/Activities/13-Stu\_RetrieveArticles/Solved/RetrieveArticles.py**

# Dependencies

import requests as req

url = "<https://api.nytimes.com/svc/search/v2/articlesearch.json?>"

api\_key = "164b73c522a8420c8e05343ef1da0a7e"

# Store a search term

q = "obama"

# Search for articles published between a begin and end date

begin\_date = "20160101"

end\_date = "20160130"

query = url + "api-key=" + api\_key + "&q=" + q + "&begin\_date=" + begin\_date\

+ "&end\_date=" + end\_date

# Retrieve articles

articles = req.get(query).json()

\_articles = [article for article in articles["response"]["docs"]]

for article in \_articles:

print("A snippet from the article: '" + article["snippet"] + "'.\n")

# BONUS: How would we get 30 results? HINT: Look up the page query param

\_articles = []

for page in range(1, 3):

query = query + "&page=" + str(page)

articles = req.get(query).json()

\_articles.extend([article for article in articles])

**06-PythonAPIs/01/Activities/14-Stu\_Combining\_Data/Solved/combining\_data.py**

import requests as req

import pandas as pd

# Create a JSON object with three links to NYT articles about it, and whatever

# other information you"d like--such as Director, Year published, etc.

# ==============================================================================

# Initialize "constants"

info = {}

name = "Gladiator"

# Initialize config variables

omdb\_url = "<http://www.omdbapi.com/?apikey=40e9cece&t=>"

nyt\_url = "<https://api.nytimes.com/svc/search/v2/articlesearch.json?>"

api\_key = "164b73c522a8420c8e05343ef1da0a7e"

# Make requests

movie = req.get(omdb\_url + name).json()

movie\_articles = req.get(nyt\_url + "api-key=" + api\_key + "&q=" + name).json()

movie\_articles = [

article for article in movie\_articles["response"]["docs"]][0:3]

# Save some information

info["Year"] = movie["Year"]

info["Director"] = movie["Director"]

# Get URLs from article data

info["Links"] = []

for article in movie\_articles:

url = article["web\_url"]

if url is not None:

info["Links"] += [url]

print("The director of " + name + " was " + info["Director"] + ".")

print("It was released in " + str(info["Year"]) + ".")

print("We collected " +

str(len(info["Links"])) +

" articles. Their links are:\n")

for url in info["Links"]:

print(url)

**06-PythonAPIs/02/Activities/01-Stu\_JSONTraversalReview/Solved/01\_JSON\_TraversalSolved.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import json

import os

# Load JSON

filepath = os.path.join("..", "Resources", "youtube\_response.json")

# this is used to close the file after we open it

with open(filepath) as jsonfile:

video\_json = json.load(jsonfile)

# In[2]:

# Isolate "data items" for easy reading

data = video\_json["data"]

data\_items = data["items"]

# Retrieve the video's title

title = data\_items[0]["title"]

print("Title: ", title)

# In[3]:

# Retrieve the video's rating

rating = data\_items[0]["rating"]

print("Rating:", rating)

# In[4]:

# Retrieve the link to the video's default thumbnail

default\_thumbnail = data\_items[0]["thumbnail"]["default"]

print("Thumbnail: ", default\_thumbnail)

# In[5]:

# Retrieve the number of views this video has

view\_count = data\_items[0]["viewCount"]

print(f"View count: {view\_count}")

**06-PythonAPIs/02/Activities/02-Stu\_RequestReview/Solved/02-Stu\_RequestReviewSolved.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import json

import requests

# In[2]:

# Specify the URL

url = "<http://nyt-mongo-scraper.herokuapp.com/api/headlines>"

# Make request and store response

response = requests.get(url)

# Print status code

print(response.status\_code)

# In[3]:

# JSON-ify response

response\_json = response.json()

# In[4]:

# Print first and last articles

print(f"The first response is {json.dumps(response\_json[0], indent=2)}.")

print(f"The last response is {json.dumps(response\_json[-1], indent=2)}.")

# In[5]:

print(f"We received {len(response\_json)} responses.")

**06-PythonAPIs/02/Activities/03-Ins\_OpenWeatherRequest/03-Ins\_OpenWeatherRequest.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import json

import requests

from config import api\_key # uses a config file to hold the api-key

# In[2]:

# Save config information

url = "<http://api.openweathermap.org/data/2.5/weather?>"

city = "London"

# Build query URL

query\_url = url + "appid=" + api\_key + "&q=" + city

# In[5]:

# Get weather data

weather\_response = requests.get(query\_url)

weather\_json = weather\_response.json()

# Get the temperature from the response

print(f"The weather API responded with: {weather\_json}.")

print(json.dumps(weather\_json))

# In[8]:

weather\_json.keys()

weather\_json['weather']

# In[9]:

weather\_json['main']['temp']

**06-PythonAPIs/02/Activities/04-Stu\_Bujumbura/Solved/04-Stu\_BujumburaSolved.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import requests

from config import api\_key

# Save config information.

url = "<http://api.openweathermap.org/data/2.5/weather?>"

city = "Bujumbura"

units = "metric"

# In[2]:

# Build query URL and request your results in Celsius

query\_url = f"{url}appid={api\_key}&q={city}&units={units}"

# Get weather data

weather\_response = requests.get(query\_url)

weather\_json = weather\_response.json()

# In[3]:

# Get temperature from JSON response

temperature = weather\_json["main"]["temp"]

# In[4]:

# Report temperature

print(f"The temperature in Bujumbura is {temperature} C.")

# In[5]:

# BONUS

# use list of units

units = ["metric", "imperial"]

# set up list to hold two different temperatures

temperatures = []

# loop throught the list of units and append them to temperatures list

for unit in units:

# Build query URL based on current element in units

query\_url = url + "appid=" + api\_key + "&q=" + city + "&units=" + unit

# Get weather data

weather\_response = requests.get(query\_url)

weather\_json = weather\_response.json()

# Get temperature from JSON response

temperature = weather\_json["main"]["temp"]

temperatures.append(temperature)

# Report temperatures by accessing each element in the list

print(

f"The temperature in Bujumbura is {temperatures[0]}C or {temperatures[1]}F.")

**06-PythonAPIs/02/Activities/05-Ins\_OpenWeatherDataFrame/cities.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import csv

import matplotlib.pyplot as plt

import requests

import pandas as pd

from config import api\_key

# In[2]:

# Save config information.

url = "<http://api.openweathermap.org/data/2.5/weather?>"

units = "metric"

# Build partial query URL

#query\_url = f"{url}appid={api\_key}&units={units}&q="

query\_url = "%sappid=%sunits%s&q="(url, api\_key, units)

# In[3]:

cities = ["Paris", "London", "Oslo", "Beijing"]

# set up lists to hold reponse info

lat = []

temp = []

# Loop through the list of cities and perform a request for data on each

for city in cities:

response = requests.get(query\_url + city).json()

lat.append(response['coord']['lat'])

temp.append(response['main']['temp'])

print("The latitude information received is: ")

print("The temperature information received is: {temp}")

# In[4]:

# create a data frame from cities, lat, and temp

weather\_dict = {

"city": cities,

"lat": lat,

"temp": temp

}

weather\_data = pd.DataFrame(weather\_dict)

weather\_data.head()

# In[5]:

# Build a scatter plot for each data type

plt.scatter(weather\_data["lat"], weather\_data["temp"], marker="o")

# Incorporate the other graph properties

plt.title("Temperature in World Cities")

plt.ylabel("Temperature (Celsius)")

plt.xlabel("Latitude")

plt.grid(True)

# Save the figure

plt.savefig("TemperatureInWorldCities.png")

# Show plot

plt.show()

**06-PythonAPIs/02/Activities/06-Stu\_TVRatings/Solved/06-Stu\_TVRatings\_Solved.ipynb**

# coding: utf-8

# In[1]:

#Dependencies

import requests

import json

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# In[2]:

# list of tv show titles to query

tv\_shows = ["Altered Carbon", "Grey's Anatomy", "This is Us", "The Flash",

"Vikings", "Shameless", "Arrow", "Peaky Blinders", "Dirk Gently"]

# tv maze show search base url

base\_url = "<http://api.tvmaze.com/search/shows?q=>"

# set up lists to hold response data for name and rating

titles = []

ratings = []

networks = []

# loop through tv show titles, make requests and parse

for show in tv\_shows:

target\_url = base\_url + show

response = requests.get(target\_url).json()

titles.append(response[0]['show']['name'])

ratings.append(response[0]['show']['rating']['average'])

# In[3]:

# create dataframe

shows\_df = pd.DataFrame({

"title": titles,

"rating": ratings

})

shows\_df

# In[4]:

# create a list of numbers for x values

tick\_locations = np.arange(len(shows\_df))

# create bar chart and set the values of xticks

plt.bar(tick\_locations, shows\_df['rating'], align="center")

plt.xticks(tick\_locations, shows\_df['title'], rotation=45, ha="right")

plt.savefig("tv\_show\_ratings.png")

plt.show()

**06-PythonAPIs/02/Activities/07-Ins\_ExceptionHandling/07-Ins\_Exception.ipynb**

students = {

# Name : Age

"James": 27,

"Sarah": 19,

"Jocelyn": 28

}

try:

print(students["Jezebel"])

except:

print("Jezebel doesnt exist as a key")

print("This line will never print.")

**06-PythonAPIs/02/Activities/07-Ins\_ExceptionHandling/07-Ins\_ExceptionHandling.ipynb**

students = {

# Name : Age

"James": 27,

"Sarah": 19,

"Jocelyn": 28

}

# Try to access key that doesn't exist

try:

students["Jezebel"]

except KeyError:

print("Oops, that key doesn't exist.")

# "Catching" the error lets the rest of our code execute

print("...But the program doesn't die early!")

**06-PythonAPIs/02/Activities/08-Stu\_MakingExceptions/Solved/08-Stu\_MakingExceptionsSolved.ipynb**

# Your assignment is to get the last line to print without changing any

# of the code below. Instead, wrap each line that throws an error in a

# try/exept block.

try:

print("Infinity looks like + " + str(10 / 0) + ".")

except ZeroDivisionError:

print("Woops. Can't do that.")

try:

print("I think her name was + " + name + "?")

except NameError:

print("Oh, I forgot to define 'name'. D'oh.")

try:

print("Your name is a nonsense number. Look: " + int("Gabriel"))

except ValueError:

print("Drat. 'Gabriel' isn't a number?")

print("I made it through the gauntlet. The message survived!")

**06-PythonAPIs/02/Activities/09-Ins\_OpenWeatherWrapper/09-Ins\_OpenWeatherWrapper.ipynb**

# coding: utf-8

# In[2]:

# Dependencies

import openweathermapy.core as owm

#config

from config import api\_key

# In[3]:

# Create settings dictionary with information we're interested in

settings = {"units": "metric", "appid": api\_key}

# In[8]:

# Get current weather

current\_weather\_paris = owm.get\_current("Paris", \*\*settings)

print("Current weather object for Paris: " + str(current\_weather\_paris))

current\_weather\_paris['main']['temp']

# In[5]:

summary = ["name", "main.temp"]

data = current\_weather\_paris(\*summary)

print(f"The current weather summary for Paris is: {data}.")

# In[ ]:

# \*summary ---list

# \*\*settings ----kwargs

**06-PythonAPIs/02/Activities/10-Stu\_MapWrap/Solved/MapWrap.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import csv

import matplotlib.pyplot as plt

import openweathermapy as ow

import pandas as pd

# import api\_key from config file

from config import api\_key

# In[2]:

# Create a settings object with your API key and preferred units

settings = {"units": "metric", "appid": api\_key}

# In[3]:

# Get data for each city in cities.csv

weather\_data = []

with open("../Resources/cities.csv") as cities:

cities = csv.reader(cities)

cities = [city[0] for city in cities]

weather\_data = [ow.get\_current(city, \*\*settings) for city in cities]

# In[4]:

# Create an "extracts" object to get the temperature, latitude,

# and longitude in each city

summary = ["main.temp", "coord.lat", "coord.lon"]

# Create a Pandas DataFrame with the results

data = [response(\*summary) for response in weather\_data]

weather\_data = pd.DataFrame(data, index=cities)

weather\_data

# In[5]:

# BONUS:

column\_names = ["Temperature", "Latitude", "Longitude"]

weather\_data = pd.DataFrame(data, index=cities, columns=column\_names)

weather\_data

**06-PythonAPIs/02/Activities/11-Ins\_WorldBankAPI/11-Ins\_WorldBankAPI.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import requests

url = "<http://api.worldbank.org/v2/>"

format = "json"

# Get country information in JSON format

countries\_response = requests.get(f"{url}countries?format={format}").json()

# First element is general information, second is countries themselves

countries = countries\_response[1]

# In[2]:

# Report the names

for country in countries:

print(country["name"])

**06-PythonAPIs/02/Activities/12-Stu\_TwoCalls/WorldBank.ipynb**

# coding: utf-8

# In[5]:

import pandas as pd

import numpy as np

import requests as req

# In[4]:

url = '<http://api.worldbank.org/v2/?>'

q= "lendingTypes"

response = requests.get(query\_url + "&q=")

response

**06-PythonAPIs/02/Activities/13-Ins\_CitiPy/13-Ins\_CitiPy.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

from citipy import citipy

# In[2]:

# Some random coordinates

coordinates = [(200, 200), (23, 200), (42, 100)]

# In[3]:

cities = []

for coordinate\_pair in coordinates:

lat, lon = coordinate\_pair

cities.append(citipy.nearest\_city(lat, lon))

# In[4]:

for city in cities:

country\_code = city.country\_code

name = city.city\_name

print(f"The country code of {name} is '{country\_code}'.")

**06-PythonAPIs/02/Activities/Extra/06-Stu\_CityPressure/Solved/CityPressure.ipynb**

# coding: utf-8

# In[1]:

# Dependencies

import csv

import matplotlib.pyplot as plt

import requests

import pandas as pd

#import api\_key

from config import api\_key

# In[2]:

# Save config information.

url = "<http://api.openweathermap.org/data/2.5/weather?>"

units = "metric"

# In[3]:

# Build partial query URL

query\_url = f"{url}appid={api\_key}&units={units}&q="

# In[4]:

# list of cities to query

cities = ["London", "Paris", "Las Vegas", "Stockholm", "Sydney", "Hong Kong"]

# list for response results

lon = []

pressure = []

# loop through cities, make API request, and append desired results

for city in cities:

response = requests.get(query\_url + city).json()

lon.append(response['coord']['lon'])

pressure.append(response['main']['pressure'])

print(f"Longitude: {lon}")

print(f"Pressure: {pressure}")

# In[5]:

# build a dataframe from the cities, lon,and pressure lists

weather\_data = {"city": cities, "pressure": pressure, "lon": lon}

weather\_data = pd.DataFrame(weather\_data)

weather\_data

# In[6]:

# Build a scatter plot for each data type

plt.scatter(weather\_data["lon"], weather\_data["pressure"], marker="o")

# Incorporate the other graph properties

plt.title("Pressure in World Cities")

plt.ylabel("Pressure (Celsius)")

plt.xlabel("Longitude")

plt.grid(True)

# Save the figure

plt.savefig("PressureInWorldCities.png")

# Show plot

plt.show()