IBM Data Science Professional Certificate

Course 4: Python for Data Science, AI, and Development.

Week 1: Python Basics

Goals:

- Demonstrate an understanding of types in python by converting/casting data types: strings, floats, integers.
- Interpret variables and solve expressions by applying mathematical operations.
- Build a program in JupyterLab to demonstrate your knowledge of types, expressions, and variables.
- Describe how to manipulate strings by using a variety of methods and operations.
- Apply your knowledge of strings using JupyterLab.

TYPES

A type is how Python represents different types of data.

There are different categories of types, such as:

- text type (str)
- numeric type (int, float, complex)
- sequence type (list, tuple, range)
- mapping type (dict)
- set type (set, frozenset)
- boolean type (bool)
- binary type (bytes, bytearray, memoryview)

Common types (and examples and data type syntax) include:

type	syntax	example
integer	int	11
float	float	21.213
string	str	"hello world"
boolean	bool	T / F
list	[]	[1, 2, 'abc']
dictionary	{}	{'dog':1, 'cat':2}
tuple	()	('apple', 'banana')

type	syntax	example
set	{}	{'apple', 'cherry'}

TYPECASTING

Typecasting is the process of changing the type of an expression in Python, i.e. converting an integer to float

Example: float(2) -> 2.0 converts integer 2 into 2.0

Example: int(1.9) -> 1 converts float 1.1 into 1 (beware of losing precision).

Example: int('1') -> 1 converts string '1' into 1

Example: int(True) -> 1 converts boolean True into 1.

Not all typecasts are possible, such as int('A').

In [1]:
```
type(True)
```

Out[1]: bool

In [2]:
```
type(23.2)
```

Out[2]: float

In [3]:
```
type("boolean")
```

Out[3]: str

In [4]:
```
bool(0)
```

Out[4]: False

In [5]:
```
type(float(2))
```

Out[5]: float

In [6]:
```
str(1.3)
```

Out[6]: '1.3'

EXPRESSIONS

Expressions describe a type of operation the computers perform and operations that Python performs, such as mathematical operations.

In [7]:
```
30+24+105-57
```

Out[7]: 102

In [8]:
```
5*2
```

Out[8]: 10

In [9]:
```
25/5
```

Out[9]: 5.0

note: one slash / indicates division that results in a float, while double slash // indicates division that results in an integer (and rounds down).

In [10]:
```
25/6
```

Out[10]: 4.166666666666667

In [11]:
```
25//6
```

Out[11]: 4

Python follows mathematical conventions of order: PEMDAS

Parenthesis, Exponents, Multiplication, Division, Addition, Subtraction

In [12]:
```
2*60+30
```

Out[12]: 150

In [13]:
```
30+2*60
```

Out[13]: 150

In [14]:
```
A = 3+2*2
print(A)

B = (3+2)/2
print(B)

C = A+B
print(C)
```

```
7
2.5
9.5
```

VARIABLES

Variables can be used to store values. Variables can also have operations performed on them.

It's good practice to use meaningful variable names so it's easier to keep track of what each one represents.

In [15]:
```
my_variable = 10

print(my_variable)
```

```
10
```

In [16]:
```
x = 20+30+40+50

print(x)
```

```
140
```

In [17]:
```
y = x/2

print(y)
```

```
70.0
```

In [18]:
```
# replace value of x

x = x/30

print(x)
```

```
4.666666666666667
```

In [19]:
```
type(x)
```

Out[19]:
```
float
```

In [20]:
```
# Example: convert total minutes to hours

total_min = 43+42+57
print(total_min)

total_hours = total_min/60
print(total_hours)

# note that if the value of total_min is changed,
# it changes the value of total_hours
```

```
142
2.3666666666666667
```

LAB: WRITING YOUR FIRST PYTHON CODE

After completing this lab you will be able to:

- Write basic code in Python
- Work with various types of data in Python

- Convert the data from one type to another
- Use expressions and variables to perform operations

In [21]:
```
# execute your first python output

print('hello, python!')

# print() is a function in python.
# you passed a string as an argument into the function to instruct python what to print
```

```
hello, python!
```

In [22]:
```
# check which version of python you're using

import sys
print(sys.version)

# sys is a built-in module that contains many system-specific parameters and functions,
# Before using sys, we must explicitly import it.
```

```
3.9.1 (tags/v3.9.1:1e5d33e, Dec  7 2020, 17:08:21) [MSC v.1927 64 bit (AMD64)]
```

It's always a good idea to add comments to your code, as it will help others understand what your code does and serves as a reminder for you. Code in python is written with a pound sign #.

In [23]:
```
# practice writing comments
print('hello, world!') # this line prints a string
```

```
hello, world!
```

Errors in Python: Python will often tell you if you have made a mistake with your code by giving you an error message.

In [24]:
```
# print string incorrectly to get error message

frint('hello, world!')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-de120d8b0953> in <module>
      1 # print string incorrectly to get error message
      2
----> 3 frint('hello, world!')

NameError: name 'frint' is not defined
```

The error code should tell you where the error occurred (line 3) and what sort of error it was (NameError).

In [ ]:
```
# try another error by not closing your string
print('hello, python!)
```

Python is an *interpreted language*, which means the script is interpreted line by line as it is executed. As such, Python will inform you of the error once it encounters it.

In contrast, *compiled languages* examine the entire program at compile time and can warn you about the errors prior to execution.

In [ ]:
```
# Print string and error to see the running order

print("This will be printed")
frint("This will cause an error")
print("This will NOT be printed")
```

STRING OPERATIONS

A string is a sequence of characters contained within two quotes (single or double).

It's helpful to think of a string as a list or tuple (which is an ordered built in data type used to store collections of data).

It is an ordered sequence of characters.

Each character in the sequence can be accessed by using an index represented by an array of numbers (starting at 0).

You can also use negative indexing (starting at -1) to reference the characters, starting from the right-most to the left-most character.

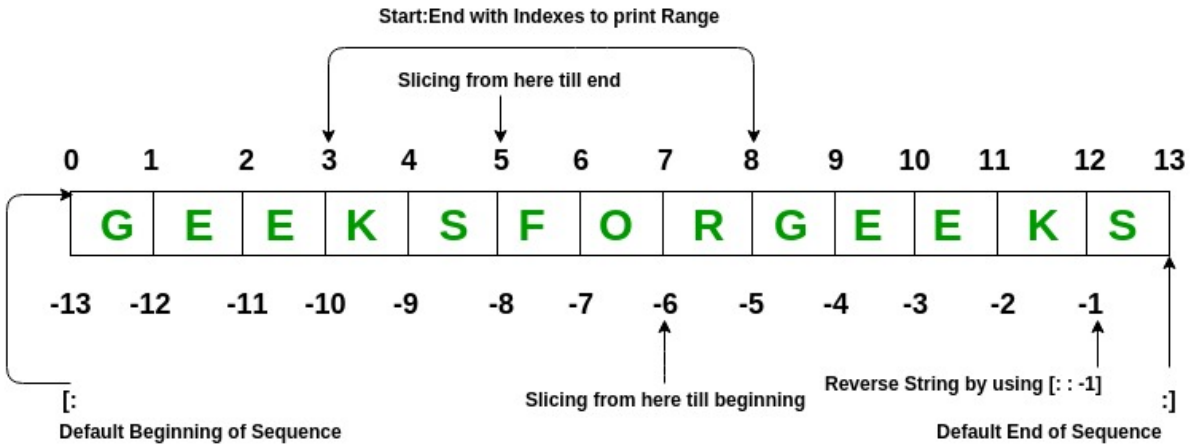
In [ ]:
```
"Michael Jackson"
```

In [ ]:
```
'Michael Buble'
```

In [ ]:
```
'1 2 3'
```

In [ ]:
```
'@ #$^!$^'
```

Here's an example of an index array for the string 'geeksforgeeks'



In [ ]:
```
Name = 'Michael Jackson'

# what is the character at index = 0?

Name [0]
```

In [ ]:
```
# what is the character at index = 8?

Name[8]
```

In [ ]:
```
Name[-7]
```

You can bind a string to another variable using *slicing* to slice up the first string into smaller strings.

In [ ]:
```
Name[0:5]
```

In [ ]:
```
Name[8:12]
```

You can also input a stride value. Stride refers to how many characters you want to move forward after the first character is retrieved from the string. It's set to 1 by default, but we can change that:

In [ ]:
```
# the ::2 indicates that we want to select every other character

Name[::2]
```

In [ ]:
```
# you can also use stride with the negative index

Name[::-2]
```

You can use slicing to grab every other letter, as well:

In [ ]:
```
Name[0:15:2]
```

In [ ]:
```
# discover the length of a string:

len(Name)
```

Combining strings is called "concatination". This can be done by using the + symbol.

In [ ]:
```
statement = Name + " is the best"
print(statement)
```

In [ ]:
```
# replicate a string

2*Name

# note this doesn't change the value of the original variable unless you explicitly set
```

In [ ]:
```
Name = Name + ' is the best!'
print(Name)
```

Strings are immutable. Backslashes \ represent the beginning of escape sequences.

Escape sequences represent strings that may be difficult to input, for example *\n* indicates a new line and *\t* indicates a tab.

In [ ]:
```
print("first line \n second line")
```

In [ ]:
```
print("first section \t second section")
```

a double backslash will put a backslash into your string:

In [ ]:
```
print ("Michael jackson \\ is the best")
```

You can place an r in front of your string, which means the string will be treated as a raw string. A raw string will treat the backslash as its own character, so it isn't treated as an escape character.

In [ ]:
```
print(r"Michael Jackson \ is the best")
```

STRING METHODS

A python method is similar to a python function, but it must be called onto an object.

Strings are sequences and so they have sequence methods that work on lists and suples. But they also have string methods that work just on strings, such as:

.upper() - converts lowercase characters to uppercase

.replace('A', 'B') - replaces the string A with string B.

.find('A') - finds the string A. The output is the first index at the start of this string. If the string A cannot be found, output will be -1.

In [ ]:
```
lowercase_variable = 'thriller is great'

uppercase_variable = lowercase_variable.upper()

print("after upper:", uppercase_variable)
```

In [ ]:
```
new_string = uppercase_variable.replace('GREAT', 'BAD')
print(new_string)

# note this method is case-sensitive.
```

In [ ]:
```
print(new_string.find('ILL'))
```

Week 2: Python Data Structures

Goals:

- Describe and manipulate tuple combinations and list data structures.
- Apply your knowledge of lists using JupyterLab.
- Apply your knowledge of tuples using JupyterLab.
- Write structures with correct keys and values to demonstrate understanding of dictionaries.
- Apply your knowledge of dictionaries using JupyterLab.
- Create sets to demonstrate understanding of the differences between sets, tuples, and lists.
- Apply your knowledge of sets using JupyterLab.

TUPLES

Lists and Tuples are called compound data types. They are one of Python's key data structures.

Tuples:

- An ordered sequence.
- Tuples are written as comma-separated elements within parentheses.
- Tuples can contain different types of values within them.
- Each element of a tuple can be accessed via an index (starting at 0), or using a negative index (starting at -1 for the right-most element).
- Tuples can be concatenated / combined.
- Tuples can be sliced so you can grab certain elements (*note: the last index number should be 1 larger than the last value you want since slicing from A:B includes A, but not B*).
- Tuple length can be calculated with len().
- Tuples are immutable and cannot be changed. As a consequence of this, you cannot change a tuple. Instead, you create a new one.
- You can use the function sorted(tuple).
- Tuples can contain other tuples within another tuple (nested tuple)

In [ ]:
```
# ratings tuple
Ratings = (10, 9, 6, 5, 10, 7, 9, 6, 2)

# different types tuple
tuple1 = ('disco', 10, 1.2)
type(tuple1)
```

In [ ]:
```
# find first element of tuple1

tuple1[0]
```

In [ ]:
```
# concatenate tuple1 with more values
tuple2 = tuple1 + ('hard rock', 10)

display(tuple2)
```

In [ ]:
```
# slice tuple2
tuple2[0:3]
```

In [ ]:
```
# length of tuple 2
len(tuple2)
```

In [ ]:
```
# immutable tuples
Ratings1 = Ratings
# now both Ratings1 and Ratings refer to the same tuple (10,...,2) object.
# they are both a variable referring to the same tuple.

# you cannot change the tuple (10,...,2), but you can assign the variable to a new tupl
Ratings = (2, 10, 1)
```

In [ ]:
```
# you can "sort" a tuple.
# the original tuple remains the same, but you can use a variable to reference the new
Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)
RatingsSorted = sorted(Ratings)
print(RatingsSorted)
```

In [ ]:
```
# nested tuple
NT = (1, 2, ("pop", "rock"), (3, 4), ("disco", (1, 2)))
# this tuple has only 5 indexes:
# 1
# 2
# ("pop", "rock")
# (3, 4)
# ("disco", (1, 2))

# you can select items within the nested tuple as below:
NT[2][1] # returns "rock"
```

In [ ]:
```
# you could even access characters in a string within the tuple
NT[2][1][0] # returns "r"
```

In [ ]:
```
NT[4][1][0] # returns 1
```

LISTS

- Lists are also ordered sequences
- A list is represented with square brackets.
- Lists can contain objects of different types.

- Lists can be nested inside of a list.
- Tuples can also be nested inside of a list.
- The same indexing conventions apply to both lists and tuples.
- Lists, like tuples, can also be sliced in the same way.
- Lists are similar to tuples in many ways, except that they are mutable and can be changed.
- Aliasing is when multiple variables or names refer to the same object, like A and B both referencing the same list. This occurs when you set B = A.
- You can clone a list, rather than reference the same list, using the syntax B = A[:] rather than B = A.

Ways to change a list include:

- .extend() method - (concatenates the argument list to the end of the original list. Each argument object is its own element.)
- .append() method - (adds the arguments as one element to the list)
- Select the index of the element you want to change and set it equal to something new.
- Delete an element of the list using del(listName[])
- .split() method - (splits strings that have spaces in them into separate strings/elements. Splits automatically on spaces, but you can identify which character you'd like to split on. This character is known as a delimiter.)
- sum(list) operation - sum up the numerical elements of a list. Note that list + list concatenates the lists.

In [ ]:
```
# list L
L = ['Michael Jackson', 10.1, 1982, [1,2], ('A', 1)]
```

In [ ]:
```
# using the + operation on lists concatenates them.
[1,2,3]+[1, 1, 1]
```

In [ ]:
```
sum([1, 2, 3])
```

In [ ]:
```
# extend the list
L.extend(["pop", 10])
display(L)
```

In [ ]:
```
# append the list
L.append(["pop", 10])
display(L)
```

In [ ]:
```
# change first element of the list
L[0] = "Janet Jackson"
```

```
display(L)
```

In [ ]:
```
# delete the last item in the list
del(L[-1])
display(L)
```

In [ ]:
```
# split
"michael jackson".split()
```

In [ ]:
```
# split on a different character
"A, B, C, D".split(",")
```

In [ ]:
```
# aliasing test
A = ["hard rock", 10, 1.2]
B = A
```

In [ ]:
```
# check values of B
display(B)
```

In [ ]:
```
# change values of A
A[0] = 'banana'

# view B
display(B)

# since A and B both reference the same object, changing an item with A also changes it
```

In [ ]:
```
# clone A instead of having B reference the same list.
A = ['hard rock', 10, 1.2]
B = A[:] # cloning

# change A to differentiate the lists
A[0] = 'apple'
display(A)
```

In [ ]:
```
display(B)
```

In [ ]:
```
# you can always use the help(object) function to get more info
help(B)
```

DICTIONARIES

- Dictionaries are a type of collection in Python.
- A dictionary has keys and values.
- The key is analogous to the index of a list. It is like an address, but it doesn't have to be an integer.

- The value is analogous to the element in a list. They contain information, usually characters.
- Dictionaries can be created using curly brackets {}
- Within the brackets, the keys are the first elements. The keys must be immutable and unique.
- Each key is followed by a colon and then the associated value.
- Values can be immutable, mutable, or duplicated.
- Each key and value pair is separated by a comma.

Actions you can take with a dictionary:

- You can use the key to look up the value using square brackets.
- You can add new entries to the dictionary.
- You can delete an entry using del(dict['key'])
- You can verify an item is in the list using the *in* command
- You can view all keys using dict.keys() method.
- You can view all values using dict.values() method.
- You can view the entire dictionary by calling its name.

In [ ]:
```
# example dictionary
dict = {'Thriller':1982, "Back in Black":1980, "The Dark Side of the Moon":1973, "The B
```

In [ ]:
```
# look up the value using the key
dict["Thriller"]
```

In [ ]:
```
# add new entry to dictionary
dict['Graduation'] = 2007
```

In [ ]:
```
dict.items()
```

In [ ]:
```
# delete an entry
del[dict['Thriller']]
dict.items()
```

In [ ]:
```
# verify The Bodyguard is in the list
'The Bodyguard' in dict
```

In [ ]:
```
# view all keys
dict.keys()
```

In [ ]:
```
# view all values
dict.values()
```

In [ ]:
```
# view dictionary
dict
```

SETS

- Sets are another type of collection in Python that can contain different types of elements.
- Unlike lists and tuples, they are unordered, which means the sets do not record the element position / index.
- Sets only have unique elements within them.
- Sets are created using curly brackets, with each item separated by commas.
- If there are duplicates listed in your set, when you actually view the set, there will be no duplicates.

Operations you can perform with sets:

- Typecasting: convert a list to a set using set(listName)
- .add() method - allows you to add a new element to the set. If a new item is added twice, nothing happens. Only unique elements are allowed.
- .remove() method - remove an element from a set
- Verify if an element is in a set using the *in* keyword
- Intersection: a new set that contains all shared elements between 2+ sets. A & B
- Union: a new set that contains all elements of 2+ sets. A.union(B)
- .issubset(): check if one set is a subset of another. A.subset(B)
- .issuperset(): check if one set is a superset of another. A.superset(B)

In [ ]:
```
# test set
set1 = {'pop', 'rock', 'hard rock', 'rock', 'R&B', 'rock', 'disco'}

# view set1 and see there are no duplicates.
set1
```

In [ ]:
```
# typecasting

# create list
album_list = ['michael jackson', 'thriller', 'thriller', 1982]

# typecast list
album_set = set(album_list)

# view new set and see there are no duplicates
album_set
```

In [ ]:
```
# add new element
album_set.add('NSYNC')
album_set
```

In [ ]:
```
# remove an element
album_set.remove('NSYNC')
album_set
```

In [ ]:
```
# verify an element
'michael jackson' in album_set
```

In [ ]:
```
# create new sets
album_set_1 = {'AC/DC', 'Back in Black', 'Thriller'}
album_set_2 = {'AC/DC', 'Back in Black', 'Dark Side of the Moon'}
```

In [ ]:
```
# find intersection of the sets
intersection_album_set = album_set_1 & album_set_2
intersection_album_set
```

In [ ]:
```
# find the union of the sets
union_album_set = album_set_1.union(album_set_2)
union_album_set
```

In [ ]:
```
# check subsets
intersection_album_set.issubset(album_set_1)
```

In [ ]:
```
# check subsets
union_album_set.issubset(album_set_1)
```

In [ ]:
```
# check superset
union_album_set.issuperset(album_set_1)
```

TABLE OF HELPFUL INFORMATION

List	Tuple	Set	Dictionary
List is a non-homogeneous data structure which stores the elements in single row and multiple rows and columns	Tuple is also a non-homogeneous data structure which stores single row and multiple rows and columns	Set data structure is also non-homogeneous data structure but stores in single row	Dictionary is also a non-homogeneous data structure which stores key value pairs
List can be represented by [ ]	Tuple can be represented by ( )	Set can be represented by { }	Dictionary can be represented by { }
List allows duplicate elements	Tuple allows duplicate elements	Set will not allow duplicate elements	Set will not allow duplicate elements but keys are not duplicated
List can use nested among all	Tuple can use nested among all	Set can use nested among all	Dictonary can use nested among all
Example: [1, 2, 3, 4, 5]	Example: (1, 2, 3, 4, 5)	Example: {1, 2, 3, 4, 5}	Example: {1, 2, 3, 4, 5}
List can be created using **list()** function	Tuple can be created using **tuple()** function.	Set can be created using **set()** function	Dictonary can be created using **dict()** function.
List is mutable i.e we can make any changes in list.	Tuple is immutable i.e we can not make any changes in tuple	Set is mutable i.e we can make any changes in set. But elements are not duplicated.	Dictionary is mutable. But Keys are not duplicated.
List is ordered	Tuple is ordered	Set is unordered	Dictionary is ordered
Creating an empty list l=[]	Creating an empty Tuple t=()	Creating a set a=set() b=set(a)	Creating an empty dictionary d={}

Week 3: Python Programming Fundamentals

Goals:

- Classify conditions and branching by identifying structured scenarios with outputs.
- Apply your knowledge of conditions and branching by using JupyterLab.
- Use visual examples to understand for loops and while loops.
- Apply your knowledge of loops using JupyterLab.
- Explain what functions do.
- Build a function using inputs and outputs.
- Apply your knowledge of functions using JupyterLab.
- Demonstrate the use of exception handling using JupyterLab.
- Explain objects and classes by identifying data types and creating a class.

CONDITIONS

Comparison operations compare some value or operand (numbers or strings). Then, based on some *condition*, a boolean is produced.

note: string comparisons are case-sensitive.

Comparison Operations:

- == (compares two values for equality)
- != (compares two values for inequality)
- <> (compares two values for inequality)
- >
- <
- >=
- <=

In [ ]:
```
# note the error
5=6
```

In [ ]:
```
# comparison test

a = 6
a != 7
```

In [ ]:
```
"BA" > "AB"
# the first letter takes precedence in ordering.
# B = 66, A = 65 in ASCII
```

In [ ]:
```
# string test

"Michael" == "Janet"
```

BRANCHING

Branching allows us to run different statements for different input. i.e., if one statement is true, then execute this action. If it isn't true, skip and go to next action.

Branching Operations:

- if (if statement is false, skip the action)
- else (if state is false, run the "else" action instead)
- elif (else-if. allows to check for additional condition if the previous statement is false)

In [ ]:
```
# if statement example

age = 17
if (age > 18):
    print("you can enter")
print("move along")
```

In [ ]:
```
# else statement example

age = 17
if (age > 18):
```

```
    print("you can go to AC/DC concert")
else:
    print("you can go to Meat Loaf concert instead")
print("move along")
```

In [ ]:

```
# elif statement example

age = 18
if (age > 18):
    print("you can go to AC/DC conert")
elif (age == 18):
    print("you can go to Pink Floyd concert")
else:
    print("you can go to Meat Loaf concert")
print("move along")
```

Logic operators take boolean values and produce different boolean values.

Logic Operators:

- and (returns True if all operators are true)
- or (returns True if any operator is true)
- not (returns True if the operators are false)

In [ ]:

```
# OR example

album_year = 1990
if (album_year <1980) or (album_year>1989):
    print("the album wasn't made in the 80s")
else:
    print("the album was made in the 80s")
```

In [ ]:

```
# AND example

album_year = 1983
if (album_year > 1979) and (album_year <1990):
    print("this album was made in the 80s")
else:
    print("this album was not made in the 80s")
```

In [ ]:

```
# NOT example

album_year = 1983
if not (album_year == 1984):
    print("album was not made in 1984")
else:
    print("album was made in 1984")
```

In [ ]:

```
# final example

album_year = 1971
if (album_year < 1980) or (album_year == 1991) or (album_year == 1993):
```

```
    print("this album came out in year %d" %album_year)

# note: the %d is for formatting numbers. %s is for strings.
# they are used when you want to include the value of an expression into a string, with
```

LOOPS

range(N):

- this function outputs an ordered sequence as a list I.
- If the input is a positive integer, the output is a sequence.
- This sequence contains the same number of elements as the input value, but starts at 0.

range(N, M):

- in this form, the function outputs an ordered sequence as a list that starts at N and ends at M-1.

Note: Python 3 doesn't generate the list explicitly like Python 2 does.

In [ ]:
```
range(-10)
```

In [ ]:
```
range(10, 15)
```

Loops:

- loops perform a task over and over, either for eternity, a set number of turns, or until some condition is met.

For Loops:

- for i in range(N, M): list[i] = __
- i is the index of the list that is output by the range function.
- The action is looped over from index N to index M-1, incremented by 1 each loop.

In [25]:
```
# for loop example

for x in range(0,3):
    print(x)
```

```
0
1
2
```

In [26]:
```
# for loop example where you don't use indeces, but the variable

squares = ['red', 'yellow', 'green']
for square in squares:
    print(square)
```

```
red
```

```
yellow
green
```

In [27]:
```
# for loop example:
for x in ['a', 'b', 'c']:
    print(x + 'Z')
```

```
aZ
bZ
cZ
```

In [28]:
```
# for loop example using both indeces and variables

squares = ['red', 'yellow','blue']
for i,square in enumerate(squares):
    print("variable: " + str(square), "index: " + str(i), sep = '\n')
```

```
variable: red
index: 0
variable: yellow
index: 1
variable: blue
index: 2
```

In [29]:
```
# for loop example

dates = [1982, 1980, 1973, 1999]
N = len(dates)
for i in range(N):
    print(dates[i])
```

```
1982
1980
1973
1999
```

In [30]:
```
# for loop used to change colors in a list to all white

squares = ['red', 'yellow', 'green', 'purple', 'blue']
for i, square in enumerate(squares):
    print("before square ", i, "was", squares[i]) # print before, index, old color
    squares[i] = 'white' # set square value to white
    print('after square ', i, 'is now', squares[i]) # print after, index, new color
```

```
before square  0 was red
after square  0 is now white
before square  1 was yellow
after square  1 is now white
before square  2 was green
after square  2 is now white
before square  3 was purple
after square  3 is now white
before square  4 was blue
after square  4 is now white
```

In [31]:
```
# for loop example

for i in range(-5, 6): # recall the range goes up to M-1
    print(i)
```

```
-5
-4
-3
-2
-1
0
1
2
3
4
5
```

While Loops:

- Similar to for loops, but instead of running a set number of times, while loops run only while a condition is met.
- while(variable[i] == 'condition'): action, i = i+1

In [32]:

```
# while loop example

squares = ['orange', 'orange', 'orange', 'purple', 'orange', 'blue']

# create new squares list from only orange squares
new_squares =[]

# index initialization
i = 0

while(squares[i] == 'orange'): # while squares is orange
    new_squares.append(squares[i]) # append the orange square to new_square
    i = i+1 # increase the index
print(new_squares) # note the loop stopped at purple, so we didn't get the 4th square
```

```
['orange', 'orange', 'orange']
```

In [33]:

```
# while loop example

dates = [1982, 1980, 1970, 1973, 2004]
i = 0
year = dates[0]

while(year !=1973): # run loop until you hit 1973
    print(year) # print year
    i = i+1 # increment index
    year = dates[i]
print("it took ", i, "repetitions to leave the loop")
```

```
1982
1980
1970
it took  3 repetitions to leave the loop
```

In [34]:

```
# while loop example

PlaylistRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 9]
i = 0
rating = PlaylistRatings[0]
```

```
while(i < len(PlaylistRatings) and (rating >=6)):
    rating = PlaylistRatings[i]
    print(rating)
    i = i+1
```

```
10
9.5
10
8
7.5
5
```

FUNCTIONS

Functions:

- Functions take some input, then produce some output or change.
- It's a reusable bit of code.
- You can use built-in functions, other people's functions, or define your own.
- def functionName(input): (output action)

Python's Built-in Functions:

- you don't need to know what tasks these functions perform or how they do it necessarily. You can just call on the function to complete a task.

Common Python Built-in Functions:

- print() : takes the input and prints it to your screen
- len() : takes in an input of types sequence (strings, lists, etc) or collection (dictionary, set, etc) and returns the length of that sequence or collection
- sum() : takes in an iterable input (tuple, list, etc) and returns the total of all the elements.
- sorted() : takes in a list or tuple and returns a new list that contains the old list's values, but sorted in ascending order.

note: recall that .sort() is a list method. Methods are similar to functions, but have to be called on an object. .sort() will not create a new list (but sort the old one instead), and sorted() will create a new sorted list (and leave the old one alone).

PYTHON QUICK REFERENCE SHEET

Here is a link to the Python Quick Reference Sheet.

In [35]:

```
# len example

album_ratings = [10, 8.5, 9.5, 7, 7, 9.5, 9, 9.5]
length = len(album_ratings)
display(length)
```

```
8
```

In [36]:
```
# sum example

album_ratings = [10, 8.5, 9.5, 7, 7, 9.5, 9, 9.5]
sum_album = sum(album_ratings)
display(sum_album)
```

```
70.0
```

In [37]:
```
# sorted example

album_ratings = [10, 8.5, 9.5, 7, 7, 9.5, 9, 9.5]
sorted_album_rating = sorted(album_ratings)
display(sorted_album_rating)

# note that this returns a new list that is sorted.
# the old list album_ratings remains unchanged.
```

```
[7, 7, 8.5, 9, 9.5, 9.5, 9.5, 10]
```

In [38]:
```
# sort method comparison

album_ratings = [10, 8.5, 9.5, 7, 7, 9.5, 9, 9.5]
album_ratings.sort()
display(album_ratings)

# note that no new list is created. The old list is rearranged.
# recall that lists are mutable.
```

```
[7, 7, 8.5, 9, 9.5, 9.5, 9.5, 10]
```

Creating Functions:

- Functions blocks begin def followed by the function name and parentheses ().
- There are input parameters or arguments that should be placed within these parentheses.
- You can also define parameters inside these parentheses.
- There is a body within every function that starts with a colon (:) and is indented.
- You can also place documentation before the body.
- The statement return exits a function, optionally passing back a value.
- at this point, you can call the function elsewhere in your program.
- it is good practice to include a comment explaining the purpose of the function.

note: not all functions have a return statement, in which case a "none" object will be returned.

note: python doesn't allow a function to have an empty body.

In [39]:
```
# example function

# create function
def add1(a):
    # add 1 to the input
```

```
    b = a+1
    print(a, "plus 1 equals", b)
    return b

# call function
print(add1(5))
```

```
5 plus 1 equals 6
6
```

In [40]:

```
# multiple parameters example

def multiply(a,b):
    # multiply parameters together
    c = a*b
    return c

# call function
print(multiply(3,5))

# note: if you were to make a string one of the parameters, the output would be that st
# since in python, multiplication symbol * can also mean repeat sequence.
```

```
15
```

In [41]:

```
# none statement example

def NoWork():
    pass

print(NoWork())
```

```
None
```

In [42]:

```
# function with loop example
Stuff = [10, 8.5, 9.35]

def printStuff(Stuff):
    # list the album number and its score
    for i,s in enumerate(Stuff):
        print("album", i, "rating is", s)

# call function
display(printStuff(Stuff))
```

```
album 0 rating is 10
album 1 rating is 8.5
album 2 rating is 9.35
None
```

In [43]:

```
# function loop example

def ArtistNames(*names):
    for name in names:
        print(name)

# call function
print(ArtistNames("MJ", "NSYNC", "AC/DC"))
```

```
MJ
NSYNC
AC/DC
None
```

Scope:

- the scope of a variable is the part of a function where that variable can be accessed.
- Global Scope: variables that are defined outside of any function and can be accessed anywhere. You can define a variable as global with keyword *global*.
- Local Scope: variables that exist only within the scope of a function.

note: if a function is not defined within a function, python will check those in the global scope

note: variables inside the global scope can have the same name as those within the local scope with no conflict. When you print in the global scope, the global variable is called. Likewise when printing within the local scope, the local variable is called.

In [44]:
```
# global scope example

def AddDC(y):
    y = y +"DC" # local scope
    print(y)
    return(y)

# call function
x = "AC" # global scope
z = AddDC(x) # global scope
print(z)
```

```
ACDC
ACDC
```

In [45]:
```
# local scope example

def Thriller():
    Date = 1982
    return Date

# call function
Thriller()

# Date < - if you try to call date outside of the function, an error occurs.
```

Out[45]: 1982

In [46]:
```
# name example
def Thriller():
    Date = 1982
    return Date

# call function
Date = 2017 # this date is a different variable to the one inside the function
```

```
Thriller() # returns 1982
print(Date) # returns 2017
```

```
2017
```

In [47]:

```
# python pulls from global scope example

def ACDC(y):
    print(Rating)
    return(Rating+y)

Rating = 9 # global scope variable is pulled by function

# call function

Z = ACDC(1)
```

```
9
```

In [48]:

```
# declaring a global variable example

def PinkFloyd():
    global ClaimedSales
    ClaimedSales = '45 million'
    return ClaimedSales

# call function
print(PinkFloyd()) # 45 million

# call global function
print("global function " + ClaimedSales)
```

```
45 million
global function 45 million
```

In [49]:

```
# function with if statements

def Equation(a,b):
    c = a+b+2*a*b-1
    print("value of c: " + str(c))
    if(c < 0):
        c = 0
    else:
        c = 5
    return(c)

# call function
Equation(3,2)
```

```
value of c: 16
```

Out[49]:

```
5
```

In [50]:

```
# function example

def type_of_album(artist, album, year_released):
    # return what sort of album based off year of release
    print(artist, album, year_released)
    if (year_released > 1980):
```

```
        return "Modern"
    else:
        return "Oldie"

# call the function
x = type_of_album("Michael Jackson", "Thriller", 1980)
print("this album is: " + x)
```

```
Michael Jackson Thriller 1980
this album is: Oldie
```

In [51]:

```
# function example

def PrintList(the_list):
    for element in the_list:
        print(element)

# call function
PrintList(['a', 2, 'anthony', 'xyz'])
```

```
a
2
anthony
xyz
```

In [52]:

```
# function with dictionary example

def printDictionary(**args):
    for key in args:
        print(key + " : " + args[key])
# *args allows you to pass multiple arguments or keywords into a function
# you don't need to use *args, it's just the asterisk and a name.
# * acts as the "unpacking operator"
printDictionary(Country='Canada',Province='Ontario',City='Toronto')
```

```
Country : Canada
Province : Ontario
City : Toronto
```

In [53]:

```
# function with unknown number of arguments

def printAll(*args): # All the arguments are 'packed' into args which can be treated li
    print("No of arguments:", len(args))
    for argument in args:
        print(argument)

#printAll with 3 arguments
printAll('Horsefeather','Adonis','Bone', '\n')
#printAll with 4 arguments
printAll('Sidecar','Long Island','Mudslide','Carriage')
```

```
No of arguments: 4
Horsefeather
Adonis
Bone


No of arguments: 4
Sidecar
```

```
Long Island
Mudslide
Carriage
```

EXCEPTION HANDLING

An exception is an error that occurs during the execution of code. This error causes the code to raise an exception and if not prepared to handle it will halt the execution of the code.

For example, when entering numbers into a string-only input field and a "please enter letters" error pops up. That exception would have been coded explicitly to handle such an event.

Try... Except Statement:

- This statement will attempt to execute the code in the 'try' block, but if an error occurs it will exit that block and search for the exception that matches the error.
- This will allow us to continue the execution of the program, even if there is an exception.
- Once the correct exception is found, that code block will be executed.
- You can add an extra except statement to handle all other errors, but this may cause more errors down the line.
- You can add an else statement to provide a notification that the code block was executed properly.
- You can add a finally statement to conclude the try...except statement, regardless of the result.

In [54]:
```
# ZeroDivisionError example

1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-54-22f81243a163> in <module>
      1 # ZeroDivisionError example
      2
----> 3 1/0

ZeroDivisionError: division by zero
```

In [ ]:
```
# NameError example - using an undefined variable

y = a+5
```

In [ ]:
```
# IndexError example - tried to access data that doesn't exist

a = [1,2, 3]
a[10]
```

In [ ]:
```
# try...except simple example

a = 1

try:
    b = int(input("please enter a number to divide a by:"))
```

```
    a = a/b
    print("success a =", a)
except:
    print('there was an error')
```

In [ ]:

```
# try...except specific example
# a specific try...except will allow you to catch specific exceptions
# and execute certain code depending on the exception.

a = 1

try:
    b = int(input("please enter a number to divide a by:"))
    integer = a/b
    print("success, a =", a)
except ZeroDivisionError:
    print("You cannot divide by zero, please select a different number")
except ValueError:
    print('you did not provide a number')
except: # general except statement
    print('something else went wrong')
```

In [ ]:

```
# try...except...else...finally example

a = 1

try:
    b = int(input("please enter a number to divide the integer by:"))
    integer = a/b
    # removed print statement here
except ZeroDivisionError:
    print("You cannot divide by zero, please select a different number")
except ValueError:
    print('you did not provide a number')
except: # general except statement
    print('something else went wrong')
# the else statement allows us to print the value of integer only if there is no error
else:
    print('success! a =', a)
# the finally statement lets the user know we finished processing their input
finally:
    print('processing complete')
```

OBJECTS

There are many different data types in Python and each is an object.

Every Object has:

- a type
- an internal data representation (blueprint)
- a set of procedures for interacting with the object (methods)

Objects:

- An object is an instance of a particular type. You can have multiple objects of any given type at any time, like 5 integers or 4 lists.
- You can find the type of an object by using command type()

In [ ]:
```
# .reverse example
L = [1 ,2 ,3 , 4, 5]
L.reverse()
print(L)
```

CLASSES

Classes:

- Classes provide a means of bundling data and functionality together.
- When you create a new class, you create a new type of object, which allows for new instances of that type to be made.
- Each class instance can have data attributes attached to it for maintaining its state.
- Class instances can also have their own methods (as defined by the class) for modifying its state.

Creating a class:

- class Name_of_Class (object):
- **init** is a special method or constructor used to initialize data attributes.

*note: init has two underscores on each side.

Methods:

- a class or type's methods are funcitons that every instance of that class provides.
- methods are how you can interact with an object or change an object.
- methods are called by adding a period and the method name() to the object.
- you can add a method to a class by defining it within the class definition.
- .sort()
- .reverse()
- you can see the object's data attributes by using the function dir(). The attributes with underscores are meant for internal use.

In [ ]:
```
# creating circle class

# import matplotlib library, which draws the objects
import matplotlib.pyplot as plt
%matplotlib inline
# puts the plots next to the cells with their code

# define the class
class Circle(object):
    # define the data attributes used to initialize each instance of this class
    # self parameter refers to the newly created instance of this class
    # radius and color each refer to their respective data attributes
```

```
    # __init_ is a special method / constructor used to initialize data attribute param

    # CONSTRUCTOR
    # you can add default parameters to the class as below
    def __init__(self, radius = 10, color = 'yellow'):
        self.radius = radius;
        self.color = color;

    # METHOD
    # define a method to adjust the radius by adding r
    def add_radius(self, r):
        self.radius = self.radius+r
        return (self.radius)

    # METHOD
    # define a new method that draws an image of the object
    def drawCircle(self):
        plt.gca().add_patch(plt.Circle((0, 0), radius=self.radius, fc=self.color))
        plt.axis('scaled')
        plt.show()
```

In [ ]:

```
# create an object from this class by introducing a variable
RedCircle = Circle(10, 'red')

# view object's attributes
RedCircle.radius
RedCircle.color

# change object's attributes
RedCircle.radius = 8
RedCircle.radius
```

In [ ]:

```
# draw RedCircle
RedCircle.drawCircle()
```

In [55]:

```
# decrease size of redcircle, then draw again
RedCircle.add_radius(-3)
RedCircle.drawCircle()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-55-97d585458665> in <module>
      1 # decrease size of redcircle, then draw again
----> 2 RedCircle.add_radius(-3)
      3 RedCircle.drawCircle()

NameError: name 'RedCircle' is not defined
```

In [ ]:

```
# create new circle object and adjust it with new methods
BlueCircle = Circle(2, 'blue')

# use method to adjust radius size
BlueCircle.add_radius(3)

# view new radius size
BlueCircle.radius
```

In [ ]:
```
# use dir function to get a list of the object's data attributes
dir(BlueCircle)
```

In [ ]:
```
# find object's color attribute
BlueCircle.color
```

In [ ]:
```
# draw bluecircle
BlueCircle.drawCircle()
```

In [ ]:
```
class Rectangle(object):

    # Constructor
    def __init__(self, width=2, height=3, color='r'):
        self.height = height
        self.width = width
        self.color = color

    # Method
    def drawRectangle(self):
        plt.gca().add_patch(plt.Rectangle((0, 0), self.width, self.height ,fc=self.colo
        plt.axis('scaled')
        plt.show()
```

In [ ]:
```
# create rectangle object

BlueRectangle = Rectangle(8, 3, 'blue')

# view info
print(BlueRectangle.height, BlueRectangle.width, BlueRectangle.color)
```

In [ ]:
```
# see attributes
dir(BlueRectangle)
```

In [ ]:
```
# draw object
BlueRectangle.drawRectangle()
```

Exercise: create a class that analyzes text, removes punctuation, lowercases all text, and then creates a dictionary that counts unique words and each word's wordcount.

In [ ]:
```
class analysedText(object):

    def __init__ (self, text):
        # remove punctuation
        formattedText = text.replace('.','').replace('!','').replace('?','').replace(',

        # make text lowercase
        formattedText = formattedText.lower()

        self.fmtText = formattedText
```

```
    def freqAll(self):
        # split text into words
        wordList = self.fmtText.split(' ')

        # Create dictionary
        freqMap = {}
        for word in set(wordList): # use set to remove duplicates in list
            freqMap[word] = wordList.count(word)

        return freqMap

    def freqOf(self,word):
        # get frequency map
        freqDict = self.freqAll()

        if word in freqDict:
            return freqDict[word]
        else:
            return 0
```

In [ ]:

```
# test the above code to see if it works
import sys

sampleMap = {'eirmod': 1,'sed': 1, 'amet': 2, 'diam': 5, 'consetetur': 1, 'labore': 1,

def testMsg(passed):
    if passed:
       return 'Test Passed'
    else :
       return 'Test Failed'

print("Constructor: ")
try:
    samplePassage = analysedText("Lorem ipsum dolor! diam amet, consetetur Lorem magna.
    print(testMsg(samplePassage.fmtText == "lorem ipsum dolor diam amet consetetur lore
except:
    print("Error detected. Recheck your function " )
print("freqAll: ")
try:
    wordMap = samplePassage.freqAll()
    print(testMsg(wordMap==sampleMap))
except:
    print("Error detected. Recheck your function " )
print("freqOf: ")
try:
    passed = True
    for word in sampleMap:
        if samplePassage.freqOf(word) != sampleMap[word]:
            passed = False
            break
    print(testMsg(passed))

except:
    print("Error detected. Recheck your function  " )
```

In [ ]:

```
# try the class
```

```
# create new object of this class
text = analysedText("Society excited by cottage private an it esteems. Fully begin on b

# view attributes
dir(text)
```

In [ ]:
```
# test dictionary method

text.freqAll()
```

In [ ]:
```
# test frequency method

text.freqOf('to')
```

Week 4: Working with Data in Python

Goals:

- Demonstrate reading files with an open function using JupyterLab.
- Demonstrate writing files with an open function using JupyterLab.
- Explain how Pandas use data frames.
- Use pandas for library and data analysis by using commands.
- Create and project in Watson Studio and load a notebook.
- Demonstrate how to use NumPy to create one-dimensional arrays using JupyterLab.
- Demonstrate how to communicate with Watson Speech to Text and Language Translator through the use of APIs.
- Demonstrate how to use NumPy to create multi-dimensional arrays using JupyterLab.
- Define the difference between APIs and REST APIs.
- Explain how APIs receive and send information.

READING FILES WITH OPEN

Open:

- Python's built in open() function can be used to create a file object and obtain data from a .txt file.
- Once the file has been opened, the .read() method can be used to read the data within the file.
- you can create a file object that refers to the file being opened.

- syntax: open(file_path, mode)
- example_file_object = open("/resources/data/exampledata.txt", 'w')
- it is good practice to open a file using a with statement, as it will automatically close the file after running everything within the code block.
- you can output every line as an element in a list using .readlines() method.
- you can read just the first N characters of the first line of the file using the .readline(N) method. If you call on it again, it will read the second line, and so forth. You can use a loop to capture all lines individually.

note: if you call only half of the characters in line 1, then next time you call .readline(), it will pull from the rest of line 1 - not line 2.

- .tell() returns the current position within the file in bytes
- . seek(offset, from) changes the position by 'offset' bytes with respect to 'from'. From can take values 0, 1, 2 corresponding to beginning, relative to current position, and end.
- .truncate() resizes the file to a given number of bytes

Modes:

- discover the mode of an object by using the data attribute mode: file1.mode
- 'r' : open text file for reading. Positioned at start of file.
- 'w' : open text file for writing. Positioned at start of file.
- 'a' : open text file for writing. Positioned at end of file.
- 'r+' : reading and writing. Cannot truncate file.
- 'w+' : writing and reading. Truncates the file.
- 'a+' : appending and reading. Creates new file if none exist.

note: to work on an existing file, it's best to use r+ and a+.

note: when using r+, it can be useful to add .truncate() at end of the data, as this will reduce the file to your data and delete everything else that follows.

with statement example:

with open("example.txt", 'r') as example_file:

```
    file_stuff = example_file.read() # store file info in variable

    print(file_stuff)
```

print(example_file.closed) # with statement closes the file

print(file_stuff)

readlines() example:

with open('example.txt', 'r') as file1:

```
    file_stuff2 = file1.readlines()

    print(file_stuff2)
```

print(file1.closed)

In [ ]:
```
# lab practice

# download the data
import urllib.request
url = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS
filename = 'Example1.txt'
urllib.request.urlretrieve(url, filename)
```

In [ ]:
```
# read file

example1 = 'Example1.txt'
file1 = open(example1, 'r')
```

In [ ]:
```
# print path of file

file1.name
```

In [ ]:
```
# print mode of file

file1.mode
```

In [ ]:
```
# read file

FileContent = file1.read()
FileContent
```

In [ ]:
```
# print file

print(FileContent)
```

In [ ]:
```
# type of file

type(FileContent)
```

In [ ]:
```
# close file after finishing with it

file1.close()
```

In [ ]:
```
# use a with statement to open the file

with open(example1, 'r') as file1:
    FileContent = file1.read()
    print(FileContent)
# the file should automatically close at the end of the with statement, after the inden
```

In [ ]:
```
# verify if file closed

file1.closed
```

In [ ]:
```
# see content of file

print(FileContent)
```

In [ ]:
```
# read only first 4 characters

with open(example1, 'r') as file1:
    print(file1.read(4))
```

In [ ]:
```
# read different amount of characters

with open(example1, "r") as file1:
    print(file1.read(4))
    print(file1.read(4))
    print(file1.read(7))
    print(file1.read(15))

# note the characters are all pulled from line 1 until completely done with line 1
```

In [ ]:
```
# Read certain amount of characters

with open(example1, "r") as file1:
    print(file1.read(16))
    print(file1.read(5))
    print(file1.read(9))
```

In [ ]:
```
# read one line with readline():

with open(example1, 'r') as file1:
    print('first line: ' + file1.readline())
```

In [ ]:
```
# use readline for specific numbers of characters

with open(example1, 'r') as file1:
```

```
    print(file1.readline(20)) # doesn't read passed end of line
    print(file1.read(20)) # does read passed the line
```

In [ ]:
```
# use loop to iterate through lines

with open(example1, 'r') as file1:
    i = 0
    for line in file1:
        print('iteration', str(i), ': ', line)
        i = i + 1
```

In [ ]:
```
# use readlines() to save text file to list
with open(example1, 'r') as file1:
    File_List = file1.readlines()

# print first line
File_List[0]
```

WRITING FILES WITH OPEN

- we can use .open() function to get a file object to create a text file
- we can then use the .write(x) method to write 'x' data to that file
- use mode 'w'
- you can use a loop to write multiple lines (from a list) to a file.
- you can copy one file to a new file

note: if you create a new file object with the same file path as another object, that file will be overwritten.

- if you want to write a file without losing any of the existing data, use mode 'a' to append the info to the end of the file, rather than overwriting it.

lab example

Write line to file

exmp2 = '/resources/data/Example2.txt'

with open(exmp2, 'w') as writefile:

```
    writefile.write("This is line A")
```

Read file

with open(exmp2, 'r') as testwritefile: print(testwritefile.read())

Write lines to file

with open(exmp2, 'w') as writefile:

```
    writefile.write("This is line A\n")

    writefile.write("This is line B\n")
```

Sample list of text

Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]

write the strings in the list to the text file

with open('Example2.txt', 'w') as writefile:

```
    for line in Lines:

        print(line)

        writefile.write(line)
```

Verify if writing to file is successfully executed

with open('Example2.txt', 'r') as testwritefile:

```
    print(testwritefile.read())
```

note: setting the mode to 'w' overwrites all existing data within the file

overwrite

with open('Example2.txt', 'w') as writefile:

```
    writefile.write("Overwrite\n")
```

with open('Example2.txt', 'r') as testwritefile:

```
    print(testwritefile.read())
```

Write a new line to text file

with open('Example2.txt', 'a') as testwritefile:

```
    testwritefile.write("This is line C\n")

    testwritefile.write("This is line D\n")

    testwritefile.write("This is line E\n")
```

Verify if the new line is in the text file

with open('Example2.txt', 'r') as testwritefile:

```
    print(testwritefile.read())
```

try a+ mode

with open('Example2.txt', 'a+') as testwritefile:

```
    testwritefile.write("This is line E\n")

    print(testwritefile.read())
```

In [ ]:

```
with open('Example2.txt', 'a+') as testwritefile:
    print("Initial Location: {}".format(testwritefile.tell()))

    data = testwritefile.read()
    if (not data):  #empty strings return false in python
            print('Read nothing')
    else:
            print(testwritefile.read())

    testwritefile.seek(0,0) # move 0 bytes from beginning.

    print("\nNew Location : {}".format(testwritefile.tell()))
    data = testwritefile.read()
    if (not data):
            print('Read nothing')
    else:
            print(data)

    print("Location after read: {}".format(testwritefile.tell()) )
```

In [ ]:

```
# try code block without truncate
with open('Example2.txt', 'r+') as testwritefile:
    data = testwritefile.readlines()
    testwritefile.seek(0,0) #write at beginning of file

    testwritefile.write("Line 1" + "\n")
    testwritefile.write("Line 2" + "\n")
    testwritefile.write("Line 3" + "\n")
    testwritefile.write("finished\n")
    #Uncomment the line below
    #testwritefile.truncate()
    testwritefile.seek(0,0)
    print(testwritefile.read())
```

In [56]:

```
# try code block with truncate

with open('Example2.txt', 'r+') as testwritefile:
    data = testwritefile.readlines()
    testwritefile.seek(0,0) #write at beginning of file

    testwritefile.write("Line 1" + "\n")
    testwritefile.write("Line 2" + "\n")
    testwritefile.write("Line 3" + "\n")
    testwritefile.write("finished\n")
    #Uncomment the line below
    testwritefile.truncate()
    testwritefile.seek(0,0)
    print(testwritefile.read())
```

```
Line 1
```

```
Line 2
Line 3
finished
```

In [57]:

```
# Copy file to another

with open('Example2.txt','r') as readfile:
    with open('Example3.txt','w') as writefile:
          for line in readfile:
                writefile.write(line)

# Verify if the copy is successfully executed

with open('Example3.txt','r') as testwritefile:
    print(testwritefile.read())
```

```
Line 1
Line 2
Line 3
finished
```

exercise: fan club membership document is updated monthly, adding active members and removing inactive members.

remove each member with 'no' in active column. add these people to the exmem file.

In [58]:

```
#Run this prior to starting the exercise
from random import randint as rnd

memReg = 'members.txt'
exReg = 'inactive.txt'
fee =('yes','no')

def genFiles(current,old):
    with open(current,'w+') as writefile:
        writefile.write('Membership No  Date Joined  Active  \n')
        data = "{:^13}  {:<11}  {:<6}\n"

        for rowno in range(20):
            date = str(rnd(2015,2020))+ '-' + str(rnd(1,12))+'-'+str(rnd(1,25))
            writefile.write(data.format(rnd(10000,99999),date,fee[rnd(0,1)]))


    with open(old,'w+') as writefile:
        writefile.write('Membership No  Date Joined  Active  \n')
        data = "{:^13}  {:<11}  {:<6}\n"
        for rowno in range(3):
            date = str(rnd(2015,2020))+ '-' + str(rnd(1,12))+'-'+str(rnd(1,25))
            writefile.write(data.format(rnd(10000,99999),date,fee[1]))


genFiles(memReg,exReg)
```

In [59]:

```
# solution

def cleanFiles(currentMem,exMem):
```

```
    with open(currentMem,'r+') as writeFile:
        with open(exMem,'a+') as appendFile:
            #get the data
            writeFile.seek(0)
            members = writeFile.readlines()
            #remove header
            header = members[0]
            members.pop(0)

            inactive = [member for member in members if ('no' in member)]
            '''
            The above is the same as

            for member in active:
            if 'no' in member:
                inactive.append(member)
            '''
            #go to the beginning of the write file
            writeFile.seek(0)
            writeFile.write(header)
            for member in members:
                if (member in inactive):
                    appendFile.write(member)
                else:
                    writeFile.write(member)
            writeFile.truncate()

memReg = 'members.txt'
exReg = 'inactive.txt'
cleanFiles(memReg,exReg)

# code to help you see the files

headers = "Membership No  Date Joined  Active  \n"

with open(memReg,'r') as readFile:
    print("Active Members: \n\n")
    print(readFile.read())

with open(exReg,'r') as readFile:
    print("Inactive Members: \n\n")
    print(readFile.read())
```

```
Active Members:


Membership No  Date Joined  Active
    85923      2020-12-18   yes
    22407      2017-7-20    yes
    52692      2015-4-24    yes
    23554      2018-4-23    yes
    64892      2020-6-9     yes

Inactive Members:


Membership No  Date Joined  Active
    91138      2015-5-10    no
    59736      2019-9-9     no
    17823      2018-11-10   no
    89794      2017-2-25    no
    18665      2016-8-14    no
```

```
43439      2016-4-25    no
10670      2020-12-10   no
41952      2019-10-22   no
78064      2015-3-18    no
82007      2017-11-5    no
63997      2019-6-23    no
82106      2019-3-23    no
85375      2019-9-8     no
65261      2019-9-21    no
70990      2017-3-1     no
88983      2018-5-4     no
18863      2018-9-6     no
11733      2018-6-18    no
```

PANDAS

- Dependencies / Libraries are pre-written code to help solve problems.
- Pandas is a popular library for data analysis.
- You can import pandas in order to gain access to its many pre-built classes and functions.
- A dataframe is comprised of rows and columns.
- we can create a dataframe out of a dictionary, where the keys are the column labels and the values are the rows.
- You can also create a new dataframe out of just one column.

PANDAS: WORKING WITH AND SAVING DATA

Commonly used pandas functions:

- df = pandas.read_csv('file/path') : loads a file
- df.head() : examine the first five rows of the dataframe
- new_df = pandas.DataFrame(dict) : turns a dictionary into a dataframe
- df.ix[x,y]: access the x-1 row and y-1 column of the dataframe df.
- new_df = old_df[['column1', 'column2']] : create new dataframe from another dataframe's columns
- df['column1'].unique() : find the unique values of column1
- new_df = old_df[old_df['column1'] >= x] : create a new dataframe from another dataframe on the condition that the values in column1 >= x
- df1.to_csv('new_CSV_name.csv') : save the dataframe to a CSV file

.LOC AND .ILOC

These methods are very similar in what they do, but .iloc uses row/column indexes while .loc uses the row/column header names.

- You can use both .iloc and .loc to slice out bits of the dataframe that you want.

recall: when you slice [x1:x2, y1:y2] using indexes, the values slice up to x2-1 and y2-1

.iloc

- .iloc[] is a method that allows you access elements in the rows and columns of a dataframe using their indexes (starting at 0).
- .iloc[x,y]: x refers to the rows, y refers to the columns.

.loc

- .loc[] is a method that allows you to access elements in the rows and columns of a dataframe using their row/column names.
- .loc['x','y']: x refers to the row headers, y refers to the column headers.

In [60]:
```
# import library

import pandas as pd
```

In [61]:
```
# dependency needed to install a file:

!pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\orgil\appdata\local\programs\python
\python39\lib\site-packages (3.0.7)
Requirement already satisfied: et-xmlfile in c:\users\orgil\appdata\local\programs\pytho
n\python39\lib\site-packages (from openpyxl) (1.1.0)
```

In [62]:
```
# Lab

# read data from CSV File

csv_path = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDevel
df = pd.read_csv(csv_path)
```

In [63]:
```
# print first five rows of data

df.head()
```

Out[63]:

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	R
0	Michael Jackson	Thriller	1982	0:42:19	pop, rock, R&B	46.0	65	30-Nov-82	NaN	
1	AC/DC	Back in Black	1980	0:42:11	hard rock	26.1	50	25-Jul-80	NaN	

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	R
2	Pink Floyd	The Dark Side of the Moon	1973	0:42:49	progressive rock	24.2	45	01-Mar-73	NaN	
3	Whitney Houston	The Bodyguard	1992	0:57:44	R&B, soul, pop	27.4	44	17-Nov-92	Y	
4	Meat Loaf	Bat Out of Hell	1977	0:46:33	hard rock, progressive rock	20.6	43	21-Oct-77	NaN	

In [64]:
```
# Read data from Excel File and print the first five rows

xlsx_path = 'https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/Cognitiv

df = pd.read_excel(xlsx_path)
df.head()
```

Out[64]:

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack	R
0	Michael Jackson	Thriller	1982	00:42:19	pop, rock, R&B	46.0	65	1982-11-30	NaN	
1	AC/DC	Back in Black	1980	00:42:11	hard rock	26.1	50	1980-07-25	NaN	
2	Pink Floyd	The Dark Side of the Moon	1973	00:42:49	progressive rock	24.2	45	1973-03-01	NaN	
3	Whitney Houston	The Bodyguard	1992	00:57:44	R&B, soul, pop	27.4	44	1992-11-17	Y	
4	Meat Loaf	Bat Out of Hell	1977	00:46:33	hard rock, progressive rock	20.6	43	1977-10-21	NaN	

In [65]:
```
# access the column length

x = df[['Length']]
x # technically x is a new dataframe

# note that capitalization matters.
```

Out[65]:

	Length
0	00:42:19

	Length
1	00:42:11
2	00:42:49
3	00:57:44
4	00:46:33
5	00:43:08
6	01:15:54
7	00:40:01

In [66]:
```
# get the column as a series

x = df['Length'] # just one bracket set
x

# think of a pandas series as a 1-dimensional dataframe
```

Out[66]:
```
0    00:42:19
1    00:42:11
2    00:42:49
3    00:57:44
4    00:46:33
5    00:43:08
6    01:15:54
7    00:40:01
Name: Length, dtype: object
```

In [67]:
```
#check type

type(x)
```

Out[67]:
```
pandas.core.series.Series
```

In [68]:
```
# get access to multiple columns

y = df[['Artist', 'Length', 'Genre']]
y
```

Out[68]:

	Artist	Length	Genre
0	Michael Jackson	00:42:19	pop, rock, R&B
1	AC/DC	00:42:11	hard rock
2	Pink Floyd	00:42:49	progressive rock
3	Whitney Houston	00:57:44	R&B, soul, pop
4	Meat Loaf	00:46:33	hard rock, progressive rock
5	Eagles	00:43:08	rock, soft rock, folk rock
6	Bee Gees	01:15:54	disco

	Artist	Length	Genre
7	Fleetwood Mac	00:40:01	soft rock

In [69]:
```
# Access the value on the first row and the first column

df.iloc[0, 0]
```

Out[69]: 'Michael Jackson'

In [70]:
```
# Access the value on the second row and the first column

df.iloc[1,0]
```

Out[70]: 'AC/DC'

In [71]:
```
# Access the value on the first row and the third column

df.iloc[0,2]
```

Out[71]: 1982

In [72]:
```
# Access the value on the second row and the third column
df.iloc[1,2]
```

Out[72]: 1980

In [73]:
```
# Access the column using the name

df.loc[1, 'Artist']
```

Out[73]: 'AC/DC'

In [74]:
```
# Access the column using the name

df.loc[1, 'Artist']
```

Out[74]: 'AC/DC'

In [75]:
```
# Access the column using the name

df.loc[0, 'Released']
```

Out[75]: 1982

In [76]:
```
# Access the column using the name

df.loc[1, 'Released']
```

Out[76]: 1980

In [77]:
```
# Slicing the dataframe

df.iloc[0:2, 0:3]
```

Out[77]:

	Artist	Album	Released
0	Michael Jackson	Thriller	1982
1	AC/DC	Back in Black	1980

In [78]:
```
# Slicing the dataframe using name

df.loc[0:2, 'Artist':'Released']
```

Out[78]:

	Artist	Album	Released
0	Michael Jackson	Thriller	1982
1	AC/DC	Back in Black	1980
2	Pink Floyd	The Dark Side of the Moon	1973

In [79]:
```
# use the below list to convert the dataframe df's index and assign it to df_new
new_index=['a','b','c','d','e','f','g','h']

df_new = df
df_new
```

Out[79]:

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack
0	Michael Jackson	Thriller	1982	00:42:19	pop, rock, R&B	46.0	65	1982-11-30	NaN
1	AC/DC	Back in Black	1980	00:42:11	hard rock	26.1	50	1980-07-25	NaN
2	Pink Floyd	The Dark Side of the Moon	1973	00:42:49	progressive rock	24.2	45	1973-03-01	NaN
3	Whitney Houston	The Bodyguard	1992	00:57:44	R&B, soul, pop	27.4	44	1992-11-17	Y
4	Meat Loaf	Bat Out of Hell	1977	00:46:33	hard rock, progressive rock	20.6	43	1977-10-21	NaN
5	Eagles	Their Greatest Hits (1971-1975)	1976	00:43:08	rock, soft rock, folk rock	32.2	42	1976-02-17	NaN

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack
6	Bee Gees	Saturday Night Fever	1977	01:15:54	disco	20.6	40	1977-11-15	Y
7	Fleetwood Mac	Rumours	1977	00:40:01	soft rock	27.9	40	1977-02-04	NaN

In [80]:
```
# convert the index

df_new.index = new_index
df_new
```

Out[80]:

	Artist	Album	Released	Length	Genre	Music Recording Sales (millions)	Claimed Sales (millions)	Released.1	Soundtrack
a	Michael Jackson	Thriller	1982	00:42:19	pop, rock, R&B	46.0	65	1982-11-30	NaN
b	AC/DC	Back in Black	1980	00:42:11	hard rock	26.1	50	1980-07-25	NaN
c	Pink Floyd	The Dark Side of the Moon	1973	00:42:49	progressive rock	24.2	45	1973-03-01	NaN
d	Whitney Houston	The Bodyguard	1992	00:57:44	R&B, soul, pop	27.4	44	1992-11-17	Y
e	Meat Loaf	Bat Out of Hell	1977	00:46:33	hard rock, progressive rock	20.6	43	1977-10-21	NaN
f	Eagles	Their Greatest Hits (1971-1975)	1976	00:43:08	rock, soft rock, folk rock	32.2	42	1976-02-17	NaN
g	Bee Gees	Saturday Night Fever	1977	01:15:54	disco	20.6	40	1977-11-15	Y
h	Fleetwood Mac	Rumours	1977	00:40:01	soft rock	27.9	40	1977-02-04	NaN

In [81]:
```
# view the table using the new index

df_new.loc['a':'d', 'Artist']
```

Out[81]:
```
a    Michael Jackson
b               AC/DC
c         Pink Floyd
d    Whitney Houston
Name: Artist, dtype: object
```

1-D NUMPY

- Numpy is a library for scientific computing.
- Numpy is the basis of pandas.
- Numpy code is generally much faster at computing problems than solving the same problems without numpy.

In [82]:
```
# import library

import numpy as np
```

The Basics and Array Creation:

- a Numpy array (ND array) is similarto a python list.
- It is normally fixed in size and each element is the same type.

See below for some common functions and methods:

In [83]:
```
# get array
a = np.array([1,2,3,4,5])

# check type
type(a)
```

Out[83]:
```
numpy.ndarray
```

In [84]:
```
# check data type of array
a.dtype
```

Out[84]:
```
dtype('int32')
```

In [85]:
```
# check size of array
a.size
```

Out[85]:
```
5
```

In [86]:
```
# check number of dimensions in array
a.ndim
```

Out[86]:
```
1
```

In [87]:
```
# check shape (size of each dimension of an array)
a.shape
```

Out[87]: (5,)

Indexing and Slicing Arrays

- like lists and tuples, you can slice an array.

In [88]:
```
# get array
c = np.array([20, 1,2,3,4])

# change first element
c[0] = 100
c
```

Out[88]: array([100, 1, 2, 3, 4])

In [89]:
```
# select elements 1-3
d = c[1:4] # the index x:y goes from x to y-1
d
```

Out[89]: array([1, 2, 3])

In [90]:
```
# change values of multiple indexes
c[3:5] = 300, 400
c
```

Out[90]: array([100, 1, 2, 300, 400])

Basic Numpy Operations:

- vector addition and subtraction (1st component x, 2nd component y)
- vector multiplication with a scalar Z (each component is multiplied by Z)
- product of two numpy arrays (1st component of x and y is multiplied, 2nd component is multiplied, etc) : *hadamard product*
- dot product (1st components multiplied + 2nd components multiplied) : indicates how similar two vectors are. If two vectors are *perpendicular*, their dot products are zero.
- adding a constant to a numpy array (adding a constant Z to each element in an array) : *broadcasting*

In [91]:
```
# vector addition using lists
u = [1,0]
v = [0,1]
z = []

for n,m in zip(u,v):
    z.append(n+m)
z
```

Out[91]: [1, 1]

In [92]:
```
# vector addition using numpy
u = np.array([1,0])
v = np.array([0,1])
z = u+v
z
```

Out[92]: array([1, 1])

In [93]:
```
# vector subtraction using numpy
u = np.array([1,0])
v = np.array([0,-3])
z = u+v
z
```

Out[93]: array([1, -3])

In [94]:
```
# vector multiplication with scalar using numpy
y = np.array([1,2])
z = 2*y
z
```

Out[94]: array([2, 4])

In [95]:
```
# vector mulitplication with scalar without numpy
y = [1,2]
z = []
for n in y:
    z.append(2*n)
z
```

Out[95]: [2, 4]

In [96]:
```
# vector multiplication of two arrays with numpy
u = np.array([1,2])
v = np.array([3,4])
z = u*v
z
```

Out[96]: array([3, 8])

In [97]:
```
# vector multiplication of two arrays without numpy
u = [1,2]
v = [3,4]
z = []
for n,m in zip(u,v):
    z.append(n*m)
z
```

Out[97]: [3, 8]

In [98]:

```
# dot product with numpy
u = np.array([1,2])
v = np.array([3,1])
result = np.dot(u,v)
result
```

Out[98]: 5

In [99]:
```
# broadcasting with numpy
u = np.array([1,2,3,-1])
z = u + 1
z
```

Out[99]: array([2, 3, 4, 0])

Universal Functions:

- a function that operates on ND arrays.
- applies to numpy arrays

List of Functions:

- .mean()
- .max()
- .min()
- .std()
- .pi
- .sin()
- .linspace(start, end, num = z) (returns evenly-spaced numbers over specified interval z)

In [100…
```
# mean
a = np.array([1,2,3])
a.mean()
```

Out[100… 2.0

In [101…
```
# max
a.max()
```

Out[101… 3

In [102…
```
# pi
np.pi
```

Out[102… 3.141592653589793

In [103…
```
# numpy array in radians
x = np.array([0, np.pi/2, np.pi])
x
```

Out[103… array([0. , 1.57079633, 3.14159265])

In [104…
```
# sin
y = np.sin(x)
```

In [105…
```
# linspace
np.linspace(-2, 2, num = 5)
```

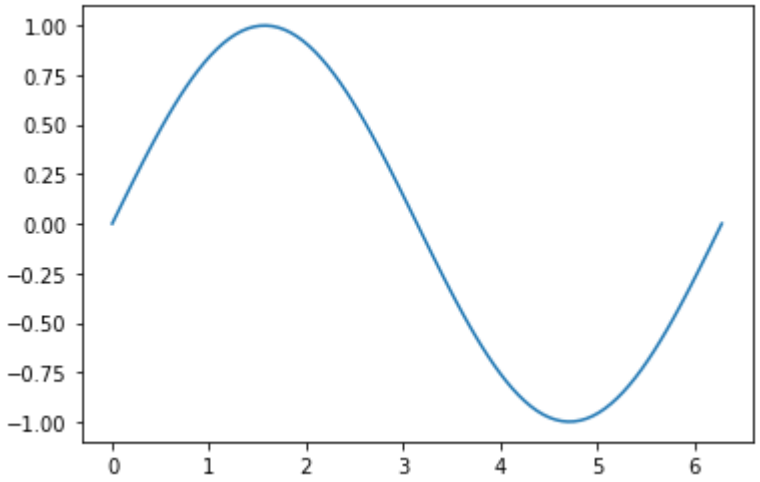
Out[105… array([-2., -1., 0., 1., 2.])

In [106…
```
np.linspace(-2, 2, num = 10)
```

Out[106… array([-2. , -1.55555556, -1.11111111, -0.66666667, -0.22222222,
0.22222222, 0.66666667, 1.11111111, 1.55555556, 2.])

In [107…
```
# generate 100 samples and graph

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(x,y)
```

Out[107… [<matplotlib.lines.Line2D at 0x21f6312d700>]



In [108…
```
# lab
# Import the libraries

import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

In [109…

```
# Plotting functions

def Plotvec1(u, z, v):

    ax = plt.axes()
    ax.arrow(0, 0, *u, head_width=0.05, color='r', head_length=0.1)
    plt.text(*(u + 0.1), 'u')

    ax.arrow(0, 0, *v, head_width=0.05, color='b', head_length=0.1)
    plt.text(*(v + 0.1), 'v')
    ax.arrow(0, 0, *z, head_width=0.05, head_length=0.1)
    plt.text(*(z + 0.1), 'z')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)

def Plotvec2(a,b):
    ax = plt.axes()
    ax.arrow(0, 0, *a, head_width=0.05, color ='r', head_length=0.1)
    plt.text(*(a + 0.1), 'a')
    ax.arrow(0, 0, *b, head_width=0.05, color ='b', head_length=0.1)
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)
    plt.xlim(-2, 2)
```

In [110…

```
# Create a python list

a = ["0", 1, "two", "3", 4]
```

In [111…

```
# Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: two
a[3]: 3
a[4]: 4
```

In [112…

```
# Create a numpy array

a = np.array([0, 1, 2, 3, 4])
a
```

Out[112… `array([0, 1, 2, 3, 4])`

In [113…

```
# Print each element

print("a[0]:", a[0])
print("a[1]:", a[1])
print("a[2]:", a[2])
print("a[3]:", a[3])
print("a[4]:", a[4])
```

```
a[0]: 0
a[1]: 1
a[2]: 2
a[3]: 3
a[4]: 4
```

In [114…
```
# Create numpy array

c = np.array([20, 1, 2, 3, 4])
c
```

Out[114… `array([20,  1,  2,  3,  4])`

In [115…
```
# Create the index list

select = [0, 2, 3]
```

In [116…
```
# Use List to select elements

d = c[select]
d
```

Out[116… `array([20,  2,  3])`

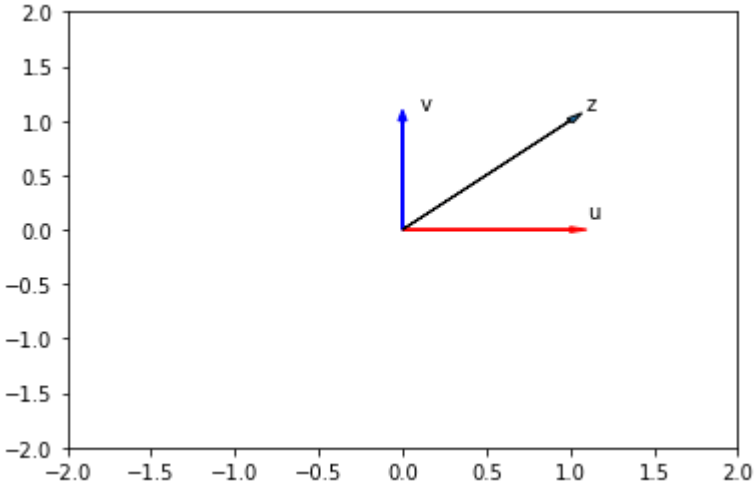
In [117…
```
# Assign the specified elements to new value

c[select] = 100000
c
```

Out[117… `array([100000,      1, 100000, 100000,      4])`

In [118…
```
# vector operations and plotting
u = np.array([1,0])
v = np.array([0,1])

# vector addition
z = u+v
z
```

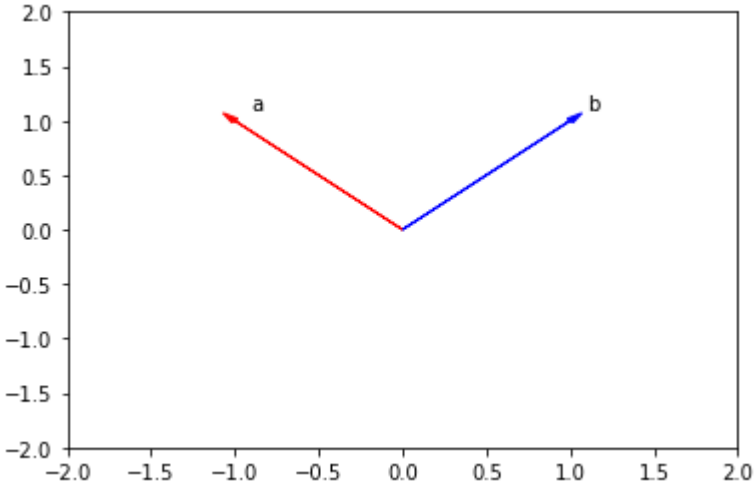
Out[118… `array([1, 1])`

In [119…
```
# plot arrays
Plotvec1(u,z,v)
```



In [120…
```
# dot product and graphing
a = np.array([-1,1])
b = np.array([1,1])
Plotvec2(a,b)
print('the dot product is', np.dot(a,b))
```

```
the dot product is 0
```



2-D Numpy

In [121…
```
# nested list
a = [[11,12, 13], [1,2,3], [31,32,33]]

# multidimensional array
A = np.array(a)
A
```

Out[121…
```
array([[11, 12, 13],
       [ 1,  2,  3],
       [31, 32, 33]])
```

In [122…
```
# retrieve number of axes / dimensions - referred to as RANK
# aka, the number of nested lists
# the first dimension is the outer brackets, the list holding 3 smaller lists
```

```
# the second dimension is the set of three lists.
A.ndim
```

Out[122… 2

In [123…
```
# the number of lists inside the 2nd dimension has to do with the shape
# (axis 0) first element corresponds to number of nested lists (3) or rows in matrix
# (axis 1) second element corresponds to number of elements within the lists (3) or col
A.shape
```

Out[123… (3, 3)

In [124…
```
# get total number of elements in matrix
A.size
```

Out[124… 9

In [125…
```
# access different elements of the array

# first element
A[0][0]
```

Out[125… 11

In [126…
```
# middle element
A[1][1]
```

Out[126… 2

In [127…
```
# bottom right element
A[2][2]
```

Out[127… 33

In [128…
```
# different formatting to access elements
A[1,1]
```

Out[128… 2

In [129…
```
A[2,1]
# 3rd row, 2nd column
```

Out[129… 32

In [130…
```
# slicing

# choose the rows you want, and the columns
A[0, 0:2]
```

Out[130… array([11, 12])

In [131…
```
# slicing
A[0:2, 0:2]
```

Out[131… array([[11, 12],
[1, 2]])

In [132…
```
# adding two multidimensional arrays
X = np.array([[1,0], [0,1]])
Y = np.array([[3,3], [4,4]])
X+Y
```

Out[132… array([[4, 3],
[4, 5]])

In [133…
```
# scalar multiplication
2*Y
```

Out[133… array([[6, 6],
[8, 8]])

In [134…
```
# matrix multiplication
X*Y
```

Out[134… array([[3, 0],
[0, 4]])

When performing matrix multiplication on multidimensional matrices / arrays, it's important to check if the number of columns of A is equal to the number of rows in B.

A*B

To compute this, you take the dot product of the ith row of A with the jth columns of B.

[first dot first] [first dot second]

[second dot first] [second dot second]

In [135…
```
# matrix multiplication using dot product
A = np.array([[0,1,1], [1, 0 , 1]])
B = np.array([[1,1], [1,1], [-1,1]])
C = np.dot(A,B)
C

# top left: 0*1 + 1*1 + 1*-1 = 0
# top right: 0*1 + 1*1 + 1*1 = 2
# bottom left: 1*1 + 0*1 + 1*-1 = 0
# bottom right: 1*1 + 0*1 + 1*1 = 2
```

Out[135… array([[0, 2],
[0, 2]])

In [136…
```
# lab
```

```
# Import the libraries

import numpy as np
import matplotlib.pyplot as plt
```

In [137…
```
# sine of C
np.sin(C)
```

Out[137…
```
array([[0.        , 0.90929743],
       [0.        , 0.90929743]])
```

In [138…
```
# matrix Z
Z = np.array([[1,4], [2,5], [3,6]])
Z
```

Out[138…
```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [139…
```
# transpose Z
Z.T
```

Out[139…
```
array([[1, 2, 3],
       [4, 5, 6]])
```

Week 5: APIs AND DATA COLLECTION

Goals:

- Explain how the URL Request Response HTTP protocol works.
- Explain the use of the HTTP protocol using the Requests Library method.
- Practice the basicsof webscraping in JupyterLab.
- Practice working with different file formats in JupyterLab.
- Work with different file formats

SIMPLE APIs

Application Program Interfaces (API):

- APIs allow two pieces of software talk to each other.
- For example, pandas is a set of software components. We use the pandas API to process the data by communicating with other software components.

In [140…
```
# using an API library
import pandas as pd

dict = {'a': [11, 21, 31], 'b': [12, 22, 33]}
df = pd.DataFrame(dict)
```

```
# call .head(), where the dataframe communicates with the API
df.head()
```

Out[140…

	a	b
0	11	12
1	21	22
2	31	33

In [141…

```
# call .mean(), where the dataframe communicates with the API to return the value
df.mean()
```

Out[141…

```
a    21.000000
b    22.333333
dtype: float64
```

Rest APIs:

- allow you to communicate via the internet, which allows for accessing more resources like storage, data, AI algorithms, etc.
- RE: Representational
- S: State
- T: Transfer

Common terms & synonyms:

- you / your code / client
- web service / resource
- input / request
- output / response
- client finds the web service via an 'endpoint'

HTTP Methods:

- A way of transmitting data over the internet.
- The client tells the Rest API's what to do via a request, which is usually communicated by a HTTP message. \
- This HTTP message usually contains a JSON file with instructions for the operation we want to perform.
- This operation is transmitted to the web service via the internet, where the operation is performed.
- Then the web service returns the response via another HTTP message, usually returned as a JSON file.

In [142…

```
# use py-coin-gecko client/wrapper for the Coin Gecko API
!pip install pycoingecko
from pycoingecko import CoinGeckoAPI
```

```
Requirement already satisfied: pycoingecko in c:\users\orgil\appdata\local\programs\pyth
```

```
on\python39\lib\site-packages (2.1.0)
Requirement already satisfied: requests in c:\users\orgil\appdata\local\programs\python
\python39\lib\site-packages (from pycoingecko) (2.25.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\orgil\appdata\local\pro
grams\python\python39\lib\site-packages (from requests->pycoingecko) (1.26.4)
Requirement already satisfied: idna<3,>=2.5 in c:\users\orgil\appdata\local\programs\pyt
hon\python39\lib\site-packages (from requests->pycoingecko) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in c:\users\orgil\appdata\local\program
s\python\python39\lib\site-packages (from requests->pycoingecko) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\orgil\appdata\local\progra
ms\python\python39\lib\site-packages (from requests->pycoingecko) (2020.12.5)
```

In [143…

```
# create client object
cg = CoinGeckoAPI()

# use a function to request data
bitcoin_data = cg.get_coin_market_chart_by_id(id = 'bitcoin', vs_currency = 'usd', days
bitcoin_data.keys()
# this retrieves info on bitcoin prices in USD from past 30 days.
# the response is returned as a JSON file expressed as a dictionary with nested lists
# the lists include price, market cap, total volumes (which contain unix timestamp and
```

Out[143… dict_keys(['prices', 'market_caps', 'total_volumes'])

In [144…

```
# retrieve only the price information as a dataframe
bitcoin_price_data = bitcoin_data['prices'] # nested list

# put the info into a dataframe for easier handling
data = pd.DataFrame(bitcoin_price_data, columns = ['Timestamp', 'Price'])
data
```

Out[144…

	Timestamp	Price
0	1621360975353	43509.903268
1	1621364548517	42815.404663
2	1621368144306	43676.079413
3	1621371807346	43372.839785
4	1621375334642	43053.724573
…	…	…
716	1623938554611	38882.667884
717	1623942186429	38956.044806
718	1623945669642	39151.981885
719	1623949296213	38474.000170
720	1623949823000	38435.866962

721 rows × 2 columns

In [145…

```
# convert timestamp column to more readable format with .to_datetime() function
data['Date'] = pd.to_datetime(data['Timestamp'], unit = 'ms')
data
```

Out[145...

	Timestamp	Price	Date
0	1621360975353	43509.903268	2021-05-18 18:02:55.353
1	1621364548517	42815.404663	2021-05-18 19:02:28.517
2	1621368144306	43676.079413	2021-05-18 20:02:24.306
3	1621371807346	43372.839785	2021-05-18 21:03:27.346
4	1621375334642	43053.724573	2021-05-18 22:02:14.642
...	...	...	...
716	1623938554611	38882.667884	2021-06-17 14:02:34.611
717	1623942186429	38956.044806	2021-06-17 15:03:06.429
718	1623945669642	39151.981885	2021-06-17 16:01:09.642
719	1623949296213	38474.000170	2021-06-17 17:01:36.213
720	1623949823000	38435.866962	2021-06-17 17:10:23.000

721 rows × 3 columns

In [146...

```
# create candlestick plot
candlestick_data = data.groupby(data.Date.dt.date).agg({'Price': ['min', 'max', 'first'
candlestick_data
# find the min, max, first, and last price of each day
```

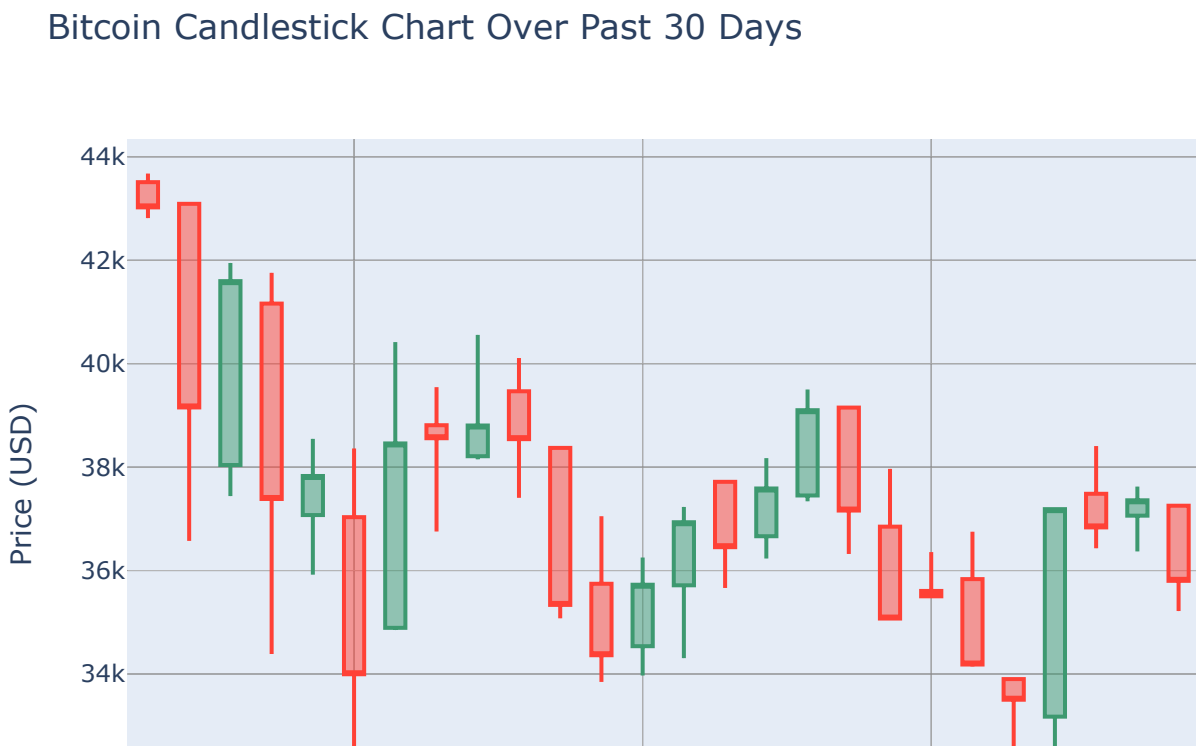
Out[146...

	Price			
	min	max	first	last
Date				
2021-05-18	42815.404663	43676.079413	43509.903268	43022.377837
2021-05-19	36573.548038	43091.041448	43091.041448	39154.208053
2021-05-20	37440.302997	41947.196711	38040.943346	41596.348073
2021-05-21	34386.609592	41757.137345	41161.937246	37384.019087
2021-05-22	35920.891747	38546.599099	37073.004560	37828.543905
2021-05-23	32458.122485	38360.641029	37032.672365	33995.040863
2021-05-24	34850.731787	40417.617244	34892.633299	38458.085643
2021-05-25	36755.333137	39546.087626	38810.164535	38558.947521
2021-05-26	38150.984549	40556.617553	38210.805218	38805.033942
2021-05-27	37406.487684	40109.301850	39466.642793	38541.079756
2021-05-28	35076.872522	38371.880070	38371.880070	35336.082856
2021-05-29	33846.607820	37050.556152	35744.457735	34363.022086
2021-05-30	33970.688099	36251.761333	34537.419075	35726.046487

	Price			
	min	max	first	last
Date				
2021-05-31	34306.896758	37230.114854	35714.752020	36934.451092
2021-06-01	35663.453003	37715.798820	37715.798820	36451.907824
2021-06-02	36233.029570	38174.225418	36663.448346	37587.034649
2021-06-03	37339.291099	39501.065673	37451.660847	39100.676932
2021-06-04	36321.025062	39151.316184	39151.316184	37160.492643
2021-06-05	35072.140405	37966.091306	36848.909698	35072.140405
2021-06-06	35502.108234	36356.780991	35605.821928	35502.108234
2021-06-07	34150.238383	36750.903651	35834.474474	34183.150733
2021-06-08	31681.562668	33901.499467	33901.499467	33502.495642
2021-06-09	32580.523479	37187.329901	33175.471094	37187.329901
2021-06-10	36430.957724	38407.469192	37484.664195	36834.548304
2021-06-11	36370.131461	37623.469170	37061.695204	37358.577094
2021-06-12	35217.561657	37253.824363	37253.824363	35801.512385
2021-06-13	34950.574994	39247.729632	35666.149779	38964.719676
2021-06-14	39037.476848	40852.104134	39147.705679	40319.243269
2021-06-15	39805.654954	40666.801420	40624.513053	40107.399334
2021-06-16	38411.140558	40517.290737	40378.202550	38596.661586
2021-06-17	38321.454657	39502.727671	38321.454657	38435.866962

In [147…

```
# create candlestick plot

# import library

import plotly.graph_objects as go


# create figure info
fig = go.Figure(data = [go.Candlestick(x = candlestick_data.index,
                                       open = candlestick_data['Price']['first'],
                                       high = candlestick_data['Price']['max'],
                                       low = candlestick_data['Price']['min'],
                                       close = candlestick_data['Price']['last'])
                        ])

# update figure layout info
fig.update_layout(xaxis_rangeslider_visible = False, xaxis_title = 'Date',
                  yaxis_title = 'Price (USD)', title = 'Bitcoin Candlestick Chart Over P

# show figure
fig.show()
```



REST APIs

Uniform Resource Locator (URL)

- made up of 3 primary parts
- Scheme: the protocol, usually http://
- Internet Addres / Base URL: used to find the location, such as www.www.ibm.com.com
- Route: location on the web server, such as /images/IDSNlogo.png

HTTP protocol:

- a general protocol for transferring information through the web, which includes many REST APIs.
- when an HTTP request is made, an HTTP method is sent which tells the server what action to perform. If the request is successful, the server sends back the object to the client as an HTTP response.

HTTP Request:

- an HTTP method is sent, tells the server what action to perform.

HTTP Response:

- contains info about the HTTP version number
- status code info & descriptive info
- response header - contains useful info
- response body - contains the requested file as an HTML document

HTTP Methods include:

- get: retrieve data from server
- post: submit data to server
- put: update data on the server
- delete: delete data on the server

Common libraries that can work with HTTP protocol:

- httplib
- urllib
- requests

In [148…
```
# using get request
import requests
import os
from PIL import Image
from IPython.display import IFrame

url_get = 'http://httpbin.org/get'
payload = {'name':'Joseph', 'ID':'123'}
r = requests.get(url_get, params = payload)
```

In [149…
```
# view request url
r.url
```

Out[149… 'http://httpbin.org/get?name=Joseph&ID=123'

In [150…
```
# view request body
r.request.body
# shows as none since the info is sent in the URL
```

In [151…
```
# view status code
r.status_code
```

Out[151… 200

In [152…
```
# view response text
r.text
```

Out[152… '{\n "args": {\n   "ID": "123", \n   "name": "Joseph"\n }, \n "headers": {\n   "Ac
cept": "*/*", \n   "Accept-Encoding": "gzip, deflate", \n   "Host": "httpbin.org", \n

```
"User-Agent": "python-requests/2.25.1", \n    "X-Amzn-Trace-Id": "Root=1-60cb8824-275f62
be2f6964ae6f171b8e"\n  }, \n  "origin": "73.98.211.168", \n  "url": "http://httpbin.org/
get?name=Joseph&ID=123"\n}\n'
```

In [153…
```
# view key 'content-type'
r.headers['Content-Type']
```

Out[153… `'application/json'`

In [154…
```
#format the content as json
r.json()
```

Out[154…
```
{'args': {'ID': '123', 'name': 'Joseph'},
 'headers': {'Accept': '*/*',
  'Accept-Encoding': 'gzip, deflate',
  'Host': 'httpbin.org',
  'User-Agent': 'python-requests/2.25.1',
  'X-Amzn-Trace-Id': 'Root=1-60cb8824-275f62be2f6964ae6f171b8e'},
 'origin': '73.98.211.168',
 'url': 'http://httpbin.org/get?name=Joseph&ID=123'}
```

In [155…
```
# view key 'args'
r.json()['args']
```

Out[155… `{'ID': '123', 'name': 'Joseph'}`

Post Requests:

- sends request in post body, not in URL (unlike get request)

In [156…
```
# post request
url_post = 'http://httpbin.org/post'
payload = {'name':'Joseph', 'ID':'123'}
r_post = requests.post(url_post, data = payload)
```

In [157…
```
# compare urls of post and get requests
print('post request: ', r_post.url)
```

```
post request:  http://httpbin.org/post
```

In [158…
```
print('get request: ', r.url)
```

```
get request:  http://httpbin.org/get?name=Joseph&ID=123
```

In [159…
```
# see that the post request has a body
r_post.request.body
```

Out[159… `'name=Joseph&ID=123'`

In [160…
```
# view payload
r_post.json()['form']
```

Out[160… {'ID': '123', 'name': 'Joseph'}

In [161…
```
# lab: requests in python

# use get request to get to ibm.com
url='https://www.ibm.com/'
r=requests.get(url)
```

In [162…
```
# view status code
r.status_code
```

Out[162… 200

In [163…
```
# view request headers
print(r.request.headers)
```

```
{'User-Agent': 'python-requests/2.25.1', 'Accept-Encoding': 'gzip, deflate', 'Accept':
'*/*', 'Connection': 'keep-alive', 'Cookie': 'bm_sz=71F60647682BE1C3AF0877AAB5BC9DCA~YAA
QLWzTF2/QNRN6AQAAVdALGwxf/eOsSDgFGuy4iF2bYX4BGhYfrUUlTTFMNDUBJSsEoOCM5e7E8pJVBiTrTOSu1iU
y4s4yDsT5enMduS8zbee12ick15UWLr0jeL1bV2Fj/PtSgmdRNcyMcX11ctZMdblFgNAnQDJnACTP1Iyihtsiygq
xdya+5Mp/; _abck=F3BF444B0F2D9968E461C424332C8CD4~-1~YAAQLWzTF3DQNRN6AQAAVdALGwbmjJzodhz
mMlFSx1NH0xi0R/cs6RhyNeqJ3+XAzbw2OuFRCb80oUtSRRjTX7cJtaM5DsCmoFYV1VvHQQkYvsZOauaCYwF99vw
tIz4tG2gqeFkdTOwDiAUz4J97apOvGM3sasGqeb9aw09+Qk1FCy7oPM1FGv8d98xI5DQZuoYHGwGSul53TGXKb8e
dfaqRT2gSt/jmbcAhk++3sj/qhoZKxZAZrXl0sQomsHykbeh2HNnDL623vaVRtbotBmaG2GO8bTKlDyX9+UVn5Sh
ZyX4SjRF9rgvyIrSQh15s56POWiahSCrBTQeJFqOx6c7fj4MEXXfnKzUDir0Bi0vlrjI6fU4=~-1~-1~-1'}
```

In [164…
```
# view request body
print('request body: ', r.request.body)
```

```
request body:  None
```

In [165…
```
# view http response header
header = r.headers
print(r.headers)
```

```
{'Cache-Control': 'max-age=301', 'Expires': 'Tue, 15 Jun 2021 13:28:38 GMT', 'Last-Modif
ied': 'Mon, 14 Jun 2021 21:42:59 GMT', 'ETag': '"18335-5c4c0bdc00463"', 'Accept-Ranges':
'bytes', 'Content-Encoding': 'gzip', 'Content-Type': 'text/html', 'X-Akamai-Transforme
d': '9 18360 0 pmb=mTOE,1', 'Date': 'Thu, 17 Jun 2021 17:36:37 GMT', 'Content-Length':
'18432', 'Connection': 'keep-alive', 'Vary': 'Accept-Encoding', 'x-content-type-option
s': 'nosniff', 'X-XSS-Protection': '1; mode=block', 'Content-Security-Policy': 'upgrade-
insecure-requests', 'Strict-Transport-Security': 'max-age=31536000'}
```

In [166…
```
# view request date
header['date']
```

Out[166… 'Thu, 17 Jun 2021 17:36:37 GMT'

In [167…
```
# view type of data
header['Content-Type']
```

Out[167… 'text/html'

In [168…
```
# check encoding
r.encoding
```

Out[168…
```
'ISO-8859-1'
```

In [169…
```
# display the HTML in the body
r.text[0:100]
```

Out[169…
```
'<!DOCTYPE html><html lang="en-US"><head><meta name="viewport" content="width=device-wid
th"/><meta ch'
```

In [170…
```
# load a non-text request
# Use single quotation marks for defining string
url='https://gitlab.com/ibm/skills-network/courses/placeholder101/-/raw/master/labs/mod
```

In [171…
```
# make get request
r=requests.get(url)
```

In [172…
```
# view response header
print(r.headers)
```

```
{'Date': 'Thu, 17 Jun 2021 17:36:37 GMT', 'Content-Type': 'image/png', 'Content-Length':
'21590', 'Connection': 'keep-alive', 'Cache-Control': 'max-age=60, public', 'Content-Dis
position': 'inline', 'Etag': 'W/"c26d88d0ca290ba368620273781ea37c"', 'Permissions-Polic
y': 'interest-cohort=()', 'Vary': 'Accept, Accept-Encoding', 'X-Content-Type-Options':
'nosniff', 'X-Download-Options': 'noopen', 'X-Frame-Options': 'DENY', 'X-Permitted-Cross
-Domain-Policies': 'none', 'X-Request-Id': '01F8DD0JH8WK855ZZJYRDVQGNQ', 'X-Runtime':
'0.064172', 'X-Ua-Compatible': 'IE=edge', 'X-Xss-Protection': '1; mode=block', 'Strict-T
ransport-Security': 'max-age=31536000', 'Referrer-Policy': 'strict-origin-when-cross-ori
gin', 'GitLab-LB': 'fe-10-lb-gprd', 'GitLab-SV': 'web-01-sv-gprd', 'CF-Cache-Status': 'R
EVALIDATED', 'Accept-Ranges': 'bytes', 'cf-request-id': '0abca4e97c000027d40e34300000000
1', 'Expect-CT': 'max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/
beacon/expect-ct"', 'Server': 'cloudflare', 'CF-RAY': '660e0a88cc2927d4-SLC'}
```

In [173…
```
# view content type
r.headers['Content-Type']
```

Out[173…
```
'image/png'
```

In [174…
```
# since an image is a response object that contains the image as a bytes-like object,
# save it to a file object.

# specify file path and name
path = os.path.join(os.getcwd(), 'image.png')
path
```

Out[174…
```
'C:\\Users\\orgil\\PycharmProjects\\OWL\\Tournament_Results\\image.png'
```

In [175…
```
# in order to access the body of the response, use attribute content
# then save it using open and write method
with open(path, 'wb') as f:
    f.write(r.content)
```

```
# then view the image
Image.open(path)
```

Out[175…

WEBSCRAPING

Webscraping:

- a process that can be used to automatically extract info from a website. This process can be completed relatively quickly (minutes, not hours).
- python can webscrape using two modules: requests, beautiful soup

Beautiful Soup:

- BeautifulSoup represents HTML as a set of tree-like objects with methods used to parse the HTML.
- .find_all('x') is a filter method that will look through a tag's descendants and retrieve all descendants that match the filter 'x'.

Process:

- first load beautiful soup and requests
- use requests to make a get request of the webpage you want
- create the beautiful soup object to parse the HTML
- use the .find_all() method to search for the info you want
- clean up your data and remove the tags

In [176…
```
# pip install bs4
```

In [177…
```
# LAB: webscraping

# import
from bs4 import BeautifulSoup
import requests
```

In [178…
```
# store the HTML info as a string
html="<!DOCTYPE html><html><head><title>Page Title</title></head><body><h3><b id='bolde
```

In [179…
```
# parse the string using beautiful soup
soup = BeautifulSoup(html, 'html.parser')
```

In [180…
```
# use prettify() method to display the html in a nested manner
print(soup.prettify())
```

```
<!DOCTYPE html>
<html>
 <head>
  <title>
   Page Title
  </title>
 </head>
 <body>
  <h3>
   <b id="boldest">
    Lebron James
   </b>
  </h3>
  <p>
   Salary: $ 92,000,000
  </p>
  <h3>
   Stephen Curry
  </h3>
  <p>
   Salary: $85,000, 000
  </p>
  <h3>
   Kevin Durant
  </h3>
  <p>
   Salary: $73,200, 000
  </p>
```

```
 </body>
</html>
```

In [181…
```
# the tag object corresponds to an HTML tag in the original document
# get the tag title
tag_object = soup.title
print('tag object: ', tag_object)
```

```
tag object:  <title>Page Title</title>
```

In [182…
```
# view tag type
print('tag object type: ', type(tag_object))
```

```
tag object type:  <class 'bs4.element.Tag'>
```

In [183…
```
# if there is more than one tag with the same name, the first one is called
tag_object = soup.h3
tag_object
```

Out[183…
```
<h3><b id="boldest">Lebron James</b></h3>
```

In [184…
```
# access the child of a tag
tag_child = tag_object.b
tag_child
```

Out[184…
```
<b id="boldest">Lebron James</b>
```

In [185…
```
# access the parent of a tag
parent_tag = tag_child.parent
parent_tag # this is identical to tag_object
```

Out[185…
```
<h3><b id="boldest">Lebron James</b></h3>
```

In [186…
```
# find tag's parent
tag_object.parent # this includes the entire body section
```

Out[186…
```
<body><h3><b id="boldest">Lebron James</b></h3><p> Salary: $ 92,000,000 </p><h3> Stephen Curry</h3><p> Salary: $85,000, 000 </p><h3> Kevin Durant </h3><p> Salary: $73,200, 000</p></body>
```

In [187…
```
# find tag sibling
sibling_1 = tag_object.next_sibling
sibling_1
```

Out[187…
```
<p> Salary: $ 92,000,000 </p>
```

In [188…
```
# find next sibling
sibling_2 = sibling_1.next_sibling
sibling_2
```

Out[188…
```
<h3> Stephen Curry</h3>
```

In [189…
```
# find next sibling
sibling_3 = sibling_2.next_sibling
sibling_3
```

Out[189…
```
<p> Salary: $85,000, 000 </p>
```

In [190…
```
# if the tag has attributes, you can access them by treating the tag like a dictionary
# id = 'boldest'
tag_child['id']
```

Out[190…
```
'boldest'
```

In [191…
```
# access the dictionary directly
tag_child.attrs
```

Out[191…
```
{'id': 'boldest'}
```

In [192…
```
# get content of the attribute with get()
tag_child.get('id')
```

Out[192…
```
'boldest'
```

In [193…
```
# the navigablestring class is used by beautifulsoup to contain a bit of text
tag_string = tag_child.string
tag_string
# navigablestring is similar to python strongs.
# The main difference is that navigable string supports beautiful soup features.
```

Out[193…
```
'Lebron James'
```

In [194…
```
# verify type
type(tag_string)
```

Out[194…
```
bs4.element.NavigableString
```

In [195…
```
# convert into string object
unicode_string = str(tag_string)
unicode_string
```

Out[195…
```
'Lebron James'
```

In [196…
```
# store a new table into a string
table="<table><tr><td id='flight'>Flight No</td><td>Launch site</td> <td>Payload mass</

# parse the string
table_bs = BeautifulSoup(table, 'html.parser')
```

In [197…
```
# .find_all() looks through a tag's descendants to retrieve all matches
# .find_all(name, attrs, recursive, string, limit, **kwargs)
```

```
# find all tags with the name tr
table_rows = table_bs.find_all('tr')
table_rows
```

Out[197…
```
[<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>,
 <tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></
td><td>300 kg</td></tr>,
 <tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg
</td></tr>,
 <tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a></
td><td>80 kg</td></tr>]
```

In [198…
```
# get first row
first_row = table_rows[0]
first_row
```

Out[198…
```
<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>
```

In [199…
```
print(type(first_row))
```

```
<class 'bs4.element.Tag'>
```

In [200…
```
# get the child
first_row.td
```

Out[200…
```
<td id="flight">Flight No</td>
```

In [201…
```
# iterate through the table_rows list
for i, row in enumerate(table_rows):
    print('row ', i, 'is ', row)
```

```
row  0 is  <tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td>
</tr>
row  1 is  <tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>300 kg</td></tr>
row  2 is  <tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td
><td>94 kg</td></tr>
row  3 is  <tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a>
</a></a></td><td>80 kg</td></tr>
```

In [202…
```
# use find_all to extract table cells using the tag 'td'
for i,row in enumerate(table_rows):
    print('row ', i)
    cells = row.find_all('td')
    for j, cell in enumerate(cells):
        print('column', j, ' cell ', cell)
```

```
row  0
column 0  cell  <td id="flight">Flight No</td>
column 1  cell  <td>Launch site</td>
column 2  cell  <td>Payload mass</td>
row  1
column 0  cell  <td>1</td>
column 1  cell  <td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></
td>
column 2  cell  <td>300 kg</td>
```

```
row  2
column 0  cell  <td>2</td>
column 1  cell  <td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td>
column 2  cell  <td>94 kg</td>
row  3
column 0  cell  <td>3</td>
column 1  cell  <td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a>
</td>
column 2  cell  <td>80 kg</td>
```

In [203…
```
# if we use a list, we cna match against any item in the list
list_input = table_bs.find_all(name = ['tr', 'td'])
list_input
```

Out[203…
```
[<tr><td id="flight">Flight No</td><td>Launch site</td> <td>Payload mass</td></tr>,
 <td id="flight">Flight No</td>,
 <td>Launch site</td>,
 <td>Payload mass</td>,
 <tr> <td>1</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></a></
td><td>300 kg</td></tr>,
 <td>1</td>,
 <td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a></a></td>,
 <td>300 kg</td>,
 <tr><td>2</td><td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td><td>94 kg
</td></tr>,
 <td>2</td>,
 <td><a href="https://en.wikipedia.org/wiki/Texas">Texas</a></td>,
 <td>94 kg</td>,
 <tr><td>3</td><td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a></a></
td><td>80 kg</td></tr>,
 <td>3</td>,
 <td><a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a></a></td>,
 <td>80 kg</td>]
```

In [204…
```
# if an argument is not recognized, it will be turned into a filter on the tag attribut
table_bs.find_all(id = 'flight')
```

Out[204…
```
[<td id="flight">Flight No</td>]
```

In [205…
```
# find all the elements that have links to the FL wiki page
list_input = table_bs.find_all(href = 'https://en.wikipedia.org/wiki/Florida')
list_input
```

Out[205…
```
[<a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a>,
 <a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a>]
```

In [206…
```
# if we set href to true, the code finds all tags with that value
table_bs.find_all(href = True)
```

Out[206…
```
[<a href="https://en.wikipedia.org/wiki/Florida">Florida<a></a></a>,
 <a href="https://en.wikipedia.org/wiki/Texas">Texas</a>,
 <a href="https://en.wikipedia.org/wiki/Florida">Florida<a> </a></a>]
```

In [207…
```
# search for strings instead of tags
table_bs.find_all(string = 'Florida')
```

Out[207…
```
['Florida', 'Florida']
```

In [208…
```
# use the .find() method if you want to just find the first element in the document ins

# store two tables of info as string
two_tables="<h3>Rocket Launch </h3><p><table class='rocket'><tr><td>Flight No</td><td>L
```

In [209…
```
# parse the string
two_tables_bs = BeautifulSoup(two_tables, 'html.parser')
```

In [210…
```
# find the first table using the tag name table
two_tables_bs.find('table')
```

Out[210…
```
<table class="rocket"><tr><td>Flight No</td><td>Launch site</td> <td>Payload mass</td></
tr><tr><td>1</td><td>Florida</td><td>300 kg</td></tr><tr><td>2</td><td>Texas</td><td>94
kg</td></tr><tr><td>3</td><td>Florida </td><td>80 kg</td></tr></table>
```

In [211…
```
# find 2nd table using class attribute
# note: class is a keyword in python, so add underscore
two_tables_bs.find('table', class_ = 'pizza')
```

Out[211…
```
<table class="pizza"><tr><td>Pizza Place</td><td>Orders</td> <td>Slices </td></tr><tr><t
d>Domino's Pizza</td><td>10</td><td>100</td></tr><tr><td>Little Caesars</td><td>12</td><
td>144 </td></tr><tr><td>Papa John's </td><td>15 </td><td>165</td></tr></table>
```

In [212…
```
# download and scrape the contents of a web page
url = "http://www.ibm.com"

# use get request to download the contents in text format
data  = requests.get(url).text

# parse with beautifulsoup
soup = BeautifulSoup(data, 'html.parser')
```

In [213…
```
# scrape all links
for link in soup.find_all('a', href = True): # in html, a link is represented by tag <a
    print(link.get('href'))
```

```
#main-content
https://www.ibm.com/
https://www.ibm.com/thought-leadership/institute-business-value/report/lgbt-inclusion?ln
k=ushpv18l1
https://www.ibm.com/cloud/go-hybrid/?lnk=ushpv18f1#chadwick
https://www.ibm.com/products/cloud-pak-for-data/scale-trustworthy-ai?lnk=ushpv18f2
https://community.ibm.com/community/user/automation/blogs/david-jenness/2021/06/03/ibm-b
uild-a-bot-challenge?lnk=ushpv18f3
https://www.ibm.com/thought-leadership/institute-business-value/report/quantum-decade?ln
k=ushpv18f4
https://www.ibm.com/products/offers-and-discounts?link=ushpv18t5&lnk2=trial_mktpl_MPDISC
https://www.ibm.com/cloud/watson-discovery?lnk=ushpv18t1&lnk2=trial_WatDiscovery&psrc=no
ne&pexp=def
https://www.ibm.com/products/workload-automation?lnk=ushpv18t2&psrc=NONE&pexp=DEF&lnk2=t
rial_WorkloadAuto
https://www.ibm.com/cloud/free?lnk=ushpv18t3&lnk2=trial_Cloud&psrc=none&pexp=def
https://www.ibm.com/products/unified-endpoint-management?lnk=ushpv18t4&lnk2=trial_MaaS36
0Wat&psrc=none&pexp=def
https://www.ibm.com/search?lnk=ushpv18srch&locale=en-us&q=
```

```
https://www.ibm.com/products?lnk=ushpv18p1&lnk2=trial_mktpl&psrc=none&pexp=def
https://developer.ibm.com/depmodels/cloud/?lnk=ushpv18ct16
https://developer.ibm.com/technologies/artificial-intelligence?lnk=ushpv18ct19
https://www.ibm.com/demos/?lnk=ushpv18ct12
https://developer.ibm.com/?lnk=ushpv18ct9
https://www.ibm.com/docs/en?lnk=ushpv18ct14
https://www.redbooks.ibm.com/?lnk=ushpv18ct10
https://www.ibm.com/support/home/?lnk=ushpv18ct11
https://www.ibm.com/training/?lnk=ushpv18ct15
https://www.ibm.com/cloud/hybrid?lnk=ushpv18ct20
https://www.ibm.com/cloud/learn/public-cloud?lnk=ushpv18ct17
https://www.ibm.com/cloud/redhat?lnk=ushpv18ct13
https://www.ibm.com/artificial-intelligence?lnk=ushpv18ct3
https://www.ibm.com/quantum-computing?lnk=ushpv18ct18
https://www.ibm.com/cloud/learn/kubernetes?lnk=ushpv18ct8
https://www.ibm.com/products/spss-statistics?lnk=ushpv18ct7
https://www.ibm.com/blockchain?lnk=ushpv18ct1
https://www-03.ibm.com/employment/technicaltalent/developer/?lnk=ushpv18ct2
https://www.ibm.com/search?lnk=ushpv18srch&locale=en-us&q=
https://www.ibm.com/products?lnk=ushpv18p1&lnk2=trial_mktpl&psrc=none&pexp=def
https://www.ibm.com/cloud/hybrid?lnk=ushpv18pt14&bv=true
https://www.ibm.com/watson?lnk=ushpv18pt17&bv=true
https://www.ibm.com/it-infrastructure?lnk=ushpv18pt19&bv=true
https://www.ibm.com/us-en/products/categories?technologyTopics%5B0%5D%5B0%5D=cat.topic:B
lockchain&isIBMOffering%5B0%5D=true&lnk=ushpv18pt4&bv=true
https://www.ibm.com/us-en/products/category/technology/security?lnk=ushpv18pt9&bv=true
https://www.ibm.com/us-en/products/category/technology/analytics?lnk=ushpv18pt1&bv=true
https://www.ibm.com/cloud/automation?lnk=ushpv18ct21
https://www.ibm.com/quantum-computing?lnk=ushpv18pt16&bv=true
https://www.ibm.com/support/home/?lnk=ushpv18ct11
https://www.ibm.com/training/?lnk=ushpv18ct15
https://www.ibm.com/demos/?lnk=ushpv18ct12
https://developer.ibm.com/?lnk=ushpv18ct9
https://www.ibm.com/garage?lnk=ushpv18pt18
https://www.ibm.com/docs/en?lnk=ushpv18ct14
https://www.redbooks.ibm.com/?lnk=ushpv18ct10
https://www.ibm.com/
```

In [214…

```
# scrape all image tags
for link in soup.find_all('img'): # in html, image is <img>
    print(link)
    print(link.get('src'))
```

```
<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iMTA1NSIgaGVpZ2h0PSI1MjcuNSIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZX
JzaW9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/
>

93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="leadspace mobile image" class="ibm-resize" decoding="async" src="https://1.da
m.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/76/6c/20210614-LGBTQ-IBV-2
5934-mobile-720x360.png" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizi
ng:border-box;padding:0;border:none;margin:auto;display:block;width:0;height:0;min-widt
h:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/76/6c/20210614
-LGBTQ-IBV-25934-mobile-720x360.png
<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="The power of a hybrid approach" class="ibm-resize ibm-ab-image featured-image"
decoding="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepa
ge/ul/g/f6/d3/20210614-f-techsplainer-jamie-chadwick-25946.jpg" style="position:absolut
```

e;top:0;left:0;bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;
display:block;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:
100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/f6/d3/20210614
-f-techsplainer-jamie-chadwick-25946.jpg
<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="AI: it’s a matter of trust" class="ibm-resize ibm-ab-image featured-
image" decoding="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-conten
t/homepage/ul/g/09/92/20210614-cloud-pak-data-25942-444x320.png" style="position:absolut
e;top:0;left:0;bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;
display:block;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:
100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/09/92/20210614
-cloud-pak-data-25942-444x320.png
<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="Build-A-Bot Challenge" class="ibm-resize ibm-ab-image featured-image" decoding
="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/
fa/24/20210615-f-build-a-bot-25944.jpg" style="position:absolute;top:0;left:0;bottom:0;r
ight:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:block;width:0;hei
ght:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/fa/24/20210615
-f-build-a-bot-25944.jpg
<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjMyMCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="Report: The Quantum Decade" class="ibm-resize ibm-ab-image featured-image" dec
oding="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/
ul/g/79/75/7975595c-c60c-4bde-8aa3bc8fe58ecf82.jpg" style="position:absolute;top:0;left:
0;bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:bloc
k;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/79/75/7975595c
-c60c-4bde-8aa3bc8fe58ecf82.jpg
<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="Watson Discovery" class="ibm-resize ibm-ab-image trials-image" decoding="asyn
c" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/9f/86/
watson-discovery-trial_700x420.png" style="position:absolute;top:0;left:0;bottom:0;righ
t:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:block;width:0;heigh
t:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/9f/86/watson-d
iscovery-trial_700x420.png
<img alt="Watson Discovery" class="ibm-resize ibm-ab-image trials-image" decoding="asyn
c" src=""
style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-box;padding:0;b
order:none;margin:auto;display:block;width:0;height:0;min-width:100%;max-width:100%;min-
height:100%;max-height:100%"/>

<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=

```
<img alt="IBM Workload Automation" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/
5e/4f/Workload-Automation-trial-444x254.png" style="position:absolute;top:0;left:0;botto
m:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:block;width:
0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/5e/4f/Workload
-Automation-trial-444x254.png
<img alt="IBM Workload Automation" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="
RAA7" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-box;paddi
ng:0;border:none;margin:auto;display:block;width:0;height:0;min-width:100%;max-width:10
0%;min-height:100%;max-height:100%"/>

<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="IBM Cloud" class="ibm-resize ibm-ab-image trials-image" decoding="async" src
="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/54/79/IBM-Cl
oud-23059-700x420.png" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizin
g:border-box;padding:0;border:none;margin:auto;display:block;width:0;height:0;min-width:
100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/54/79/IBM-Clou
d-23059-700x420.png
<img alt="IBM Cloud" class="ibm-resize ibm-ab-image trials-image" decoding="async" src
="" style
="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-box;padding:0;borde
r:none;margin:auto;display:block;width:0;height:0;min-width:100%;max-width:100%;min-heig
ht:100%;max-height:100%"/>

<img alt="" aria-hidden="true" role="presentation" src="
B3aWR0aD0iNDQwIiBoZWlnaHQ9IjI2MCIgeG1sbnM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW
9uPSIxLjEiLz4=" style="max-width:100%;display:block;margin:0;border:none;padding:0"/>

cudzMub3JnLzIwMDAvc3ZnIiB2ZXJzaW9uPSIxLjEiLz4=
<img alt="IBM MaaS360 with Watson" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/
4c/a5/10072019-t-bt-MaaS360-watson-23210-700x420.png" style="position:absolute;top:0;lef
t:0;bottom:0;right:0;box-sizing:border-box;padding:0;border:none;margin:auto;display:blo
ck;width:0;height:0;min-width:100%;max-width:100%;min-height:100%;max-height:100%"/>
https://1.dam.s81c.com/public/content/dam/worldwide-content/homepage/ul/g/4c/a5/10072019
-t-bt-MaaS360-watson-23210-700x420.png
<img alt="IBM MaaS360 with Watson" class="ibm-resize ibm-ab-image trials-image" decoding
="async" src="
RAA7" style="position:absolute;top:0;left:0;bottom:0;right:0;box-sizing:border-box;paddi
ng:0;border:none;margin:auto;display:block;width:0;height:0;min-width:100%;max-width:10
0%;min-height:100%;max-height:100%"/>

```

In [215…

```
# scrape data from html tables

#The below url contains an html table with data about colors and color codes.
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DA0321EN-
```

In [216…

```
# get the contents of the webpage in text format and store in a variable called data
data  = requests.get(url).text

# parse
soup = BeautifulSoup(data, 'html.parser')
```

In [217…
```
# find html table on webpage
table = soup.find('table') # in html, table is <table>
```

In [218…
```
# get all rows from table
for row in table.find_all('tr'): # in html, table rows are <tr>
    # get all columns
    cols = row.find_all('td') # in html, column is <td>
    color_name = cols[2].string # store value in column 3 as color name
    color_code = cols[3].string # store value in column 4 as color code
    print("{} --->{}". format(color_name, color_code))
```

```
Color Name --->None
lightsalmon --->#FFA07A
salmon --->#FA8072
darksalmon --->#E9967A
lightcoral --->#F08080
coral --->#FF7F50
tomato --->#FF6347
orangered --->#FF4500
gold --->#FFD700
orange --->#FFA500
darkorange --->#FF8C00
lightyellow --->#FFFFE0
lemonchiffon --->#FFFACD
papayawhip --->#FFEFD5
moccasin --->#FFE4B5
peachpuff --->#FFDAB9
palegoldenrod --->#EEE8AA
khaki --->#F0E68C
darkkhaki --->#BDB76B
yellow --->#FFFF00
lawngreen --->#7CFC00
chartreuse --->#7FFF00
limegreen --->#32CD32
lime --->#00FF00
forestgreen --->#228B22
green --->#008000
powderblue --->#B0E0E6
lightblue --->#ADD8E6
lightskyblue --->#87CEFA
skyblue --->#87CEEB
deepskyblue --->#00BFFF
lightsteelblue --->#B0C4DE
dodgerblue --->#1E90FF
```

In [219…
```
# scrape data from html tables into a dataframe
import pandas as pd

#The below url contains html tables with data about world population.
url = "https://en.wikipedia.org/wiki/World_population"

# get the contents of the webpage in text format and store in a variable called data
data  = requests.get(url).text

# parse
soup = BeautifulSoup(data, 'html.parser')
```

In [220…
```
# find all html tables in the page
tables = soup.find_all('table')
```

In [221…
```
# how many tables did we find?
len(tables)
```

Out[221… 27

In [222…
```
# search the tables to find the correct ones
for index, table in enumerate(tables):
    if ('10 most densely populated countries' in str(table)):
        table_index = index
print(table_index)
```

5

In [223…
```
# try to locate the table name
# clean up
print(tables[table_index].prettify())

# create data frame
population_data = pd.DataFrame(columns=["Rank", "Country", "Population", "Area", "Densi

# find countries with highest population density
for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col != []):
        rank = col[0].text
        country = col[1].text
        population = col[2].text.strip()
        area = col[3].text.strip()
        density = col[4].text.strip()
        population_data = population_data.append({"Rank":rank, "Country":country, "Popu

population_data
```

```
<table class="wikitable sortable" style="text-align:right">
 <caption>
  10 most densely populated countries
  <small>
   (with population above 5 million)
  </small>
 </caption>
 <tbody>
  <tr>
   <th>
    Rank
   </th>
   <th>
    Country
   </th>
   <th>
    Population
   </th>
   <th>
    Area
    <br/>
    <small>
     (km
     <sup>
      2
     </sup>
```

```
     )
     </small>
    </th>
    <th>
     Density
     <br/>
     <small>
      (pop/km
      <sup>
       2
      </sup>
      )
     </small>
    </th>
   </tr>
   <tr>
    <td>
     1
    </td>
    <td align="left">
     <span class="flagicon">
      <img alt="" class="thumbborder" data-file-height="3456" data-file-width="5184" deco
ding="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/4/48/Flag_o
f_Singapore.svg/23px-Flag_of_Singapore.svg.png" srcset="//upload.wikimedia.org/wikipedi
a/commons/thumb/4/48/Flag_of_Singapore.svg/35px-Flag_of_Singapore.svg.png 1.5x, //uploa
d.wikimedia.org/wikipedia/commons/thumb/4/48/Flag_of_Singapore.svg/45px-Flag_of_Singapor
e.svg.png 2x" width="23"/>
     </span>
     <a href="/wiki/Singapore" title="Singapore">
      Singapore
     </a>
    </td>
    <td>
     5,704,000
    </td>
    <td>
     710
    </td>
    <td>
     8,033
    </td>
   </tr>
   <tr>
    <td>
     2
    </td>
    <td align="left">
     <span class="flagicon">
      <img alt="" class="thumbborder" data-file-height="600" data-file-width="1000" decod
ing="async" height="14" src="//upload.wikimedia.org/wikipedia/commons/thumb/f/f9/Flag_of
_Bangladesh.svg/23px-Flag_of_Bangladesh.svg.png" srcset="//upload.wikimedia.org/wikipedi
a/commons/thumb/f/f9/Flag_of_Bangladesh.svg/35px-Flag_of_Bangladesh.svg.png 1.5x, //uplo
ad.wikimedia.org/wikipedia/commons/thumb/f/f9/Flag_of_Bangladesh.svg/46px-Flag_of_Bangla
desh.svg.png 2x" width="23"/>
     </span>
     <a href="/wiki/Bangladesh" title="Bangladesh">
      Bangladesh
     </a>
    </td>
    <td>
     170,850,000
    </td>
    <td>
     143,998
    </td>
```

```
   <td>
    1,186
   </td>
  </tr>
  <tr>
   <td>
    3
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" decodi
ng="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/59/Flag_of_
Lebanon.svg/23px-Flag_of_Lebanon.svg.png" srcset="//upload.wikimedia.org/wikipedia/commo
ns/thumb/5/59/Flag_of_Lebanon.svg/35px-Flag_of_Lebanon.svg.png 1.5x, //upload.wikimedia.
org/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/45px-Flag_of_Lebanon.svg.png 2x" wi
dth="23"/>
    </span>
    <a href="/wiki/Lebanon" title="Lebanon">
     Lebanon
    </a>
   </td>
   <td>
    6,856,000
   </td>
   <td>
    10,452
   </td>
   <td>
    656
   </td>
  </tr>
  <tr>
   <td>
    4
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" decodi
ng="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_
the_Republic_of_China.svg/23px-Flag_of_the_Republic_of_China.svg.png" srcset="//upload.w
ikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/35px-Flag_of
_the_Republic_of_China.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/7/7
2/Flag_of_the_Republic_of_China.svg/45px-Flag_of_the_Republic_of_China.svg.png 2x" width
="23"/>
    </span>
    <a href="/wiki/Taiwan" title="Taiwan">
     Taiwan
    </a>
   </td>
   <td>
    23,604,000
   </td>
   <td>
    36,193
   </td>
   <td>
    652
   </td>
  </tr>
  <tr>
   <td>
    5
   </td>
   <td align="left">
    <span class="flagicon">
```

```
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" decodi
ng="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/0/09/Flag_of_
South_Korea.svg/23px-Flag_of_South_Korea.svg.png" srcset="//upload.wikimedia.org/wikiped
ia/commons/thumb/0/09/Flag_of_South_Korea.svg/35px-Flag_of_South_Korea.svg.png 1.5x, //u
pload.wikimedia.org/wikipedia/commons/thumb/0/09/Flag_of_South_Korea.svg/45px-Flag_of_So
uth_Korea.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/South_Korea" title="South Korea">
     South Korea
    </a>
   </td>
   <td>
    51,781,000
   </td>
   <td>
    99,538
   </td>
   <td>
    520
   </td>
  </tr>
  <tr>
   <td>
    6
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="720" data-file-width="1080" decod
ing="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/1/17/Flag_of
_Rwanda.svg/23px-Flag_of_Rwanda.svg.png" srcset="//upload.wikimedia.org/wikipedia/common
s/thumb/1/17/Flag_of_Rwanda.svg/35px-Flag_of_Rwanda.svg.png 1.5x, //upload.wikimedia.or
g/wikipedia/commons/thumb/1/17/Flag_of_Rwanda.svg/45px-Flag_of_Rwanda.svg.png 2x" width
="23"/>
    </span>
    <a href="/wiki/Rwanda" title="Rwanda">
     Rwanda
    </a>
   </td>
   <td>
    12,374,000
   </td>
   <td>
    26,338
   </td>
   <td>
    470
   </td>
  </tr>
  <tr>
   <td>
    7
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="1000" decod
ing="async" height="14" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/56/Flag_of
_Haiti.svg/23px-Flag_of_Haiti.svg.png" srcset="//upload.wikimedia.org/wikipedia/commons/
thumb/5/56/Flag_of_Haiti.svg/35px-Flag_of_Haiti.svg.png 1.5x, //upload.wikimedia.org/wik
ipedia/commons/thumb/5/56/Flag_of_Haiti.svg/46px-Flag_of_Haiti.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Haiti" title="Haiti">
     Haiti
    </a>
   </td>
   <td>
```

```
    11,578,000
   </td>
   <td>
    27,065
   </td>
   <td>
    428
   </td>
  </tr>
  <tr>
   <td>
    8
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="600" data-file-width="900" decodi
ng="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/2/20/Flag_of_
the_Netherlands.svg/23px-Flag_of_the_Netherlands.svg.png" srcset="//upload.wikimedia.or
g/wikipedia/commons/thumb/2/20/Flag_of_the_Netherlands.svg/35px-Flag_of_the_Netherlands.
svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/2/20/Flag_of_the_Netherland
s.svg/45px-Flag_of_the_Netherlands.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/Netherlands" title="Netherlands">
     Netherlands
    </a>
   </td>
   <td>
    17,610,000
   </td>
   <td>
    41,526
   </td>
   <td>
    424
   </td>
  </tr>
  <tr>
   <td>
    9
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="800" data-file-width="1100" decod
ing="async" height="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Flag_of
_Israel.svg/21px-Flag_of_Israel.svg.png" srcset="//upload.wikimedia.org/wikipedia/common
s/thumb/d/d4/Flag_of_Israel.svg/32px-Flag_of_Israel.svg.png 1.5x, //upload.wikimedia.or
g/wikipedia/commons/thumb/d/d4/Flag_of_Israel.svg/41px-Flag_of_Israel.svg.png 2x" width
="21"/>
    </span>
    <a href="/wiki/Israel" title="Israel">
     Israel
    </a>
   </td>
   <td>
    9,360,000
   </td>
   <td>
    22,072
   </td>
   <td>
    424
   </td>
  </tr>
  <tr>
   <td>
```

```
    10
   </td>
   <td align="left">
    <span class="flagicon">
     <img alt="" class="thumbborder" data-file-height="900" data-file-width="1350" decod
ing="async" height="15" src="//upload.wikimedia.org/wikipedia/en/thumb/4/41/Flag_of_Indi
a.svg/23px-Flag_of_India.svg.png" srcset="//upload.wikimedia.org/wikipedia/en/thumb/4/4
1/Flag_of_India.svg/35px-Flag_of_India.svg.png 1.5x, //upload.wikimedia.org/wikipedia/e
n/thumb/4/41/Flag_of_India.svg/45px-Flag_of_India.svg.png 2x" width="23"/>
    </span>
    <a href="/wiki/India" title="India">
     India
    </a>
   </td>
   <td>
    1,378,250,000
   </td>
   <td>
    3,287,240
   </td>
   <td>
    419
   </td>
  </tr>
 </tbody>
</table>
```

Out[223…

	Rank	Country	Population	Area	Density
0	1	Singapore	5,704,000	710	8,033
1	2	Bangladesh	170,850,000	143,998	1,186
2	3	Lebanon	6,856,000	10,452	656
3	4	Taiwan	23,604,000	36,193	652
4	5	South Korea	51,781,000	99,538	520
5	6	Rwanda	12,374,000	26,338	470
6	7	Haiti	11,578,000	27,065	428
7	8	Netherlands	17,610,000	41,526	424
8	9	Israel	9,360,000	22,072	424
9	10	India	1,378,250,000	3,287,240	419

In [225…

```
# scrape data from html tables into dataframe using read_html
import pandas as pd
pd.read_html(str(tables[5]), flavor='bs4')
# read_html always returns a list of dataframes, so choose the correct one
```

Out[225…

```
[   Rank      Country  Population  Area(km2)  Density(pop/km2)
 0     1    Singapore     5704000        710              8033
 1     2   Bangladesh   170850000     143998              1186
 2     3      Lebanon     6856000      10452               656
 3     4       Taiwan    23604000      36193               652
 4     5  South Korea    51781000      99538               520
 5     6       Rwanda    12374000      26338               470
 6     7        Haiti    11578000      27065               428
 7     8  Netherlands    17610000      41526               424
```

```
 8     9       Israel     9360000      22072              424
 9    10        India  1378250000    3287240              419]
```

In [226…

```
# choose dataframe
population_data_read_html = pd.read_html(str(tables[5]), flavor = 'bs4')[0]
population_data_read_html
```

Out[226…

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5704000	710	8033
1	2	Bangladesh	170850000	143998	1186
2	3	Lebanon	6856000	10452	656
3	4	Taiwan	23604000	36193	652
4	5	South Korea	51781000	99538	520
5	6	Rwanda	12374000	26338	470
6	7	Haiti	11578000	27065	428
7	8	Netherlands	17610000	41526	424
8	9	Israel	9360000	22072	424
9	10	India	1378250000	3287240	419

In [227…

```
# you can also use read_html to directly get dataframes from a URL
dataframe_list = pd.read_html(url, flavor = 'bs4')
```

In [228…

```
# find how many dataframes there are
len(dataframe_list)
```

Out[228… 27

In [229…

```
# choose the dataframe we want
dataframe_list[5]
```

Out[229…

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5704000	710	8033
1	2	Bangladesh	170850000	143998	1186
2	3	Lebanon	6856000	10452	656
3	4	Taiwan	23604000	36193	652
4	5	South Korea	51781000	99538	520
5	6	Rwanda	12374000	26338	470
6	7	Haiti	11578000	27065	428
7	8	Netherlands	17610000	41526	424
8	9	Israel	9360000	22072	424

	Rank	Country	Population	Area(km2)	Density(pop/km2)
9	10	India	1378250000	3287240	419

In [232…
```
# use the match parameter to select the specific table we want
pd.read_html(url, match = '10 most densely populated countries', flavor = 'bs4')[0]
```

Out[232…

	Rank	Country	Population	Area(km2)	Density(pop/km2)
0	1	Singapore	5704000	710	8033
1	2	Bangladesh	170850000	143998	1186
2	3	Lebanon	6856000	10452	656
3	4	Taiwan	23604000	36193	652
4	5	South Korea	51781000	99538	520
5	6	Rwanda	12374000	26338	470
6	7	Haiti	11578000	27065	428
7	8	Netherlands	17610000	41526	424
8	9	Israel	9360000	22072	424
9	10	India	1378250000	3287240	419

OTHER FILE FORMATS

Python Libraries can be helpful in identifying and reading different file types.

Some common file formats:

- .csv
- .json
- .xml
- excel
- hdf
- SQL

In [235…
```
# .CSV example
import pandas as pd

# get data
url ='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSk
df = pd.read_csv(url)
df
```

Out[235…

	John	Doe	120 jefferson st.	Riverside	NJ	08075
0	Jack	McGinnis	220 hobo Av.	Phila	PA	9119
1	John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	8075

	John	Doe	120 jefferson st.	Riverside	NJ	08075
2	Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234
3	NaN	Blankman	NaN	SomeTown	SD	298
4	Joan "the bone", Anne	Jet	9th, at Terrace plc	Desert City	CO	123

In [236…

```
# add columns
df. columns = ['first name', 'last name', 'location', 'city', 'state', 'area code']
df
```

Out[236…

	first name	last name	location	city	state	area code
0	Jack	McGinnis	220 hobo Av.	Phila	PA	9119
1	John "Da Man"	Repici	120 Jefferson St.	Riverside	NJ	8075
2	Stephen	Tyler	7452 Terrace "At the Plaza" road	SomeTown	SD	91234
3	NaN	Blankman	NaN	SomeTown	SD	298
4	Joan "the bone", Anne	Jet	9th, at Terrace plc	Desert City	CO	123

In [240…

```
# find name columns
# pass a list of column names
df[['first name', 'last name']]
```

Out[240…

	first name	last name
0	Jack	McGinnis
1	John "Da Man"	Repici
2	Stephen	Tyler
3	NaN	Blankman
4	Joan "the bone", Anne	Jet

In [241…

```
# alternatively, use loc and iloc
# first row
df.loc[0]
```

Out[241…

```
first name           Jack
last name         McGinnis
location      220 hobo Av.
city                Phila
state                  PA
area code            9119
Name: 0, dtype: object
```

In [242…

```
df.loc[[0,1,2], 'first name']
```

Out[242…

```
0            Jack
1    John "Da Man"
```

```
2          Stephen
Name: first name, dtype: object
```

In [243…

```
# To select the 0th,1st and 2nd row of "First Name" column only
df.iloc[[0,1,2], 0]
```

Out[243…

```
0             Jack
1    John "Da Man"
2          Stephen
Name: first name, dtype: object
```

In [244…

```
# the transform function returns a dataframe with transformed values after applying the
#import library
import pandas as pd
import numpy as np

#creating a dataframe
df=pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), columns=['a', 'b', 'c'])
df
```

Out[244…

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

In [245…

```
#applying the transform function
df = df.transform(func = lambda x : x + 10)
df
```

Out[245…

	a	b	c
0	11	12	13
1	14	15	16
2	17	18	19

In [246…

```
# find sqrt
result = df.transform(func = ['sqrt'])
result
```

Out[246…

	a	b	c
	sqrt	**sqrt**	**sqrt**
0	3.316625	3.464102	3.605551
1	3.741657	3.872983	4.000000
2	4.123106	4.242641	4.358899

In [247…

```
# .JSON example
```

```
import json

# serialization is the process of converting an object into a special format suitable f
# json uses dump() or dumps() function to convert python objects into json objects

person = {
    'first_name' : 'Mark',
    'last_name' : 'abc',
    'age' : 27,
    'address': {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

In [249…

```
# use dump function
with open('person.json', 'w') as f:  # writing JSON object
    json.dump(person, f)
person
```

Out[249…

```
{'first_name': 'Mark',
 'last_name': 'abc',
 'age': 27,
 'address': {'streetAddress': '21 2nd Street',
  'city': 'New York',
  'state': 'NY',
  'postalCode': '10021-3100'}}
```

In [251…

```
# Serializing json
json_object = json.dumps(person, indent = 4)

# Writing to sample.json
with open("sample.json", "w") as outfile:
    outfile.write(json_object)

print(json_object)

# the python object is now serialized to the file.
# deserialize it back to the python object by using load() function
```

```
{
    "first_name": "Mark",
    "last_name": "abc",
    "age": 27,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021-3100"
    }
}
```

In [252…

```
# deserialization using .load()
# Opening JSON file
with open('sample.json', 'r') as openfile:

    # Reading from json file
```

```
    json_object = json.load(openfile)

print(json_object)
print(type(json_object))
```

```
{'first_name': 'Mark', 'last_name': 'abc', 'age': 27, 'address': {'streetAddress': '21 2
nd Street', 'city': 'New York', 'state': 'NY', 'postalCode': '10021-3100'}}
<class 'dict'>
```

In [253…

```
# .XLSX example
import pandas as pd
import urllib.request

# load data
urllib.request.urlretrieve("https://cf-courses-data.s3.us.cloud-object-storage.appdomai
df = pd.read_excel("sample.xlsx")
df
```

Out[253…

	0	First Name	Last Name	Gender	Country	Age	Date	Id
0	1	Dulce	Abril	Female	United States	32	15/10/2017	1562
1	2	Mara	Hashimoto	Female	Great Britain	25	16/08/2016	1582
2	3	Philip	Gent	Male	France	36	21/05/2015	2587
3	4	Kathleen	Hanner	Female	United States	25	15/10/2017	3549
4	5	Nereida	Magwood	Female	United States	58	16/08/2016	2468
5	6	Gaston	Brumm	Male	United States	24	21/05/2015	2554
6	7	Etta	Hurn	Female	Great Britain	56	15/10/2017	3598
7	8	Earlean	Melgar	Female	United States	27	16/08/2016	2456
8	9	Vincenza	Weiland	Female	United States	40	21/05/2015	6548

In [255…

```
# use etree.elementtree module to parse and create .xlsx files
import xml.etree.ElementTree as ET

# create the file structure
employee = ET.Element('employee')
details = ET.SubElement(employee, 'details')
first = ET.SubElement(details, 'firstname')
second = ET.SubElement(details, 'lastname')
third = ET.SubElement(details, 'age')
first.text = 'Shiv'
second.text = 'Mishra'
third.text = '23'

# create a new XML file with the results
mydata1 = ET.ElementTree(employee)
# myfile = open("items2.xml", "wb")
# myfile.write(mydata)
with open("new_sample.xml", "wb") as files:
    mydata1.write(files)
```

Out[255…

```
<xml.etree.ElementTree.ElementTree at 0x21f66afc760>
```

In [256…

```
# reading .xlsx files
import pandas as pd

import xml.etree.ElementTree as etree

# get file
!wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSk
```

```
--2021-06-17 11:55:52--  https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cl
oud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Sample-employe
e-XML-file.xml
Resolving cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-data.s
3.us.cloud-object-storage.appdomain.cloud)... 198.23.119.245
Connecting to cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud (cf-courses-dat
a.s3.us.cloud-object-storage.appdomain.cloud)|198.23.119.245|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1016 [application/xml]
Saving to: 'Sample-employee-XML-file.xml'

     0K                                                       100%  510M=0s

2021-06-17 11:55:52 (510 MB/s) - 'Sample-employee-XML-file.xml' saved [1016/1016]
```

In [258…

```
# sample code to parse .xml file

tree = etree.parse("Sample-employee-XML-file.xml")

root = tree.getroot()
columns = ["firstname", "lastname", "title", "division", "building","room"]

datatframe = pd.DataFrame(columns = columns)

for node in root:

    firstname = node.find("firstname").text

    lastname = node.find("lastname").text

    title = node.find("title").text

    division = node.find("division").text

    building = node.find("building").text

    room = node.find("room").text

    datatframe = datatframe.append(pd.Series([firstname, lastname, title, division, bui

# view file
datatframe
```

Out[258…

	firstname	lastname	title	division	building	room
0	Shiv	Mishra	Engineer	Computer	301	11
1	Yuh	Datta	developer	Computer	303	02
2	Rahil	Khan	Tester	Computer	304	10
3	Deep	Parekh	Designer	Computer	305	14

In [259…
```
# you can save the dataset to CSV
datatframe.to_csv('employee.csv', index = False)
```

In [260…
```
# BINARY example
# this type includes image files, audio files, pdf etc
# use the PIL library

# importing PIL
from PIL import Image

import urllib.request
# Downloading dataset
urllib.request.urlretrieve("https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/image
```

Out[260… ('dog.jpg', <http.client.HTTPMessage at 0x21f66af7b50>)

In [261…
```
# Read image
img = Image.open('dog.jpg')

# Output Images
display(img)
```

In [262…
```
# data analysis
```

```
# Import pandas library
import pandas as pd

# data
path = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloper
df = pd.read_csv(path)

# show the first 5 rows using dataframe.head() method
print("The first 5 rows of the dataframe")
df.head(5)
```

```
The first 5 rows of the dataframe
```

Out[262…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	O
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [263…

```
# view dimensions
df.shape
```

Out[263…

```
(768, 9)
```

In [264…

```
# get info about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [265…

```
# get some basic statistics
df.describe()
```

Out[265…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	76
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreel
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [266…

```
# identify and handle missing values
# two methods for detecting missing data
# .isnull()
# .notnull()

missing_data = df.isnull()
missing_data.head(5)
```

Out[266…

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False

In [268…

```
# count missing values in each column
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
# false means the value is not null / not missing
# so we have 768 rows of info and 768 non-null rows.
# no missing information.
```

```
Pregnancies
False    768
Name: Pregnancies, dtype: int64

Glucose
False    768
Name: Glucose, dtype: int64

BloodPressure
False    768
Name: BloodPressure, dtype: int64

SkinThickness
False    768
Name: SkinThickness, dtype: int64
```

```
Insulin
False    768
Name: Insulin, dtype: int64

BMI
False    768
Name: BMI, dtype: int64

DiabetesPedigreeFunction
False    768
Name: DiabetesPedigreeFunction, dtype: int64

Age
False    768
Name: Age, dtype: int64

Outcome
False    768
Name: Outcome, dtype: int64
```

In [270…

```
# visualize the data with seaborn and matplotlib
# import libraries
import matplotlib.pyplot as plt
import seaborn as sns

# create plot
labels= 'Diabetic','Not Diabetic'
plt.pie(df['Outcome'].value_counts(),labels=labels,autopct='%0.02f%%')
plt.legend()
plt.show()
```