Tecnológico de Monterrey

Victor Hugo Avila Felipe - A01794425

IBM: PY0101EN Python for Data Science

It’s also interactive, object-oriented - Python 3

- first module: types, expressions, variables, strings and string operations.
- second :data structures, namely lists and tuples, sets and dictionaries. Lists and tuples, are called compound data types and are one of the key types of data structures in Python. Sets and dictionaries are types of collections.
- In the 3rd module we'll cover Python programming fundamentals, including conditions, branching, loops, functions, and objects and classes. Specifically, you’ll learn about comparison and logic operators, and the if, else, and elif statements, for and while loops, how to use some of Python’s built-in functions, as well as how build your own functions, and how to create objects and classes using object constructors and attributes.
- And finally, in the 4th module we'll show you how to work with data in Python. Specifically, how to read and write with the open method, and how to load, work with, and save data in pandas.

  A statement or expression is an instruction the computer will run or execute. The value in the parentheses is called the argument. If you are using a Jupiter notebook in this course, you will see a small rectangle with the statement. This is called a cell.

11 - int 21.213 - float "hello" - str true - boolean

true - 1 False - 0 type("str")

type casting float(2) int(1.1)

operands 56 78569 46 78 operators *- - /- // +

variables my_variable = 1

Strings are sequences and, as such, have apply methods that work on lists and tuples. Strings also have a second set of methods that just work on strings.

the method upper. This method converts lower case characters to upper case characters.

```
In [1]: print('Hello Python 101')

        Hello Python 101

In [3]: #Imprimir Hola mundo
        print("Hello World")

        Hello World

In [2]: print("Hello\nWorld!")

        Hello
        World!

In [3]: # print('Hello World!')

In [1]: type(11)

Out[1]: int

In [2]: type(21.45)

Out[2]: float

In [3]: type("str")

Out[3]: str

In [10]: float(2)

Out[10]: 2.0

In [8]: print(int("1"))

        1

In [9]: int("1")

Out[9]: 1

In [11]: bool(int("1"))

Out[11]: True

In [12]: my_variable = 1

In [13]: my_variable + 1

Out[13]: 2
```

In [14]:
```
import sys
print(sys.version)
```
```
3.7.0 (default, Jun 28 2018, 13:15:42)
[GCC 7.2.0]
```

In [12]:
```
print("Hello, Python!")
```
```
Hello, Python!
```

In [16]:
```
print("This will be printed")
frint("This will cause an error")
print("This will NOT be printed")
```
```
This will be printed

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-aa3f0d14d353> in <module>()
      1 print("This will be printed")
----> 2 frint("This will cause an error")
      3 print("This will NOT be printed")

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

In [17]:
```
print("Hello, world!")
```
```
Hello, world!
```

In [18]:
```
type(12.0)
```
Out[18]:
```
float
```

In [19]:
```
sys.float_info
```
Out[19]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min
=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=5
3, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

In [22]:
```
type(float(2))
```
Out[22]:
```
float
```

In [23]:
```
type(float(2))
```
Out[23]:
```
float
```

In [24]:
```
type(6/2) # float
```
Out[24]:
```
float
```

In [25]:
```
type(6//2)
```
Out[25]:
```
int
```

```
In [26]: 6/2
Out[26]: 3.0
```

```
In [27]: 160//60
Out[27]: 2
```

```
In [28]: total_min = 43 + 42 + 57 # Total length of albums in minutes
         total_min
Out[28]: 142
```

```
In [29]: x = 3 + 2 * 2
         x
Out[29]: 7
```

```
In [30]: y = (3 + 2) * 2
         y
Out[30]: 10
```

```
In [31]: Letters="ABCDEFGHIJK"
         Letters[0:4]
Out[31]: 'ABCD'
```

```
In [32]: Good="GsoAo+d"
         Good[::2]
Out[32]: 'Good'
```

```
In [33]: "uppercase"
         "uppercase".upper()
Out[33]: 'UPPERCASE'
```

```
In [34]: "0123456".find('1')
Out[34]: 1
```

```
In [35]: name = "Michael Jackson"
         name.find('el')
Out[35]: 5
```

```
In [36]: name.find('Jasdfasdasdf')
Out[36]: -1
```

```
In [37]: a = "1"
         a
Out[37]: '1'
```

```
In [38]: b = "2"
         b
Out[38]: '2'
```

In [39]:
```
c = a + b
```
Out[39]:
```
'12'
```

In [40]:
```
d = "ABCDEFG"
```
```
ABC
```

In [41]:

In [42]:
```
correct
```

In [43]:

Out[43]:
```
7
```

In [45]:
```
name = 'Lizz'
```
```
Li
```

In [46]:

In [48]:
```
0246
```

In [49]:

Out[49]:
```
'12'
```

In [50]:

In [51]:

Out[51]:
```
<function str.upper()>
```

In [52]:

Out[52]:
```
'HELLO'
```

In [53]:

Out[53]:
```
'HELLO'
```

MODULE 2

Tuples are an ordered sequence. Here is a Tuple “Ratings.” Tuples are expressed as comma-separated elements within parentheses. These are values inside the parentheses. Tuple[1]

TUPLE A = (0,1,2,3) print(A)

Lists are also an ordered sequence. Here is a list L. A list is represented with square brackets. In many respects lists are like tuples, one key difference is they are mutable.

LIST B=["a","b","c"]

Sets are a type of collection. This means that like lists and tuples, you can input different python types. Unlike lists and tuples they are unordered. This means sets do not record element position. Sets only have unique elements. This means there is only one of a particular element in a set.

Dictionaries are a type of collection in Python. If you recall, a list has integer indexes. These are like addresses. A list also has elements. A dictionary has keys and values. The key is analogous to the index, they are like addresses but they don't have to be integers. They are usually characters; the values are similar to the elements in a list and contain information.

```
In [55]: A=(0,1,2,3)

In [56]: A

Out[56]: (0, 1, 2, 3)

In [57]: print(A)

         (0, 1, 2, 3)

In [58]: A:[0,2]

In [60]: A:[0,2]

In [61]: A

Out[61]: (0, 1, 2, 3)

In [62]: A[0:2]

Out[62]: (0, 1)

In [63]: B ["a","b","c"]

In [64]: B[0:2]

Out[64]: ['a', 'b']

In [66]: B=["a","b","c"]

In [68]: B[1:]

Out[68]: ['b', 'c']
```

```
In [69]: genres_tuple = ("pop", "rock", "soul", "hard rock", "soft rock", \
                         "R&B", "progressive rock", "disco")
         genres_tuple
```

```
Out[69]: ('pop',
          'rock',
          'soul',
          'hard rock',
          'soft rock',
          'R&B',
          'progressive rock',
          'disco')
```

```
In [71]: len(genres_tuple)
```

```
Out[71]: 8
```

```
In [72]: genres_tuple[3]
```

```
Out[72]: 'hard rock'
```

```
In [75]: genres_tuple[3:6]
```

```
Out[75]: ('hard rock', 'soft rock', 'R&B')
```

```
In [77]: genres_tuple[0:3]
```

```
Out[77]: ('pop', 'rock', 'soul')
```

```
In [79]: genres_tuple[0:2]
```

```
Out[79]: ('pop', 'rock')
```

```
In [81]: genres_tuple[-1]
```

```
Out[81]: 'disco'
```

```
In [82]: genres_tuple.index("disco")
```

```
Out[82]: 7
```

```
In [83]: C_tuple = (-5, 1, -3)
         C_list = sorted(C_tuple)
         C_list
```

```
Out[83]: [-5, -3, 1]
```

```
In [85]: L = ["Michael Jackson", 10.1, 1982]
         L
```

```
Out[85]: ['Michael Jackson', 10.1, 1982]
```

```
In [87]: L = ["Michael Jackson", 10.1,1982,"MJ",1]
         L
```

```
Out[87]: ['Michael Jackson', 10.1, 1982, 'MJ', 1]
```

In [88]:
```
L[3:5]
```
Out[88]:
```
['MJ', 1]
```

In [89]:
```
# Use append to add elements to list

L = [ "Michael Jackson", 10.2]
L.append(['pop', 10])
L
```
Out[89]:
```
['Michael Jackson', 10.2, ['pop', 10]]
```

In [90]:
```
L = [ "Michael Jackson", 10.2]
L.extend(['pop', 10])
L
```
Out[90]:
```
['Michael Jackson', 10.2, 'pop', 10]
```

In [91]:
```
# Use append to add elements to list

L.append(['a','b'])
L
```
Out[91]:
```
['Michael Jackson', 10.2, 'pop', 10, ['a', 'b']]
```

In [92]:
```
# Copy (copy by reference) the list A

A = ["hard rock", 10, 1.2]
B = A
print('A:', A)
print('B:', B)
```
```
A: ['hard rock', 10, 1.2]
B: ['hard rock', 10, 1.2]
```

In [93]:
```
# Examine the copy by reference

print('B[0]:', B[0])
A[0] = "banana"
print('B[0]:', B[0])
```
```
B[0]: hard rock
B[0]: banana
```

In [95]:
```
print('A[0]:', A[0])
```
```
A[0]: banana
```

In [96]:
```
# Clone (clone by value) the list A

B = A[:]
B
```
Out[96]:
```
['banana', 10, 1.2]
```

In [97]:
```
a_list = [1,'hello',[1,2,3],'True']
```

```
In [98]: a
```

```
Out[98]: '1'
```

```
In [99]: a_list[1]
```

```
Out[99]: 'hello'
```

```
In [100]: a_list[1:4]
```

```
Out[100]: ['hello', [1, 2, 3], 'True']
```

```
In [101]: A = [1, 'a']
          B = [2, 1, 'd']
          A + B
```

```
Out[101]: [1, 'a', 2, 1, 'd']
```

```
In [102]: 


          S={'A','B','C'}

          U={'A','Z','C'}

          U.union(S)
```

```
Out[102]: {'A', 'B', 'C', 'Z'}
```

```
In [103]: D = {'a':0,'b':1,'c':2}
```

```
In [104]: D.values()
```

```
Out[104]: dict_values([0, 1, 2])
```

```
In [105]: D = {'a':0,'b':1,'c':2}
```

```
In [ ]: 
```

```
In [106]: D['b']
```

```
Out[106]: 1
```

```
In [107]: A = ('a','b','c')
```

```
In [109]: A[1]
```

```
Out[109]: 'b'
```

```
In [110]: L.append(['a','b'])
In [111]: '
Out[111]: ['Michael Jackson', 10.2, 'pop', 10, ['a', 'b'], ['a', 'b']]
```

```
In [112]: Dict={"A":1,"B":"2","C":[3,3,3],"D":(4,4,4),'E':5,'F':6}
```

```
In [113]: Dict["D"]
Out[113]: (4, 4, 4)
```

MODULE 3

Comparison operations compare some value or operand, then, based on some condition they produce a Boolean. /= <

> =

The inequality test uses an explanation mark preceding the equal sign, if two operands are not equal, then the condition becomes true. We can use a number line. When the condition is true the corresponding numbers are marked in green and red for where the condition is false. i!=

Branching allows us to run different statements for a different input. It's helpful to think of an “if statement” as a locked room: If the statement is true, you can enter the room, and your program can run some predefined task. If the statement is false, your program will skip the task.

The “else statement” will run a different block of code, if the same condition is false.

The elif statement, short for “else if,” allows us to check additional conditions, if the proceeding condition is false. If the condition is true, the alternate expressions will be run.

Logic operations take Boolean values and produce different Boolean values. The first operation is the not operator. If the input is true, the result is a false. Similarly, if the input is false, the result is a true.

QUESTION 1 1 point possible (ungraded)

Select the values of i that produces a True for the following:

```
In [5]: i = 1
        i!=0
Out[5]: True
```

```
In [6]: i = 0
        i!=0
Out[6]: False
```

```
In [7]: i = -1
        i!=0
Out[7]: True
```

In [8]:
```
i = 100000000
i!=0
```
Out[8]:
```
True
```

QUESTION 2 1 point possible (ungraded)

What is the output of the following:

In [10]:
```
x='a'

if(x!='a'):

    print("This is not a.")

else:

    print("This is a.")
```
```
This is a.
```

Quiz on Conditions

Write an if statement to determine if an album had a rating greater than 8. Test it using the rating for the album **"Back in Black"** that had a rating of 8.5. If the statement is true print "This album is Amazing!"

In [11]:
```
rating = 8.5
if rating > 8:
    print ("This album is Amazing!")
```
```
This album is Amazing!
```

Write an if-else statement that performs the following. If the rating is larger then eight print "this album is amazing". If the rating is less than or equal to 8 print "this album is ok".

In [13]:
```
rating = 8.5
if rating > 8:
    print ("this album is amazing")
else:
    print ("this album is ok")
```
```
this album is amazing
```

Write an if statement to determine if an album came out before 1980 or in the years: 1991 or 1993. If the condition is true print out the year the album came out.

In [14]:
```
album_year = 1979

if album_year < 1980 or album_year == 1991 or album_year == 1993:
    print("This album came out in year", album_year)
```
```
This album came out in year 1979
```

LOOPS The range function outputs an ordered sequence as a list “i”. If the input is a positive integer, the output is a sequence; the sequence contains the same number of elements as the input, but starts at zero.

Each element in the list is a string representing the color. We want to change the name of the color in each element to white. Each element in the list has the following index. This is the syntax to perform a loop in Python. Notice the indent. The range function generates a list. The code will simply repeat everything in the indent 5 times. If you were to change the value to 6, it would do it 6 times, however, the value of “i” is incremented by one each time.

ENUMERATE()

“While loops” are similar to “for loops”, but instead of executing a statement a set number of times, a while loop will only run if a condition is met.

QUESTION 1 1 point possible (ungraded)

What is the output of the following lines of code:

In [16]:
```
A=[3,4,5]

for a in A:

    print(a)
```

```
3
4
5
```

QUESTION 2 1 point possible (ungraded)

What is the output of the following lines of code:

In [1]:
```
x=3

y=1

while(y!=x):

    print(y)
    y=y+1
```

```
1
2
```

Loops

Range

Sometimes, you might want to repeat a given operation many times. Repeated executions like this are performed by **loops**. We will look at two types of loops, `for` loops and `while` loops.

Before we discuss loops lets discuss the `range` object. It is helpful to think of the range object as an ordered list. For now, let's look at the simplest case. If we would like to generate an object

In [3]:
```
# Use the range
```

Out[3]:
```
range(0, 3)
```

What is for loop?

The for loop enables you to execute a code block multiple times. For example, you would use this if you would like to print out every element in a list. Let's try to use a for loop to print all the years presented in the list dates:

This can be done as follows:

In [4]:
```
# For loop example

dates = [1982,1980,1973]
N = len(dates)

for i in range(N):
```

```
1982
1980
1973
```

What is `while` loop?

As you can see, the `for` loop is used for a controlled flow of repetition. However, what if we don't know when we want to stop the loop? What if we want to keep executing a code block until a certain condition is met? The `while` loop exists as a tool for repeated execution based on a condition. The code block will keep being executed until the given logical condition returns a **False** boolean value.

Let’s say we would like to iterate through list `dates` and stop at the year 1973, then print out the number of iterations. This can be done with the following block of code:

In [5]:
```
# While Loop Example

dates = [1982, 1980, 1973, 2000]

i = 0
year = dates[0]

while(year != 1973):
    print(year)
    i = i + 1
    year = dates[i]


print("It took ", i, "repetitions to get out of loop.")
```

```
1982
1980
It took  2 repetitions to get out of loop.
```

Quiz on Loops

Write a `for` loop the prints out all the element between **-5** and **5** using the range function.

In [6]:
```
for i in range(-4, 5):
    print(i)
```

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

Print the elements of the following list: `Genres=[ 'rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']` Make sure you follow Python conventions.

In [7]:
```
Genres = ['rock', 'R&B', 'Soundtrack', 'R&B', 'soul', 'pop']
for Genre in Genres:
    print(Genre)
```

```
rock
R&B
Soundtrack
R&B
soul
pop
```

Write a for loop that prints out the following list: `squares=['red', 'yellow', 'green', 'purple', 'blue']`

In [8]:
```
squares=['red', 'yellow', 'green', 'purple', 'blue']
for square in squares:
    print(square)
```

```
red
yellow
green
purple
blue
```

Write a while loop to display the values of the Rating of an album playlist stored in the list `PlayListRatings` . If the score is less than 6, exit the loop. The list `PlayListRatings` is given by: `PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]`

In [9]:
```
PlayListRatings = [10, 9.5, 10, 8, 7.5, 5, 10, 10]
i = 0
Rating = PlayListRatings[0]
while(i < len(PlayListRatings) and Rating >= 6):
    print(Rating)
    i = i + 1 # This prints the value 10 only once
    Rating = PlayListRatings[i]
    i = i + 1 #Try uncommenting the line and comment the previous i = i + 1, a
```

```
10
9.5
8
```

Write a while loop to copy the strings `'orange'` of the list `squares` to the list `new_squares` . Stop and exit the loop if the value on the list is not `'orange'` :

In [10]:
```
squares = ['orange', 'orange', 'purple', 'blue ', 'orange']
new_squares = []
i = 0
while(i < len(squares) and squares[i] == 'orange'):
    new_squares.append(squares[i])
    i = i + 1
print (new_squares)
```

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

FUNCTIONS

Functions take some input then produce some output or change. The function is just a piece of code you can reuse. You can implement your own function, but in many cases, you use other people's functions.

QUESTION 1 1 point possible (ungraded)

What is the value of c after the following block of code is run ?

In [18]:
```
a=1

def add(b):

    return a+b

c = add(10)
```

```
11
```

QUESTION 2 1 point possible (ungraded)

What is the value of c after the following block of code is run with proper numerical input?

In [20]:
```
def f(*x):
    return sum(x)
c=f(c)
```

```
11
```

Return the total of a variable amount of parameters.

What is a Function?

You can define functions to provide the required functionality. Here are simple rules to define a function in Python:

- 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.

An example of a function that adds on to the parameter `a` prints and returns the output as `b` :

In [22]:
```
# First function example: Add 1 to a and store as b

def add(a):
    """
    add 1 to a
    """
    b = a + 1
    print(a, "if you add one", b)
```

In [23]:
```
# Call the function add()

```

```
2 if you add one 3
```

Out[23]:
```
3
```

Quiz on Functions

Come up with a function that divides the first input by the second input:

In [ ]:
```
def div(a, b):
    return(a/b)
```

Use the function con for the following question.

In [24]:
```
# Use the con function for the following question

def con(a, b):
    return(a + b)
```

Can the `con` function we defined before be used to add two integers or strings?

In [25]:
```
con(2, 2)
```

Out[25]:
```
4
```

Can the `con` function we defined before be used to concatenate lists or tuples?

In [26]:
```
con(['a', 1], ['b', 1])
```

Out[26]:
```
['a', 1, 'b', 1]
```

Objects and Classes

Python has many different kinds of data types: Integers, Floats, Strings, Lists, Dictionaries, Booleans. In Python each is an object. Every object has the following: a type, internal representation, a set of functions called methods to interact with the data. An object is an instance of a particular type.

Every object has the following: a type, internal representation, a set of functions called methods to interact with the data. An object is an instance of a particular type.

We can find out the type of an object by using the type command.

METHODS

A class or type's methods are functions that every instance of that class or type provides. It's how you interact with the object. Sorting is an example of a method that interacts with the data in the object.

You can create your own type or class in Python. In this section, you will create a class. The class has data attributes. The class has methods. We then create an instances or instances of that class or objects. The class data attributes define the class.

You will need the class Car for the next exercises. The class Car has four data attributes: make, model, colour and number of owners (owner_number). The method `car_info()` prints out the data attributes and the method sell() increments the number of owners by 1

In [4]:
```python
class Car(object):
    def __init__(self,make,model,color):
        self.make=make;
        self.model=model;
        self.color=color;
        self.owner_number=0
    def car_info(self):
        print("make: ",self.make)
        print("model:", self.model)
        print("color:",self.color)
        print("number of owners:",self.owner_number)
    def sell(self):
        self.owner_number=self.owner_number+1
```

Create a Car Object

Create a `Car` object "my_car" with the given data attributes:

In [5]:
```python
make="BMW"
model="M3"
color="red"
my_car = Car(make,model,color)
```

Data Attributes

Use the method `car_info()` to print out the data attributes.

In [7]:
```python
my_car.car_info()
```

```
make:  BMW
model: M3
color: red
number of owners: 0
```

Methods

Call the method `sell()` in the loop, then call the method `car_info()` again

In [8]:
```python
for i in range(5):
    my_car.sell()

my_car.car_info()
```

```
make:  BMW
model: M3
color: red
number of owners: 5
```

QUESTION 1 1 point possible (ungraded)

Using the Class Car in the lab, create a Car object with the following attributes:

In [10]:
```
class Car(object):
    def __init__(self,make,model,color):
        self.make=make;
        self.model=model;
        self.color=color;
        self.owner_number=0
    def car_info(self):
        print("make: ",self.make)
        print("model:", self.model)
        print("color:",self.color)
        print("number of owners:",self.owner_number)
    def sell(self):
        self.owner_number=self.owner_number+1

make="BMW"
model="M3"
color="red"
my_car = Car(make,model,color)

print(my_car)
```

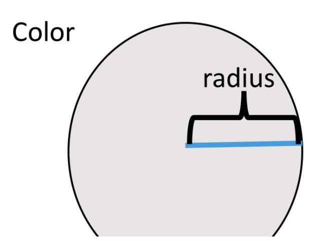
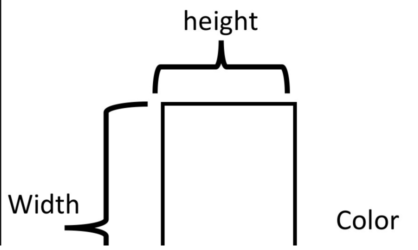
```
<__main__.Car object at 0x7f1c6afd48d0>
```

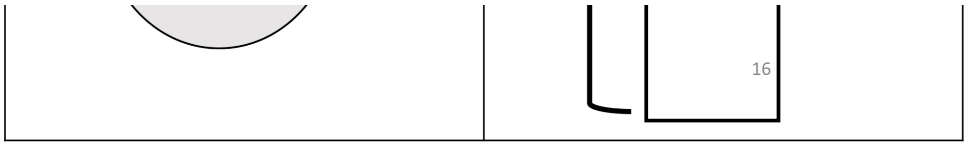
In [ ]:
```
make="Honda"

model="Accord"

color="blue"
```

Introduction to Classes and Objects

Creating a Class

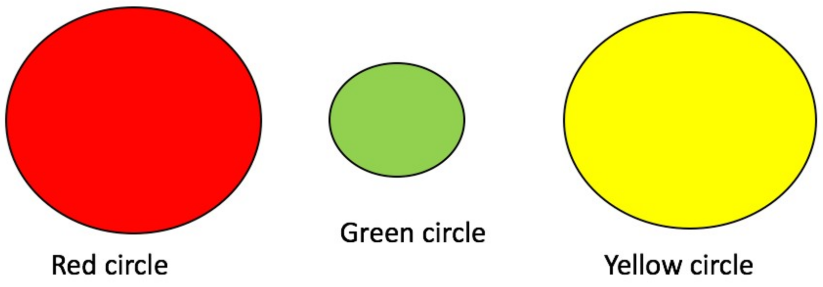
The first step in creating a class is giving it a name. In this notebook, we will create two classes: Circle and Rectangle. We need to determine all the data that make up that class, which we call *attributes*. Think about this step as creating a blue print that we will use to create objects. In figure 1 we see two classes, Circle and Rectangle. Each has their attributes, which are variables. The class Circle has the attribute radius and color, while the Rectangle class has the attribute height and width. Let's use the visual examples of these shapes before we get to the code, as this will help you get accustomed to the vocabulary.

Class Circle	**Class Rectangle**
Attributes: radius, Color	Attributes: Color, height and Width
	



Instances of a Class: Objects and Attributes

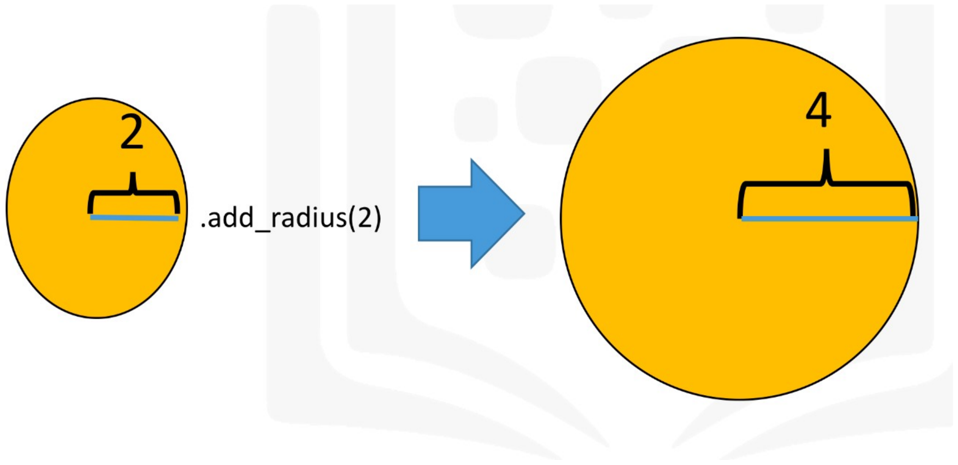
An instance of an object is the realisation of a class, and in Figure 2 we see three instances of the class circle. We give each object a name: red circle, yellow circle, and green circle. Each object has different attributes, so let's focus on the color attribute for each object.



The colour attribute for the red Circle is the colour red, for the green Circle object the colour attribute is green, and for the yellow Circle the colour attribute is yellow.

Methods

Methods give you a way to change or interact with the object; they are functions that interact with objects. For example, let’s say we would like to increase the radius of a circle by a specified amount. We can create a method called **add_radius(r)** that increases the radius by **r**. This is shown in figure 3, where after applying the method to the "orange circle object", the radius of the object increases accordingly. The “dot” notation means to apply the method to the object, which is essentially applying a function to the information in the object.

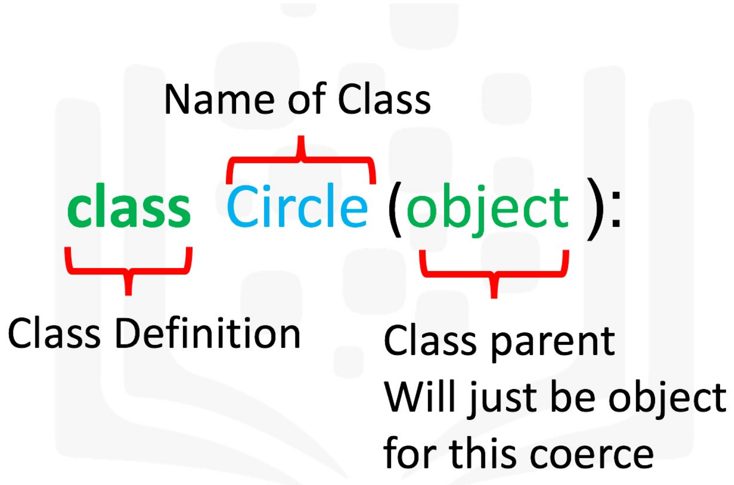


Creating a Class

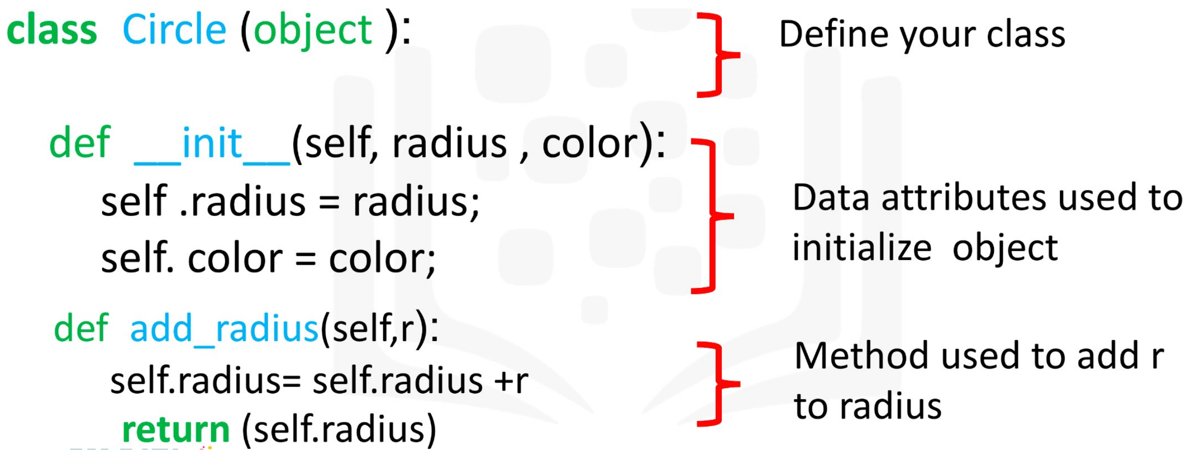
Now we are going to create a class Circle, but first, we are going to import a library to draw the objects:

In [11]:
```
# Import the library

import matplotlib.pyplot as plt
```

The first step in creating your own class is to use the `class` keyword, then the name of the class as shown in Figure 4. In this course the class parent will always be object:



The next step is a special method called a constructor `_init_\` , which is used to initialize the object. The inputs are data attributes. The term `self` contains all the attributes in the set. For example the `self.color` gives the value of the attribute color and `self.radius` will give you the radius of the object. We also have the method `add_radius()` with the parameter `r` , the method adds the value of `r` to the attribute radius. To access the radius we use the syntax `self.radius` . The labeled syntax is summarized in Figure 5:



class Circle (object):
Define your class
def __init__(self, radius , color):
self .radius = radius;
self. color = color;
Data attributes used to initialize object
def add_radius(self,r):
self.radius= self.radius +r
return (self.radius)
Method used to add r to radius

The actual object is shown below. We include the method `drawCircle` to display the image of a circle. We set the default radius to 3 and the default colour to blue:

In [12]:
```
# Create a class Circle

class Circle(object):

    # Constructor
    def __init__(self, radius=3, color='blue'):
        self.radius = radius
        self.color = color

    # Method
    def add_radius(self, r):
        self.radius = self.radius + r
        return(self.radius)

    # Method
    def drawCircle(self):
        plt.gca().add_patch(plt.Circle((0, 0), radius=self.radius, fc=self.col
        plt.axis('scaled')
        plt.show()
```

Creating an instance of a class Circle

Let’s create the object `RedCircle` of type Circle to do the following:

In [13]:
```
# Create an object RedCircle

RedCircle = Circle(10, 'red')
```

We can use the `dir` command to get a list of the object's methods. Many of them are default Python methods.

In [14]:
```
# Find out the methods can be used on the object RedCircle

dir(RedCircle)
```

Out[14]:
```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'add_radius',
 'color',
 'drawCircle',
 'radius']
```

In [15]:
```
# Print the object attribute radius

RedCircle.radius
```

Out[15]:
```
10
```

In [16]:
```
# Print the object attribute color

RedCircle.color
```

Out[16]:
```
'red'
```

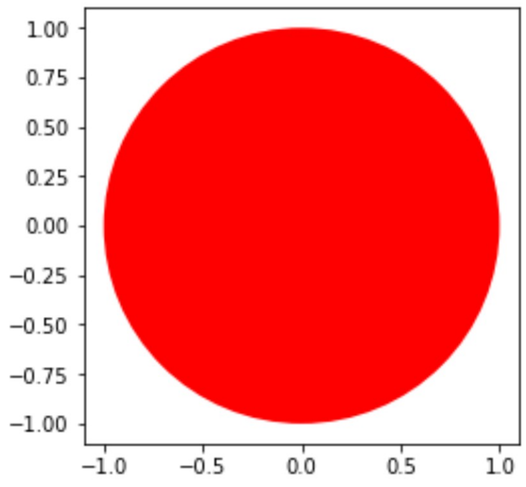
We can change the object's data attributes:

In [17]:
```
# Set the object attribute radius

RedCircle.radius = 1
RedCircle.radius
```

Out[17]:
```
1
```

In [ ]:
```
We can draw the object by using the method <code>drawCircle()</code>:
```

In [18]:
```python
# Call the method drawCircle

RedCircle.drawCircle()
```



We can increase the radius of the circle by applying the method `add_radius()` . Let's increases the radius by 2 and then by 5:

In [19]:
```python
# Use method to change the object attribute radius

print('Radius of object:',RedCircle.radius)
RedCircle.add_radius(2)
print('Radius of object of after applying the method add_radius(2):',RedCircle
RedCircle.add_radius(5)
print('Radius of object of after applying the method add_radius(5):',RedCircle
```

```
Radius of object: 1
Radius of object of after applying the method add_radius(2): 3
Radius of object of after applying the method add_radius(5): 8
```

Exercises

Text Analysis
You have been recruited by your friend, a linguistics enthusiast, to create a utility tool that can perform analysis on a given piece of text. Complete the class 'analysedText' with the following methods -

- Constructor (__init__) - This method should take the argument `text` , make it lower case, and remove all punctuation. Assume only the following punctuation is used: period (.), exclamation mark (!), comma (,) and question mark (?). Assign this newly formatted text to a new attribute called `fmtText` .
- freqAll - This method should create and **return** dictionary of all unique words in the text, along with the number of times they occur in the text. Each key in the dictionary should be the unique word appearing in the text and the associated value should be the number of times it occurs in the text. Create this dictionary from the `fmtText` attribute.
- freqOf - This method should take a word as an argument and **return** the number of occurrences of that word in `fmtText` .

The skeleton code has been given to you. Docstrings can be ignored for the purpose of the exercise.
Hint: Some useful functions are `replace()`, `lower()`, `split()`, `count()`

In [20]:
```
class analysedText(object):

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

        # 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
```

You can run the code cell below to test your functions to ensure they are working correctly. First execute the code cell in which you implemented your solution, then execute the code cell to test your implementation.

In [21]:
```
import sys

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

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

print("Constructor: ")
try:
    samplePassage = analysedText("Lorem ipsum dolor! diam amet, consetetur Lor
    print(testMsg(samplePassage.fmtText == "lorem ipsum dolor diam amet conset
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  " )
```

```
Constructor:
Test Passed
freqAll:
Test Passed
freqOf:
Test Passed
```

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.

Examples

In [22]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-22-9e1622b385b6> in <module>()
----> 1 1/0

ZeroDivisionError: division by zero
```

`ZeroDivisionError` occurs when you try to divide by zero.

In [24]:
```
y = a + 5
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-6ddcec040107> in <module>()
----> 1 y = a + 5

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

`NameError` -- in this case, it means that you tried to use the variable a when it was not defined.

In [25]:
```
a = [1, 2, 3]
a[10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-25-3f911ca4e3d3> in <module>()
      1 a = [1, 2, 3]
----> 2 a[10]

IndexError: list index out of range
```

`IndexError` -- in this case, it occured because you tried to access data from a list using an index that does not exist for this list.

Exception Handling

A `try except` will allow you to execute code that might raise an exception and in the case of any exception or a specific one we can handle or catch the exception and execute specific code. This will allow us to continue the execution of our program even if there is an exception.

Python tries to execute the code in the `try` block. In this case if there is any exception raised by the code in the `try` block, it will be caught and the code block in the `except` block will be executed. After that, the code that comes *after* the try except will be executed.

In [ ]:
```
# potential code before try catch

try:
    # code to try to execute
except:
    # code to execute if there is an exception

# code that will still execute if there is an exception
```

In [27]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except:
    print("There was an error")
```

```
Please enter a number to divide a2
Success a= 0.5
```

Try Except Specific

A specific `try except` allows you to catch certain exceptions and also execute certain code depending on the exception. This is useful if you do not want to deal with some exceptions and the execution should halt. It can also help you find errors in your code that you might not be aware of. Furthermore, it can help you differentiate responses to different exceptions. In this case, the code after the try except might not run depending on the error.

In [28]:
```
# potential code before try catch

try:
    # code to try to execute
except (ZeroDivisionError, NameError):
    # code to execute if there is an exception of the given types

# code that will execute if there is no exception or a one that we are handlin
```

```
  File "<ipython-input-28-e61cf3848eb0>", line 5
    except (ZeroDivisionError, NameError):
         ^
IndentationError: expected an indented block
```

In [29]:
```
# potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError

# code that will execute if there is no exception or a one that we are handlin
```

```
  File "<ipython-input-29-22d931ca9af7>", line 5
    except ZeroDivisionError:
         ^
IndentationError: expected an indented block
```

In [30]:
```
# potential code before try catch

try:
    # code to try to execute
except ZeroDivisionError:
    # code to execute if there is a ZeroDivisionError
except NameError:
    # code to execute if there is a NameError
except:
    # code to execute if ther is any exception

# code that will execute if there is no exception or a one that we are handlin
```

```
  File "<ipython-input-30-a22ac9e35695>", line 5
    except ZeroDivisionError:
         ^
IndentationError: expected an indented block
```

In [34]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
    print("Success a=",a)
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
```

```
Please enter a number to divide aq
You did not provide a number
```

Try Except Else and Finally Example

`else` allows one to check if there was no exception when executing the try block. This is useful when we want to execute something only if there were no errors.

In [35]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
```

```
Please enter a number to divide a1
success a= 1.0
```

In [36]:
```
a = 1

try:
    b = int(input("Please enter a number to divide a"))
    a = a/b
except ZeroDivisionError:
    print("The number you provided cant divide 1 because it is 0")
except ValueError:
    print("You did not provide a number")
except:
    print("Something went wrong")
else:
    print("success a=",a)
finally:
    print("Processing Complete")
```

```
Please enter a number to divide a0
The number you provided cant divide 1 because it is 0
Processing Complete
```

Question 1 What is the output of the following lines of code:

In [37]:
```
x=1

if(x!=1):

    print('Hello')

else:

    print('Hi')

print('Mike')
```

```
Hi
Mike
```

In [ ]:
```
Question 2
What is the output of the following few lines of code?
```

In [38]:
```
A = ['1','2','3']

for a in A:

    print(2*a)
```

```
11
22
33
```

Question 3 1 point possible (graded)

Consider the function Delta, when will the function return a value of 1

In [39]:
```


def Delta(x):

    if x==0:

        y=1;

    else:

        y=0;

return(y)
```

```
  File "<ipython-input-39-8da9de6afdb4>", line 13
    return(y)
             ^
SyntaxError: 'return' outside function
```

Question 4

What is the correct way to sort the list 'B' using a method? The result should not return a new list, just change the list 'B'.

In [47]:
```
B=['red', 'yellow', 'green', 'purple', 'blue']
```

In [48]:
```
B.sort()
```

In [49]:
```
B
```

Out[49]:
```
['blue', 'green', 'purple', 'red', 'yellow']
```

Question 5

What are the keys of the following dictionary: {'a':1,'b':2}?

MODULE 4

We will use Python's open function to get a file object. We can apply a method to that object to read data from the file. We can open the file Example1 ". txt"

Exercise 1: Pandas: DataFrame and Series

Pandas is a popular library for data analysis built on top of the Python programming language. Pandas generally provide two data structures for manipulating data, They are:

- DataFrame
- Series

A **DataFrame** is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

- A Pandas DataFrame will be created by loading the datasets from existing storage.
- Storage can be SQL Database, CSV file, an Excel file, etc.
- It can also be created from the lists, dictionary, and from a list of dictionaries.

Series represents a one-dimensional array of indexed data. It has two main components :

1. An array of actual data.
2. An associated array of indexes or data labels.

The index is used to access individual data values. You can also get a column of a dataframe as a **Series**. You can think of a Pandas series as a 1-D dataframe.

In [50]:
```
# Let us import the Pandas Library
import pandas as pd
```

In [51]:
```
#Define a dictionary 'x'

x = {'Name': ['Rose','John', 'Jane', 'Mary'], 'ID': [1, 2, 3, 4], 'Department'
      'Salary':[100000, 80000, 50000, 60000]}

#casting the dictionary to a DataFrame
df = pd.DataFrame(x)

#display the result df
df
```

Out[51]:

	Name	ID	Department	Salary
0	Rose	1	Architect Group	100000
1	John	2	Software Group	80000
2	Jane	3	Design Team	50000
3	Mary	4	Infrastructure	60000

Column Selection:

To select a column in Pandas DataFrame, we can either access the columns by calling them by their columns name.

Let's Retrieve the data present in the `ID` column.

In [53]:
```
#Retrieving the "ID" column and assigning it to a variable x
x = df[['ID']]
```

Out[53]:

	ID
0	1
1	2
2	3
3	4

In [54]:
```
#check the type of x
```

Out[54]:
```
pandas.core.frame.DataFrame
```

In [55]:
```
#Retrieving the Department, Salary and ID columns and assigning it to a variab

z = df[['Department','Salary','ID']]
```

Out[55]:

	Department	Salary	ID
0	Architect Group	100000	1
1	Software Group	80000	2
2	Design Team	50000	3
3	Infrastructure	60000	4

Problem 1: Create a dataframe to display the result as below:

In [56]:
```
a = {'Student':['David', 'Samuel', 'Terry', 'Evan'],
     'Age':['27', '24', '22', '32'],
     'Country':['UK', 'Canada', 'China', 'USA'],
     'Course':['Python','Data Structures','Machine Learning','Web Development'
     'Marks':['85','72','89','76']}
df1 = pd.DataFrame(a)
```

Out[56]:

	Student	Age	Country	Course	Marks
0	David	27	UK	Python	85
1	Samuel	24	Canada	Data Structures	72
2	Terry	22	China	Machine Learning	89
3	Evan	32	USA	Web Development	76

Problem 2: Retrieve the Marks column and assign it to a variable b

In [57]:
```
b = df1[['Marks']]
```

Out[57]:

	Marks
0	85
1	72
2	89
3	76

Problem 3: Retrieve the Country and Course columns and assign it to a variable c

In [58]:
```
c = df1[['Country','Course']]
c
```

Out[58]:

	Country	Course
0	UK	Python
1	Canada	Data Structures
2	China	Machine Learning
3	USA	Web Development

In [59]:
```
# Get the Name column as a series Object

x = df['Name']
x
```

Out[59]:
```
0    Rose
1    John
2    Jane
3    Mary
Name: Name, dtype: object
```

In [60]:
```
#check the type of x
type(x)
```

Out[60]:
```
pandas.core.series.Series
```

Exercise 2: `loc()` and `iloc()` functions

`loc()` is a label-based data selecting method which means that we have to pass the name of the row or column that we want to select. This method includes the last element of the range passed in it.

Simple syntax for your understanding:

- loc[row_label, column_label]

`iloc()` is an indexed-based selecting method which means that we have to pass integer index in the method to select a specific row/column. This method does not include the last element of the range passed in it.

Simple syntax for your understanding:

- iloc[row_index, column_index]

Let us see some examples on the same.

In [61]:
```
# Access the value on the first row and the first column

df.iloc[0, 0]
```

Out[61]:
```
'Rose'
```

In [62]:
```
# Access the value on the first row and the third column

df.iloc[0,2]
```
Out[62]:
```
'Architect Group'
```

In [63]:
```
# Access the column using the name

df.loc[0, 'Salary']
```
Out[63]:
```
100000
```

In [64]:
```
df2=df
df2=df2.set_index("Name")
```

In [65]:
```
#To display the first 5 rows of new dataframe
df2.head()
```
Out[65]:

	ID	Department	Salary
Name			
Rose	1	Architect Group	100000
John	2	Software Group	80000
Jane	3	Design Team	50000
Mary	4	Infrastructure	60000

In [68]:
```
#Now, let us access the column using the name
df2.loc['Jane', 'Salary']
```
Out[68]:
```
50000
```

Try it yourself

Use the `loc()` function,to get the Department of Jane in the newly created dataframe df2.

In [69]:
```

df2.loc['Jane', 'Department']
```
Out[69]:
```
'Design Team'
```

Use the `iloc()` function,to get the Salary of Mary in the newly created dataframe df2.

In [70]:
```
df2.iloc[3,2]
```
Out[70]:
```
60000
```

Exercise 3: Slicing

Slicing uses the [] operator to select a set of rows and/or columns from a DataFrame.

To slice out a set of rows, you use this syntax: data[start:stop],

here the start represents the index from where to consider, and stop represents the index one step BEYOND the row you want to select. You can perform slicing using both the index and the name of the column.

> NOTE: When slicing in pandas, the start bound is included in the output.

So if you want to select rows 0, 1, and 2 your code would look like this: df.iloc[0:3].

It means you are telling Python to start at index 0 and select rows 0, 1, 2 up to but not including 3.

> NOTE: Labels must be found in the DataFrame or you will get a KeyError.

Indexing by labels(i.e. using `loc()`) differs from indexing by integers (i.e. using `iloc()`). With `loc()` , both the start bound and the stop bound are inclusive. When using `loc()` , integers can be used, but the integers refer to the index label and not the position.

For example, using `loc()` and select 1:4 will get a different result than using `iloc()` to select rows 1:4.

We can also select a specific data value using a row and column location within the DataFrame and iloc indexing.

In [72]:
```
# let us do the slicing using old dataframe df

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

Out[72]:
```
   Name  ID       Department
0  Rose   1  Architect Group
1  John   2   Software Group
```

In [73]:
```
#let us do the slicing using loc() function on old dataframe df where index co
df.loc[0:2,'ID':'Department']
```

Out[73]:
```
   ID       Department
0   1  Architect Group
1   2   Software Group
2   3      Design Team
```

In [74]:
```
#let us do the slicing using loc() function on new dataframe df1 where index c
df2.loc['Rose':'Jane', 'ID':'Department']
```

Out[74]:
```
      ID       Department
Name
Rose   1  Architect Group
John   2   Software Group
Jane   3      Design Team
```

Try it yourself

using `loc()` function, do slicing on old dataframe df to retrieve the Name, ID and department of index column having labels as 2,3

In [75]:
```
df.loc[2:3,'Name':'Department']
```

Out[75]:

	Name	ID	Department
2	Jane	3	Design Team
3	Mary	4	Infrastructure

Writing Files

We can open a file object using the method `write()` to save the text file to a list. To write to a file, the mode argument must be set to **w**. Let's write a file **Example2.txt** with the line: ***“This is line A”**

In [77]:
```
# Write line to file
exmp2 = 'Example2.txt'
with open(exmp2, 'w') as writefile:
    writefile.write("This is line A")
```

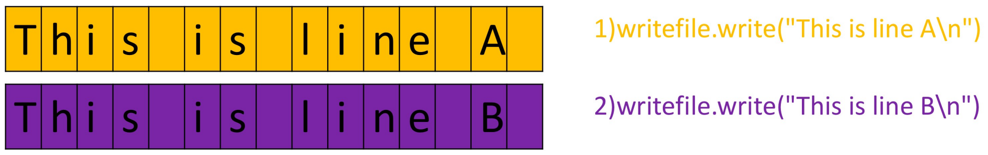
We can read the file to see if it worked:

In [79]:
```
# Read file

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

```
This is line A
```

In [80]:
```
# Write lines to file

with open(exmp2, 'w') as writefile:
    writefile.write("This is line A\n")
    writefile.write("This is line B\n")
```

The method `.write()` works similar to the method `.readline()` , except instead of reading a new line it writes a new line. The process is illustrated in the figure. The different colour coding of the grid represents a new line added to the file after each method call.


This is line A
1)writefile.write("This is line A\n")
This is line B
2)writefile.write("This is line B\n")

You can check the file to see if your results are correct

In [81]:
```
# Check whether write to file

with open(exmp2, 'r') as testwritefile:
```
```
This is line A
This is line B
```

In [82]:
```
# Sample list of text

Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]
```
Out[82]:
```
['This is line A\n', 'This is line B\n', 'This is line C\n']
```

In [83]:
```
# Write the strings in the list to text file

with open('Example2.txt', 'w') as writefile:
    for line in Lines:
        print(line)
```
```
This is line A

This is line B

This is line C
```

In [84]:
```
# Verify if writing to file is successfully executed

with open('Example2.txt', 'r') as testwritefile:
```
```
This is line A
This is line B
This is line C
```

In [85]:
```
with open('Example2.txt', 'w') as writefile:
    writefile.write("Overwrite\n")
with open('Example2.txt', 'r') as testwritefile:
```
```
Overwrite
```

Exercise

Your local university's Raptors fan club maintains a register of its active members on a .txt document. Every month they update the file by removing the members who are not active. You have been tasked with automating this with your Python skills.
Given the file `currentMem`, Remove each member with a 'no' in their Active column. Keep track of each of the removed members and append them to the `exMem` file. Make sure that the format of the original files in preserved. (*Hint: Do this by reading/writing whole lines and ensuring the header remains*)

Run the code block below prior to starting the exercise. The skeleton code has been provided for you. Edit only the `cleanFiles` function.

In [87]:
```
#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)
```

Now that you've run the prerequisite code cell above, which prepared the files for this exercise, you are ready to move on to the implementation.

Exercise: Implement the cleanFiles function in the code cell below.

In [88]:
```
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 members:
            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  
    47889      2020-5-18    yes
    74681      2018-11-17   yes
    46535      2020-2-2     yes
    86544      2016-9-19    yes
    25975      2019-12-25   yes
    90929      2020-6-20    yes
    35600      2017-8-7     yes
    18448      2015-11-17   yes
    76183      2016-3-1     yes

Inactive Members:


Membership No  Date Joined  Active
    38525      2020-9-7     no
    37573      2018-9-3     no
    56609      2020-10-16   no
    21197      2019-10-21   no
    69294      2018-3-18    no
    41254      2016-7-6     no
    81682      2020-2-22    no
    14869      2015-4-25    no
    87779      2020-1-11    no
    71832      2016-12-9    no
    99558      2015-1-9     no
    13782      2020-4-19    no
    99891      2019-9-8     no
    30767      2019-5-9     no
```

The code cell below is to verify your solution. Please do not modify the code and run it to test your implementation of `cleanFiles` .

In [90]:
```
def testMsg(passed):
    if passed:
       return 'Test Passed'
    else :
       return 'Test Failed'

testWrite = "testWrite.txt"
testAppend = "testAppend.txt"
passed = True

genFiles(testWrite,testAppend)

with open(testWrite,'r') as file:
    ogWrite = file.readlines()

with open(testAppend,'r') as file:
    ogAppend = file.readlines()

try:
    cleanFiles(testWrite,testAppend)
except:
    print('Error')

with open(testWrite,'r') as file:
    clWrite = file.readlines()

with open(testAppend,'r') as file:
    clAppend = file.readlines()

# checking if total no of rows is same, including headers

if (len(ogWrite) + len(ogAppend) != len(clWrite) + len(clAppend)):
    print("The number of rows do not add up. Make sure your final files have t
    passed = False

for line in clWrite:
    if  'no' in line:
        passed = False
        print("Inactive members in file")
        break
    else:
        if line not in ogWrite:
            print("Data in file does not match original file")
            passed = False
```

```
Test Passed
```

Get to Know a Pandas Array

You will use the dataframe df for the following:

In [91]:
```
import pandas as pd

df=pd.DataFrame({'a':[11,21,31],'b':[21,22,23]})
```

In [ ]:
```
1.  Display the first three rows:
```

In [92]:
```
df.head(3)
```

Out[92]:
```
    a   b
0  11  21
1  21  22
2  31  23
```

```
2.  Obtain column  <code> 'a' </code>:
```

QUESTION 1

Consider the dataframe df. How would you find the element in the second row and first column?

In [93]:
```
df['a']
```

Out[93]:
```
0    11
1    21
2    31
Name: a, dtype: int64
```

In [98]:
```
df.iloc[1,0]
```

Out[98]:
```
21
```

QUESTION 2

Will the following code run?

In [99]:
```
import pandas as banana

df=banana.DataFrame({'a':[11,21,31],'b':[21,22,23]})

df.head()
```

Out[99]:
```
    a   b
0  11  21
1  21  22
2  31  23
```

Get to Know a Pandas Array

You will use the dataframe `df` for the following:

In [100]:
```
import pandas as pd

df=pd.DataFrame({'a':[1,2,1],'b':[1,1,1]})
```

1. Find the unique values in column `'a'` :

In [101]:
```
df['a'].unique()
```

Out[101]:
```
array([1, 2])
```

2. Return a dataframe with only the rows where column `a` is less than two:

In [102]:
```
df[df['a']<2]
```

Out[102]:

	a	b
0	1	1
2	1	1

QUESTION 1

Consider the dataframe: df=pd.DataFrame({'a':[1,2,1],'b':[1,1,1]})

What type does the following return: df['a']==1 ?

In [103]:
```
df=pd.DataFrame({'a':[1,2,1],'b':[1,1,1]})
```

In [106]:
```
df['a']==1
```

Out[106]:
```
0     True
1    False
2     True
Name: a, dtype: bool
```

QUESTION 2 1 point possible (ungraded)

What task does the following method perform: df.to_csv("file.csv")?

Save a dataframe to a csv file.

Quiz on DataFrame

Use a variable `q` to store the column **Rating** as a dataframe

In [108]:
```
# Dependency needed to install file

# If running the notebook on your machine, else leave it commented
# !pip install xlrd

import piplite
await piplite.install(['xlrd','openpyxl'])
# Import required library

import pandas as pd

q = df[['Rating']]
q
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-108-53090bd8cf53> in <module>()
      6 import pandas as pd
      7
----> 8 q = df[['Rating']]
      9 q

~/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in __getitem__(s
elf, key)
   2680         if isinstance(key, (Series, np.ndarray, Index, list)):
   2681             # either boolean or fancy integer index
-> 2682             return self._getitem_array(key)
   2683         elif isinstance(key, DataFrame):
   2684             return self._getitem_frame(key)

~/anaconda3/lib/python3.7/site-packages/pandas/core/frame.py in _getitem_arra
y(self, key)
   2724             return self._take(indexer, axis=0)
   2725         else:
-> 2726             indexer = self.loc._convert_to_indexer(key, axis=1)
   2727             return self._take(indexer, axis=1)
   2728

~/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py in _convert_t
o_indexer(self, obj, axis, is_setter)
   1325                 if mask.any():
   1326                     raise KeyError('{mask} not in index'
-> 1327                                    .format(mask=objarr[mask]))
   1328
   1329                 return com._values_from_object(indexer)

KeyError: "['Rating'] not in index"
```

Assign the variable `q` to the dataframe that is made up of the column **Released** and **Artist**:

In [ ]:
```
q = df[['Released', 'Artist']]
q
```

Access the 2nd row and the 3rd column of `df`:

In [ ]:
```
df.iloc[1, 2]
```

Use the following list to convert the dataframe index `df` to characters and assign it to `df_new`; find the element corresponding to the row index `a` and column `'Artist'`. Then select the rows `a` through `d` for the column `'Artist'`

In [109]:
```
new_index=['a','b','c','d','e','f','g','h']
df_new=df
df_new.index=new_index
df_new.loc['a', 'Artist']
df_new.loc['a':'d', 'Artist']
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-109-6c461f214021> in <module>()
      1 new_index=['a','b','c','d','e','f','g','h']
      2 df_new=df
----> 3 df_new.index=new_index
      4 df_new.loc['a', 'Artist']
      5 df_new.loc['a':'d', 'Artist']

~/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in __setattr__
(self, name, value)
   4387         try:
   4388             object.__getattribute__(self, name)
-> 4389             return object.__setattr__(self, name, value)
   4390         except AttributeError:
   4391             pass

pandas/_libs/properties.pyx in pandas._libs.properties.AxisProperty.__set__()

~/anaconda3/lib/python3.7/site-packages/pandas/core/generic.py in _set_axis(s
elf, axis, labels)
    644 
    645     def _set_axis(self, axis, labels):
--> 646         self._data.set_axis(axis, labels)
    647         self._clear_item_cache()
    648 

~/anaconda3/lib/python3.7/site-packages/pandas/core/internals.py in set_axis
(self, axis, new_labels)
   3321             raise ValueError(
   3322                 'Length mismatch: Expected axis has {old} elements, n
ew '
-> 3323                 'values have {new} elements'.format(old=old_len, new=
new_len))
   3324 
   3325         self.axes[axis] = new_labels

ValueError: Length mismatch: Expected axis has 3 elements, new values have 8
elements
```

Review Questions

Question 1 1 point possible (graded)

What do the following lines of code do?

```
In [111]: with open("Example1.txt","r") as file1:

              FileContent=file1.readlines()

              print(FileContent)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-111-5ade4f5c8733> in <module>()
----> 1 with open("Example1.txt","r") as file1:
      2 
      3     FileContent=file1.readlines()
      4 
      5     print(FileContent)

FileNotFoundError: [Errno 2] No such file or directory: 'Example1.txt'
```

Read the file "Example1.txt"

Question 2

What do the following lines of code do?

```
In [113]: with open("Example2.txt","w") as writefile:

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

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

Write to the file "Example2.txt".

Question 3

What do the following lines of code do?

```
In [114]: with open("Example3.txt","a") as file1:

              file1.write("This is line C\n")
```

Append the file "Example3.txt".

Question 4

What is the result of applying the following method df.head() to the dataframe "df"? Prints the first row of the dataframe.

Prints the first 5 rows of the dataframe.

MODULE 5 working with Numpy Arrays & simple APIs NUMPY 1D arrays

1) The Basics and Array Creation 2) Indexing and Slicing 3) Basic Operations 4) Universal Functions

Cast the following list to a numpy array:

In [2]:
```
import numpy as np
a=[1,2,3,4,5]
x = np.array(a)
```

1. Find the type of `x` using the function `type()`.

In [3]:
```
type(x)
```

Out[3]:
```
numpy.ndarray
```

2. Find the shape of the array:

In [4]:
```
x.shape
```

Out[4]:
```
(5,)
```

In [ ]:
```
3. Find the type of data in the array:
```

In [ ]:
```
x.dtype
```

In [ ]:
```
4. Find the mean of the array:
```

In [5]:
```
x.mean()
```

Out[5]:
```
3.0
```

QUESTION 1 1/1 point (ungraded)

What is the result of the following operation: np.array([1,-1])*np.array([1,1])? array([1, -1])

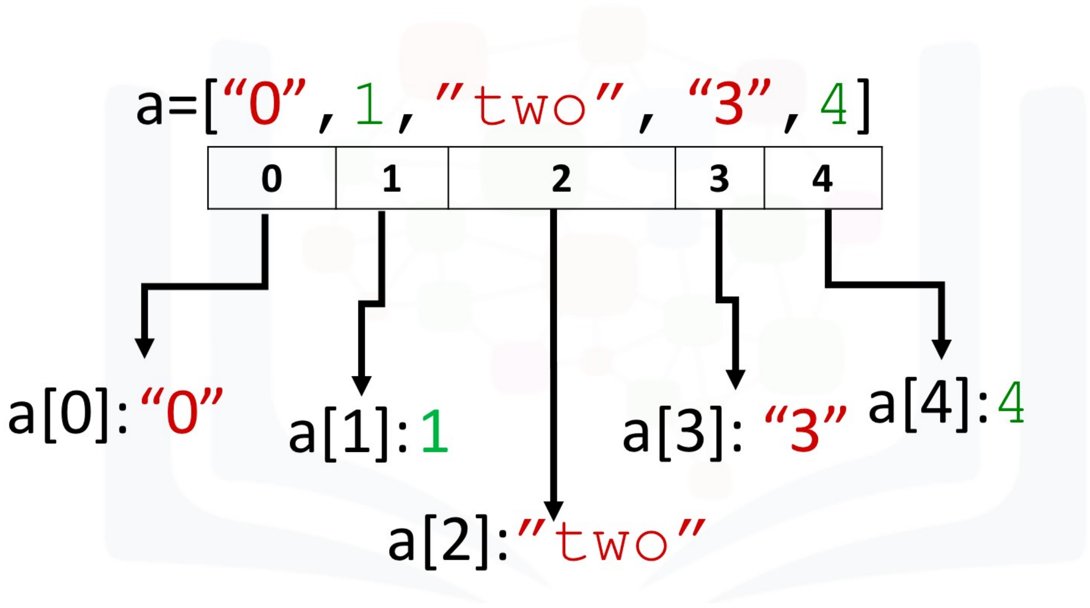
QUESTION 2 1 point possible (ungraded)

What is the result of the following operation: np.dot(np.array([1,-1]),np.array([1,1]))?

In [6]:
```
np.dot(np.array([1,-1]),np.array([1,1]))
```

Out[6]:
```
0
```

Create a Python List as follows:



In [7]:
```python
# Create a python list
```

In [8]:
```python
# Print each element

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

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

What is Numpy?

NumPy is a Python library used for working with arrays, linear algebra, fourier transform, and matrices.A numpy array is similar to a list. NumPy stands for Numerical Python and it is an open source project.The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with ndarray very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

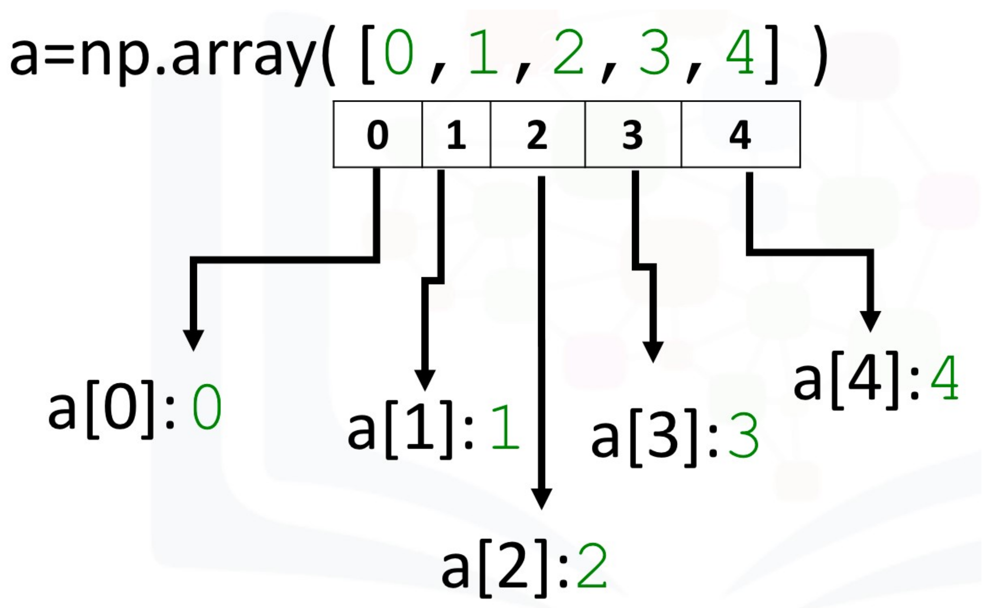
NumPy is usually imported under the np alias.

It's usually fixed in size and each element is of the same type. We can cast a list to a numpy array by first importing `numpy` :

In [10]:
```
# import numpy library

import numpy as np
```

In [11]:
```
# Create a numpy array

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

Out[11]:
```
array([0, 1, 2, 3, 4])
```



In [13]:
```
# 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 [14]:
```
print(np.__version__)
```

```
1.15.1
```

In [15]:
```
# Check the type of the array

type(a)
```

Out[15]:
```
numpy.ndarray
```

In [16]:
```
# Check the type of the values stored in numpy array

a.dtype
```
Out[16]:
```
dtype('int64')
```

Try it yourself

Check the type of the array and Value type for the given array **c**

In [18]:
```
b = np.array([3.1, 11.02, 6.2, 213.2, 5.2])

# Enter your code here
type(b)

b.dtype
```
Out[18]:
```
dtype('float64')
```

Try it yourself

Assign the value 20 for the second element in the given array.

In [19]:
```
a = np.array([10, 2, 30, 40,50])

# Enter your code here
a[1]=20
a
```
Out[19]:
```
array([10, 20, 30, 40, 50])
```

Try it yourself

Print the even elements in the given array.

In [20]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

# Enter your code here
print(arr[1:8:2])
```
```
[2 4 6 8]
```

Try it yourself

Find the size ,dimension and shape for the given array **b**

In [30]:
```
b = np.array([10, 20, 30, 40, 50, 60, 70])

# Enter your code here
b.size
```
Out[30]:
```
7
```

In [29]:
```
b.ndim
```
Out[29]:
```
1
```
In [31]:
```
b.shape
```
Out[31]:
```
(7,)
```

Try it yourself

Find the sum of maximum and minimum value in the given numpy array

In [32]:
```
c = np.array([-10, 201, 43, 94, 502])

# Enter your code here
max_c = c.max()
max_c
```
Out[32]:
```
502
```
In [33]:
```
min_c = c.min()
min_c
```
Out[33]:
```
-10
```
In [34]:
```
Sum = (max_c +min_c)
Sum
```
Out[34]:
```
492
```

Try it yourself

Perform addition operation on the given numpy array arr1 and arr2:

In [35]:
```
arr1 = np.array([10, 11, 12, 13, 14, 15])
arr2 = np.array([20, 21, 22, 23, 24, 25])

# Enter your code here
arr3 = np.add(arr1, arr2)
arr3
```
Out[35]:
```
array([30, 32, 34, 36, 38, 40])
```

Try it yourself

Perform subtraction operation on the given numpy array arr1 and arr2:

In [36]:
```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

# Enter your code here
arr3 = np.subtract(arr1, arr2)
arr3
```

Out[36]:
```
array([-10,  -1,   8,  17,  26,  35])
```

Try it yourself

Perform multiply operation on the given numpy array arr1 and arr2:

In [37]:
```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([2, 1, 2, 3, 4, 5])

# Enter your code here
arr3 = np.multiply(arr1, arr2)
arr3
```

Out[37]:
```
array([ 20,  20,  60, 120, 200, 300])
```

Try it yourself

Perform division operation on the given numpy array arr1 and arr2:

In [38]:
```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([3, 5, 10, 8, 2, 33])

# Enter your code here
arr3 = np.divide(arr1, arr2)
arr3
```

Out[38]:
```
array([ 3.33333333,  4.        ,  3.        ,  5.        , 25.        ,
        1.81818182])
```

Try it yourself

Perform dot operation on the given numpy array ar1 and ar2:

In [39]:
```
arr1 = np.array([3, 5])
arr2 = np.array([2, 4])

# Enter your code here
arr3 = np.dot(arr1, arr2)
arr3
```

Out[39]:
```
26
```

Try it yourself

Add Constant 5 to the given numpy array ar:

```
In [40]: arr = np.array([1, 2, 3, -1])

         # Enter your code here
         arr + 5
```

```
Out[40]: array([6, 7, 8, 4])
```

Try it yourself

Make a numpy array within [5, 4] and 6 elements

```
In [41]: np.linspace(5, 4, num=6)
```

```
Out[41]: array([5. , 4.8, 4.6, 4.4, 4.2, 4. ])
```

Quiz on 1D Numpy Array

Implement the following vector subtraction in numpy: u-v

```
In [42]: # Write your code below and press Shift+Enter to execute

         u = np.array([1, 0])
         v = np.array([0, 1])

         u-v
```

```
Out[42]: array([ 1, -1])
```

Multiply the numpy array z with -2:

```
In [43]: # Write your code below and press Shift+Enter to execute

         z = np.array([2, 4])
         z* 2
```

```
Out[43]: array([-4, -8])
```

Consider the list `[1, 2, 3, 4, 5]` and `[1, 0, 1, 0, 1]`. Cast both lists to a numpy array then multiply them together:

```
In [45]: a = np.array([1, 2, 3, 4, 5])
         b = np.array([1, 0, 1, 0, 1])
         a*b
```

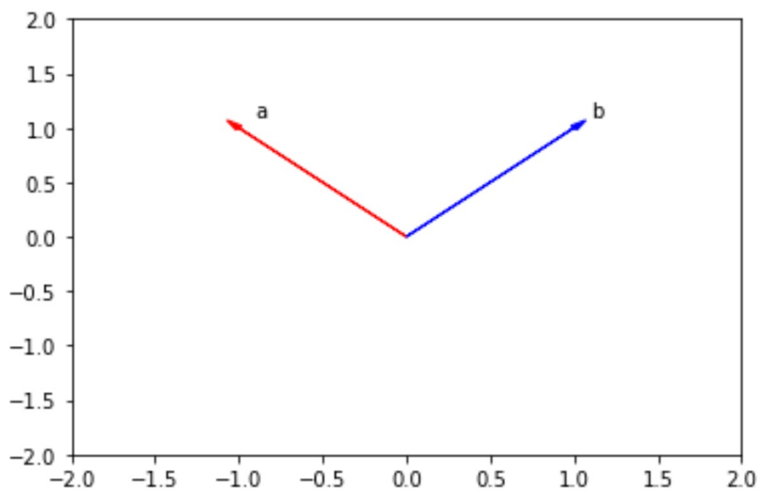
```
Out[45]: array([1, 0, 3, 0, 5])
```

In [46]:
```python
# Import the libraries

import time
import sys
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

def Plotvec2(a,b):
    ax = plt.axes()# to generate the full window axes
    ax.arrow(0, 0, *a, head_width=0.05, color ='r', head_length=0.1)#Add an ar
    plt.text(*(a + 0.1), 'a')
    ax.arrow(0, 0, *b, head_width=0.05, color ='b', head_length=0.1)#Add an ar
    plt.text(*(b + 0.1), 'b')
    plt.ylim(-2, 2)#set the ylim to bottom(-2), top(2)
    plt.xlim(-2, 2)#set the xlim to left(-2), right(2)
```

Convert the list `[-1, 1]` and `[1, 1]` to numpy arrays `a` and `b`. Then, plot the arrays as vectors using the fuction `Plotvec2` and find their dot product:

In [47]:
```python
a = np.array([-1, 1])
b = np.array([1, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a,b))
```

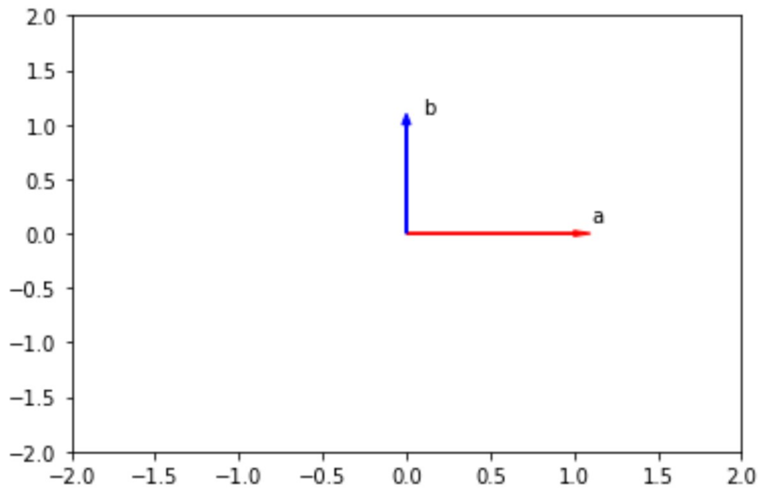
```
The dot product is 0
```



Convert the list `[1, 0]` and `[0, 1]` to numpy arrays `a` and `b`. Then, plot the arrays as vectors using the function `Plotvec2` and find their dot product:

In [48]:
```
a = np.array([1, 0])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

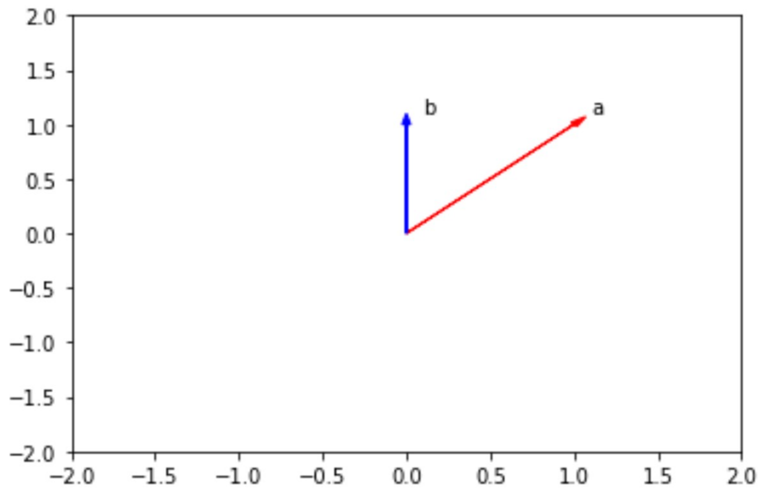
```
The dot product is 0
```



Convert the list `[1, 1]` and `[0, 1]` to numpy arrays `a` and `b`. Then plot the arrays as vectors using the fuction `Plotvec2` and find their dot product:

In [49]:
```
a = np.array([1, 1])
b = np.array([0, 1])
Plotvec2(a, b)
print("The dot product is", np.dot(a, b))
```

```
The dot product is 1
```



Why are the results of the dot product for `[-1, 1]` and `[1, 1]` and the dot product for `[1, 0]` and `[0, 1]` zero, but not zero for the dot product for `[1, 1]` and `[0, 1]`?

Hint: Study the corresponding figures, pay attention to the direction the arrows are pointing to.
The vectors used for question 4 and 5 are perpendicular. As a result, the dot product is zero.

Convert the list `[1, 2, 3]` and `[8, 9, 10]` to numpy arrays `arr1` and `arr2`. Then perform `Addition` , `Subtraction` , `Multiplication` , `Division` and `Dot Operation` on the `arr1` and `arr2`.

In [51]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([8, 9, 10])

arr3 = np.add(arr1, arr2)
arr3

arr4 = np.subtract(arr1, arr2)
arr4

arr5 = np.multiply(arr1, arr2)
arr5


arr6 = np.divide(arr1, arr2)
arr6

arr7 = np.dot(arr1, arr2)
arr7
```
Out[51]:
```
56
```

Convert the list `[1, 2, 3, 4, 5]` and `[6, 7, 8, 9, 10]` to numpy arrays `arr1` and `arr2`. Then find the even and odd numbers from `arr1` and `arr2`.

In [52]:
```
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.array([6, 7, 8, 9, 10])

x = arr1[1:8:2]
x

y = arr2[0:8:2]
```
Out[52]:
```
array([ 6,  8, 10])
```

Create a 2D Numpy Array

In [55]:
```
# Import the libraries

import numpy as np
import matplotlib.pyplot as plt
# Create a list

a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]
```
Out[55]:
```
[[11, 12, 13], [21, 22, 23], [31, 32, 33]]
```

In [56]:
```
# Convert list to Numpy Array
# Every element is the same type

A = np.array(a)
A
```
Out[56]:
```
array([[11, 12, 13],
       [21, 22, 23],
       [31, 32, 33]])
```

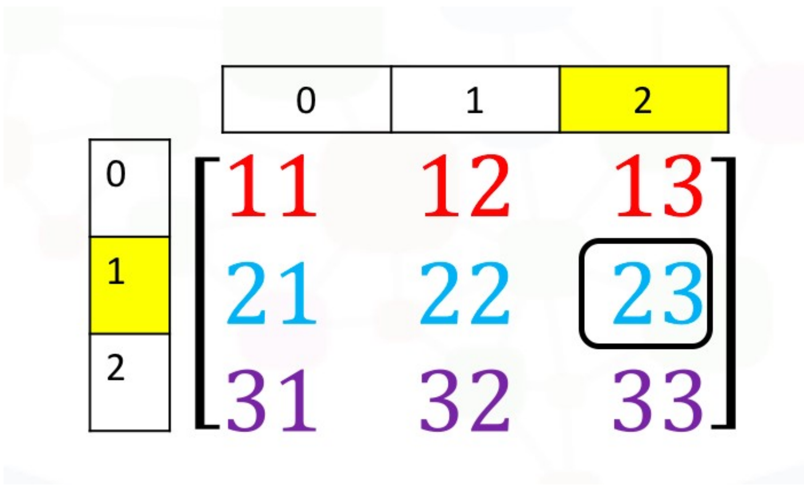
In [57]:
```
# Show the numpy array dimensions

A.ndim
```
Out[57]:
```
2
```

In [58]:
```
A.shape
```
Out[58]:
```
(3, 3)
```

In [59]:
```
# Show the numpy array size

A.size
```
Out[59]:
```
9
```

Accessing different elements of a Numpy Array

We can use rectangular brackets to access the different elements of the array. The correspondence between the rectangular brackets and the list and the rectangular representation is shown in the following figure for a 3x3 array:



We simply use the square brackets and the indices corresponding to the element we would like:

In [61]:
```
# Access the element on the second row and third column

A[1, 2]
```
Out[61]:
```
23
```

In [62]:
```
# Access the element on the second row and third column

A[1][2]
```
Out[62]:
```
23
```

Quiz on 2D Numpy Array

Consider the following list `a` , convert it to Numpy Array.

In [63]:
```
# Write your code below and press Shift+Enter to execute

a = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
```

In [64]:
```
A = np.array(a)
A
```
Out[64]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Calculate the numpy array size.

In [65]:
```
# Write your code below and press Shift+Enter to execute
A.size
```
Out[65]:
```
12
```

Access the element on the first row and first and second columns.

In [66]:
```
# Write your code below and press Shift+Enter to execute
A[0][0:2]
```
Out[66]:
```
array([1, 2])
```

Perform matrix multiplication with the numpy arrays `A` and `B` .

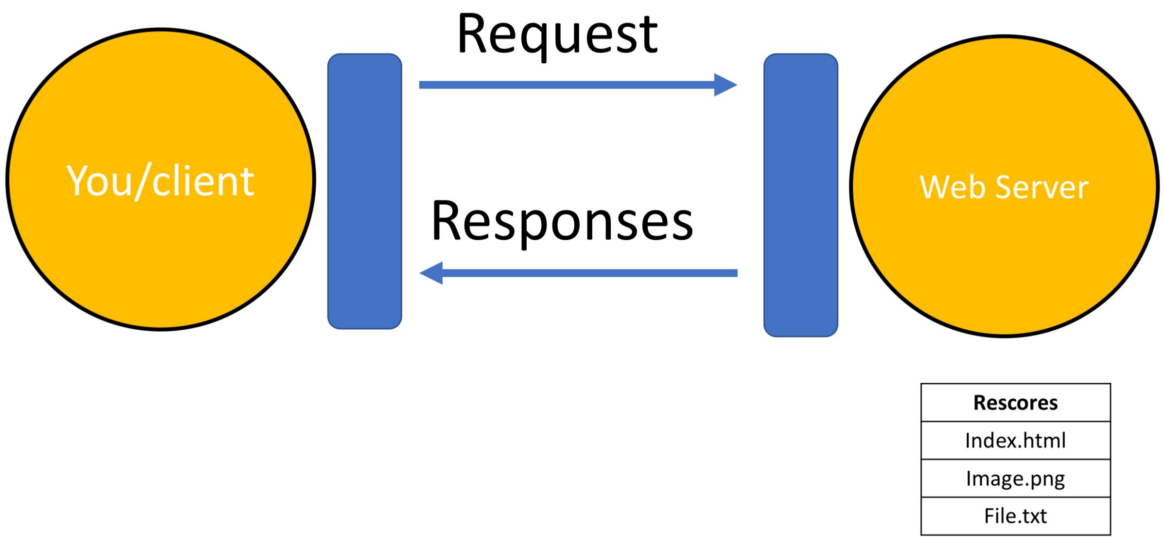
In [67]:
```
B = np.array([[0, 1], [1, 0], [1, 1], [-1, 0]])
X = np.dot(A,B)
X
```
Out[67]:
```
array([[ 1,  4],
       [ 5, 12],
       [ 9, 20]])
```

Overview of HTTP

When you, the **client**, use a web page your browser sends an **HTTP** request to the **server** where the page is hosted. The server tries to find the desired **resource** by default " `index.html` ". If your request is successful, the server will send the object to the client in an **HTTP response**. This includes information like the type of the **resource**, the length of the **resource**, and other information.

The figure below represents the process. The circle on the left represents the client, the circle on the right represents the Web server. The table under the Web server represents a list of resources stored in the web server. In this case an `HTML` file, `png` image, and `txt` file .

The **HTTP** protocol allows you to send and receive information through the web including webpages, images, and other web resources. In this lab, we will provide an overview of the Requests library for interacting with the `HTTP` protocol.



Uniform Resource Locator: URL

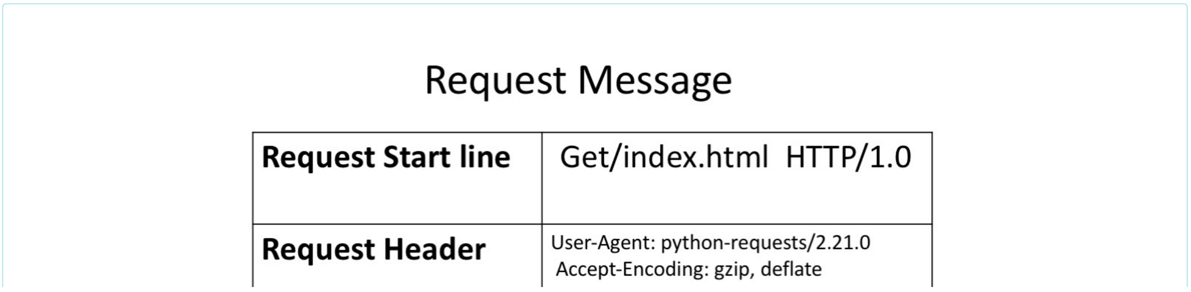
Uniform resource locator (URL) is the most popular way to find resources on the web. We can break the URL into three parts.

- **scheme** this is this protocol, for this lab it will always be `http://`
- **Internet address or Base URL** this will be used to find the location here are some examples: `www.ibm.com` and `www.gitlab.com`
- **route** location on the web server for example: `/images/IDSNlogo.png`

You may also hear the term Uniform Resource Identifier (URI), URL are actually a subset of URIs. Another popular term is endpoint, this is the URL of an operation provided by a Web server.

Request

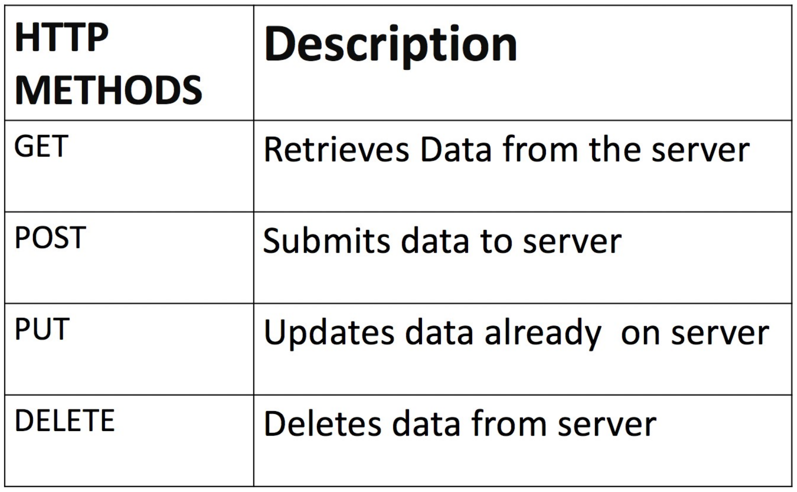
The process can be broken into the **request** and **response** process. The request using the get method is partially illustrated below. In the start line we have the `GET` method, this is an `HTTP` method. Also the location of the resource `/index.html` and the `HTTP` version. The Request header passes additional information with an `HTTP` request:



Request Message

Request Start line	Get/index.html HTTP/1.0
Request Header	User-Agent: python-requests/2.21.0 Accept-Encoding: gzip, deflate

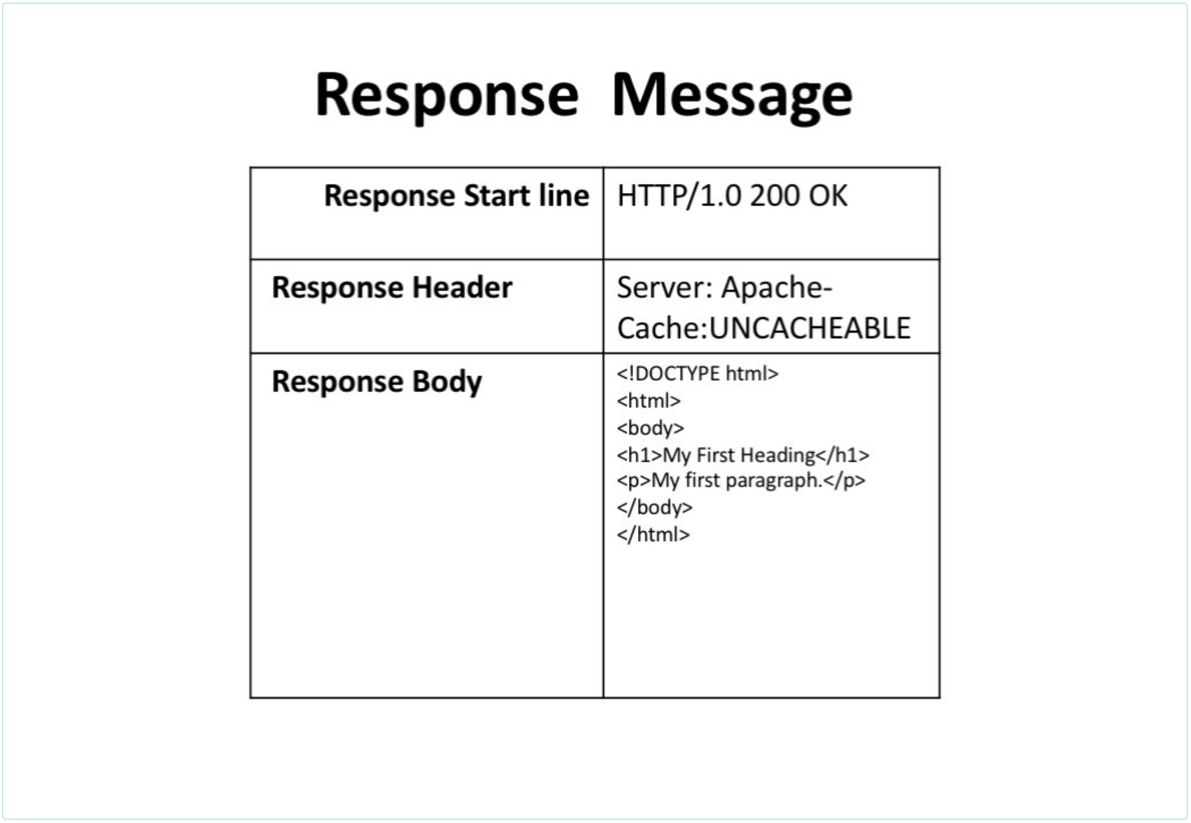
When an `HTTP` request is made, an `HTTP` method is sent, this tells the server what action to perform. A list of several `HTTP` methods is shown below. We will go over more examples later.



HTTP METHODS	**Description**
GET	Retrieves Data from the server
POST	Submits data to server
PUT	Updates data already on server
DELETE	Deletes data from server

Response

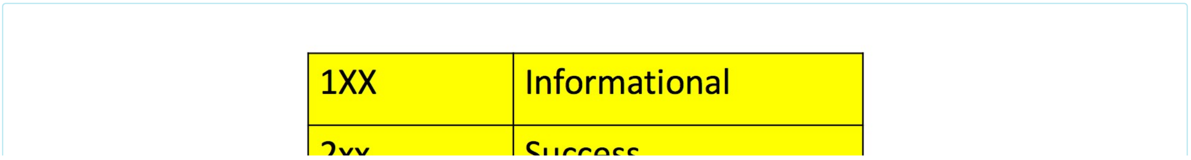
The figure below represents the response; the response start line contains the version number `HTTP/1.0` , a status code (200) meaning success, followed by a descriptive phrase (OK). The response header contains useful information. Finally, we have the response body containing the requested file, an `HTML` document. It should be noted that some requests have headers.



Response Message

Response Start line	HTTP/1.0 200 OK
Response Header	Server: Apache- Cache:UNCACHEABLE
Response Body	<!DOCTYPE html> <html> <body> <h1>My First Heading</h1> <p>My first paragraph.</p> </body> </html>

Some status code examples are shown in the table below, the prefix indicates the class. These are shown in yellow, with actual status codes shown in white. Check out the following link (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2022-01-01 for more descriptions.



1XX	Informational

In [68]:
```
import requests
import os
from PIL import Image
```

You can make a `GET` request via the method `get` to www.ibm.com (http://www.ibm.com/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2022-01-01

In [69]:
```
url='https://www.ibm.com/'
r=requests.get(url)
```

We have the response object `r` , this has information about the request, like the status of the request. We can view the status code using the attribute `status_code`

In [70]:
```
r.status_code
```

Out[70]:
```
200
```

You can view the request headers:

In [71]:
```
print(r.request.headers)
```

```
{'User-Agent': 'python-requests/2.19.1', 'Accept-Encoding': 'gzip, deflate',
'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie': '_abck=A5353D1CF03F144
A81EFBE170E02FEC5~-1~YAAQOS0tF63v6FuDAQAAaLa8lgj7WoHf7QU4EtatpSwaQymY7HqLsoHO
24/NM0XZ2cdJEvqU+Y4Pn3bAzJ/3k9QPNhf1pPzVUcGQ1Tm3QNKvoTV6AAo+ZaQTom+Tw2HjjdWgZ
SZI1IhWTZIAz4A6Pd2yFu8QToB5eCY/ZKpz60AvuHCfCgr5qeiDUXAnXdzRQTWn7njB80H2ZNTIlm
N3W0Qz+fzFIOgpogLJS5elIgLWO38RvU74A0FfhkuGN7SJJDyq2mTC1AaoOG6EEHzdQM5RvIra136
RvoWLoEItC9e/ovnrZVqeQYtzbVgrDrJxds1oHnxiEmmwsVk6hXki1i/Frzo3kPfKHQphU2lK30T
j+HXFAjY=~-1~-1~-1; bm_sz=38B3C4B5DBFB0400CED3C9EC21CDAC95~YAAQOS0tF67v6FuDAQ
AAaLa8lhGDhwUILvhx9kPVvRHfbe7wwcfy4DYwAFCd50g6UBkH3CLMbalVp1R2EtG9/RW/4g+nVvj
ImAEhZq+INSPvszZ8eq69IKKUitGWVTBRUaRw7FSHdD6X/9WkbkKKnMxl0GgnIStLBxNCPSLEja/a
hplnHzBTxviBGoLsSRsNSh+b6Mv0ZeOxfves2UAQHeXkz3J8eZmJk4shSpJtr/Anba3dRJMDT4xxZ
95wnF725bdz25jYv5EkEsMssGepDpVLS1mCAleWqbFSoyEUWHc=~4539206~4534341'}
```

You can view the request body, in the following line, as there is no body for a get request we get a `None` :

In [72]:
```
print("request body:", r.request.body)
```

```
request body: None
```

You can view the `HTTP` response header using the attribute `headers` . This returns a python dictionary of `HTTP` response headers.

In [74]:
```
header=r.headers
print(r.headers)
```

```
{'Server': 'Apache', 'x-drupal-dynamic-cache': 'UNCACHEABLE', 'Link': '<http
s://www.ibm.com/mx-es>; rel="canonical", <//1.cms.s81c.com>; rel=preconnect;
crossorigin, <//1.cms.s81c.com>; rel=dns-prefetch', 'x-ua-compatible': 'IE=ed
ge', 'Content-Language': 'es-mx', 'Permissions-Policy': 'interest-cohort=()',
'x-generator': 'Drupal 9 (https://www.drupal.org)', 'x-dns-prefetch-control':
'on', 'x-drupal-cache': 'MISS', 'Last-Modified': 'Sat, 01 Oct 2022 11:24:59 G
MT', 'ETag': '"1664623499"', 'Content-Type': 'text/html; charset=UTF-8', 'x-a
cquia-host': 'www.ibm.com', 'x-acquia-path': '/mx-es', 'x-acquia-site': '', '
x-acquia-purge-tags': '', 'x-varnish': '639174596 639270922', 'x-cache-hits':
'5', 'x-age': '3313', 'Accept-Ranges': 'bytes', 'Content-Encoding': 'gzip', '
Cache-Control': 'public, max-age=300', 'Expires': 'Sun, 02 Oct 2022 03:33:14
GMT', 'X-Akamai-Transformed': '9 10850 0 pmb=mTOE,2', 'Date': 'Sun, 02 Oct 20
22 03:28:14 GMT', 'Content-Length': '11017', 'Connection': 'keep-alive', 'Var
y': 'Accept-Encoding', 'x-content-type-options': 'nosniff', 'X-XSS-Protection
': '1; mode=block', 'Content-Security-Policy': 'upgrade-insecure-requests', '
Strict-Transport-Security': 'max-age=31536000', 'x-ibm-trace': 'www-dipatche
r: dynamic rule'}
```

We can obtain the date the request was sent using the key `Date`

In [75]:
```
header['date']
```

Out[75]:
```
'Sun, 02 Oct 2022 03:28:14 GMT'
```

`Content-Type` indicates the type of data:

In [76]:
```
header['Content-Type']
```

Out[76]:
```
'text/html; charset=UTF-8'
```

You can also check the `encoding` :

In [77]:
```
 r.encoding
```

Out[77]:
```
'UTF-8'
```

As the `Content-Type` is `text/html` we can use the attribute `text` to display the `HTML` in the body. We can review the first 100 characters:

In [78]:
```
r.text[0:100]
```

Out[78]:
```
'<!DOCTYPE html>\n<html lang="es-mx" dir="ltr">\n  <head>\n    <meta charset
="utf-8" />\n<script>digitalD'
```

You can load other types of data for non-text requests, like images. Consider the URL of the following image:

In [80]:
```
# Use single quotation marks for defining string
url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDev
```

We can make a get request:

In [81]:
```
r=requests.get(url)
```

We can look at the response header:

In [82]:
```
print(r.headers)
```

```
{'Date': 'Sun, 02 Oct 2022 03:31:08 GMT', 'X-Clv-Request-Id': 'e26b4f8d-d2e0-
4fc6-9748-0fa162b80127', 'Server': 'Cleversafe', 'X-Clv-S3-Version': '2.5', '
Accept-Ranges': 'bytes', 'x-amz-request-id': 'e26b4f8d-d2e0-4fc6-9748-0fa162b
80127', 'Cache-Control': 'max-age=0,public', 'ETag': '"a831e767d02efd21b904ec
485ac0c769"', 'Content-Type': 'image/png', 'Last-Modified': 'Wed, 14 Sep 2022
05:47:46 GMT', 'Content-Length': '21590'}
```

We can see the `'Content-Type'`

In [83]:
```
r.headers['Content-Type']
```

Out[83]:
```
'image/png'
```

An image is a response object that contains the image as a bytes-like object (https://docs.python.org/3/glossary.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2022-01-01 bytes-like-object). As a result, we must save it using a file object. First, we specify the file path and name

In [85]:
```
path=os.path.join(os.getcwd(),'image.png')
path
```

Out[85]:
```
'/home/flynn/IBM Course notes/image.png'
```

Question 1: write `wget` (https://www.gnu.org/software/wget/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetv

In the previous section, we used the `wget` function to retrieve content from the web server as shown below. Write the python code to perform the same task. The code should be the same as the one used to download the image, but the file name should be `'Example1.txt'`.

In [87]:
```
url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDev
path=os.path.join(os.getcwd(),'example1.txt')
r=requests.get(url)
with open(path,'wb') as f:
    f.write(r.content)
```

Skills Network

Simple APIs

Random User and Fruitvice API Examples

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Load and use RandomUser API, using `RandomUser()` Python library
- Load and use Fruitvice API, using `requests` Python library

The purpose of this notebook is to provide more examples on how to use simple APIs. As you have already learned from previous videos and notebooks, API stands for Application Programming Interface and is a software intermediary that allows two applications to talk to each other.

The advantages of using APIs:

- **Automation**. Less human effort is required and workflows can be easily updated to become faster and more
  productive.
- **Efficiency**. It allows to use the capabilities of one of the already developed APIs than to try to independently implement some functionality from scratch.

The disadvantage of using APIs:

- **Secirity**. If the API is poorly integrated, it means it will be vulnerable to attacks, resulting in data
  breeches or losses having financial or reputation implications.

One of the applications we will use in this notebook is Random User Generator. RandomUser is an open-source, free API providing developers with randomly generated users to be used as placeholders for testing purposes. This makes the tool similar to Lorem Ipsum, but is a placeholder for people instead of text. The API can return multiple results, as well as specify generated user details such as gender, email, image, username, address, title, first and last name, and more. More information on RandomUser (https://randomuser.me/documentation?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2022-01-01 can be found here.

Another example of simple API we will use in this notebook is Fruitvice application. The Fruitvice API webservice which provides data for all kinds of fruit! You can use Fruityvice to find out interesting information about fruit and educate yourself. The webservice is completely free to use and contribute to.

Example 1: RandomUser API

Bellow are Get Methods parameters that we can generate. For more information on the parameters, please visit this documentation (https://randomuser.me/documentation?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2022-01-01 page.

Get Methods

- get_cell()
- get_city()
- get_dob()
- get_email()
- get_first_name()
- get_full_name()
- get_gender()
- get_id()
- get_id_number()
- get_id_type()
- get_info()
- get_last_name()
- get_login_md5()
- get_login_salt()
- get_login_sha1()
- get_login_sha256()
- get_nat()
- get_password()
- get_phone()
- get_picture()
- get_postcode()
- get_registered()
- get_state()
- get_street()
- get_username()

In [88]:
```
!pip install randomuser
```

```
Collecting randomuser
  Downloading https://files.pythonhosted.org/packages/b3/98/6e2d0a77d8c420c0d
9eba678868ff4978edfa0c78ae5a183efbf29819c5d/randomuser-1.6.tar.gz (https://fi
les.pythonhosted.org/packages/b3/98/6e2d0a77d8c420c0d9eba678868ff4978edfa0c78
ae5a183efbf29819c5d/randomuser-1.6.tar.gz)
Building wheels for collected packages: randomuser
  Running setup.py bdist_wheel for randomuser ... done
  Stored in directory: /home/flynn/.cache/pip/wheels/5f/89/e1/54edea993ef3aa6
2c958b8823d93dcd86758decf1ae1be86eb
Successfully built randomuser
twisted 18.7.0 requires PyHamcrest>=1.9.0, which is not installed.
Installing collected packages: randomuser
Successfully installed randomuser-1.6
You are using pip version 10.0.1, however version 22.2.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In [89]:
```
from randomuser import RandomUser
import pandas as pd
```

First, we will create a random user object, r.

In [90]:
```
r = RandomUser()
```

Then, using `generate_users()` function, we get a list of random 10 users.

In [92]:
```
some_list = r.generate_users(10)
some_list
```

Out[92]:
```
[<randomuser.RandomUser at 0x7fcc3863c978>,
 <randomuser.RandomUser at 0x7fcc3863c9b0>,
 <randomuser.RandomUser at 0x7fcc3863c9e8>,
 <randomuser.RandomUser at 0x7fcc3863ca20>,
 <randomuser.RandomUser at 0x7fcc3863ca58>,
 <randomuser.RandomUser at 0x7fcc3863ca90>,
 <randomuser.RandomUser at 0x7fcc3863cac8>,
 <randomuser.RandomUser at 0x7fcc3863cb00>,
 <randomuser.RandomUser at 0x7fcc3863cb38>,
 <randomuser.RandomUser at 0x7fcc3863cb70>]
```

The **"Get Methods"** functions mentioned at the beginning of this notebook, can generate the required parameters to construct a dataset. For example, to get full name, we call `get_full_name()` function.

In [96]:
```
name = r.get_full_name()
name
```

Out[96]:
```
'Robert Michels'
```

Let's say we only need 10 users with full names and their email addresses. We can write a "for-loop" to print these 10 users.

In [95]:
```
for user in some_list:
    print (user.get_full_name()," ",user.get_email())
```

```
Allen Craig   allen.craig@example.com
Arnold Howell   arnold.howell@example.com
Milla Kauppila   milla.kauppila@example.com
Veeti Wainio   veeti.wainio@example.com
Mirko Perrin   mirko.perrin@example.com
Frank-Peter Herzig   frank-peter.herzig@example.com
Iker Cruz   iker.cruz@example.com
Roope Kivisto   roope.kivisto@example.com
Leo Tuomala   leo.tuomala@example.com
Tina Rodrigues   tina.rodrigues@example.com
```

Exercise 1

In this Exercise, generate photos of the random 5 users.

In [97]:
```
for user in some_list:
    print (user.get_picture())
```

```
https://randomuser.me/api/portraits/men/6.jpg (https://randomuser.me/api/port
raits/men/6.jpg)
https://randomuser.me/api/portraits/men/18.jpg (https://randomuser.me/api/por
traits/men/18.jpg)
https://randomuser.me/api/portraits/women/95.jpg (https://randomuser.me/api/p
ortraits/women/95.jpg)
https://randomuser.me/api/portraits/men/43.jpg (https://randomuser.me/api/por
traits/men/43.jpg)
https://randomuser.me/api/portraits/men/56.jpg (https://randomuser.me/api/por
traits/men/56.jpg)
https://randomuser.me/api/portraits/men/55.jpg (https://randomuser.me/api/por
traits/men/55.jpg)
https://randomuser.me/api/portraits/men/69.jpg (https://randomuser.me/api/por
traits/men/69.jpg)
https://randomuser.me/api/portraits/men/45.jpg (https://randomuser.me/api/por
traits/men/45.jpg)
https://randomuser.me/api/portraits/men/70.jpg (https://randomuser.me/api/por
traits/men/70.jpg)
https://randomuser.me/api/portraits/women/62.jpg (https://randomuser.me/api/p
ortraits/women/62.jpg)
```

To generate a table with information about the users, we can write a function containing all desirable parameters. For example, name, gender, city, etc. The parameters will depend on the requirements of the test to be performed. We call the Get Methods, listed at the beginning of this notebook. Then, we return pandas dataframe with the users.

In [98]:
```
def get_users():
    users =[]

    for user in RandomUser.generate_users(10):
        users.append({"Name":user.get_full_name(),"Gender":user.get_gender(),"

    return pd.DataFrame(users)

get_users()

df1 = pd.DataFrame(get_users())
```

In [99]:
```
df1
```

Out[99]:

	City	DOB	Email	Gender	Name	
0	Mendrisio	1979-08-10T09:10:37.624Z	hildegard.laurent@example.com	female	Hildegard Laurent	https /a
1	Lasalle	1952-06-30T07:31:04.921Z	theo.french@example.com	male	Theo French	https
2	Madison	1977-12-05T01:52:23.164Z	lucille.hopkins@example.com	female	Lucille Hopkins	https /a
3	Portlaoise	1989-11-20T04:20:10.708Z	jacob.hawkins@example.com	male	Jacob Hawkins	https
4	Rockhampton	1994-04-15T14:28:03.192Z	dennis.fisher@example.com	male	Dennis Fisher	https
5	Florianópolis	1960-04-09T10:24:31.452Z	iven.fernandes@example.com	male	Íven Fernandes	https
6	Bath	1991-05-09T03:11:16.307Z	megan.bouchard@example.com	female	Megan Bouchard	https /a
7	Portarlington	1965-07-18T20:01:12.584Z	eliza.mckinney@example.com	female	Eliza Mckinney	https /a
8	Reims	1990-11-26T18:20:55.557Z	tony.dasilva@example.com	male	Tony Da Silva	https
9	Móstoles	1985-06-14T15:36:37.886Z	felix.santos@example.com	male	Felix Santos	https

In [101]:
```
import requests
import json
data = requests.get("https://www.fruityvice.com/api/fruit/all")
results = json.loads(data.text)
pd.DataFrame(results)
df2 = pd.json_normalize(results)
```

```
---------------------------------------------------------------------------
Error                                     Traceback (most recent call last)
~/anaconda3/lib/python3.7/site-packages/urllib3/contrib/pyopenssl.py in wrap_
socket(self, sock, server_side, do_handshake_on_connect, suppress_ragged_eof
s, server_hostname)
    443             try:
--> 444                 cnx.do_handshake()
    445             except OpenSSL.SSL.WantReadError:

~/anaconda3/lib/python3.7/site-packages/OpenSSL/SSL.py in do_handshake(self)
   1906         result = _lib.SSL_do_handshake(self._ssl)
-> 1907         self._raise_ssl_error(self._ssl, result)
   1908

~/anaconda3/lib/python3.7/site-packages/OpenSSL/SSL.py in _raise_ssl_error(se
lf, ssl, result)
   1638         else:
-> 1639             _raise_current_error()
   1640
```

Exercise 2

In this Exercise, find out how many calories are contained in a banana.

In [ ]:
```

```

In [100]:
```
import requests
import json
data = requests.get("https://www.fruityvice.com/api/fruit/all")
results = json.loads(data.text)
pd.DataFrame(results)
df2 = pd.json_normalize(results)



cal_banana = df2.loc[df2["name"] == 'Banana']
cal_banana.iloc[0]['nutritions.calories']
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-100-992eafee6e5d> in <module>()
----> 1 cal_banana = df2.loc[df2["name"] == 'Banana']
      2 cal_banana.iloc[0]['nutritions.calories']

NameError: name 'df2' is not defined
```

Review Questions

Multiple Choice 1 point possible (graded)

What is the result of the following lines of code:

a=np.array([0,1,0,1,0])

b=np.array([1,0,1,0,1])

a*b

In [102]:
```
a=np.array([0,1,0,1,0])

b=np.array([1,0,1,0,1])

a*b
```
Out[102]:
```
array([0, 0, 0, 0, 0])
```

In [103]:
```
a=np.array([0,1])

b=np.array([1,0])

np.dot(a,b)
```
Out[103]:
```
0
```

In [104]:
```
a=np.array([1,1,1,1,1])

a+10
```
Out[104]:
```
array([11, 11, 11, 11, 11])
```

Reference

```
Lutz, Mark. Learning Python: Powerful Object-Oriented Programming. "
O'Reilly Media, Inc.", 2013.
Ana Bell, Eric Grimson and John Guttag.  Introduction to Computer Sci
ence and Programming in Python, MIT Open Course Ware, MIT 2017
```

In [105]:
```
3+2*2
```
Out[105]:
```
7
```

In [106]:
```
A='1234567'
A[1::2]
```
Out[106]:
```
'246'
```

In [107]:
```
Name="Michael Jackson"
Name.find('el')
```
Out[107]:
```
5
```

In [108]:
```
F="You are wrong"
F.upper()
```
Out[108]:
```
'YOU ARE WRONG'
```

In [110]:
```
 A=((11,12),[21,22])
A[1]
```
Out[110]:
```
[21, 22]
```

In [111]:
```
A[0][1]
```
Out[111]:
```
12
```

In [112]:
```
'1,2,3,4'.split(',')
```
Out[112]:
```
['1', '2', '3', '4']
```

In [115]:
```
V={'A','B','C' }
V.add('C')
V
```
Out[115]:
```
{'A', 'B', 'C'}
```

In [116]:
```


x="Go"

if(x=="Go"):

    print('Go ')

else:

    print('Stop')

print('Mike')
```
```
Go
Mike
```

In [118]:
```
for n in range(3):

    print(n)
```
```
0
1
2
```

In [119]:

```
for n in range(3):

    print(n+1)
```

```
1
2
3
```

In [120]:

```
A=['1','2','3']

for a in A:

    print(2*a)
```

```
11
22
33
```

In [121]:

```
class Points(object):

    def __init__(self,x,y):

        self.x=x

        self.y=y

    def print_point(self):

        print('x=',self.x,' y=',self.y)

p1=Points(1,2)

p1.print_point()
```

```
x= 1  y= 2
```

In [ ]:

```
class Points(object):

def __init__(self,x,y):

self.x=x

self.y=y

def print_point(self):

print('x=',self.x,' y=',self.y)

p2=Points(1,2)

p2.x=2

p2.print_point()
```

In [ ]: