Name - Akshat Shah

Roll no - 21BCP322

Computer Engineering

Div-3, Group-10,

Semester-2

Fundamentals of Python programming

List of Lab Assignments done during the course:

1 Basic data types, variables and standard functions.

2 Operators.

3 Loops and conditional statements

4 Functions

5 Exception handling

6 Data collections

7 Graph plotting

8 Object oriented programming in python

9 Principles of Object-oriented programming

10 Types of sorting methods

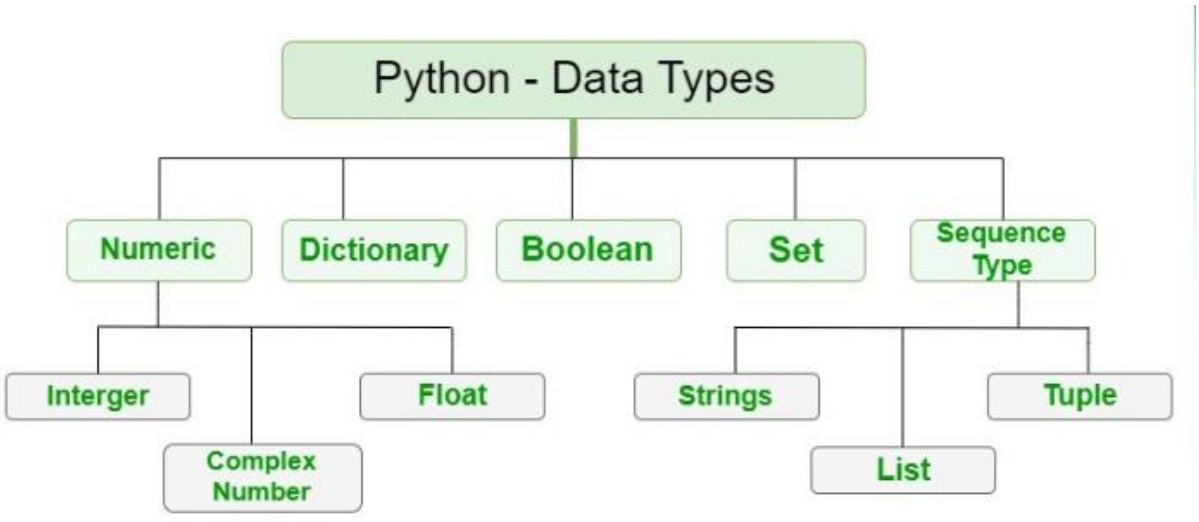
1 Basic data types, variables and standard functions.

Objectives: To know about the basic data types, variables and standard functions and their notations.

Theory:

Data types are the classification or categorization of data items.

Data types are actually classes and variables are instance (object) of these classes.



Certain functions are available for use in the Python environment.

•Similar to #include in C, Python uses import statements to import modules to the Python environment.

•Standard functions are those functions which are already present without any import.

•There are 75 standard functions available.

List of problems:

Q1. WAP to take 2 float values to add, subtract, multiply and divide them and print their fractional representation.

```
print("Enter 2 floating values")
#Taking inputs
x=float(input("Enter first value "))
y=float(input("Enter second value "))
```

```
Enter 2 floating values
Enter first value 20.99
Enter second value 30.77
```

```
sum=x+y
diff=x-y
mul=x*y
div=x/y
```

```
from fractions import Fraction
print (Fraction(sum))
print (Fraction(diff))
print (Fraction(mul))
print (Fraction(div))
```

```
7284572397271777/140737488355328
-344103159028777/35184372088832
5681064870337209/8796093022208
3072166271644677/4503599627370496
```

Q2 WAP to take 5 float values and print maximum and minimum of them

```
lst = []
n = int(input("Enter number of elements : "))

for i in range(0, n):
    ele = int(input())
    lst.append(ele)

print('The maximum is '+ str(max(lst)))
print('The minimum is '+ str(min(lst)))
```

```
Enter number of elements : 4
1
2
3
4
The maximum is 4
The minimum is 1
```

Q3 WAP to print your name in uppercase, lowercase and titlecase

```
name=input("Enter your name : ")
print(name.upper())
print(name.lower())
print(name.title())
```

```
Enter your name : Jos
JOS
jos
Jos
```

2 Operators in python

Objectives: To understand the types and uses of operators in python.

Theory:

Types:

Arithmetic operators

Assignment operators

Comparison operators

Logical operators

Identity operators

Membership operators

The Higher the Position, the Higher the Precedence
**
~ + (unary) – (unary)
* / % //
+ -
&
^ |
<= >= < >
<> == !=
= %= /= //= -= += *= **=
not and or

List of problems:

Q1. WAP to take two floats as input and print all the permutations of the applicable operators on the

In [1]:
```
a=float(input("Enter first value "))
b=float(input("Enter second value "))
```

```
Enter first value 10.37
Enter second value 36.99
```

In [2]:
```
#1. Addition (+)
print(a+b)
```

```
47.36
```

In [3]:
```
#2. Subtraction (-)
print(a-b)
```

```
-26.620000000000005
```

In [4]:
```
#3. Multiplication (*)
print(a*b)
```

```
383.5863
```

In [5]:
```
#4. Division (/)
print(a/b)
```

```
0.28034603947012704
```

In [6]:
```
#5. Modulus(%)
print(a%b)
```

```
10.37
```

In [7]:
```
#6. Exponentiation (**)
print(a**b)
```

```
3.746762792880301e+37
```

```
#7. Floor Division (//)
print(a//b)
```

```
0.0
```

2. Assignment Operators:

Assignment operators are used to assign values to variables:

```
# "+=" Operator
a+=b
print(a)
```

```
47.36
```

```
# '-=' Operator
a-=b
print(a)
```

```
10.369999999999997
```

```
# '*=' Operator
a*=b
print(a)
```

```
383.58629999999994
```

```
# '/=' Operator
a/=b
print(a)
```

```
10.369999999999997
```

```
# '%=' Operator
a%=b
print(a)
```

```
10.369999999999997
```

```
# '//=' Operator
a//=b
print(a)
```

```
0.0
```

```
# "**="
a**=b
print(a)
```

```
0.0
```

3. Comparison Operators:

Comparison operators are used to compare two values:

```
# Equal (==) Operator
print(a==b)
```

```
False
```

```
# Not equal (!=) Operator
print(a!=b)
```

```
True
```

```
# Greater than (>) Operator
print(a>b)
```

```
False
```

```
# Less than (<) Operator
print(a<b)
```

```
True
```

```
# Greater than or equal to (>=) Operator
print(a>=b)
```

```
False
```

```
# Less than or equal to (<=) Operator
print(a<=b)
```

```
True
```

4. Logical Operators:

Logical operators are used to combine conditional statements:

```
# 'and' Operator : Returns True if both statements are true
print(a and b)
print(a < 5 and  b < 10)
```

```
0.0
False
```

```
# 'or' Operator : Returns True if one of the statements is true
print(a or b)
print(a < 5 or b < 4)
```

```
36.99
True
```

```
# 'not' Operator: Reverse the result, returns False if the result is true
print(not(a < 5 and b < 10))
```

```
True
```

Q2. WAP to take 5 integer values as input and print the quotient and remainder when the maximum of them is divided by the minimum of them.

```
MyList=[]


# iterating till the range
for i in range(0, 4):
    ele = int(input())

    MyList.append(ele) # adding the element

maxEle=max(MyList)
minEle=min(MyList)

# The quotient
print(maxEle/minEle)
# The remainder
print(maxEle%minEle)
```

```
3
4
5
2
2.5
1
```

3 Loops and conditional statements

Objectives: To know about different types of loops and other statements.

Theory:

Conditional Statements:

General Format:

```
if (conditions to be met):
tab {do this}
{and this} ...

else:
{do this}
```

Else if in python:

```
if(condition_1):
{dothis}
elif(condition_2):
{dothis}
```

Elif....

else:

{dothis}

Loops:

1 While Loop:

while(condition):

tab {dothis}

{andthis}

{...}

Nesting while:

while loop inside another while loop.

while (condition_1): while(condition_2):

tab {tab} {do this}

else:

{do this instead}

else:

{do this}

List of problems:

Q1. Input an integer, and print the multiplication table for the number.

```
x=int(input("Enter the number "))
```

```
Enter the number 139
```

```
for i in range (1,11):
    mul=x*i
    print("{} x {} = {}".format(x,i,mul))
```

```
139 x 1 = 139
139 x 2 = 278
139 x 3 = 417
139 x 4 = 556
139 x 5 = 695
139 x 6 = 834
139 x 7 = 973
139 x 8 = 1112
139 x 9 = 1251
139 x 10 = 1390
```

Q2. Input 10 characters using while loop and count how many vowels are there.

```
n=10
while n:
    c= (input("Enter character : "))
    if c=='a' or c=='e' or c=='i' or c=='o' or c=='u' or c=='A' or
c=='E' or c=='I' or c=='O' or c=='U':
        print("It is a vowel")
    else:
        print("It is Not a vowel")
    n-=1
```

```
Enter character : w
It is Not a vowel
Enter character : e
It is a vowel
Enter character : r
It is Not a vowel
Enter character : t
It is Not a vowel
Enter character : y
It is Not a vowel
Enter character : u
It is a vowel
```

4 Functions

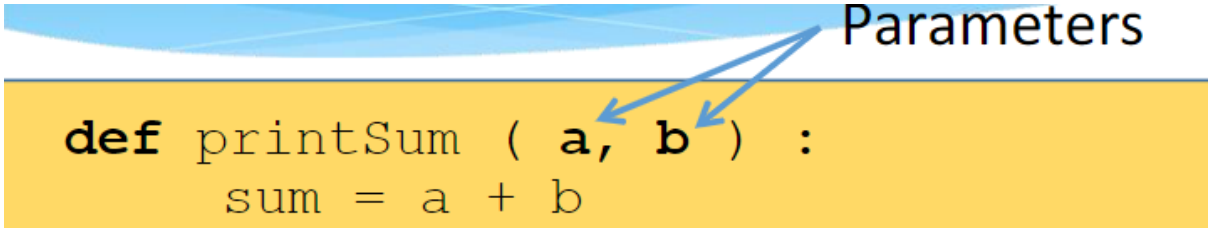
Objectives: To understand different types of functions and how are they implemented.

Theory:

Functions are self -contained "modules" of code that take inputs, do a computation, and produce outputs.

Syntax:

```
def <function name> (< param1 > ,
< param2 >):
< line1 >
< line2 >
< line3 >
```



The number of arguments "passed" into a function must exactly match the number of parameters required for the function.

List of problems:

Q1. Take a single input from user and return its integer value.

```
def MyInt(a):
    return int(a)

inp=float(input("Enter the value : "))

# Checking
# print(type(inp))

fun=MyInt(inp)
print(fun)
```

```
Enter the value : 343.442366
343
```

Q2. Print the results of the arithmetic operations as shown

Output:

Enter the value of a

25

Enter the value of b

5

Adding 25 with 5 Ans: 30

Subtracting 5 from 25 Ans: 20

Multiplying 25 with 5 Ans: 125

Dividing 25 by 5 Ans: 5

```
a=int(input("Enter the value of a "))
b=int(input("Enter the value of b "))
```

```
Enter the value of a 25
Enter the value of b 5
```

```
def add(a,b):
    return a+b
def sub(a,b):
    return a-b
def mul(a,b):
    return a*b
def div(a,b):
    return a/b

print('Adding {0} with {1} Ans: {2}'.format(a,b,add(a,b)))
print('Subtracting {0} from {1} Ans: {2}'.format(b,a,sub(a,b)))
print('Multiplying {0} with {1} Ans: {2}'.format(a,b,mul(a,b)))
print('Dividing {0} by {1} Ans: {2}'.format(a,b,add(a,b)))
```

```
Adding 25 with 5 Ans: 30
Subtracting 5 from 25 Ans: 20
Multiplying 25 with 5 Ans: 125
Dividing 25 by 5 Ans: 30
```

5 Exception handling

Objectives: To understand the concept of exceptions in python.

Theory:

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them-

1 Exception handling

2 Assertion

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

If you have some suspicious code that may raise an exception, you can defend your program by code that may raise an exception, you can defend your program by placing the suspicious code in a placing the suspicious code in a try: block. After the try: block, include an block. After the try: block, include an except: statement, followed statement, followed by a block of code which handles the problem as elegantly as possible.by a block of code which handles the problem as elegantly as possible.

Simple syntax:

You do your operations here;

.....

except Exception I:

If there is Exception I, then execute this block.

except Exception II:

If there is Exception II, then execute this block.

.....

else:

If there is no exception then execute this block.

List of problems:

1. Handling the zero division error

```
try:
    a=int(input("Enter number one "))
    b=int(input("Enter number two "))
    div= a/b
    print(div)

except:
    print("Error try again")
```

```
Enter number one 1
Enter number two 0
Error try again
```

2.To handle error while working with file

```
#FileNotFound Error --> When files are not present

with open("exception.txt") as f:
    for i in f:
        print(i)
```

```
---------------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
/var/folders/_j/759_qxxj5g78z3vbyyd3zb6c0000gn/T/ipykernel_68899/13770
49080.py in <module>
----> 1 with open("exception.txt") as f:
      2     for i in f:
      3         print(i)

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

```
# Using try except block to solve problem

try:
    with open("exception.txt") as f:
        for i in f:
            print(i)
except:
    print("File is not present. Please check file path or existence of
file")
```

```
File is not present. Please check file path or existence of file
```

3.To handle value error

```
try:
    raise ValueError
except:
    print("Value error handled")
```

```
Value error handled
```

6 Collections of data

Objectives:

To understand about tuples and how to append on going list.

Theory:

Python tuples:

Tuples are used to store multiple items in a single variable.

A tuple is a collection which is ordered and unchangeable.

Example:

```
mytuple =("apple", "banana", "cherry")
print(mytuple)
```

Accessing a list:

If you insert less items then you replace

Insert a new list item: insert()

Append Items: append()

```
mylist =["apple",  "banana", "cherry"]
mylist[1:3] = ["watermelon"]
```

```
print(mylist)
```

Sets:

Sets are used to store multiple items in a single variable.

A set is a collection which is both unordered and unindexed.

```
myset = {"apple", "banana", "cherry"}
print(myset)
```

Dictionary Methods

clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

List of problems:

1. Practice all the examples for List, Tuple, Set and Dictionaries. provided in the attached pdf and submit the notebook.
2. Write a Python program to read the admission number and names of 10 students from the keyboard. Create a dictionary of these admission number and names and then display them on the screen.

Question 1

```
#List Examples
a=[1,2,3,4]
print(type(a))

a.append('apple')
print(a)
```

```
<class 'list'>
[1, 2, 3, 4, 'apple']
```

```
mylist = ["apple", "banana", "cherry"]
mylist[1:3] = ["watermelon"]
print(mylist)
```

```
['apple', 'watermelon']
```

```
mylist = ["apple", "banana", "cherry"]
i = 0
while i < len(mylist):
   print(mylist[i])
   i = i + 1
```

```
apple
banana
cherry
```

```
insert(1,'cupcake')
int(a)
list = ["apple", "banana", "cherry"]
list.insert(2, "watermelon")
int(mylist)
```

```
[1, 'cupcake', 'cupcake', 'cupcake'
['apple', 'banana', 'watermelon', '
```

```
mylist = ["apple", "banana", "cherry"]
morefruits = ["mango", "pineapple", "papaya"]
mylist.extend(morefruits)
print(mylist)
```

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

```
mylist = ["apple", "banana", "cherry"]
for x in mylist:
  print(x)
```

```
apple
banana
cherry
```

```
# Tuple

b=(1,2,3,4)
```

```
print(type(b))
mytuple = ("apple", "banana", "cherry")
print(mytuple[1])
```

```
<class 'tuple'>
banana
```

```
mylist.append("orange")
print(mylist)
```

```
['apple', 'banana', 'cherry', 'orange']
```

```
b.count(3)

print(b)
```

```
(1, 2, 3, 4)
```

```python
# Dictionary
myset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
```

```python
myset.update(tropical)
print(myset)
```

```
{'pineapple', 'apple', 'banana', 'cherry', 'mango', 'papaya'}
```

```python
c={
    'three':3,
    'four':4
}
```

```python
myset = {"apple", "banana", "cherry"}
myset.remove("banana")
print(myset)
```

```
{'cherry', 'apple'}
```

```python
myset = {"apple", "banana", "cherry"}
myset.discard("banana")
print(myset)
```

```
{'cherry', 'apple'}
```

```python
print(type(c))
```

```
<class 'dict'>
```

```python
c.get(1)
print(c.keys())
```

```
dict_keys(['three', 'four'])
```

```
car ={
    "brand":"Ford","model":"Mustang"
}
x =car.keys()
print(x) #before thechange
car["color"] = "white"
print(x) #after thechange
```

```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year',
```

```
thisdict ={
    "brand":"Ford","model":"Mustang"
}
print(thisdict)
print(thisdict["brand"])
```

```
{'brand': 'Ford', 'model': 'Mustang',
Ford
```

Question 2

```
My_dict={}
for i in range(1,11):
    Ad_No=int(input("Enter admission r
    Name=input("Enter name ")
    My_dict[Ad_No]=Name
    print()


for keys,values in My_dict.items():
    print(keys)
    print(values)
    print()
```

```
Enter admission number 102
Enter name Eoin

Enter admission number 103
Enter name Ben

Enter admission number 104
Enter name Livingstone

Enter admission number 105
Enter name Anderson

Enter admission number 106
Enter name Joe root

Enter admission number 107
Enter name Shane
```

```
Enter admission number 107
Enter name Shane

Enter admission number 108
Enter name Mike

Enter admission number 109
Enter name Steve

Enter admission number 110
Enter name David

101
Jos

102
Eoin

103
Ben

104
Livingstone

105
Anderson

106
Joe root

107
Shane
```

```
108
Mike

109
Steve

110
David
```

7 Graph plotting:

Objectives: To learn how we can illustrate various informations in forms of charts or graphs in python.

Theory:

The two parameters of pylab.plot must be sequences of the same length.

•The first specifies the x-coordinates of the points to be plotted, and the second specifies the y-coordinates.

•Together, they provide a sequence of four <x,y> coordinate pairs.

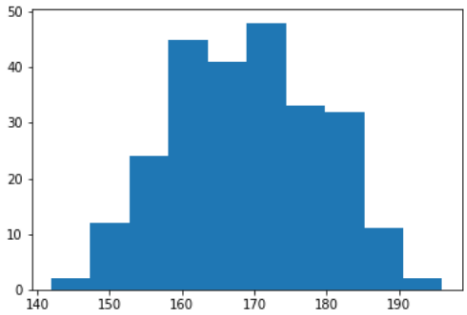
•These are plotted in order.

•As each point is plotted, a line is drawn connecting it to the previous point.

•pylab.show() causes the window to appear on the computer screen.

- There is an optional argument that is a format string indicating the colour and line type of the plot.

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```



A Histogram

List of problems:

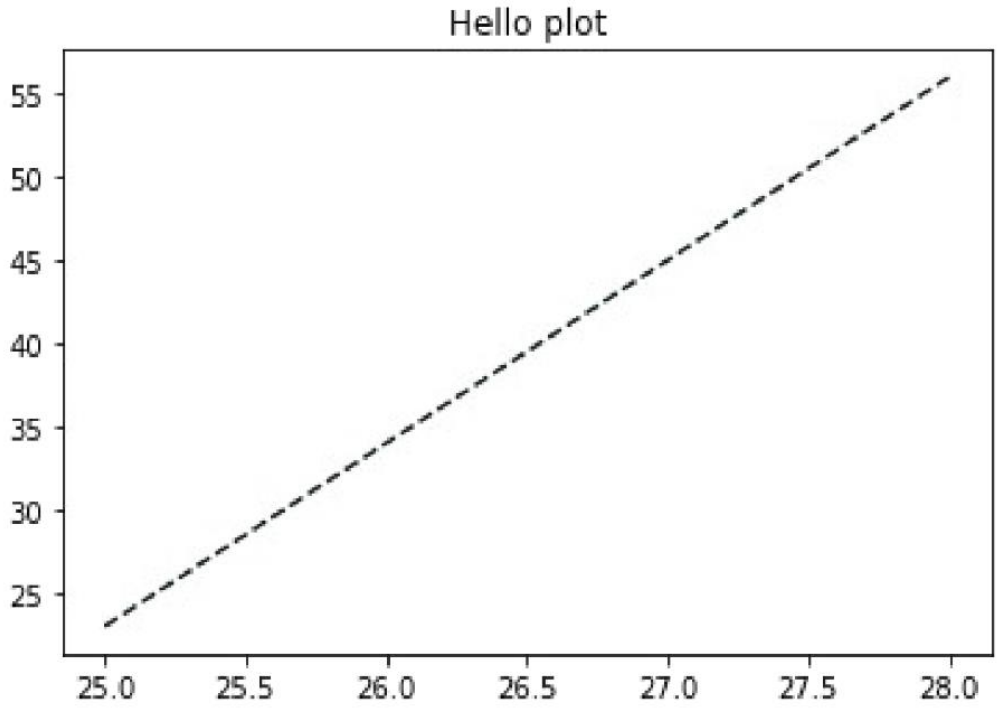
1. Practice the examples provided in the ppt.
2. Draw two different plots in the same graph (multiple plots), consisting of following:

a) Pie Chart

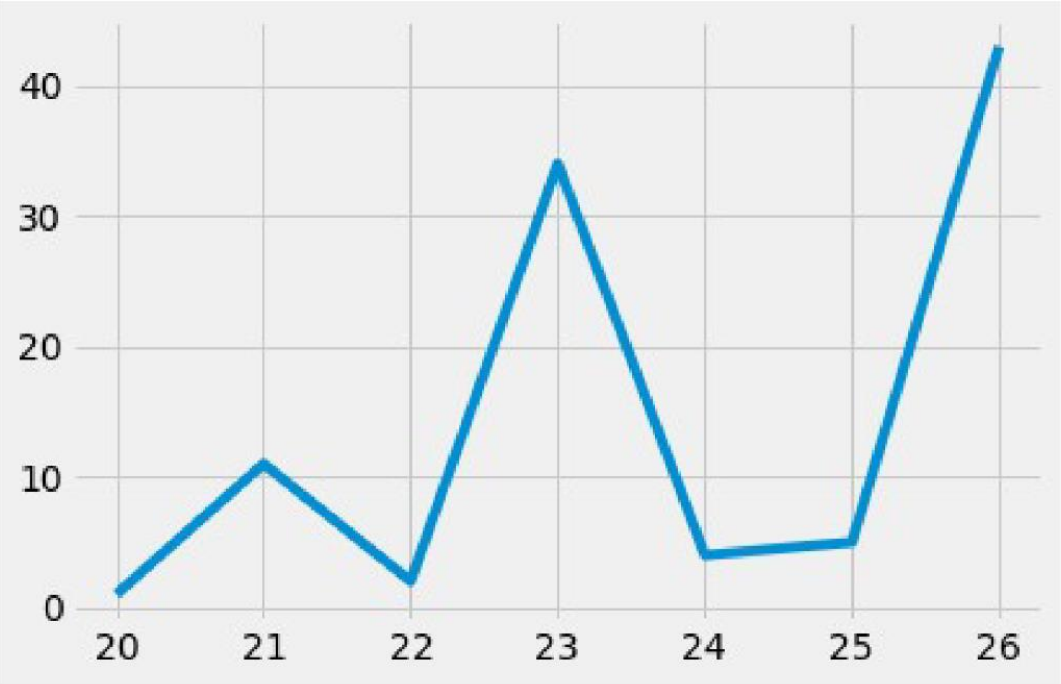
b) Line Graph

Part 1:-

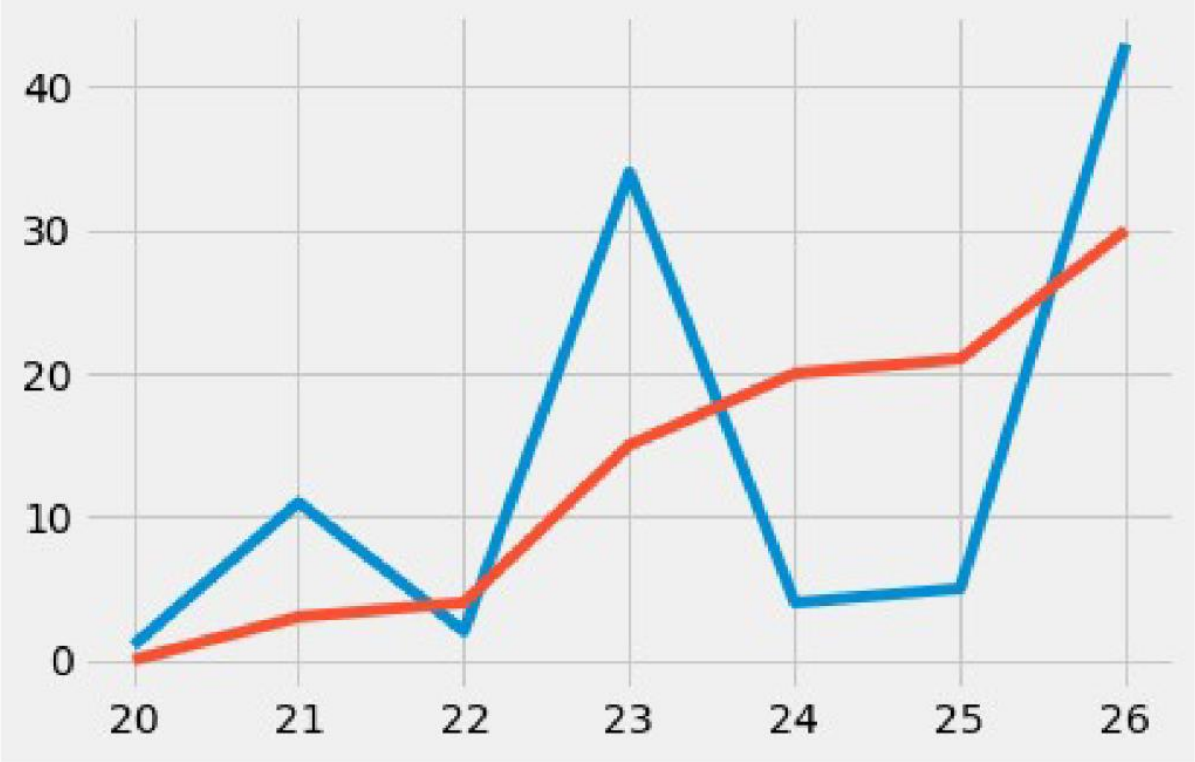
```
from matplotlib import pyplot as plt

import numpy as np

ax_x=np.array([25,26,27,28])
ax_y=np.array([23,34,45,56])

plt.plot(ax_x,ax_y,color='#111F1F',linestyle='--')


plt.title("Hello plot" )
plt.show()
```



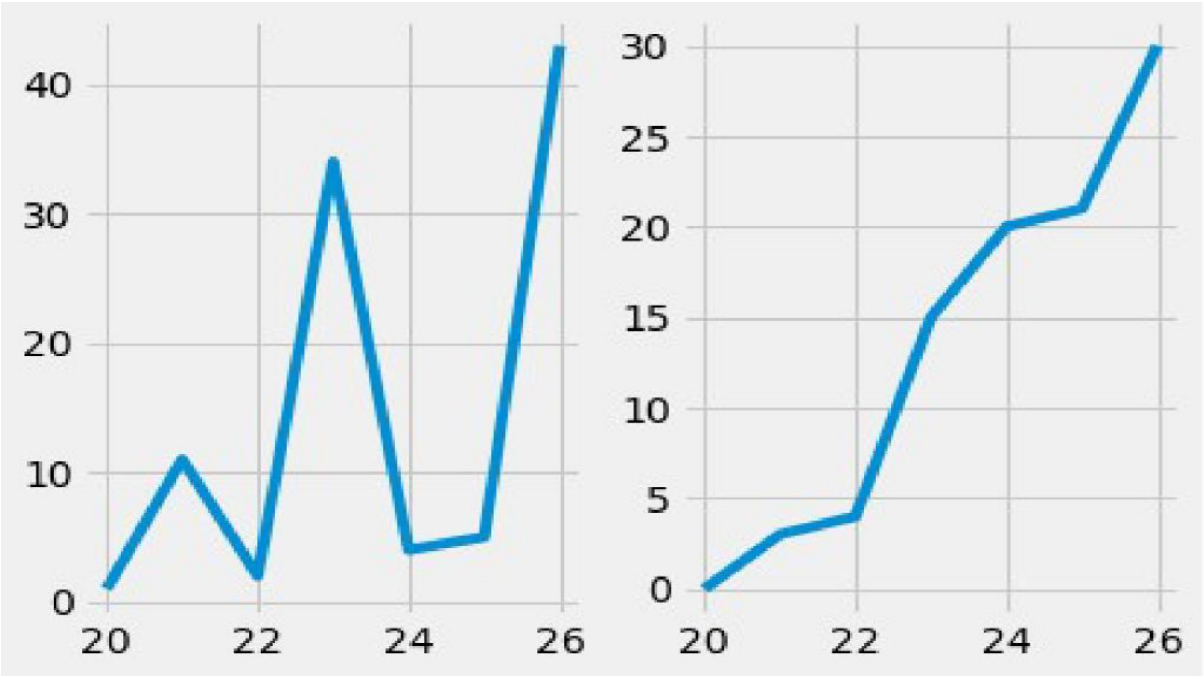
```
## Another plot
plt.style.use('fivethirtyeight')
age=[20,21,22,23,24,25,26]
money=[1,11,2,34,4,5,43]
plt.plot(age,money)
plt.show()
```




```
## 2 line graph

age=[20,21,22,23,24,25,26]
money=[1,11,2,34,4,5,43]
marriage=[0,3,4,15,20,21,30]
plt.plot(age,money,label="money")
plt.plot(age,marriage,label="marriage")
plt.show()
```

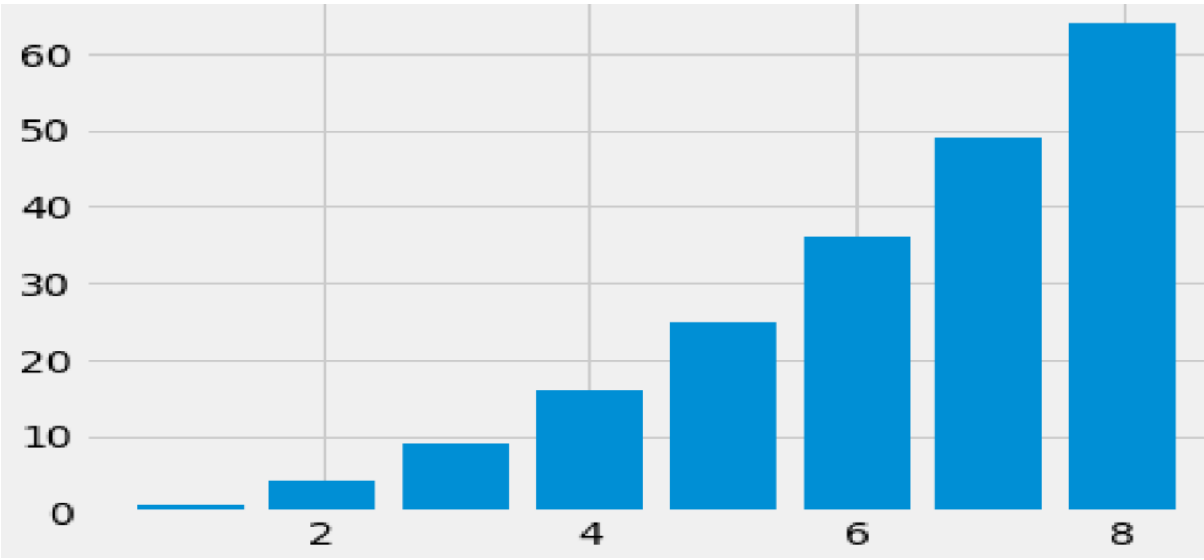



```
# Using the subplot function to make 2 plots

age=[20,21,22,23,24,25,26]
money=[1,11,2,34,4,5,43]
marriage=[0,3,4,15,20,21,30]
plt.subplot(1,2,1)
plt.plot(age,money,label="money")
plt.subplot(1,2,2)
plt.plot(age,marriage,label="marriage")
plt.show()
```

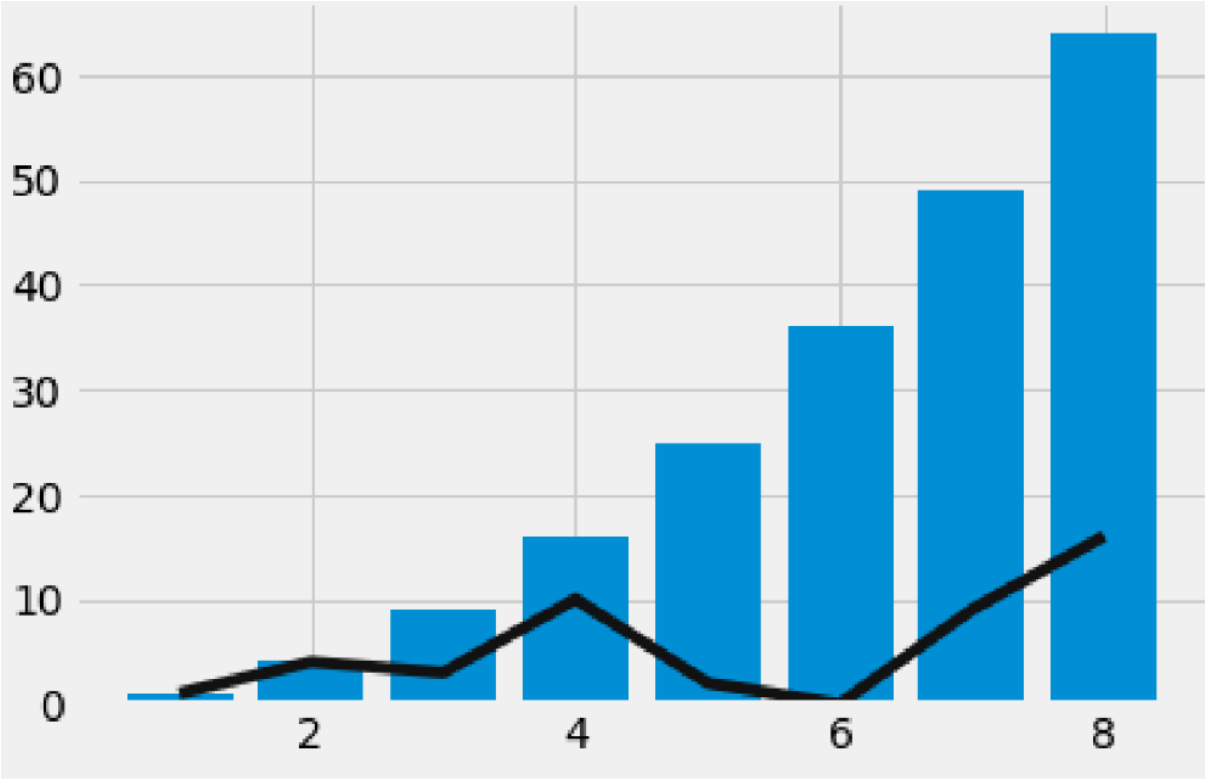


Bar Graph in matplotlib

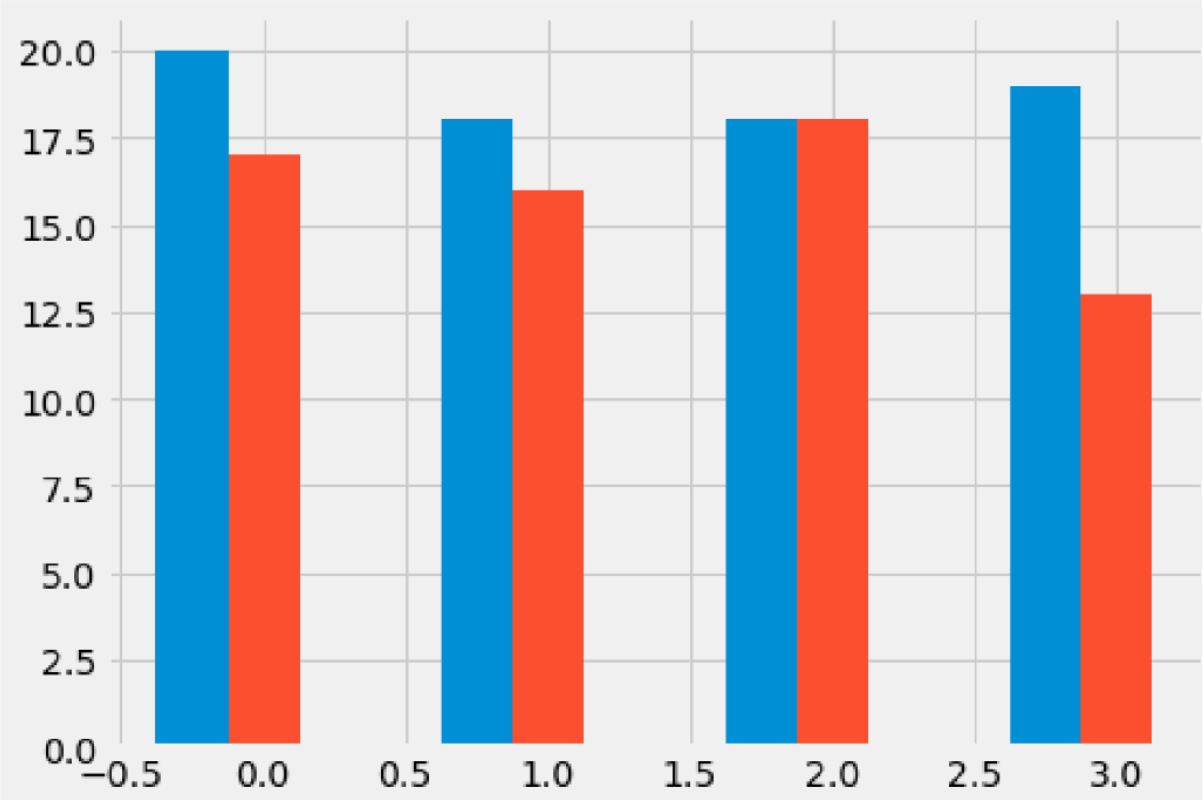
```
x=[1,2,3,4,5,6,7,8]
y=[1,4,9,16,25,36,49,64]
plt.bar(x,y)
plt.title("x vs y")
plt.show()
```



```
x=[1,2,3,4,5,6,7,8]
y=[1,4,9,16,25,36,49,64]
z=[1,4,3,10,2,0,9,16]
plt.bar(x,y)
plt.plot(x,z,color='#111111')
plt.title("x vs y")
plt.show()
```

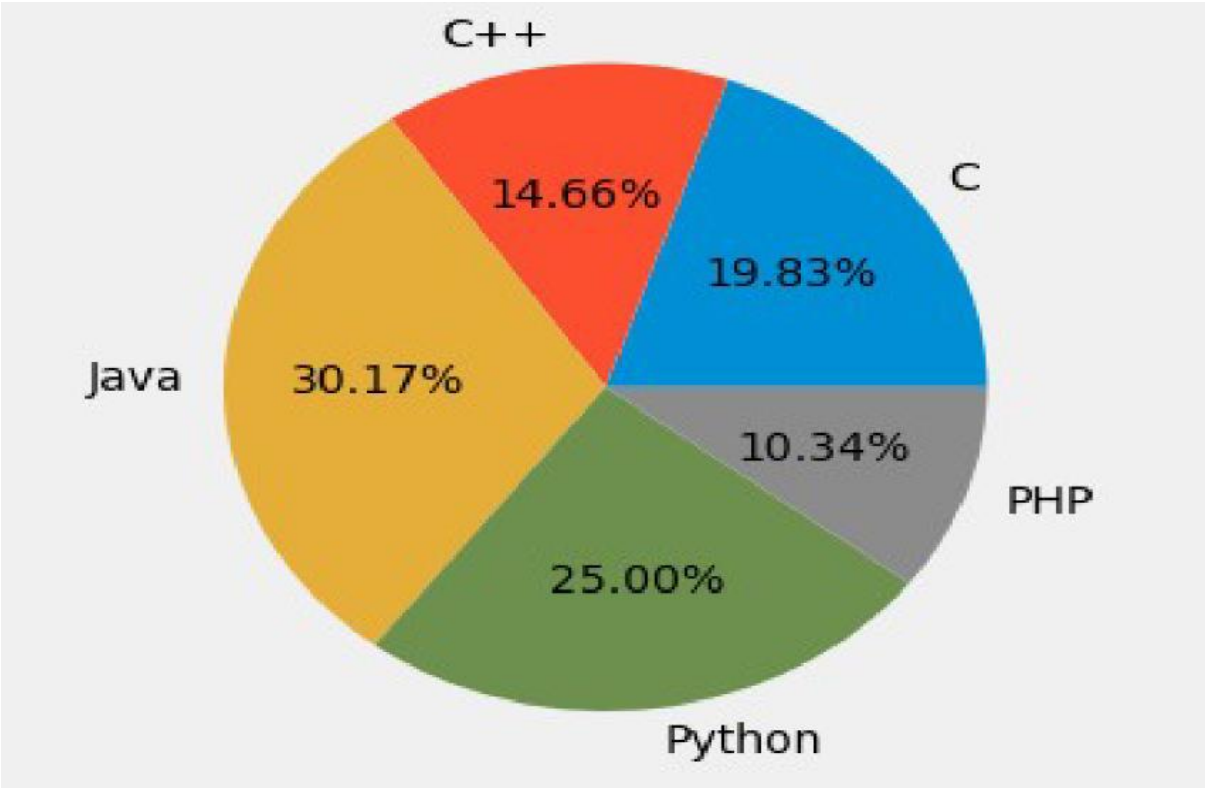



```
# Now what is we want to use 2 bar graphs and compare them side by side
# making the data for the plot

x=['maths',"physics",'chem','bio']
y_my=[20,18,18,19]
y_avg=[17,16,18,13]
index = np.arange(4)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
bar_width = 0.25
plt.bar(index-bar_width,width=bar_width,height=y_my)
plt.bar(index-bar_width+0.25,width=bar_width,height=y_avg)
plt.show()
```



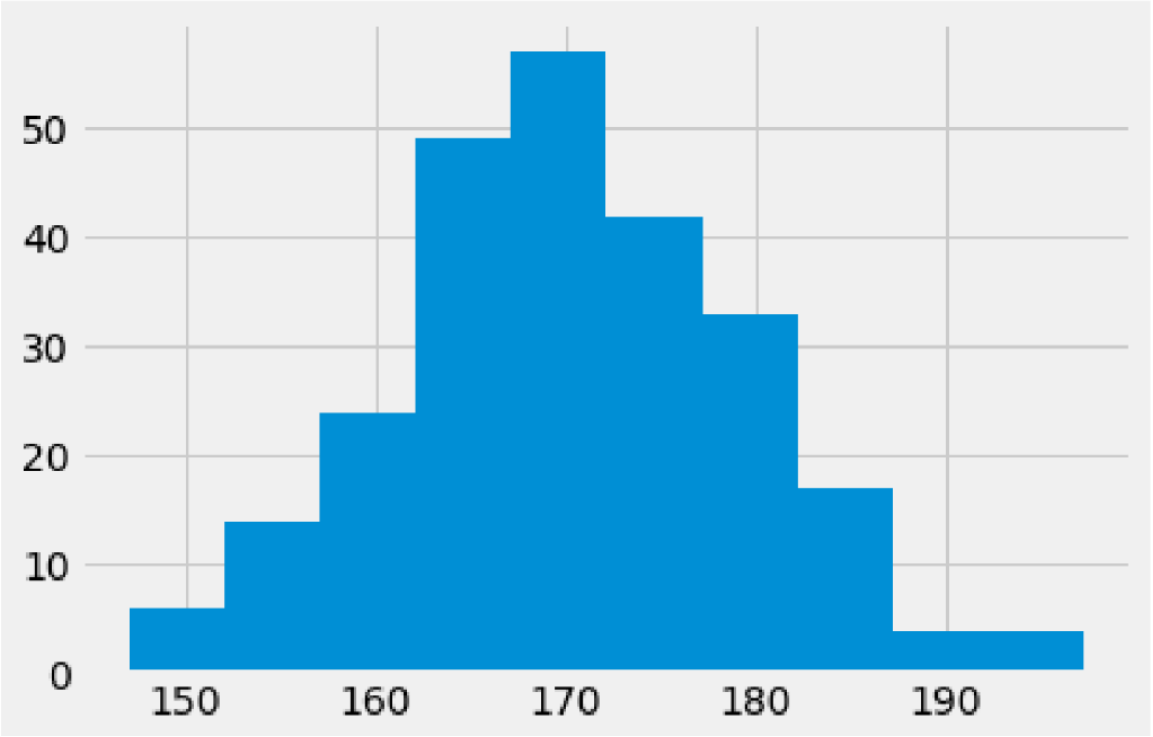
Pie chart

```
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
ax.pie(students, labels = langs,autopct='%1.2f%%')
plt.show()
```



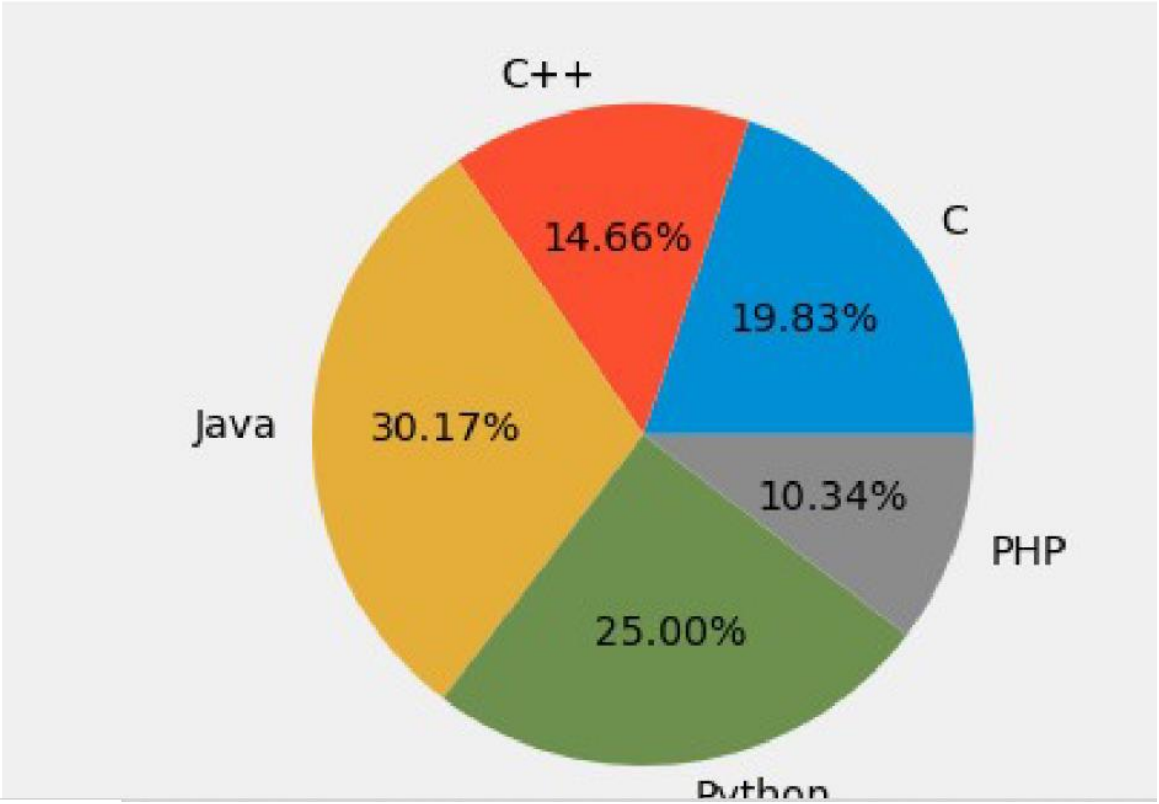
Histogram

```
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

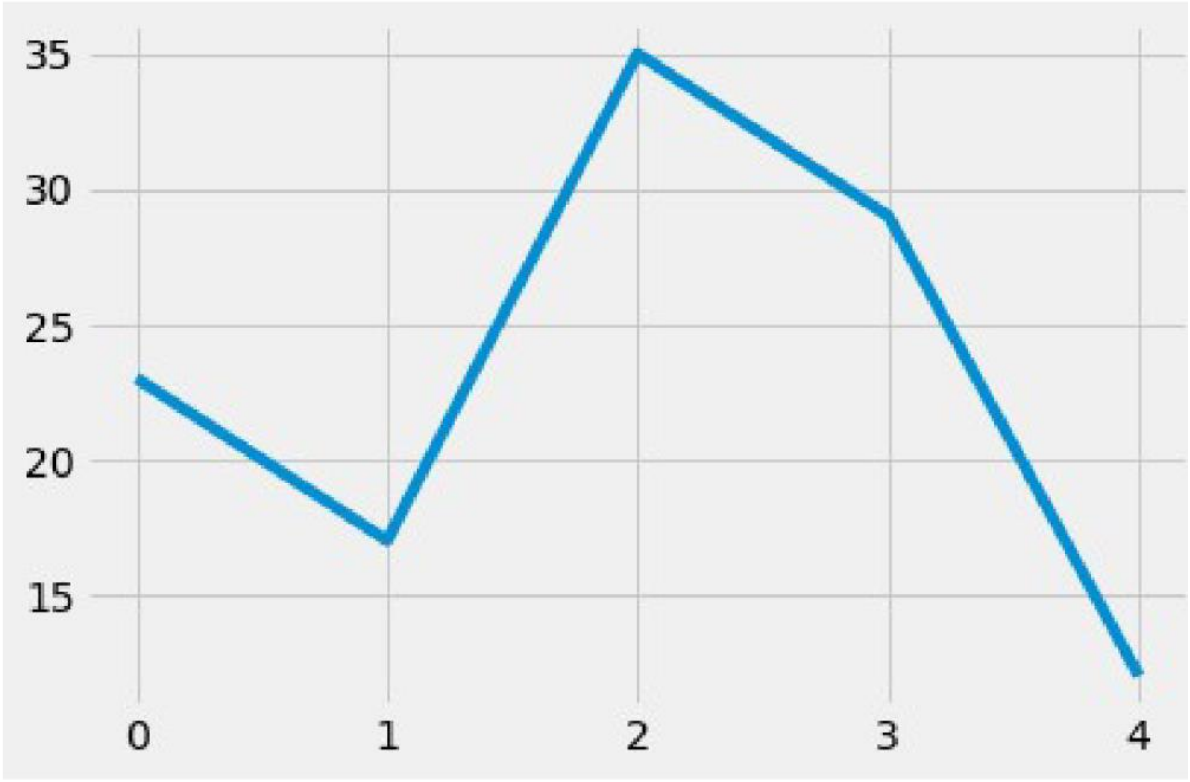


Part 2

```
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.axis('equal')
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23, 17, 35, 29, 12]
ax.pie(students, labels=langs, autopct='%1.2f%%')
plt.show()
```




```
students = [23,17,35,29,12]
plt.plot(students)
plt.show()
```


35
30
25
20
15
0
1
2
3
4

8 Object oriented Programming

Objectives: To study how python is constituted with principles of OOP.

Theory:

Python is an object-oriented programming language.

•Almost everything in Python is an object, with its properties and methods.

•Class: like an object constructor, or a "blueprint" for creating objects; data-type

•Object: It is an instance of a particular class.

The init() Function

•All classes have this function which is always executed when the class is being initiated.

Self parameter:

•reference to the current instance of the class.

List of Problems:

Part 1 Implement the OOPs examples provided in the PPT.

In [20]:

```
class Dog:
    def __init__(self,n,sc,el,spot):
        self.name = n
        self.skinColor = sc
        self.earLength = el
        self.isSpotted =spot
        print("Dog has been created!!")
    def walk(self):
        print("{0} is walking!".format(self.name))
    def eat(self):
        print("{0} is eating!".format(self.name))
print("Hello dog is created")
```

```
Hello dog is created
```

In [22]:

```
doggie1 = Dog("ScoobyDoo","brown","short",True)
doggie1.walk()
print()
doggie2 =Dog('Tommy', 'white', 'long', False )
doggie2.walk()
```

```
Dog has been created!!
ScoobyDoo is walking!

Dog has been created!!
Tommy is walking!
```

In [24]:

```
class Person:
    def __init__(self, name,age):
        self.name =name
        self.age = age
    def myfunc(self):
        print("Hello my name is " +self.name)

# Creating John with age 36
p1 = Person("John",36)
p1.myfunc()
```

```
Hello my name is John
```

Part 2 Write a python code using the concept of OOPs to add two complex numbers

In [38]:

```
class Complex:
    def __init__(self, real,img):
        self.real=int(real)
        self.img=int(img)
```

```
    def printCom(self):
        print("Created complex number : {0} + {1}i".format(self.real,self.img))
    def comSum(self,number):
        re=self.real+number.real
        im=self.img+number.img
        result=Complex(re,im)
        return result
```

In [39]:

```
c1=Complex(2,3)
c2=Complex(4,5)
```

In [40]:

```
c1.printCom()
c2.printCom()
```

```
Created complex number : 2 + 3i
Created complex number : 4 + 5i
```

```
res=c1.comSum(c2)
```

In [43]:

```
res.printCom()
```

```
Created complex number : 6 + 8i
```

9 Principles of object-oriented programming

Objectives:

To understand the principles of OOP.

Theory

There are 3 principles that provide mechanisms to help implement the object-oriented model:

1.Inheritance: It is the process by which one object acquires the properties of another object.

2.Encapsulation: To restrict the access of the data (attributes) and the code (methods), their scope of access can be set as private or public.

3.Polymorphism: Polymorphism in python defines methods in the child class that have the same name as the methods in the parent class.

List of problems:

Part 1 Implement the OOPs examples provided in the PPT.

```
class Animal:
    def __init__(self, name,age1):
        self.givenname = name
        self.age =age1
    def printname(self):
        print(self.givenname,self.age)

x =Animal("Scooby", "5")
x.printname()
```

```
Scooby 5
```

```
class Dog(Animal):
    pass

x =Dog("Scooby", "5")
x.printname()
```

```
Scooby 5
```

```
class Dog(Animal):
    def __init__(self,name,age):
        super().__init__(name,age)
        self.owner='Sam'
x =Dog("Scooby", "5")
x.printname()
print(x.owner)
```

```
Scooby 5
Sam
```

```
# Understanding the Encapsulation
class Rectangle:
    __length = 0 #privatevariable
    __breadth = 0 #privatevariable
    def __init__(self):
        #constructor
        self.__length=5
        self.__breadth=3
        #printingvaluesoftheprivatevariablewithintheclass
        print(self.length)


        print(self.breadth)
rec = Rectangle()
#object created for the class 'Rectangle
#printing values of the private variable outside the class using the
object created for the class 'Rectangle'
print(rec.length)
print(rec.breadth)
```

```
---------------------------------------------------------------------------
-----
AttributeError                            Traceback (most recent call
last)
/var/folders/_j/759_qxxj5g78z3vbyyd3zb6c0000gn/T/ipykernel_60124/39092
80180.py in <module>
     10         print(self.length)
     11         print(self.breadth)
---> 12 rec = Rectangle()
     13 #object created for the class 'Rectangle
     14 #printing values of the private variable outside the class
using the object created for the class 'Rectangle'

/var/folders/_j/759_qxxj5g78z3vbyyd3zb6c0000gn/T/ipykernel_60124/39092
80180.py in __init__(self)
      8         self.__breadth=3
      9         #printingvaluesoftheprivatevariablewithintheclass
---> 10         print(self.length)
     11         print(self.breadth)
     12 rec = Rectangle()
```

```
AttributeError: 'Rectangle' object has no attribute 'length'
```

Part 2: Practice the examples of Inheritance , Encapsulation and Polymorphism provided in the link:

```
class Parrot:

    # class attribute
    species = "bird"

    # instance attribute
    def __init__(self, name, age):
        self.name = name
        self.age = age

# instantiate the Parrot class
blu = Parrot("Blu", 10)
woo = Parrot("Woo", 15)

# access the class attributes
print("Blu is a {}".format(blu.__class__.species))
print("Woo is also a {}".format(woo.__class__.species))


# access the instance attributes
print("{} is {} years old".format( blu.name, blu.age))
print("{} is {} years old".format( woo.name, woo.age))
```

```
Blu is a bird
Woo is also a bird
Blu is 10 years old
Woo is 15 years old
```

```
class Parrot:

    # instance attributes
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
    def sing(self, song):
        return "{} sings {}".format(self.name, song)

    def dance(self):
        return "{} is now dancing".format(self.name)

# instantiate the object
blu = Parrot("Blu", 10)
```

```
# call our instance methods
print(blu.sing("'Happy'"))
print(blu.dance())

Blu sings 'Happy'
Blu is now dancing

# parent class
class Bird:

    def __init__(self):
        print("Bird is ready")

    def whoisThis(self):
        print("Bird")

    def swim(self):
        print("Swim faster")

# child class
class Penguin(Bird):

    def __init__(self):
```

```
            # call super() function
            super().__init__()
            print("Penguin is ready")

        def whoisThis(self):
            print("Penguin")

        def run(self):
            print("Run faster")

peggy = Penguin()
peggy.whoisThis()
peggy.swim()
peggy.run()
```

```
Bird is ready
Penguin is ready
Penguin
Swim faster
Run faster
```

```
class Computer:

    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))
```

```
    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()

# change the price
c.__maxprice = 1000
c.sell()

# using setter function
c.setMaxPrice(1000)
c.sell()

Selling Price: 900
Selling Price: 900
Selling Price: 1000

class Parrot:

    def fly(self):
```

```
            print("Parrot can fly")

        def swim(self):
            print("Parrot can't swim")

    class Penguin:

        def fly(self):
            print("Penguin can't fly")

        def swim(self):
            print("Penguin can swim")

    # common interface
    def flying_test(bird):
        bird.fly()

    #instantiate objects
    blu = Parrot()
    peggy = Penguin()

    # passing the object
    flying_test(blu)
    flying_test(peggy)
```

```
Parrot can fly
Penguin can't fly
```

10 Sorting

Objectives:

To implement different types of sorting arrangements in python.

Theory:

Arrange the numbers in ascending order.

Selection sort:

repeatedly finding the minimum element from unsorted part putting it at the beginning.

Python has built-in sort() method

Most Python implementations runs in roughly O(n*log(n))time.

Bubble sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Insertion sort:

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands.

Merge sort:

Merge Sort is a divide and conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.

List of problems:

Implementing the Bubble Sort

```
def bubbleSort(arr):

    n = len(arr)

    # For loop to traverse through all element in an array
    for i in range(n):
        for j in range(0, n - i - 1):

            # Range of the array is from 0 to n-i-1
            # Swap the elements if the element found is greater than
the adjacent element
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

def printArr(arr):
    for i in range(len(arr)):
        print("%d" % arr[i])

myArr=[9,8,7,6,5,4,3,2,1]
bubbleSort(myArr)
printArr(myArr)
```

```
1
2
3
4
5
6
7
```

Implementing the Selection Sort

```
def selectionSort(array):
     size=len(array)
     for s in range(size):
        min_idx = s

        for i in range(s + 1, size):

            # For sorting in descending order for minimum element in
```

```
    each loop
                if array[i] < array[min_idx]:
                    min_idx = i

            # Arranging min at the correct position
            (array[s], array[min_idx]) = (array[min_idx], array[s])


    myArr=[9,8,7,6,5,4,3,2,1]
    selectionSort(myArr)

    printArr(myArr)
```

```
1
2
3
4
5
6
7
8
```

Implementing the Insertion Sort

```
def insertion_sort(list1):

        # Outer loop to traverse on len(list1)
        for i in range(1, len(list1)):

            a = list1[i]

            # Move elements of list1[0 to i-1],which are greater to
one position
            # ahead of their current position
            j = i - 1

            while j >= 0 and a < list1[j]:
                list1[j + 1] = list1[j]
                j -= 1

            list1[j + 1] = a

myArr=[9,8,7,6,5,4,3,2,1]
insertion_sort(myArr)
printArr(myArr)
```

```
1
2
3
4
5
```

Implementing the Merge Sort

```
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]

    # Merge the temp arrays back into arr[l..r]
    i = 0     # Initial index of first subarray
    j = 0     # Initial index of second subarray
    k = l     # Initial index of merged subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # Copy the remaining elements of L[], if there
    # are any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if there
    # are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

# l is for left index and r is right index of the
```

```
def mergeSort(arr, l, r):
    if l < r:

        # Same as (l+r)//2, but avoids overflow for
        # large l and h
        m = l+(r-l)//2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)


myArr=[9,8,7,6,5,4,3,2,1]
mergeSort(myArr,0,len(myArr)-1)
printArr(myArr)
```

```
1
2
3
4
5
6
7
8
9
```

```
# merge sort used to sort some values in between EXAMPLE:
myArr=[9,8,7,6,5,4,3,2,1]
mergeSort(myArr,3,len(myArr)-4)
printArr(myArr)
#this will sort the elements from position 4(3+1) to 6(10-4)
```

```
9
8
7
4
5
6
3
2
1
```