**Programs/Syntax**

[API/JSON](#_API/JSON)

[CSS](#_CSS)

[GitBash](#_GITBASH)

[HTML](#_HTML)

[Java Script](#_Java_Script)

[Matplotlib](#_MATPLOTLIB)

[Pandas](#_PANDAS_1)

[Python](#_PYTHON)

[VBA](#_VBA)

# **API/JSON**

**ANATOMY**

1. Retrieve initial requests
   1. response=requests.get(url)
   2. response=requests.get(url).json()
2. Save the initial response as a variable
   1. response\_json=response.json()
3. Retrieve a deeper-level request
   1. response = requests.get(url + “Whatever key you want” + api\_key).json()
   2. url2 = url + some criteria (followed by) response = requests.get(url2)
   3. response = requests.get(url + "/5ef094188566f88").json() (URL in quotes found in json file)

**Anatomy example**

API

“show”:{

“name”: name of show

“rating”: {

“average”: average of show

for eachrow in tv\_shows:

url2=url+eachrow

response=requests.get(url2).json()

titles.append(response[0]["show"]["name"])

ratings.append(response[0]["show"]["rating"]["average"])

**Dependencies**

* import requests
* import json
* from pprint import pprint
* from config import api\_key

**Standard**

* url = “ url here “

print(requests.get(url)) (maybe tells you the # rows of data?)

print(requests.get(url).json()) (returns the ugly version)

* response = requests.get(url).json() (This is “making the request and storing the response”)

pprint(response)

**Using API Key**

* Create a python file called config that has your API key
* from config import api\_key

url = “ url” (or sometimes url = url + api\_key)

api\_key = "&apikey=" + api\_key

response = requests.get(url + “Whatever key you want” + api\_key).json()

**Store the API response within a variable**

* response = requests.get(url)

response\_json = response.json() (does exact same thing as adding .json() to the line above it)

#print(json.dumps(response\_json, indent=4, sort\_keys=True))

pprint(response\_json)

**Narrow down to specific part of file**

* response=requests.get(url).json()

print(json.dumps(response, indent = 4, sort\_keys=True)) (to find the specific URL)

response = requests.get(url + "/5e9e4502f509094188566f88").json() (the URL in quotes was found after looking at the json file)

#print(json.dumps(response, indent=4, sort\_keys=True))

pprint(response)

* base url = “ url “

key\_value\_you\_want\_to\_narrow\_down\_to = “Value”

url = base\_url + key\_value\_you\_want\_to\_narrow\_down\_to

#print(url)

response = requests.get(url)

#print(response.url)

data = response.json()

#print(json.dumps(data, indent = 4, sort\_keys=True))

pprint(data)

**Get particular data**

* print(response\_json[“name of key”])
* new\_var = len(response\_json[“name of key”])

print(new\_var)

* new\_var = response\_json[“name of key”][*list* index number of var you want]

print(new\_var)

* new\_var = response\_json[“name of dictionary”][“name of key”]

**For Loop**

* movies = ["Aliens", "Sing", "Moana"]

responses = []

for eachrow in movies:

movie\_data = requests.get(url + eachrow).json()

responses.append(movie\_data)

print(f'The director of {eachrow} is {movie\_data["Director"]}')

* API

“show”:{

“name”: name of show

“rating”: {

“average”: average of show

for eachrow in tv\_shows:

url2=url+eachrow

response=requests.get(url2).json()

titles.append(response[0]["show"]["name"])

ratings.append(response[0]["show"]["rating"]["average"])

# **CSS**

Used to control spacing, color, fonts, style

Always begins with a rule set: a ***selector*** that selects which HTML elements to style, and a ***declaration block***, which describes the styles to apply

**Tag Selectors**

P = select ***all*** paragraph tags

**Other Selectors**

Id: #id

* <style> #site-description { } </style>

Class: .class

* <style> .book-summary { } </style>

**Syntax**

Color = font color

**Font text**

Font-family

**Font Style and Weight and Variants**

Italicize: font-style: italic;

Underline: Text-decoration: underline;

Uppercase: Text-transform: uppercase;

Bold: font-weight: bold;

**Units: Absolute**

Pixels: px

* <style> {width: 100 px;}

Margins: margin:

**Units: Relative**

Percentages

**Colors**

By RGB, 0-255

* <style> body {background-color: rgb(255, 0, 255);} </style>

By hexadecimal, 00-FF:

* <style> body {background-color: #ff00ff;} </style>
* <style> body {background-color: #00f;} </style>

**Design**

Border: border: describe; (ex: solid, dashed red, 1rem solid, thick double)

Cursor: cursor: describe; (ex: auto, default, pointer, not-allowed)

Box-shadow

**Comment**

/\* add comments that it will ignore like this \*/ (put in between <style> tags)

# **GITBASH**

**DIRECTORY**

Create new directory

* mkdir [name the folder]
* mkdir test

Change to a different directory

* cd [name of file location]
* cd Desktop

List contents of active directory

* ls

Parent directory

* cd ..

User directory

* cd ~

Identify what directory you are in

* pwd

Remove directories

* rm -r

**FILE**

Create new file

* touch [name of file]
* touch sample.txt

Change file name

* mv [old name] [new name]
* mv folder1 folder2

Open file

* explorer [filename]

Move to a directory

* mv [file name] [directory name]
* mv sample.txt test folder2

Open to see/change (should open in VSCode)

* code [name of file]
* code index.html

**Run Python file (.py)**

Python [file name]

Remove files

rm

**LOG**

Navigate within a log (like when you have the : symbol only)

* j: move down one line at a time
* k: move up one line at a time
* d: move down half the page
* u: move up half the page
* f: move down the whole page
* b: move up the whole page
* q: quit, go back

Show commits on the *active* branch

* git log

Show *all* commits

* git log --all

Show info about JUST ONE commit (where you give an ID)

* git show

Show list of files that have been changed BUT NOT THE ACTUAL CHANGES

* git log --stat

Show the *actual changes* to a file

* git log -p

Show commits by text (USE SINGLE QUOTE!)

* git log --grep=’[text]’
* git log --grep=’Facebook’

**LOG - DIFFERENT WAYS TO DISPLAY**

Show commits - one header for each change

* git log --oneline

Show commits/logs with bullets

* git log --graph

**REPO-CREATE**

Create new

* git init

**REPO - CHANGES**

Add/stage changes (all)

* git add -A

Add/stage changes (specific)

* git add [name of file to add/stage]
* git add index.html

Commit staged changes

* git commit -m “[describe changes being committed since last commit]”
* git commit -m “Added an index file”

Pull over changes

* git pull (do while in main branch)

Push changes to main branch

* git push origin main

Alter the *most recent* commit

* git commit --amend

Revert changes (this just undoes what you did)

* git revert [SHA]
* git revert db7e7a

Reset change (CAUTION - this undoes and then creates a new commit)

* git reflog

**REPO-BRANCHES**

Create *new* branch

* git branch [name the branch]
* git branch sidebar

Checkout *existing* branch

* git checkout [name the branch]
* git checkout main
* git checkout sidebar

Create *new* branch and check out (-b = a new branch that does not currently exist)

* git checkout -b [name of branch]
* git checkout -b header

View all the branches (a \* will appear next to the active one)

* git branch

Delete a branch (cannot delete a branch you are currently on; use capital D to force it - git will not delete if changes are pending)

* git branch -d [name of branch]
* git branch -d sidebar

**REPO - MERGE BRANCHES**

Merge

* git merge [name of branch to bring into existing branch - brings the changes *into* current branch]
* git merge footer

Undo merge

* git reset --hard HEAD^

**REPO - WHERE AM I AND WHAT IS GOING ON?**

Identify what directory/branch/area you are in

* pwd

Status of existing git file - changes that have been *made/saved* but not yet *committed*

* git diff

Status of existing git file (do you have pending changes, what branch you are on, etc.)

* git status

**REPO - OTHER**

Keep a file in the directory but make sure it is not accidentally added

* End file name with .gitignore

**REPO - TAGS**

Create annotated tag

* git tag -a [the tag you want to create]
* git tag -a v1.0

Tag an old commit

* git tag -a [the tag you want to create] [the SHA]
* git tag -a v1.0 a87984

Delete tag

* git tag -d (or) --delete [what the tag is you want to delete]
* git tag -d v1.0
* git tag --delete v1.0

Make tag display in log

* git tag --decorate

# **HTML**

**Elements: Structure**

p = paragraph

h = heading (use 1 for largest, 2 for smaller…)

span = (part of a line)

div = groups chunks of content together

ul = unordered list

li - ordered list

a = hyperlink

* <a href=”url of site”>Text to Display</a>

**Elements: Style**

em = italicize text

strong = bold

**Elements: Graphics**

Buttons: button

img = image

* <img src=”url of image” alt=”Description of image”>

figcaption = caption for a figure/image

**Elements: Fundamental**

DOCTYPE = is this a txt, html, etc.

head: Meta info like title, links to scripts, etc. that the site needs

body: actual content users will see

**Lists**

Ordered = with numbers or letters

Unordered = bullets, nothing

**Chrome, Display**

Chrome: CTR + O

**Syntax**

* <div><h1>Article Title</h1></div>

is the same thing as

* <div>

<h1>Article Title</h1>

</div>

**Comments**

* <!-- put in comments this way -->

# **Java Script**

NOT the same thing as Java

In Chrome: Right click, inspect, Console

**Numbers**

Numbers

Arithmetic Operations

Comparing Numbers (<, >, <=, >=, ==, !=)

**Booleans**

True

False

**String**

In quotes (can use single or double)

Can contain text or numbers

To concatenate just use a + between two strings

To use quotes inside a string, use \

* “The man whispered, \”please speak to me.\””

**Special Characters**

\\ = backslash

\” = double quote

\’ =- single quote

\n = newline

\t = tab

**Variables**

Var

* var name = “Randy the cat”;
* var age = 10;

Use “camel case”

* var catName = “Randy”
* thisIsCamelCase

**String Indexing**

* “James”[0]; returns “J”

or

* Var name = “James”;

name[0];

**Comments**

// comment (or) /\* comment \*/

# **MATPLOTLIB**

**Always start with**

* import numpy as np
* import matplotlib.pyplot as plt
* import pandas as pd

**Maybe start with?**

* import seaborn as sns

**General Tips**

* Put all graphic stuff (colors, markers, style, etc.) in the same box
* plt.plot = MPL plot (as you import MPL as plt)
* nameof\_df= pd.read\_csv("Resources/filename.csv")

nameof\_df.head()

or

* file\_to\_load ="../Resources/filename.csv"

nameof\_df =pd.read\_csv(file\_to\_load)

nameof\_df.head()

**Create x-axis - Line graph**

* x\_axis\_name = np.arange(0, 13, 1) (first # on axis, second # -1, increments - so this gives x-axis 1-12 in increments of 1)
* months = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
* months = [“Jan”, “Feb”, “March”]

**Create x-axis: Bar chart**

* county\_pop = [2000, 10000, 150000, 1500, 1956]

x\_axis = np.arange(len(county\_pop))

**Plot a line graph**

* plt.plot(x\_axis\_varname, y\_axis\_varname)

plt.show() (to plot 2 or more on same graph, add a second statement before the show command)

**Plot line graph and make it better looking**

* fahrenheit, = plt.plot(months, temp\_f, marker="+",color="blue", linewidth=1, label="Fahreneit")

celsius, = plt.plot(months, temp\_c, marker="s", color="Red", linewidth=1, label="Celsius")

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

plt.xlabel("Months")

plt.ylabel("Degrees")

**Plot a bar chart**

* plt.bar(x\_axis\_var, y\_axis\_var, color="xxx", align = "xxx")

**Plot a bar chart**

* plt.bar(x\_axis\_var, y\_axis\_var, color="xxx", align = "xxx")

tick\_locations = [value for value in x\_axis\_varname]

plt.xticks(tick\_locations, x\_axis\_ varname)

**Plot a bar chart using Pandas**

* gender\_chart = gender\_trips.plot(kind="bar", title="Bike Trips by Gender")

**Plot bar chart using CSV file**

* import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sn

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

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

tick\_locations=[value+0.4 for value in x\_axis] (pushes ticks towards center)

plt.figure(figsize=(18,5)) (makes it a smaller chart)

plt.bar(x\_axis, rain\_df["Inches"], color="r", alpha=0.2, align = "edge") (alpha is opacity)

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

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

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

plt.tight\_layout()

**Plot a pie chart**

* pies = ["Apple", "Pumpkin", "Chocolate Creme", "Cherry”]

pie\_votes = [47, 37, 32, 27]

colors = ["yellow", "green", "lightblue", "orange"]

explode = (0.1, 0, 0, 0)

plt.pie(pie\_votes, explode=explode, labels=pies, colors=colors, autopct="%1.1f%%", shadow=True, startangle=140)

plt.axis("equal")

plt.show()

**Plot a scatter plot**

* temp = [14.2, 16.4, 11.9, 15.2, 18.5]

icecream\_sales = [215, 325, 185, 332, 406]

plt.scatter(“temp”, “icecream\_sales”, marker="o", facecolors="red", edgecolors="black")

plt.xlim(min(“temp”)-1, max(“temp”)+1)

plt.ylim(min(“icecream\_sales”) -10, max(“icecream\_sales”)+10)

**Plot a scatter plot from a df**

* temp\_icecream\_df.plot(kind = "scatter", x = "temp", y = "icecream\_sales”, marker="o", c="red")

**Set limits of x and y axis**

* plt.xlim(#, #)
* plt.ylim(#, #)
* plt.xlim(-0.75, len(x\_axis)-0.25) (bar chart)
* plt.ylim(0, max(y\_axisvar)+0.4) (bar chart)

**Labels**

* plt.xlabel(“text”)
* plt.ylabel(“text”)
* plt.title(“text”)

**Add a background grid**

* plt.grid()

# **PANDAS**

Begin with import pandas as pd

* nameof\_df= pd.read\_csv("Resources/filename.csv")

nameof\_df.head()

or

* file\_to\_load ="../Resources/filename.csv"

nameof\_df =pd.read\_csv(file\_to\_load)

nameof\_df.head()

**Concatenate DF**

* brandnew\_df = pd.concat([old\_df1, old\_df2, old\_df3], axis=1)

**Merge**

* merge\_df = pd.merge(first\_df, second\_df, on="joining var") (the first df is “left”, the second is “right”, and “inner” is implied unless you specify)
* merge\_df = pd.merge(first\_df, second\_df, on="joining var", how=”inner, outer, etc”)
* “Inner join” is where you have a matching field in each df - THIS IS THE DEFAULT
* “Left join” = all record from left, and matched from right (tip: just put the one you want the most on the left and *always* do a left join - then no need to do a right, ever)
* “Right join”= all from right, and matched from left
* “Outer join”: All records where a match on either side

**Merge, then delete a duplicate column**

* del merged\_df\_name[“Column to delete”]

**Add two variables together**

* new\_var=existing\_var.add(other\_existing\_var, fill\_value=0) (fill\_value means to replace NaN with 0)

**Give an overview**

List of variables and type

* data\_filename\_df.dtypes
* print(dfname.dtypes) (use for dataframes)

Statistical overview of NUMERIC data

* data\_filename\_df.describe()

Mean

* average = data\_filename\_df[“Var name”].mean()
* data\_filename\_df[“Var name”].mean()
* if you do not specify a var, data\_filename\_df.mean() it gives you mean for everything

Sum

* total = data\_filename\_df["Var name"].sum()
* data\_filename\_df["Var name"].sum()
* sum\_all\_values = data\_filename\_df[[“Var 1 to sum”, “Var 2 to sum”, “Var 3 to sum”]].values.sum() (sums all values from those 3 vars)

Min and max

* data\_filename\_df ["Record Release Date"].max()
* data\_filename\_df ["Record Release Date "].min()

Count items for a particular column

* count = data\_filename\_df["Gender"].value\_counts()
* data\_filename\_df["Gender"].value\_counts()

Show every element that appears only once in a series

* unique = data\_filename\_df["Names of Bands”].unique() (gives list of all bands in dataset)
* data\_filename\_df[“Names of Bands”].unique()

Count unique elements for one column (i.e., gives you a single number that = num unique items)

* band\_count = data\_filename.df.nameofbands.nunique()
* new\_var\_df=pd.DataFrame({"New New For Result" : [data\_filename\_df ["Column to look for values in"].nunique()]})

Calculate a value and add it in as a new column

* thousands\_of\_dollars = data\_filename\_df["Amount"]/1000

data\_filename\_df["Thousands of Dollars"] = data\_filename\_df["Amount"]/1000

(Divides # in column “Amount” by 1000 and adds a new col called “Thousands of Dollars” with the values)

* pokemon\_comparison\_df["Total"] = pokemon\_comparison\_df.sum(axis=1) (adds new column, “Total”, which sums the value of all the columns)
* name\_df["Total of Whatevs"] = name\_df ["Thing to Sum1"] + name\_df ["Thing to Sum2"]

**Re-organize shit**

Move cols around

* reorg\_data\_df = existing\_dataset\_df [[“Put” , “Col” , “In” , “OrderYouWant” , “Using” , “ExistingNames”]] (this also creates a new df called “reorg\_data” but you can just use same file name and overwrite; also, if you exclude vars and rename the df, the other vars will be deleted)

Rename shit

* reorg\_data\_df = existing\_dataset\_df.rename(columns = {‘Existing name’ : ‘New name’ , ‘Existing name’ : ‘New name’})

Change to current/decimals. Etc.

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

Delete one row based on index number

* trip\_df = trip\_df.drop(trip\_df.index[50792])

Delete rows based on value in a column

* name\_df.drop(name\_df.loc[name\_df[“name of column”]=="value that indicates to delete that row"].index, inplace=True)

Convert object to int

* name\_df[“column to convert”] = pd.to\_numeric(name\_df[“column to convert”],errors = 'coerce')

Replace “NaN” with 0

* name\_df = name\_df.fillna(0)

**Series: like a list that gets indexed**

* series\_variable = pd.series ([“UCLA”, “UC Berkeley”, “Bridgewater”]) will return
  + 0 UCLA
  + 1 UC Berkeley
  + 2 Bridgewater

**Dataframe (think “Excel sheet with headers and index”)**

* Strings go in quotes, integers do not
* It must be spelled “DataFrame” not “dataframe”

Convert a list of dictionaries into a DF

* state\_dict = [{“State” : “New York” , “ABBREVIATON” : “NY”} , {“STATE” : “Utah” , “ABBREVIATION” : “UT”}]

states\_df = pd.DataFrame(state\_dict) (creates a new df called states\_df)

Convert a single dictionary with lists into a DF

* pharaoh\_df = pd.DataFrame({“Dynasty”: [“Early Period” , “Old Kingdom”],   
  “Pharaoh” : [“Thinis”, “Mephis”]})

Create a df

* benfranklin\_df = pd.DataFrame({

“Frames”: [“Ornate” , “Classical” , “Modern”] ,

“Price” : [15.00, 12.50, 10.00] ,

“Sales” : [100, 210, “N/A”]

}]

Create a df from one dictionary

* artstuff\_df = pd.DataFrame({

"Painting" : ["Mona Lisa", "Van Gogh self portraint" , "Starving Artist" , "Toddler Drawing"] ,

"Popularity" : ["Very Popular" , "Popular" , "Average" , "Not Popular"] ,

"Price" : [25, 20, 11, 1]

})

Create a df from a list of dictionaries

* moreart\_df = pd.DataFrame([

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

{"Painting": "Van Gogh self portraint", "Price" : 20 , "Popularity" : "Popular"} ,

{"Painting": "Starving Artist", "Price" : 11 , "Popularity" : "Average"} ,

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

])

Create a new df from existing df using specific criteria

* new\_df = exisiting\_df.loc [ (exisiting\_df ["Column you care about"] >=100) &

(exisiting\_df ["Another Column"] >0) &

(exisiting\_df ["Yet another column”] <0) ]

**Read portions of a dataset and make a dataframe**

* marcs\_file\_df = pd.read\_csv(data\_file)

marcs\_file\_df.tail() returns the last 5 rows of data with headers

* marcs\_file\_df = pd.read\_csv(data\_file)

marcs\_file\_df.head() returns the first 5 rows of data with headers

* marcs\_file\_df = pd.read\_csv(data\_file)

marcs\_file\_df[“Gender”].head() returns the first 5 rows of data with headers just for gender

* marcs\_file\_df = pd.read\_csv(data\_file)

marcs\_file\_df[[“Gender” , “Age”]].head() returns the first 5 rows of data with headers for gender and age only - must use the [[ ]] if you have > 1 items in there \

**GroupBy and Index**

**Groupby**

* list\_of\_local\_bands\_df = list\_of\_bands\_df.groupby(["Local"]) (this indexes df by local bands)

**Index**

* name\_df = name\_df.set\_index("Var name to use to set index")

**Bin - MUST HAVE ONE FEWER LABEL THAN BIN**

* age\_bins = [0, 9.9, 14.9, 19.9, 24.9, 29.9, 34.9, 39.9, 45] (creates age bins for ages 0-9.9, 10 to 14.9, 15 to 19.9…)

label\_names = ["<10", "10-14", "15-19", "20-24", "25-29", "30-34", "35-39", "40+"] (creates bin labels and auto-assigns by order)

marcs\_file\_df ["Age Group"]=pd.cut(marcs\_file\_df ["Age"], age\_bins, labels=label\_names)

or, to auto generate it

* bin\_inc=round(dfname[“Var”].max() - dfname[“Var”].min())/(# of bins you want)

bins = [0, bin\_inc, bin\_inc\*2, bin\_inc\*3…) (and so on until you get to the number of bins you want)

# **PYTHON**

Spacing matters

Zero-based indexing

Var names only have letters, numbers or \_ and must start with a letter or \_, and no spaces

General method is all lowercase letters with \_ separating words

**Data Types**

* int, float, bool, string

**Mutable/Immutable**

* Mutability - can be changed after it was created
* Immutable - to change them you have to create a whole new string

**Ordered**

* Ordered - does the order of content matter
* Can you use the position of an element to access the element

**Strings (str) - IF IT IS IN QUOTES IT IS SEEN AS A STRING**

* Immutable and ordered
* Single or double quotes work
* Use single quotes inside double quotes to return quotes: (“ ‘This part will print in quotes’ “)
* Use a \ to print an apostrophe: (“I think you\’re an idiot”)
* + puts strings together: print(first sentence + second sentence)
* \* repeats strings: print(word \* 5) will print the word 5 times
* len provides string length: var\_length = len(“var name”)

**Lists**

* Mutable and ordered
* Uses square brackets, words in single quotes, separated by commas

**Tuple**

* Immutable and ordered
* Can use parenthesis for the values or not
* Two or more elements closely related and always used together (like latitude and longitude)
* dimensions = 52, 40, 100

length, width, height = dimensions [this is called “tuple unpacking”]

[could also just do one line: length, width, height = 52, 40, 100]

print(“The dimensions are {}x{}x{}” .format(length, width, height))

**Assignment Operators**

* = assigns value (x =2 means value of x is 2; population = 100 means value of var “population” is 100)
* += increments (population += 400 + 50 means take existing population var (n = 400) and add 50)
* -= increments (population -= 400 + 50 means take existing population var (n = 400) and subtract 50)
* <, >, <=, >=
* == (equal to)
* != (not equal to)
* and, or
* not (inverses Boolean type)

**Functions**

* print(name of variable) (displays value of variable )
* print(type(name of variable)) (displays type of variable)

**Methods: A function that “belongs” to an object**

* is lower (are all the letters lowercase): print(full\_name.islower())
* count

print(“One fish, two fish, red fish, blue fish.”.count(‘fish’)) returns value of 4

* Format

print(“Marc has {} balloons”.format(27)) returns “Marc has 27 balloons”

maria\_string=”Maria loves {} and {}”

print (maria\_string.fomat(“math”, “statistics”))

returns “Maria loves math and statistics”

* f strings

actor = {

"name": "Sylvester Stallone",

"age": 62,

"married": True,

"best movies": [

"Rocky",

"Rocky 2",

"Rocky 3"]}

print(f’{actor["name"]} was in {actor["best movies"][0]}”) (“Sylvester Stallone was in Rocky’)

#Here is blown up detailed version AND YES YOU MUST START END THE STRING IN SINGLE QUOTES

print (f ‘ {dict name [“dict element”] } was in { dict name [“dict element”] [index #] } ‘ )

**Replace**

* name = “India”

result = name.replace(“d” , “\*”) (returns “Ind\*ia”)

**Numeric Values**

* Integers(“int”) - whole number

x = int(4.7) assigns x the value of 4; cut off the number after the decimal

* Float - has a decimal

x = float(4) assigns x the value of 4.0; always add .0 to the end of the number

**General Syntax**

* X=2

Y=3

Z=5

is the same as

X, Y, Z = 2, 3, 5

* grams = “35.0” makes 35.0 a string

grams = float(grams) makes grams a float

* monday\_sales = “121” is a string

monday\_sales = int(“121”) makes it an int - you don’t have to redefine the var or create a new one, you can just say print(int(monday\_sales)) or weekly\_sales = int(monday\_sales) + ….

* (“I am” + Str(age1) + “ years old.”) [returns “I am 9 years old” by turning age 1, an int into a string]
* Candy\_choice = [“Snickers”, “Gum”, “Pepsi”]
* Grocery\_cart [] (creates newvar grocery\_cart, specifies it as a list, and leaves the contents blank)

**Input - lets someone else input answers**

* my\_first\_name = input() (run, at bottom of screen will prompt to enter name)

can also do as my\_first\_name = input(“What is your first name”)

* options = ["r", "p", "s"]

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

* user\_input = input("Do you like (p)izza or (t)acos better?")

options = ["p", "t"]

if (user\_input == "p"):

print("Duh, everyone loves pizza")

else:

     print("Tacos rule!")

**Conditionals**

* if x == 1: [where x = 5 and y = 14]

print(“x is equal to 1)

* if x == 1 and y ==10: [where x = 5 and y = 14]

print(“Both values returned true”

* if x < 45 or y > 5: [where x = 5 and y = 14]

print(“One or more statements were true”)

* if x < 10: [where x = 5 and y = 14]

if y < 5:

print(“x is less than 10 and y is less than 5”)

elif y ==5: [this is “else if”]

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”) [this one gets returned]

* name = Dan

group\_one = [“Greg”, “Dan”, “Bob”]

if name in group\_one

print(name + “is in group one”)

**Loops: For**

* for pepsi in range(10):

print(pepsi) (returns 1, 2, 3, 4, 5, 6, 7, 8, 9 in vertical list)

* for pepsi in range(20, 30):

print(pepsi) (returns 20, 21, 22, 23, 24, 25, 26, 27, 28, 29 in vertical list)

* words = ["Warriors", "Come", "Out", "And", "Play"]

for pepsi in words:

     print(pepsi) (returns Warriors Come Out And Play in vertical list)

* candy\_choice = [“Snickers”, “Gum”, “Pepsi”]

for x in candy\_choice:

print(x) [prints out each in a row, “x” is a random name]

* candy\_choice = ["Snickers", "Gum", "Pepsi", "Popcorn", "Drugs"]

for index in range(len(candy\_choice)):

print(index, candy\_choice[index]) [prints index and string starting at 0]

* candy\_choice = ["Snickers", "Gum", "Pepsi", "Popcorn", "Drugs"]

for index in range(len(candy\_choice)):

print(index +1, candy\_choice[index]) [prints index and string starting at 1]

* cities = [‘new york’, ‘mountain view’, ‘detroit’, ‘york’]

for city in cities: [“city” is the iteration variable, but you could name it anything you want]

print(city.title())

* cities = [‘new york’, ‘mountain view’, ‘detroit’, ‘york’]

capitalized\_cities = []

for city in cities:

capitalized\_cities.append(city.title()) [capitalizes city name as it loops]

* cities = [‘new york’, ‘mountain view’, ‘detroit’, ‘york’]

for index in range(len(cities)):

cities [index] = cities[index].title()

print(cities)

* names = [“Marc Leslie”, “Liz Kolonay”, “Your Mom”]

usernames = []

for poop in names:

usernames.append(poop.lower().replace(“ “, “\_”))

print(usernames) (Returns [marc\_leslie, liz\_kolonay, your\_mom])

* names = [“Marc Leslie”, “Liz Kolonay”, “Your Mom”]

for poop in range(len(names)):

names[poop] = names[poop].lower().replace(“ “, “\_”)

print(names) (Returns [marc\_leslie, liz\_kolonay, your\_mom])

* names = ["Marc Leslie", "Liz Kolonay", "Your Face"]

for YOLO in range(len(names)):

names[YOLO] = names[YOLO].lower().replace(" " , "\_")

print(names) (returns names in lower case with a \_ in between first/last name)

* currency = ["$500$", "76$", "$230$", "Po$tato"]

count = 0

for dude in currency:

if dude[0] == "$" and dude[-1] == "$":

        count +=1

print(count) (returns answer of 2, since only 2 of the strings have $ at start and end)

* for poop in range(5, 35, 5):

print (poop) (returns list 5, 10, 15, 20, 25, 30)

* fish = “bass”

letters = []

for each\_letter in fish:

letters.append(each\_letter)

print(letters) (returns a list: [‘b’, ‘a’, ‘s’, ‘s’]

**Ranges**

* To alter, use a range which is in order of start, stop, and step
* Start is first number of sequence, stop is one above the last, step tells how many to separate by

Start defaults to 0 and stop defaults to 1 unless you specify

* print(list(range(4))) returns 0, 1, 2, 3
* print(list(range(2,6))) returns 2, 3, 4, 5 (start 2, end 6-1, no step given)
* print(list(range(1, 10, 2))) returns 1, 3, 5, 7, 9 (start 2, end 10-1, 2 places between each)

**Loop: While (loops until a condition is met)**

* card\_deck = [4, 11, 8, 5, 13, 2, 8, 10,]

hand = []

while sum(hand) <=17:

hand.append(card\_deck.pop())

print(hand) (returns 10, 8 - 8 is the number where sum is >17, 10 is last number popped out???)

* To continue or stop a while loop (see “House of pies” script):

while user\_says == “Yes”:

user\_says = int(input(“Type Y or N.”))

if user\_says == “y”:

continue

else: [insert whatever you then want to happen, like a print statement]

**Working with CSV files**

* (Shift + right click on file in explorer gives you file path)
* You have to do 2 slashes for file names

C:\Users\USER\Desktop\LearnPython\CSV files\ must be written as

C:\\Users\\USER\\Desktop\\LearnPython\\CSV files\\

Basic import syntax

* import os

import csv

tv\_shows\_csv = os.path.join (“insert path file here”)

#Opens and reads the csv file

with open(tv\_shows\_csv) as csv\_file:

csv\_reader = csv.reader(csv\_file, delimeter=”,”)

#Reads the header row first- skip if no header row

csv\_header = next(csv\_file)

print(f”Header: {csv\_header}”)

#Read through each row of data after the header

for row in csv\_reader:

#Convert row to float and compare to get what you want

if float(row[insert col #to look at]) > 5: [in this one, we want where value in col 7 >5, and I think b/c zero indexing col 7 is actually col 8)

print(row) [returns all rows where col 7 is greater than 5]

**Dictionaries**

* Ordered, stores info in keys/value pairs
* my\_dict = {} (creates a blank dictionary called “my\_dict”)
* dict = {

“brand”: “Ford”,

“model”: “Mustang”,

“year”: 1964

“colors”: [“red”, “white”, “blue”]

}

print(dict) or

print(dict[“brand”])

* elements = {‘hydrogen’: 1, ‘helium’: 2, ‘carbon’: 6}

elements[‘lithium’] =3 [adds lithium to dictionary]

* elements.get(‘lithium’) (looks up values, returns “none” if not in there)
* elements[‘lithium’] (looks up values, returns “key error” if not in there)
* print(len(dict)) [tells number of items in dictionary]
* actor = {

"name": "Sylvester Stallone",

"age": 62,

"married": True,

"best movies": [

"Rocky",

"Rocky 2",

"Rocky 3"]}

print(f’{actor["name"]} was in {actor["best movies"][0]}”) (“Sylvester Stallone was in Rocky’)

#Here is blown up detailed version AND YES YOU MUST START END THE STRING IN SINGLE QUOTES

print (f ‘ {dict name [“dict element”] } was in { dict name [“dict element”] [index #] } ‘ )

# VBA

Create the code in VB using a “module”

A module then becomes available under “Macros”  
Macros are saved as a module

To modify that macro, you can go back into VB under module

i = row

j = column

**Sub (subroutine)**

* Sub cannot return a value - it returns an *action* like a message box or a pre-formatted report

Begins with Sub [nameofsub]():

Ends with End Sub

**Function (math, etc.)**

* Begins with Function [nameoffunction]():

Ends with End Function

**Add in an Excel function**

* Application.WorksheetFunction.[name of Excel function]

**Add a Button**

* Insert a button from the Developer tab
* The dialog box pops up for Assign Macro
* Assign it a subroutine that you want it to do (you would have had to already made the sub)
* This does not create a *new* macro or new module - it just adds the *existing* macro to that button
* If you want to create a new macro, then record a macro when you add the button - hit “record” when that dialog box pops up, then CLICK THE BUTTON THEN do whatever it is you want to happen when you click that button.

**Assign a Macro: Box that says “Hello World.” This will save as a module that will be assigned “Module 1” in VB. In the Macro section, it will be available as the macro name “HelloWorld.”**

* Sub HelloWorld():

MsgBox (“Hello World”)

End Sub

**Dim: Names a variable - it declares a variable, which you then have to name**

* Dim variablename
* Dim ing1 As String (text)

ing1 = “Peanut Butter”

* Dim budget As Double (number with decimals)

budget = 5.00

* Dim age1 as Integer

age1 = 5

* Dim mileage as long [long = integer where you may end up with a lengthy number]

mileage = 564325

**Assign multiple variables of same type at same time**

* Dim ingredients (0 to 2) as String

ingredients (0) = “Peanut Butter”

ingredients (1) = “Jelly”

ingredient (2) = “Bread”

* Dim ingredients ing1, ing2, ing3 as String

ing1 = “Peanut Butter”

ing2 = “Jelly”

ing3 = “Bread”

**Boolean**

* Dim money\_growns\_on\_trees As Boolean

money\_grows\_on\_trees = False

**Put values or text in particular cells (i.e, like you would to create headers in a table)**

* Sub CatInTheHat():

Cells (2, 1).Value = “Cat” [this puts it at cell A2]

Cells (2, 2).Value = “In”

Cells (2, 3).Value = “The”

Cells (2, 4).Value = “Hat”

End Sub

* Sub CatInTheHat():

Range(“A2”).Value = “Cat”

Range (“B2”).Value = “In”

Range (“C2”).Value = “The”

Range (“C3”).Value = “Hat”

End Sub

**Put the same values or text in multiple cells**

* Sub Chessboard():

Range(“A1, H1”).Value = “Rook” [will place Rook in cells A1 *and* H1]

Range(“A1:H1”).Value = “Pawn” [will place Pawn in cells A1 *through* H1]

**Add an array of variables at once**

* Dim Ingredients (5) as string

Ingredients (0) = “Chocolate”

Ingredients (1) = “Peanut Butter”

Ingredients (2) = “Jelly”

Ingredients (3) = “Macaroni”

Ingredients (4) = “Potato Salad”

Ingredients (5) = “Fruit”

MsgBox (Ingredients (4)) [this returns “Potato salad”]

**Message Box - Make words separate**

* MsgBox(“Congrats “ & Range(“A8”) & “ “ & Range(“C5”) [the space between congrats and “ adds a space, and the & “ “ & between the two ranges adds spaces between those two words]

**Create a variable, put it in a cell, and tell VB where you will find a value you later enter or calculate**

* Dim ItemPrice as Double

ItemPrice = Range(“B1”).Value [this allows you to use “ItemPrice” to refer to cell B1]

Range(“B1”) = ItemPrice [if you are calculating item price, this will make it appear in cell B1]

**Concatenate variables**

* Sub NameTest():

Dim firstname As String

firstname = “Gandalf”

Dim title As String

title = “The Great”

Dim fullname As String

fullname = firstname + “ “ + title [number of spaces between “ and “ = number of spaces in result]

MsgBox (fullname)

* Sub NameAgeTest():

Dim age1 as integer

Age1 = 9

MsgBox (“I am” + Str(age1) + “ years old.”) [returns “I am 9 years old”]

**Worksheets**

* use “ws” - for example, “ws.Name” will reference the name of the worksheet
* For Each ws in Worksheets

ws.activate [Use at start of script to run script through all worksheets]

* ws.Range(“A1”).EntireColumn.Insert [inserts new column in Column A]

**Split up a String**

* Sub SplitPhrase()

Dim Words () As String

Dim Quicksand As String

Quicksand = “Loading my questions like a shotgun”

Words = Split (Quicksand, “ “)

MsgBox (Words(3)) [will return “like”]

**Math with variables**

* Dim price As Double

price = 5.00

Dim tax as double

tax = 0.05

Range(“A1”).Value = price \* tax [multiplies them, puts value in cell A1]

**Conditionals - Simple, one argument**

* If Range (“A2”).Value > Range (“B2”).Value

Then MsgBox (“Num 1 is greater than Num 2”)

End If [this will only return a message if A2 is > B2; otherwise it returns nothing).

**Conditionals - Three Arguments**

* If Range (“A5”).Value > Range (“B5”).Value Then MsgBox (“Num 3 is greater than Num4”)

ElseIf Range(“A5”).Value < Range(“B5”).Value Then MsgBox (“Num 4 is greater than Num3”)

Else MsgBox (“Num3 and Num4 are equal”)

End If

**Conditionals with Operators - And**

* If (Range (“A8”).Value > Range (“C8”).Value And Range (“B8”).Value > Range(“C8”).Value) Then MsgBox (“Both Num 5 and Num 6 are greater than Num 7”)

End If

**Conditionals with Operators - Or**

* If (Range (“A8”).Value > Range (“C8”).Value or Range (“B8”).Value > Range (“C8”).Value) Then

MsgBox (“Either Num5 and/or Num6 is greater than Num 7”)

End If

**For Loop**

* Sub forLoop():

Dim HeyThere as Integer [HeyThere is random, could be “I” or any word you want]

For HeyThere = 1 to 20 [Loop 20 spaces - rows and columns]

Cells(HeyThere, 1).Value=1 [Iterate through column 1 and put a value of 1 in each cell]

Cells(1, HeyThere).Value=5 [Iterate through Row 1 and put a value of 5 in each cell]

Cells(HeyThere +1, 2).Value=HeyThere+1 [in column 2, start with the number 1 on row 1 and then add one number for each row you loop through but start this 1 cell below where the loop starts - i.e. row 2]

Next HeyThere [calls the next iteration]

**Loop - create a variable name to reference in syntax**

* VarName1 = Cells(HeyThere, 1).Value [everything in the loop in column A is now called “VarName1”]
* VarName2=Cells(HeyThere, 2).Value [everything in the loop in column A is now called “VarName2”]

**Modulus - Gives the remainder of *what could not be divided completely***

* Sub modulo():

Range(“A2”).Value = 4 Mod 2 [in A2, put remainder of 4/2, which is 0]

Range(“A3”).Value = 50 Mod 44 [in A3, put remainder of 50/44, which is 6]

**Conditional Loops/Modulus**

* Sub conditional\_loops():

For DudeAbides = 1 to 10 [random variable name, creates loop for spaces 1 to 10]

If Cells(DudeAbides, 1).Value Mod 2 = 0 Then Cells(DudeAbides, 2).Value=”Even Number” [If the value of the number within the conditional loop in column A can be divided evenly by 2 thus creating 0 remainder, then in the next cell over in column B, write Even Number]

Else [otherwise…USE “ELSEIF” IF YOU ARE GOING TO USE 2 OR MORE ADDITIONAL CONDITIONS]

Cells(DudeAbides, 2).Value = “Odd Number” [any other outcome is an odd number]

End If

Next DudeAbides

**Loops to find something specific and return it somewhere specific**

* If Cells(LottoNums, 3)=WinningNumbers Then [if winning lotto numbers are in column 3 of the loop]

Range("F2").Value = Cells(MyNumbers, 1).Value [return first name of winner, located in column 1 of the loop, who has the winning numbers into cell F2]

Range("G2").Value = Cells(MyNumbers, 2).Value [return last name of winner, located in column 1 of the loop, who has the winning numbers into cell G2]

**Nested Loop (loop first through the rows, then columns)**

* Sub NestedLoop():

For TheRows = 1 to 3 [TheRows is random variable name, there are 3 rows of text]

For The Columns = 1 to 5 [TheColumns is random variable name, there are 3 rows of text]

Next TheColumns

Next TheRows

End Sub

**For Loop/Nested Loop: Count occurrence and report it in a specific cell**

* Dim CountTheButts as Integer

CountTheButts=0 [to zero it out]

For TheRows = 1 to 3 [TheRows is random variable name, there are 3 rows of text]

For The Columns = 1 to 5 [TheColumns is random variable name, there are 3 rows of text]

If Cells(TheRows, TheColumns).Value=”Butts” Then CountTheButts= CountTheButts+1 [the 1 tells it to add each occurrence once]

End If

Next TheColumns

Range(“L5”) = CountTheButts [puts the total count of “Butts” in cell L5]

Next TheRows

End Sub

**Have VBA tell you # of rows and columns that have data**

* LastRow=Cells(Rows.Count, 1).End(xlUp).Row [then use “LastRow” as the end of your for statements, like CountTheButts = 1 to LastRow]
* LastColumn = Cells(2, Columns.Count).End(xlToLeft).Column [then use “LastColumn” as the end of your for statements, like CountTheButts = 2 to LastColumn]

Or, for when going across worksheets

* LastRow=ws.Cells(Rows.Count, 1).End(xlUp).Row
* LastColumn = ws.Cells(2, Columns.Count).End(xlToLeft).Column

**Change Font Color or Cell Shade (color index at http://dmcritchie.mvps.org/excel/colors.htm)**

* Sub formatter():
* Range (“A1”).Font.ColorIndex=3 [turns fonts in cell A1 red]
* Range(“B2:B5”).Interior.ColorIndex=4 [shades cells B2-B5 green]

**If/Then, Put a String Somewhere Specific (MUST GO IN THIS ORDER)**

* Dim PassOrWarn as String

If Grade>90 Then

PassOrWarn=”Pass”

Range(“C2”).Value=PassOrWarn [Puts the string “Pass” in C2 - it must go after the If/Then statement for it to work]

**Resize Columns to Autofit**

* Columns (“A:F”).Select

Columns (“A:F”).EntireColumn.AutoFit