***Please study the slides of CS1101 (first-year python lab) shared with you at WeLearn before trying to solve the following questions.**

**** You must try to execute the solutions of these questions in your Python3 IDEs/compilers.**

*****These are ungraded and are only for practice (you don't have to submit their solutions to WeLearn)**

****** The initial Sets are to help students brush up on their CS1101 knowledge.**

******* The next sets will contain some new concepts taught in CS2201 Theory/Tutorial classes (Wed)**

Set 1

Q1) Lists

(a) Create a list of integers from 0 to 9 and store the list in variable x

```
x = list(range(10))
```

(b) Create a list of integers from 3 to 12 and store the list in variable y

```
y=list(range(3,13,1))
```

(c) Using a single print command, print the list in x in reverse

```
print(x[-1:-len(x)-1:-1])
```

(d)Using a single print command, print the list of odd entries (starting from index 1) in x and then the list of even entries (starting from index 0) in x

```
print(x[1:10:2])
print(x[0:10:2])
```

(e) Check whether the 4th item of x (0th index element is the 1st element) is same as the 1st item of y by extracting those items

```
if (x[3]==y[0]):
        print ("True")
else:
        print ("False")
```

(f) Print the location of the number 10 is in the list x using index()

print (x.index(10))

(g) Print the location of the number 7 is in the list y using index()

print (y.index(7))

(h) Get a combined (appended) list of the items of x and y

print(list(x)+list(y))

(i) Create a combined list that consists of x and y. Find the location of the maximum and minimum numbers in this combined list (use max() and min()).

(x+y).index(max(x+y))

(x+y).index(min(x+y))

Q2) Strings are lists

(a) Store a string:“The quick brown fox jumps over the lazy dog” in a variable x

x="The quick brown fox jumps over the lazy dog"

(b) Check whether the word fox is in this sentence

```
x="The quick brown fox jumps over the lazy dog"

if 'fox' in x:
   print("Present")
```

(c) Print the sentence in reverse order

print (x[-1:-len(x)-1:-1])

(d) Print every third character of the above sentence (0th, 3rd, 6th….characters).

print (x[0:len(x):3])

(e) Print every fourth character of the above sentence (0th, 4th, 8th…characters).

```
print (x[0:len(x):4])
```

(f) Find how many characters are there in the sentence (iprint(Including spaces)

```
len(x)
```

(g) Print every second character of the sentence starting from the last character in reverse order

```
print (x[-1:-len(x)-1:-2])
```

(h) Store the first four characters of x in a variable y and the last three letters in a variable z.

```
>>> y=x[0:4]
>>> print (y)
The
>>> z=x[40:]
>>> print (z)
dog
```

Print the output of y + z

```
>>> print (y+z)
The dog
```

(i) Print the output of y*10

```
print (y*10)
```

Q.3 Miscellaneous

a) Print 'Hello World' on the screen

b) Add a comment to the above program

c) Store your name (e.g. Tintin), age (e.g. 20) and roll number (e.g. 20MS1234) in three variables and print them separately and also together like *Hello! My name is Tintin . I am 20 years old. My roll number is 20MS1234* using these variables. Use string concatenation. Also, use formatted output.

```
Name='Tintin'
Age=20
RollNo='20MS1234'

print('Name=', Name,' Age=', Age, ' Roll Number=', RollNo)
print('Hello! My name is', Name,'. I am',Age,'years old. My roll number is', RollNo)
```

```
print("Hello! My name is {}. I am {} years old. My roll number is
{}".format(Name, Age, RollNo))
```

d) Take two number strings as input and print their sum

```
a=input('Give a number:')
b=input('Give another number')
print('The sum=', a+b)
```

e) Take two integers as input and print their sum

```
a=int(input('Give a number:'))
b=int(input('Give another number'))
print('The sum=', a+b)
```

f) Use print() to print the statement -- It's good to learn Python

```
print("It's good to learn Python")
```

g) Use print() to print the statement -- The man asked, "Where to meet you?" I said, "Well, use Google Meet!"

```
print('The man asked, "Where to meet you?" I said, "Well, use Google Meet!"' )
```

h) Store an integer, floating point number and character in different variables and print the data type of each variable.

```
integer = 1
floating = 1.5
str = 'c'

print(type(integer))
print(type(floating))
print(type(str))
```

i) Take your name (e.g. Feluda) in the variable 'Name' as input and print *My name is Feluda* using Name and string concatenation. Do the same using %s.

```
name=input("Give your name: ")
print("My name is " + name)

name=input("Give your name: ")
print("My name is %s" %(name))
```

Q4. Write a Python function that takes two lists and returns True if they have at least one common member.

```
# Define a function called 'common_data' that takes two lists, 'list1' and 'list2', as input
def common_data(list1, list2):
    # Initialize a variable 'result' to False to indicate no common elements initially
    result = False

    # Iterate through each element 'x' in 'list1'
    for x in list1:
        # Iterate through each element 'y' in 'list2'
        for y in list2:
            # Check if the current elements 'x' and 'y' are equal
            if x == y:
                # If there's a common element, set 'result' to True and return it
                result = True
                return result

# Call the 'common_data' function with two lists and print the result
print(common_data([1, 2, 3, 4, 5], [5, 6, 7, 8, 9]))
# Call the 'common_data' function with two lists and print the result
print(common_data([1, 2, 3, 4, 5], [6, 7, 8, 9]))
```

Q5. Write a Python program to convert a list of characters into a string.

```
# Define a list 's' containing individual characters
s = ['a', 'b', 'c', 'd']

# Use the 'join' method to concatenate the characters in the list 's' into a single string
str1 = ''.join(s)

# Print the concatenated string 'str1'
print(str1)
```

Q6. Write a Python program to remove the K'th element from a given list, and print the updated list.
Original list:
[1, 1, 2, 3, 4, 4, 5, 1]
After removing an element at the kth position of the said list:
[1, 1, 3, 4, 4, 5, 1]

```
# Define a function 'remove_kth_element' that takes a list 'n_list' and an integer 'L' as input
def remove_kth_element(n_list, L):
    # Return a modified list by removing the element at the kth position (L-1) from the input list
    return n_list[:L - 1] + n_list[L:]

# Create a list 'n_list' containing integers
n_list = [1, 1, 2, 3, 4, 4, 5, 1]

# Print a message indicating the original list
print("Original list:")
# Print the original list
print(n_list)

# Assign an integer 'kth_position' with the value 3
kth_position = 3

# Call the 'remove_kth_element' function with 'n_list' and 'kth_position'
# and store the result in the 'result' variable
result = remove_kth_element(n_list, kth_position)

# Print a message indicating the list after removing an element at the kth position
print("\nAfter removing an element at the kth position of the said list:")
# Print the 'result' list
print(result)
```

Set 2

1. Write a program which prints the following sequence of strings using a for loop

mitochondria
mitochondria
mitochondri
mitochondr
mitochond
mitochon
mitocho
mitoch
mitoc
mito
mit
mi
m

```
a = "mitochondria"

for i in range(len(a)+1, 0, -1):
    print(a[:i:])
```

2. Consider a list of names of chemical elements (e.g. sodium, potassium etc.) and write a program that finds the longest name in that list. Using “format()” print the longest name thus found with its length.

```
words = ["sodium", "potassium", "phosphorus", "calcium", "iodine"]

l = 0


for w in words:
    if len(w) > l:
        word_longest = w
        l = len(w)

print("The longest word is {} with length {}".format(word_longest, l))
```

3. Take your full name as input and make an abbreviation of your name based on the initials of the names.

```
name=input("Your name: ")
l = name.split()
print(l)
abbre=""

for n in range(0, len(l)-1):
    abbre = abbre + l[n][0] + ". "

abbre = abbre + l[len(l)-1]
print(abbre)
```

4. Write a program that prints the first 10 elements of a Fibonacci series 1, 1, 2, 3,5, 8, 13,....., where each element is the sum of the two previous elements (the first two numbers are defined to be 1).

```
f1 = 1
f2 = 1
print(str(f1))
print(str(f2))


for i in range(1, 9, 1):
    f = f1 + f2
    print(str(f))
    f1 = f2
    f2 = f
```

5. Print the pattern using nested for loops (for n lines):

1

1 2
1 2 3
1 2 3 4
1 2 3 4 5

```
n = 5
for i in range(1, n+1):
   for j in range(1, i+1):
       print(str(j), end = " ")
   print("\n")
```

6. Print the pattern using nested for loops (for n lines):

1

1 3

1 3 5

1 3 5 7

1 3 5 7 9

```
n = 5

for i in range(1, n+2, 1):
    for j in range(1, 2*i-1, 2):
        print(str(j), end=" ")

    print()
```

7. Print the pattern for n lines:

1 2 3 4 5

1 2 3 4

1 2 3

1 2

1

8. Print the pattern for n lines (mind the left spaces):

1 2 3 4 5 4 3 2 1

  2 3 4 5 4 3 2

    3 4 5 4 3

      4 5 4

        5

```
n = 5
for i in range(1, n+1):
    for l in range(1, i):
        print("  ", end="")
    for j in range(i, n+1):
        print(str(j) + " ", end="")
    for j in range(n-1, i-1, -1):
        print(str(j) + " ", end="")

    print("\n")
```

9. Without taking using input create a dictionary with some userid: password pairs as key:value pairs.
   i) Now take a userid from the user (use input()). If this userid is in your dictionary, print, ask for password; if the password is also correct print “Welcome to the portal!”; otherwise (if userid or password is wrong) print “Invalid credentials!”
   ii) Create a service for password change. Take a userid from the user (use input()). If this userid is in your dictionary, ask for a new password (twice). If the provided passwords are the same (and different from the existing password),

update the corresponding password in the dictionary. If the userid is not in your dictionary, print “Invalid username”.

i)

```
useridpassword = {"kripa": "kr123", "arun": "xyz34", "asha": "asha1990", "disha": "dish99"}

userid = input("Give user id: ")

if userid in useridpassword:
    password = input("Password? ")
    if useridpassword[userid] == password:
        print("Welcome to the portal!")
    else:
        print("Invalid credentials!")
```

ii)

```
useridpassword = {"kripa": "kr123", "arun": "xyz34", "asha": "asha1990", "disha": "dish99"}

userid = input("Give user id: ")

if userid in useridpassword:
    password = input("New Password? ")
    password1 = input("New password (again)? ")
    if password == password1 and password != useridpassword[userid]:
        useridpassword[userid] = password
        print(useridpassword)
    else:
        print("Updation failed!")



else:
   print("Invalid username!")
```

10. Using the Tabulation Method find an interval containing a root of the equation $\sin(x) + x^2 - 1$ in the interval [0, 1]. You may use the sin() function in the math module.

```
import math

def f(x):
    return math.sin(x) + x*x  - 1


import numpy as np

i1 = 0.0 #left interval point
i2 = 1.0 #right interval point

l = i1
r = i2
1
f1=f(l) #value at 0.0
l1 = l #0.0

step = 0.1

l+=step #0.1

for n in np.arange(l, r+step, step):
    f2=f(n)
    print("l1=" + str(l1) + " f(l1)=" + str(f1) + " n=" + str(n) + " f2=" + str(f2))
    if f1*f2 < 0.0:
        print("Desired interval :" + str(l1)+ " " + str(n))

        break



    l1 = n
    f1 = f(l1)
```

11. Using Tabulation method find a better approximate interval in the interval [0, 1] and then on this better interval use Bisection to find the root of the equation sin(x) + x^2 -1 = 0 . You may use the sin() function in the math module.

```
def f(x):
     return math.sin(x) + x*x -1

def bisection(a, b):
   if(f(a)*f(b)>=0):
      print("Invalid interval!")
      return

   mid = a


   while (b - a) >= 0.001:

      mid = (a + b)/2.0

    print("%f %f %f %f %f %f" %(a, f(a), b, f(b), mid, f(mid)))
      if(f(a)*f(mid) < 0):
         b = mid
      else:
         a = mid



   return mid

sol = bisection(0.6, 0.7)
print("The root is ", "%.4f"%sol)
```

12. Consider a ball dropping from a height ‘h’ (value, in cms, taken as input from the user). Each time the ball hits the ground, it bounces up half the immediate previous height it was at. That is after it is dropped from height h, it hits the ground and bounces up height h/2, drops down again, hits the ground, and bounces up height h/4, and so on. Write a program that prints these heights h, h/2, h/4…n times, where n is taken as input from the user. However, if the height reaches 0.05 cm, you should stop immediately. Also, you should print these heights as floating-point numbers using formatted output.

```
h = int(input("Height: "))
n = int(input("Bounce no: "))
```

```
i = 1
ht = h

while ht > 0.05 and i <= n:
    print("Height is %f" %ht)
    ht = ht/2
    n = n + 1
```

Set 3

1. Given a list L = [1, 3, 5, 7] add the elements 2, 4, 6 after 1, 3, 5 respectively.

```
L = [1, 3, 5, 7]

L.insert(1, 2)
L.insert(3, 4)
L.insert(5, 6)
```

2. In the above list, remove the last element.

```
L.pop(len(L)-1)
```

3. Write a Push() function that adds an element at the end of a List L and returns the updated List L.

```
def Push(L, e):
    L.append(e)

L = []
Push(L, 1)
Push(L, 2)
Push(L, 3)
Push(L, 4)
Push(L, 5)
print(L)
```

4. Write a Pop(L) function that removes the last element in a List L and returns the deleted element. Apply Pop() on list L in Q3 to remove the last element.

```
def Pop(L):
```

```
    return L.pop(len(L)-1)

print(Pop(L))
print(L)
```

5. Use Push() to add five names to a list Name and using Pop() display the names in the reverse order.

```
def Push(L, e):
    L.append(e)

def Pop(L):
    return L.pop(len(L)-1)

l = ["messi", "neymar", "ronaldo", "lukaku"]
Names = []
for i in range(0, len(l)):
    Push(Names, l[i])

print(Names)

for i in range(0, len(l)):
    print(Pop(Names))
```

6. Given a list L with repeating elements, remove the duplicates. You should not use any temporary list.

```
L = [1, 2, 2, 3, 3, 3, 4, 5]

L = list(set(L))
```

7. Given a list L = [1, 2, 3, 4, 5], use list comprehension to generate another list L1 containing the even numbers in L

```
L1 = [1, 2, 3, 4, 5]

L2 = [x for x in L1 if x%2 == 0]
```

8. Given two lists L1 = [1, 2, 3, 4, 5] and L2 = [5, 4, 10, 12], use list comprehension to generate another list L3 containing the sum of the odd elements in L1 and L2.

```
L1 = [1, 2, 3, 4, 5]
L2 =  [5, 4, 10, 12]
L3 = [(x+y) for x in L1 for y in L2 if x%2 == 1 and y%2 == 1]
```

9. Use a while loop to take a number as input from the user and continue the loop until the user inputs an even number.

```
while True:
    n = int(input("Give an even number: "))
    if n%2 == 1:
        print("The number is odd!")
    else:
        break
```

10. Use while loop to reverse an integer stored in a variable n and store the reversed number in a variable rev. You can't use a list, string operations or any Python functions for reversal.

```
n = 123
rev = 0

while n != 0:
    rev = rev*10 + n%10
    n = n//10


print(rev)
```

11. Take a number as input (use input()) and using the dictionary d = {1 : 'ONE', 2 : 'TWO', 3 : 'THREE', 4 : 'FOUR', 5 : 'FIVE', 6 : 'SIX', 7 : 'SEVEN', 8 : 'EIGHT', 9 : 'NINE'} only print the number as words. If the input is 1234, the output should be ONE TWO THREE FOUR. Note that, you have to use only this dictionary (and no

other source) to convert each digit of the input number to the corresponding word equivalent.

```
d = {1 : 'ONE', 2 : 'TWO', 3 : 'THREE', 4 :
'FOUR', 5 : 'FIVE', 6 : 'SIX', 7 : 'SEVEN', 8 :
'EIGHT', 9 : 'NINE'}

s = "3456"

for e in s:
   print(d[eval(e)], end=" ")

print()
```

12. Write a Python program to determine the direction (‘increasing’ or ‘decreasing’ or ‘not monotonic’) of monotonic sequence numbers.

```
def test_monotone(nums):
   # Check if all elements in the list 'nums' are in increasing order
   if all(nums[i] < nums[i + 1] for i in range(len(nums) - 1)):
      return "Increasing."
   # Check if all elements in the list 'nums' are in decreasing order
   elif all(nums[i + 1] < nums[i] for i in range(len(nums) - 1)):
      return "Decreasing."
   else:
      return "Not a monotonic sequence!"

# Assign a specific list of numbers 'nums' to the variable
nums = [1, 2, 3, 4, 5, 6]

# Print the original list of numbers 'nums'
print("Original list:")
print(nums)

# Print a message indicating the operation to be performed
```

```
print("Check the direction ('increasing' or 'decreasing') of the said list:")

# Print the result of the test function applied to the 'nums' list
print(test_monotone(nums))
```

https://www.w3resource.com/python-exercises/puzzles/index.php

13. Using Newton-Raphson method find all the roots of the equation $f(x) = x^3 + x^2 - x = 0$ in the interval [0, 1]. Note that the first order derivative of f(x) is $3x^2 + 2x - 1$.

```
import math

def f(x):
    return x**3 + x**2 - x

def df(x):
     return 3.0*x**2 + 2*x - 1


def newtonRaphson(x):
    h = f(x)/df(x)




    while abs(f(x)) >= 0.0001:

        h = f(x)/df(x)

        x = x - h



        print('x, f, df, h values: %f %f %f %f' %(x, f(x), df(x), h))



    return x
```

```
print("The root is ", "%.4f"%newtonRaphson(1.0))
```

14. Design an automatic Quizzing System (choose an apt name like “Proshnobaan”) that asks science questions. Use a Python dictionary (KB) to store the QAs with correct and wrong options and scores for the correct option. Each element in KB is a key:value pair, key being the question (e.g. "What is the smallest known organism?") and the value is a list of tuples. Each element of the list is a tuple in the format (option, correct/wrong, score), e.g. ("Mycoplasma gallicepticum", 1, 2), where "Mycoplasma gallicepticum" is an option for the question, ‘1’ indicates that this is the CORRECT option for the question and ‘2’ is the score that the participant will get if s/he chooses this option. On the other hand, the tuple for a wrong option is ("Valonia ventricosa", 0, 0). For each question, the system should ask the question, show the options and ask for the answer. After the participant enters the answer, the system will print an appropriate message like “Correct answer” or “Wrong answer”.

The system will ask a mix of easy-to-tough questions (the score will be higher for the tougher questions) one after the other and add up the scores obtained by the participant. If the total score at the end of these questions is less than or equal to a threshold, the system will announce a consolation prize. If this total score is more than the threshold, the system will ask a JACKPOT question (an unusual question like "What is the smallest resolvable unit of distance by a given computer mouse pointing device called?") WITHOUT any option. If the participant answers the jackpot question correctly, the system should print “Congratulations” with the announcement of a mega prize; otherwise, it should print an appropriate message.

```
KB = {"What is the smallest known organism?":
     [("Mycoplasma gallicepticum", 1, 2),
      ("Valonia ventricosa", 0, 0)],
     "What is the unit of luminous intensity?":
        [("Candela", 1, 1), ("Kelvin", 0, 0)]
        }

#print(KB["What is the smallest known organism?"])
score_obtained = 0
#while True:
for q in KB:
   print("Question: " + q)
```

```
    print("Your options:")
    correct_ans = ""
    score_allotted = 0
    for op in KB[q]:
        print(op[0])
        if op[1] == 1:
            correct_ans = op[0]
            score_allotted = op[2]

    print()
    ans = input("Your answer: ")
    #print(ans)
    if ans == correct_ans:
        print("Correct")
        print("Your score is %d" %score_allotted)
        score_obtained = score_obtained + score_allotted
    else:
        print("Wrong!")


print("Total score %d" %score_obtained)

if score_obtained == 3:
    print("Your Jackpot question:")
    ans_jackpot = input("What is the smallest resolvable unit of distance by a given
computer mouse pointing device called?")
    ans_jackpot = ans_jackpot.lower()
    if ans_jackpot == "mickey":
        print("Congratulations! Here comes your mega prize!!")
    else:
        print("Well played, but you missed the Jackpot :-( We still have an exciting prize for
you..")
else:
    print("Good try, please collect your consolation prize.")
```

Set 4

1. Use list comprehension to create a list containing all the elements in list L1 and not in L2.

```
L1 = [1, 2, 3, 4]
L2 = [1, 5, 6, 7]

L3 = [e for e in L1 if e not in L2]
```

2. Input the name and basic salary (B) of an employee. The total salary T = DA + HRA + MA, where DA = 17% of B, HRA = 8% of B and MA = 10% of B. Print name right adjusted with 2 padding spaces (assume the length of name is 5) and the total salary right-adjusted (assume that the salary will not be more than 10 digits) and corrected to 3 places of decimal using formatted output applying % operator.

```
name = input('Name:')
basic = float(input('Basic salary:'))

DA = basic*(17/100)
HRA = basic*(8/100)
MA = basic*(10/100)

Total_salary = DA + HRA + MA

print("The salary of %7s is %-10.3f" %(name, Total_salary))
A more generic solution if the length of name is not known:

p_n = len(name)+2
print("The salary of %{}s is %-10.3f".format(p_n) %(name, Total_salary))
```

3. Do as (3) using the format() method

```
name = input('Name:')
basic = float(input('Basic salary:'))

DA = basic*(17/100)
HRA = basic*(8/100)
MA = basic*(10/100)

Total_salary = DA + HRA + MA

print("The salary of {0:7s} is {1:-10.3f}".format(name, Total_salary))
```

4. Use numpy array to generate the following pattern for n (taken from user) lines (shown for n = 4):

1.0

1.0 1.5 2.0

1.0 1.5 2.0 2.5 3.0

1.0 1.5 2.0 2.5 3.0 3.5 4.0

```
import numpy as np

n = 5

for i in range(1, n+1):
    for j in np.arange(1, i+0.5, 0.5):
        print(str(j), end = ' ')

    print("\n")
```

5. Write a function encrypt() that accepts a string and maps each character of the string to the corresponding alphabet in the opposite order of alphabets. E.g. ‘a’ will be mapped to ‘z’, ‘b’ will be mapped to ‘y’, ‘z’ will be mapped to ‘a’ and so on. So, encrypt() will map ‘zbc’ to ‘ayx’. Assume that the input string is in the lower case. Note that ord(‘a’) gives the ASCII value of the character ‘a’ (i.e. 97) and chr(97) gives the character equivalent of 97 (i.e. ‘a’).

```
def encrypt(s):
    s_out = ''
    for c in s:
        s_out += chr(122-ord(c)+ 97)
     return s_out

  print(encrypt("zbc"))
```

6. Write a python program to swap the left and right halves of an input string s; the middle element will be unchanged for an odd length string. E.g. ‘kripa’ will be converted to ‘paikr’; ‘aman’ will be converted to ‘anam’.

```
s = 'kripa'
mid=len(s)//2

if(len(s)%2 == 0): #even
```

```
    sout = s[mid:len(s)] + s[0:mid]
else:
    sout = s[mid+1:len(s)] + s[mid] + s[0:mid]



print(sout)
```

7. Input a list from the user (containing duplicates) and create another list containing only the non-repeating elements. E.g. if the input list is [1, 2, 2, 3, 4, 4, 5], the output list will be [1, 3, 5].

```
lst = input('Give a list: (space separated, put an enter to end)').split(" ")

print(lst)


l = len(lst)

lst_out = []

for i in range(0, l):
    r = 0
    for j in range(0, l):
        if i != j:
            if lst[i] == lst[j]:
                r = 1
                break
    if r == 0:
        lst_out.append(lst[i])

print(lst_out)
```

8. Print the numpy array [[1,2,3], [4, 5, 6]] column-wise by using slicing.

```
import numpy as np


A = np.array([[1,2,3], [4, 5, 6]])
c = A.shape[1] #no of cols
for i in range(0, c):
```

```
        print(A[:,i])
```

9. Define a function calc_area (using def) that takes a radius (r) and returns the area of a circle (use math.pi). Take a list radius_list containing five radii values and use calc_area to store the corresponding areas in another list area_list.

```
import math
import numpy as np

def calc_area(r):
    return math.pi*r*r

radius_list = [1, 2, 3]
area_list = []
for l in radius_list:
    area_list.append(calc_area(l))
```

10. Repeat the problem in (2) using a numpy array radius_np containing five radii values (same as in radius_list) to store the corresponding areas in another numpy array area_np.

```
import numpy as np
radius_np = np.array([1, 2, 3])
area_np = calc_area(radius_np)
```

11. Input (use input()) the roll number and marks of three courses for 5 students and store it in a dictionary (d) such that the roll number is the key and the marks are stored in a numpy array of type float64 (the type of a numpy array A can be changed to float64 type by A.astype('float64')). For example, for a student with roll number 'ms1901' and marks 70, 80, 90, the dictionary entry will look like 'ms1901': array([70., 80., 90.])

12. Continue with Q6 to create another dictionary 'd_sum' that contains the sum of the marks. So, the corresponding dictionary entry will look like 'ms1901': 240.0. To compute sum use numpy function sum() (Note: The sum of a numpy array A is calculated as numpy.sum(A))

```
d = {}
for i in range(0, 6):
```

```
    rollno = input("Roll no ")
    marks = input("Three marks, space separated, press enter to end: ").split(" ")
    d[rollno] = np.array(marks).astype('float64')


#print(d)
d_sum = {}

for e in d:
    d_sum[e] = np.sum(d[e])


#print(d_sum)
```

13. Check if two 1-D numpy arrays are equal or not using a for loop. Verify the result with the numpy function numpy.array_equal(a1, a2) that returns True if the arrays a1, a2 are equal.

```
import numpy as np

a1 = np.array([1, 2, 3, 4])
a2 = np.array([1, 2, 3, 5])

if len(a1) != len(a2):
    print("The arrays are not equal!")
else:
    unequal = False
    for i in range(0, len(a1)):
        if a1[i] != a2[i]:
            unequal = True
            break
    if unequal == False:
        print("The arrays are equal!")
    else:
        print("The arrays are not equal!")

print(np.array_equal(a1, a2))
```

14. Consider a game “Give me thy name” with n players P1, P2, …, Pn. Each player has a dedicated variable ‘sum’ (n ‘sum’s for n players). The game goes like this: P2 takes the length of P1’s name and adds up to his/her existing sum (initially it is zero for all the players). Then this continues to P3 who adds the length of P2’s name to his/her sum and so on. This goes from left to right (P1 to Pn) and then right to left (where the summing is

done right to left; Pn-1 adds the length of the name of Pn to his/her current sum and passes the control to Pn-2 and so on) to constitute one pass. There can be ‘m’ such passes (left to right and then right to left). Each time a sum is updated for a player, the sum is checked to be prime or not. If the sum is prime, the corresponding player is eliminated. Then the process continues from the next player.

You should stop the game, the moment you are left with one player – who wins the game. If at the end of m passes, you have more than one player, the one with the highest sum wins. You should print the winner’s name.

You should not use another variable to store the sum; it should be stored with each player's information. You should have a list of lists, where each element is a list of three elements: player name, name length, and current sum. As an example,

Initial players:

[['Soumyadeep Sar', 12, 0], ['Anurag Sharma', 13, 0], ['Aakash Ghosh', 12, 0], ['Tisha Dash', 10, 0], ['Sirswa Kuldeep Shree Ram', 24, 0]]

Pass: 1
Left to right==>

[['Soumyadeep Sar', 12, 0], ['Anurag Sharma', 13, **12**], ['Aakash Ghosh', 12, 0], ['Tisha Dash', 10, 0], ['Sirswa Kuldeep Shree Ram', 24, 0]]

[['Soumyadeep Sar', 12, 0], ['Anurag Sharma', 13, 12], ['Aakash Ghosh', 12, **13**], ['Tisha Dash', 10, 0], ['Sirswa Kuldeep Shree Ram', 24, 0]]

Aakash Ghosh gets eliminated as ‘13’ is prime to produce the updated list as:

[['Soumyadeep Sar', 12, 0], ['Anurag Sharma', 13, 12], ['Tisha Dash', 10, 0], ['Sirswa Kuldeep Shree Ram', 24, 0]]

This continues and finally Anurag Sharma wins.

```
team = [ ["Kripa", 5, 0], ["Anurag", 6, 0], ["Aakash", 6, 0], ["Tisha", 5, 0], ["Richa", 5, 0]]

def isPrime(num):

    # define a flag variable
    flag = False
```

```
    if num <= 1:
        return False
    elif num > 1:
        # check for factors
        for i in range(2, num):
            if (num % i) == 0:
                # if factor is found, set flag to True
                flag = True
                # break out of loop
                break


    return not flag


def max_candidate(lst):
    max = 0
    max_name = ""

    for l in lst:
        if l[2] > max:
            max_name = l[0]
            max = l[2]

    return max_name

#print(isPrime(4))
print(team)
i = 0
is_single_flag = 0

for n in range(0, 5):
    print("Pass: %d" %(n+1))
    print("Left to right==>")

    i = 0
    length = len(team)
    while i < length:
        if i != 0:
            team[i][2] = team[i][2] + team[i-1][1]
            print(team)
        if isPrime(team[i][2]):
            print("Is Prime: ")
            print(team[i])
```

```
            #team[i][2] = 0
            team.remove(team[i])
            length = len(team)
            i = i - 1
            print(team)
        i = i + 1
        if length == 1:
            is_single_flag = 1
            break
        print("End: ")
        print(team)
    if(is_single_flag):
         break

    print("Right to left <==")


    length = len(team)
    i = length - 1

    while i >= 0:
        if i != length - 1:
            team[i][2] = team[i][2] + team[i+1][1]
            print(team)
        if isPrime(team[i][2]):
            print("Is Prime: ")
            print(team[i])
            #team[i][2] = 0
            team.remove(team[i])
            length = len(team)
            i = i + 1
            print(team)
        i = i - 1
        if length == 1:
            is_single_flag = 1
            break
        print("End: ")
        print(team)
    if(is_single_flag):
         break

if is_single_flag:
    print(team[0][0])
else:
```

```
print(max_candidate(team))
```

Set 5

1. Input (using input()) a list of alternating names and ages e.g. [‘kripa’, ‘37’, ‘arun’, ‘45’, ‘dipa’, ‘40’] and create a dictionary such the (i, i+1)th elements (i = 0, 2, …) form the key value pairs. For the mentioned example, the dictionary will be {‘kripa’: ‘37’, ‘arun’: ‘45’, ‘dipa’: ‘40’}

```
inpt = input("Give alternative names and ages, press enter to end: ").split(" ")
lst = list(inpt)
d = {}

for i in range(0, len(lst)-1, 2):
    d[lst[i]] = lst[i+1]
```

2. For the same input of (1), create a list of tuples of the (i, i+1)th elements (i = 0, 2, …) of the input list. For the mentioned example, the output list will be [('kripa', ‘37’), ('arun', ‘32’), ('dipa', ‘40)]

```
inpt = input("Give alternative names and ages, press enter to end: ").split(" ")
lst = list(inpt)
lst_out = []

for i in range(0, len(lst)-1, 2):
    lst_out.append((lst[i], lst[i+1]))
```

3. Use numpy linalg.solve to find the point of intersection of the three planes x + 2y + 3z = 2, 4x + 8y +66z = 3 and 7x + 81y + 9z = 4. Perform the said operation if the coefficient determinant is non-singular. This can be checked by the function np.linalg.det(). You may consider a determinant to be singular if np.linalg.det() is close to zero (say abs() value less than 0.00001).

```
A1 = np.array([[1, 2, 3], [4, 8, 66], [7, 81, 9]])
B1 = np.array([2, 3, 4])
```

```
dval = np.linalg.det(A1)

if(abs(dval) > 0.00001):
    print(np.linalg.solve(A1, B1))
else:
    print("Singular!")
```

4. Write a function d2b() that takes a decimal number as argument and returns its binary equivalent. Write a numpy ufunc function numpy_DecimalToBinary that uses d2b() and applies the same operation elementwise on a numpy array A. E.g. if A is 1, 2, 3, 4, 5, numpy_DecimalToBinary(A) will produce **1 10 11 100 101**. Note that all the elements of numpy_DecimalToBinary(A) should be of 'int' type and not 'str' type.

```
def d2b(x):

    dgts = ""
    while x != 0:
        digit = x%2
        dgts = str(digit) + "" + dgts
        x = x//2

    return int(dgts)

arr1 = np.array([1, 2, 3, 4, 5])
numpy_DecimalToBinary = np.frompyfunc(d2b, 1, 1)
arr2 = numpy_DecimalToBinary(arr1)
print(arr2)
```

5. Consider a numpy 2-D array of the form [[50, 60, 70], [67, 88, 90], [60, 78, 97]] where the ith 1-D array contains the marks of the ith student in three subjects (in this order. E.g. 50, 60, 70 are the subject1, subject2, subject3 marks respectively of student-0. Use numpy sum function only to i) create a 1-D numpy array with the sum of the marks of individual students ii) create a 1-D numpy array with the sum of subject-wise marks. Also, do these operations to produce 2-D numpy arrays with the same content.

6. Given two 1-D numpy arrays A and B, remove the elements in A which are also in B and store the resulting array in C. Use numpy set operations.

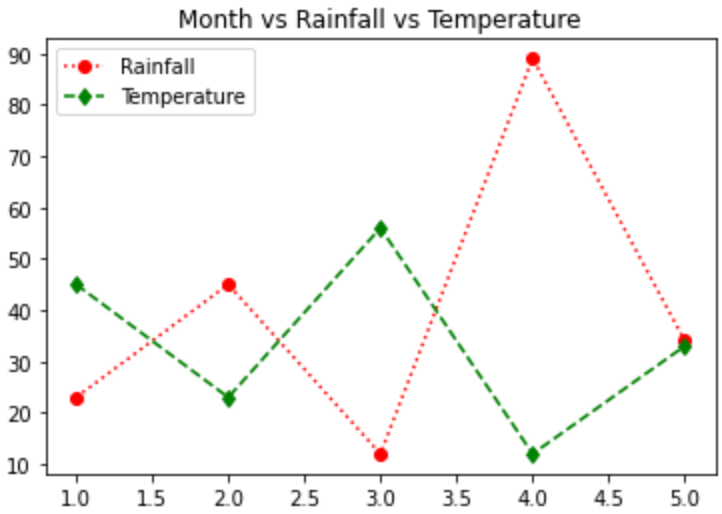
7. Given two numpy 2-D arrays arr1 = np.array([[1, 2], [4, 5]]), arr2 = np.array([[3, 3], [1,1]]) explore the difference between np.multiply(arr1, arr2) and np.matmul(arr1, arr2).

8. Take months (say 2, 3 etc.) and rainfall (in millimeters) as input from the user and plot months (x-axis) vs rainfall (y-axis) using plot in matplotlib.

9. Add another series for temperature (in degree celsius) and add it in the plot in (1) against months. Make sure you use different colors and markers for the two series (rainfall and temperature) against months.

10. Play around with line styles and colors of the two series. Also add title and legend.

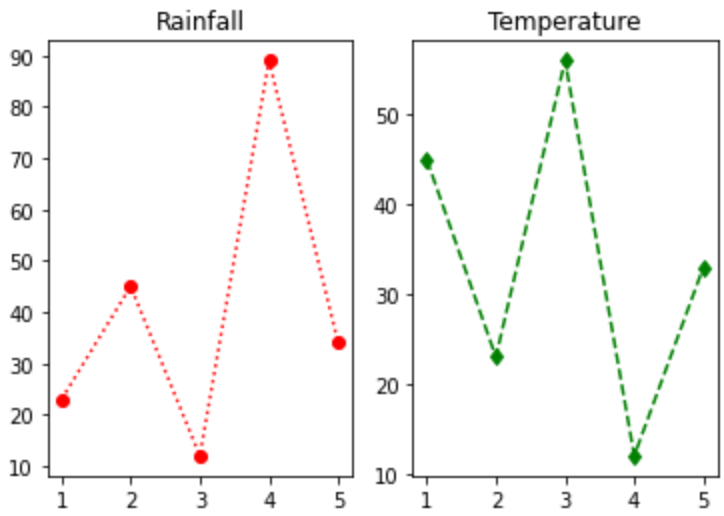
```
month = input("Give space-separated values: ").split(" ")
rain = input("Give space separated marks: ").split(" ")
temp = input("Give space separated marks: ").split(" ")

x = np.array(month)
y = np.array(rain)
z = np.array(temp)


import matplotlib.pyplot as pt

pt.title("Month vs Rainfall vs Temperature")

pt.plot(x, y, 'o:r', label="Rainfall")
pt.plot(x, z, 'd--g', label = "Temperature")

pt.legend()
```



11. Show the above two plots as subplots i) along the same row and ii) along the same column. Put the title on each subplot.

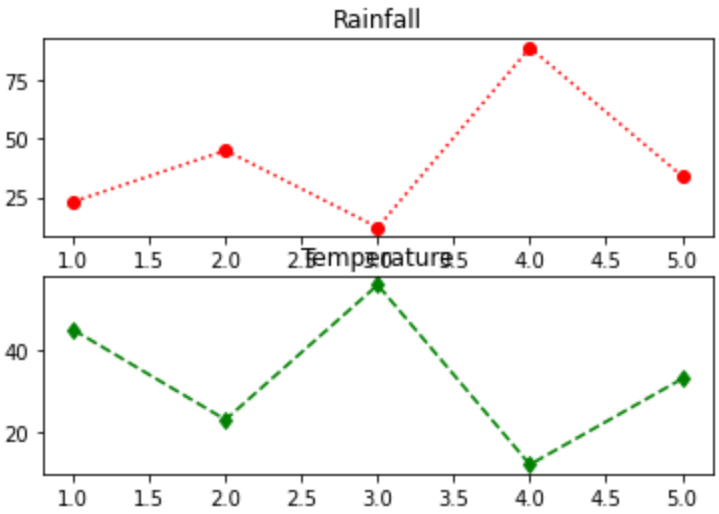
i)

```
month = input("Give space-separated values: ").split(" ")
rain = input("Give space separated marks: ").split(" ")
temp = input("Give space separated marks: ").split(" ")

x = np.array(month)
y = np.array(rain)
z = np.array(temp)


import matplotlib.pyplot as pt

pt.subplot(1, 2, 1)
pt.title("Rainfall")
pt.plot(x, y, 'o:r', label="Rainfall")

pt.subplot(1, 2, 2)
pt.title("Temperature")
pt.plot(x, z, 'd--g', label = "Temperature")
```



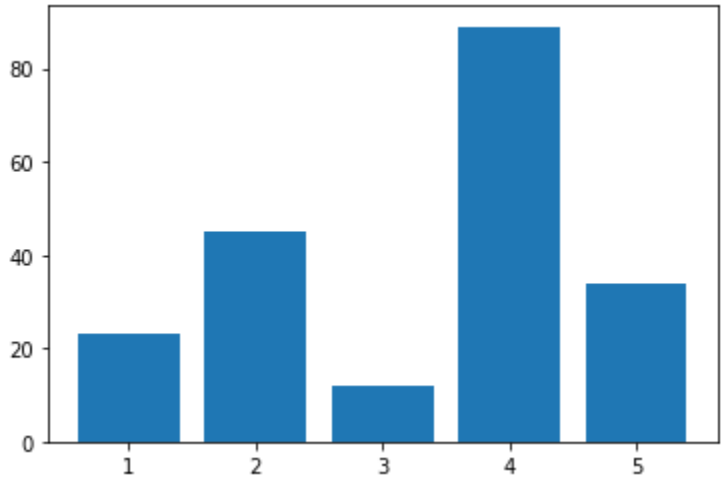
ii)

```
month = input("Give space-separated values: ").split(" ")
rain = input("Give space separated marks: ").split(" ")
temp = input("Give space separated marks: ").split(" ")

x = np.array(month)
y = np.array(rain)
z = np.array(temp)


import matplotlib.pyplot as pt

pt.subplot(2, 1, 1)
pt.title("Rainfall")
pt.plot(x, y, 'o:r', label="Rainfall")

pt.subplot(2, 1, 2)
pt.title("Temperature")
pt.plot(x, z, 'd--g', label = "Temperature")
```



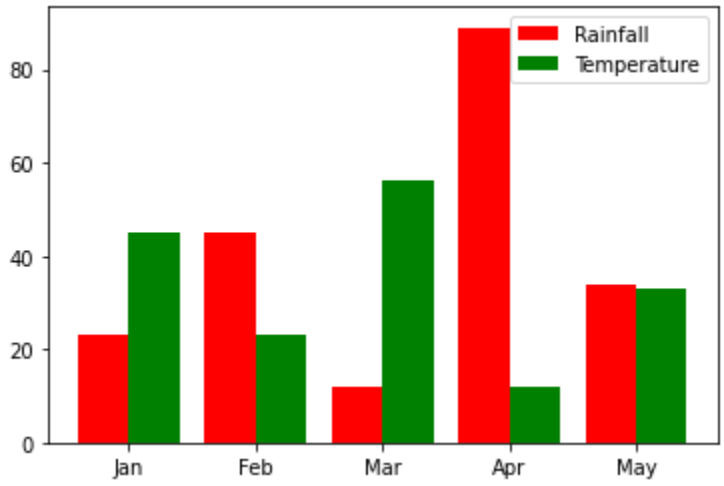
12. Use the data of Q8 to draw a bar chart.

```
x = np.array(month)
y = np.array(rain)

pt.bar(x, y)
```



13. Use the same data to draw a bar graph showing the rainfall and temperature for each month.

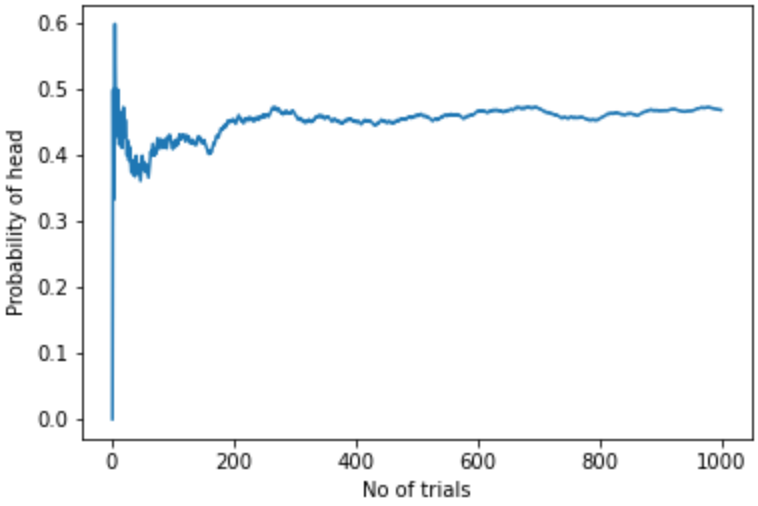
```
x = np.array(month)
y = np.array(rain)
z = np.array(temp)

X = ["Jan", "Feb", "Mar", "Apr", "May"]

pt.bar(x-0.2, y, 0.4, label = "Rainfall", color = "red")
pt.bar(x+0.2, z, 0.4, label = "Temperature", color = "green")
pt.xticks(x, X)
pt.legend()
```



14. Use the same data to draw a pie chart.

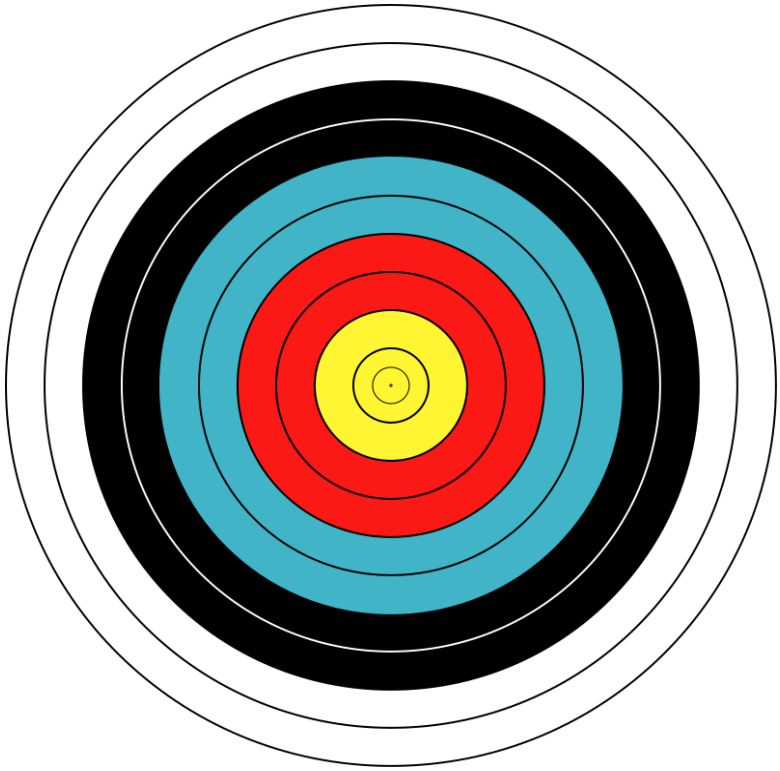
15. Consider a trial of tossing an unbiased coin n (a large value) times. Use random.random() [generates a random float number between 0 and 1] to simulate the observation after each coin toss as follows: if the random number is <0.5, assume that H has occurred, tail otherwise. Count the number of heads and compute the fraction of heads generated after each coin toss. This is basically to estimate the probability of head (that is 0.5). Plot the estimated probability (using matplotlib.pyplot.plot) along the y-axis, while the trial numbers are plotted along the x-axis.

```
import random
head = 0
x = []
y = []
n = 1000

for i in range(1, n+1):
    x.append(i)
    r = random.random()
    if r < 0.5:
        print("head")
        head += 1

    y.append(head/i)


print(x)
print(y)

import matplotlib.pyplot as plt
plt.xlabel("No of trials")
plt.ylabel("Probability of head")
plt.plot(x, y)
#plt.show()
plt.savefig("head.png")
```



Set 6

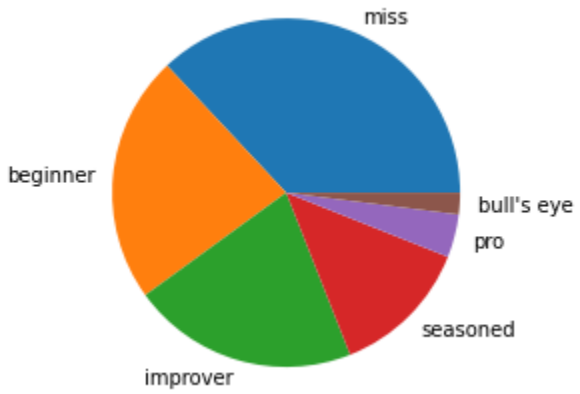
1. Consider an archery competition where a target (see figure below)



is shot at by the participants. Consider that the maximum points are obtained by hitting the “bull’s eye” (the centre spot in the yellow zone), where any hit on the white zone (or outside) fetches no points. So, each attempt (in the increasing order of success) can be termed as 'miss', 'beginner', 'improver', 'seasoned', 'pro' or 'bull’s eye' according as the participant hits the white (or outside), black, blue, red, yellow or the bull’s eye respectively. Simulate the competition by a normal distribution (use np.random.normal()) such that the probability of hitting the white zone is the maximum (occurrence near the mean) and that of hitting the bull’s eye is the minimum (maximum distance away from the mean). Finally, plot the distribution of performances after n (large value) attempts as 'miss', 'beginner', 'improver', 'seasoned', 'pro', 'bull's eye' on a pie chart.

```
import numpy as np


s = abs(np.random.normal(0, 0.5, 100)) #random sample of size 100 generated from a normal distribution of mean 0 and s.d. 0.5
print(s)
max1 = np.max(s)
min1 = np.min(s)

s1 = [(x-min1)/(max1-min1) for x in s]
print(s1)
sc = np.zeros(6)

for x in s1:
    if x < 0.2:
        sc[0] = sc[0] + 1
    elif x < 0.4:
        sc[1] = sc[1] + 1
    elif x < 0.6:
        sc[2] = sc[2] + 1
    elif x < 0.8:
        sc[3] = sc[3] + 1
    elif x < 0.95:
        sc[4] = sc[4] + 1
    else:
        sc[5] = sc[5] + 1

print(sc)

import matplotlib.pyplot as plt
```

```
plt.pie(sc, labels = ['miss', 'beginner', 'improver', 'seasoned', 'pro', 'bulls eye']) #Note that ‘miss’ is modelled on an event with the highest probability, with x-value close to the mean (0); similarly ‘bull’s eye’ is modelled on the rare events, i.e. with low probabilities

plt.show()
```



2. Take a sentence (e.g. "Information Retrieval is the science of search engines") as input. Consider a list of stopwords (unimportant words) stop = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"]. Write a function **preprocess** that converts the input string to lowercase and removes all the stopwords. Finally print the preprocessed input string on the terminal. The output for the example input will be “information retrieval science search engines”.

```
def preprocess(s):
                    s = s.lower()
                    s = s.split(" ")
                    stop = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you",
                "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she",
```

```
"her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs",
"themselves", "what", "which", "who", "whom", "this", "that", "these",
"those", "am", "is", "are", "was", "were", "be", "been", "being", "have",
"has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and",
"but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with",
"about", "against", "between", "into", "through", "during", "before", "after",
"above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over",
"under", "again", "further", "then", "once", "here", "there", "when", "where",
"why", "how", "all", "any", "both", "each", "few", "more", "most", "other",
"some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too",
"very", "s", "t", "can", "will", "just", "don", "should", "now"]
    s_out = []
    for w in s:
        if w not in stop:
            s_out.append(w)
    return s_out

dd=input("Give a text: ")
print(' '.join(preprocess(dd)))
```

3. Consider five documents

   doc1 : "Information Retrieval is the science of search engines",
   doc2 : "This is the age of information technology",
   doc3 : "Mathematics in the language of science",
   doc4 : "Efficient retrieval of important data is the feature of any sound search system.",
   doc5 : "Gerard Salton is the father of Information Retrieval"

   Use the **preprocess** function of Q2 to lowercase and remove stopwords from these documents.
   Now take a query string q as input from the user and preprocess it using the preprocess function.
   Now, print the documents that contain
   (i) at least one word in q and
   (ii) all the words in q

   For example, if the query q is “Information Retrieval” for (i) doc1, doc2, doc4 and doc5 will be printed while for (ii) doc1 and doc5 will be printed.

```
def preprocess(s):
    s = s.lower()
    s = s.split(" ")
```

```
    stop = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your",
"yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers",
"herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what",
"which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was",
"were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did",
"doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of",
"at", "by", "for", "with", "about", "against", "between", "into", "through", "during",
"before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off",
"over", "under", "again", "further", "then", "once", "here", "there", "when", "where",
"why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some",
"such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t",
"can", "will", "just", "don", "should", "now"]
    s_out = []
    for w in s:
        if w not in stop:
            s_out.append(w)
    return s_out

def BAND(coll, q):
    length = len(q)

    for key in coll:
        match = 0
        for w in q:
            #print("w: "+w)
            if w in coll[key]:
                match = match + 1
        #print(str(match))
        if match == length:
            print(key)

def BOR(coll, q):
    length = len(q)

    for key in coll:
        match = 0
        for w in q:
            #print("w: "+w)
            if w in coll[key]:
                match = match + 1
        #print(str(match))
        if match > 0:
            print(key)
```

```
coll = {"doc1" : "Information Retrieval is the science of search engines",
        "doc2" : "This is the age of information technology",
        "doc3" : "Mathematics in the language of science",
        "doc4" : "Efficient retrieval of important data is the feature of any sound
search system.",
        "doc5" : "Gerard Salton is the father of Information Retrieval"}

for key in coll:
    coll[key] = preprocess(coll[key])
    print(coll[key])

q = "information retrieval".split(" ")

print(q)

BAND(coll, q)
#BOR(coll, q)
```

4. In the setup of Q3, print the sorted list of documents for a given query in the decreasing order of the degree of match (number of words matched between query and document). That is a document with more number of word matches with the query will the ranked higher. Ties are broken arbitrarily. For example, if the query is “Information Retrieval”, the list should be:

   doc1
   doc5
   doc2
   doc4
   doc3

   Note that sorted_dict = dict (sorted(rl.items(), key = lambda item: item[1], reverse=True)) sorts the dictionary “rl” based on the value in the reverse order and stores it in sorted_dict.

```
def preprocess(s):
        s = s.lower()
```

```
    s = s.split(" ")
    stop = ["i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your",
"yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers",
"herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what",
"which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was",
"were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did",
"doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of",
"at", "by", "for", "with", "about", "against", "between", "into", "through", "during",
"before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off",
"over", "under", "again", "further", "then", "once", "here", "there", "when", "where",
"why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some",
"such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t",
"can", "will", "just", "don", "should", "now"]
    s_out = []
    for w in s:
        if w not in stop:
            s_out.append(w)
    return s_out

def BAND(coll, q):
    length = len(q)

    for key in coll:
        match = 0
        for w in q:
            #print("w: "+w)
            if w in coll[key]:
                match = match + 1
        #print(str(match))
        if match == length:
            print(key)

def BOR(coll, q):
    length = len(q)

    for key in coll:
        match = 0
        for w in q:
            #print("w: "+w)
            if w in coll[key]:
                match = match + 1
        #print(str(match))
        if match > 0:
            print(key)
```

```
def RankedRet(coll, q):
    length = len(q)
    ranked_list = {}

    for key in coll:
        match = 0
        for w in q:
            #print("w: "+w)
            if w in coll[key]:
                match = match + 1
        #print(str(match))
        ranked_list[key] = match

    return ranked_list

coll = {"doc1" : "Information Retrieval is the science of search engines",
        "doc2" : "This is the age of information technology",
        "doc3" : "Mathematics in the language of science",
        "doc4" : "Efficient retrieval of important data is the feature of any sound
search system.",
        "doc5" : "Gerard Salton is the father of Information Retrieval"}

for key in coll:
    coll[key] = preprocess(coll[key])
    print(coll[key])

q = "information retrieval".split(" ")

print(q)

#BAND(coll, q)
#BOR(coll, q)

rl = RankedRet(coll, q)
sorted_dict = dict ( sorted(rl.items(), key = lambda item: item[1], reverse=True) )

for k in sorted_dict:
    print(k)
```

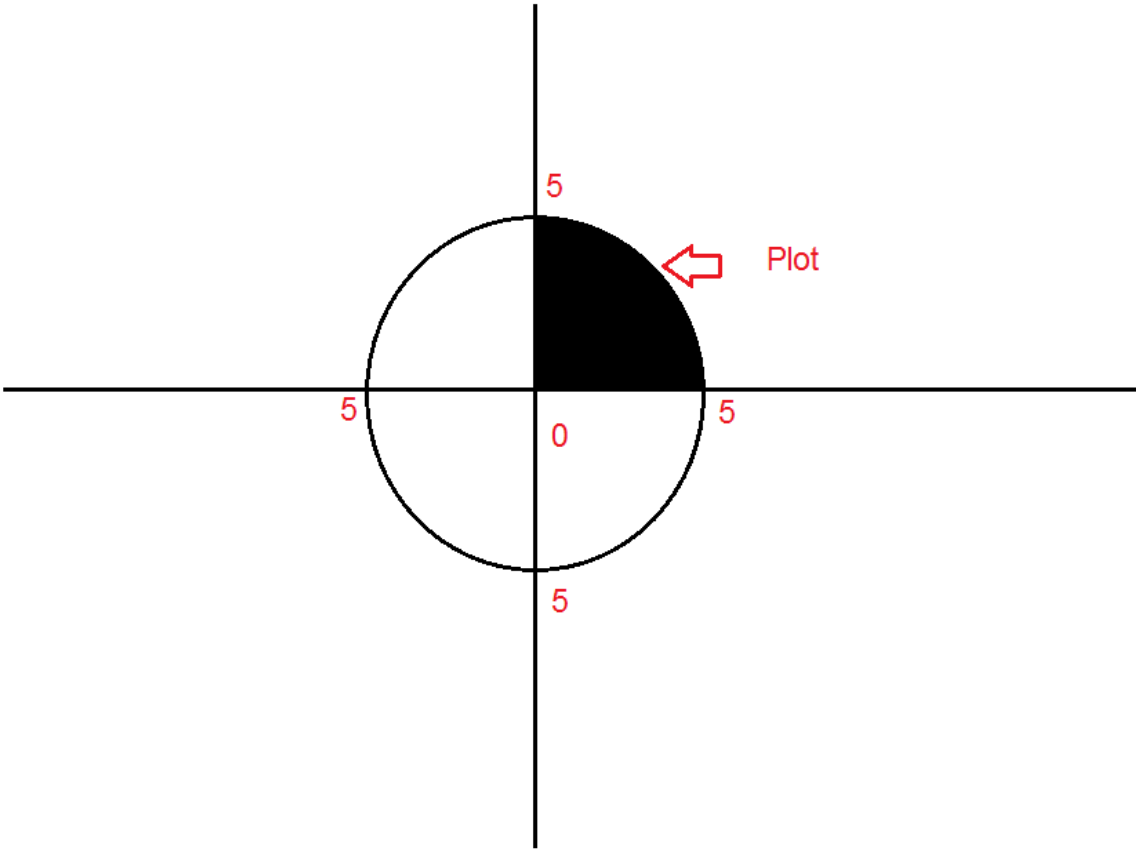
5. Use Rectangular method, Trapezoidal rule and Simpson's rule applying the original formulas to find the approximate integral of the function 1/x in the interval [1, 2]. Compare the approximate values with other actual values of the integral.

   Repeat the above using scipy.integrate for Trapezoidal and Simpson's rules.

```
import numpy as np
N = 10
xs, h = np.linspace(1, 2, N, endpoint=False, retstep=True)

#print(xs)
#print(h)

ys = 1.0/xs
#print(ys)
#Rectangular
Irect = h*np.sum(ys)

print("Rectangular=%.5f" %(Irect))

import scipy.integrate as spi

#Trapezoidal rule

#print("Actual=%f" %(1/11.))
xs, h = np.linspace(1, 2, N, endpoint=True, retstep=True)
ys = 1.0/xs
#print(ys)
Itrap = h*(0.5*(ys[0] + ys[-1]) + np.sum(ys[1:-1]))
print("Trapezoidal=%.5f" %(Itrap))

Itrap1 = spi.trapz(ys, xs)
print("Trapezoidal scipy=%f" %(Itrap1))

#from scipy.integrate import simps

#Simpson's rule

xs, h = np.linspace(1, 2, N+1, endpoint=True, retstep=True)
ys = 1.0/xs
print(ys)
print(xs)

Isimp = (h/3.)*(ys[0] + ys[-1] + 4*np.sum(ys[1:-1:2]) +
2*np.sum(ys[2:-1:2]))
```

```
print("Simp=%.5f" %(Isimp))

import math

actual = math.log(2)

print("Actual=%f" %(actual))

print("Error simp = %.5f" %(abs(Isimp-actual)))
print("Error trap = %.5f" %(abs(Itrap-actual)))



Isimp1 = spi.simps(ys, xs)
print("Simp scipy = %f" %(Isimp1))
```

6. Given the plot of land in the figure below, use the formula of Rectangular rule to approximate the area (shown in black). Do the same using both the original formulas and scipy implementations of Trapezoidal rule and Simpson's rule. Compute the actual area (hint: use the formula of a circle) and compare the approximate values for this area by printing the percentage accuracy (abs(approx-actual)/actual)*100) for all the above approximate values.



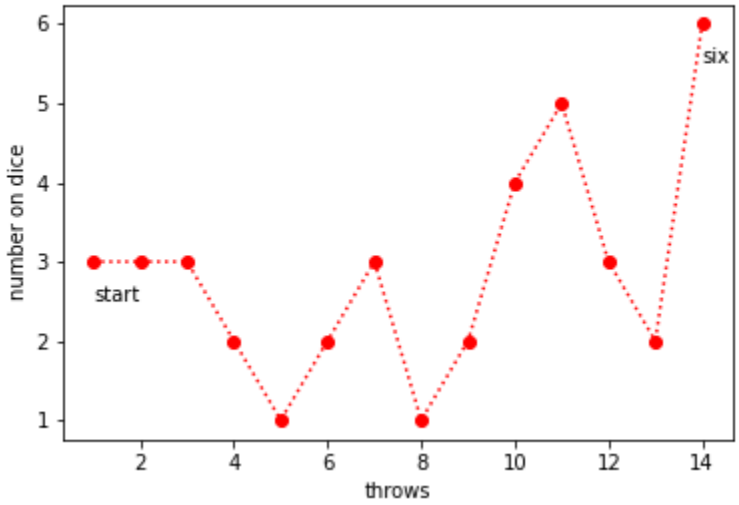

```
scipy=%f

import numpy as np
N = 10
xs, h = np.linspace(0, 5, N, endpoint=False, retstep=True)

#print(xs)
#print(h)

ys = np.sqrt(25 - xs*xs)
print(ys)
#Rectangular
Irect = h*np.sum(ys)

print("Rectangular=%.5f" %(Irect))

import scipy.integrate as spi

#Trapezoidal rule
```

```
#print("Actual=%f" %(1/11.))
xs, h = np.linspace(0, 5, N, endpoint=True, retstep=True)
ys = np.sqrt(25 - xs*xs)
#print(ys)
Itrap = h*(0.5*(ys[0] + ys[-1]) + np.sum(ys[1:-1]))
print("Trapezoidal=%.5f" %(Itrap))

Itrap1 = spi.trapz(ys, xs)
print("Trapezoidal " %(Itrap1))

#from scipy.integrate import simps

#Simpson's rule

xs, h = np.linspace(0, 5, N+1, endpoint=True, retstep=True)
ys = np.sqrt(25 - xs*xs)
print(ys)
print(xs)

Isimp = (h/3.)*(ys[0] + ys[-1] + 4*np.sum(ys[1:-1:2]) +
2*np.sum(ys[2:-1:2]))
print("Simp=%.5f" %(Isimp))

import math

actual = math.pi*5*5*0.25

print("Actual=%f" %(actual))




Isimp1 = spi.simps(ys, xs)
print("Simp scipy = %f" %(Isimp1))

print("Error rect = %.5f" %((abs(Irect-actual)/actual)*100))
print("Error trap original = %.5f" %((abs(Itrap-actual)/actual)*100))
print("Error trap scipy = %.5f" %((abs(Itrap1-actual)/100)*100))

print("Error simp original = %.5f" %((abs(Isimp-actual)/actual)*100))
print("Error trap scipy = %.5f" %((abs(Isimp1-actual)/actual)*100))
```

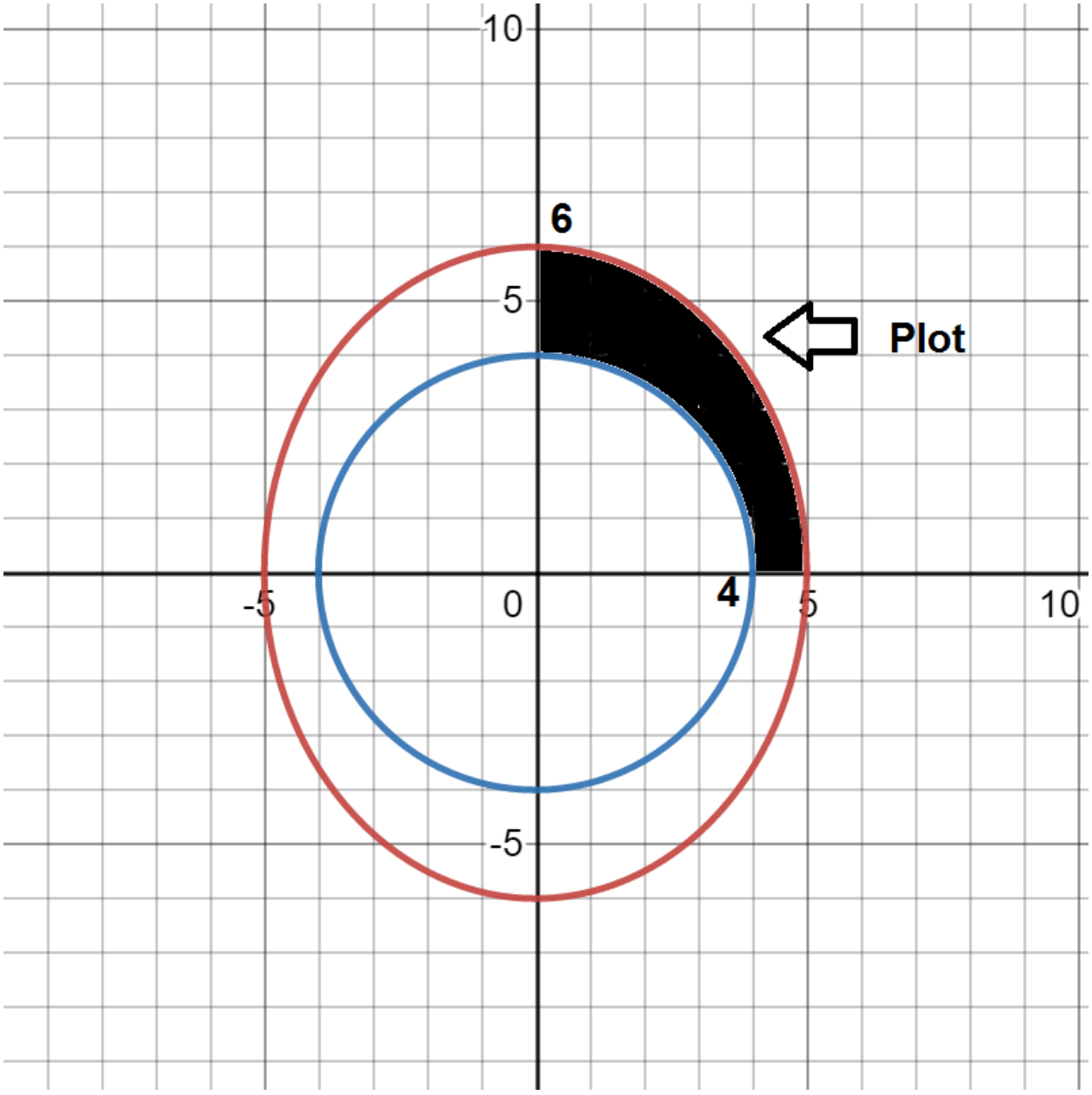
7. Simulate a dice game for a single player named Shakuni. The player throws the dice until he gets a six, when he stops. Use random.uniform(a, b) [generates uniformly distributed random number between a & b] to simulate the outcome using the scheme: random number < 1 ⇒ side 'one' appears, random number >=1 and <2 ⇒ side 'two' appears and so on. Plot the outcomes (along y-axis, with trial number along x-axis) using matplotlib.pyplot.plot. Also, indicate the first and last throws as "start" and "stop" respectively on the plot using matplotlib.pyplot.text(x, y, t) [prints t on the plot at (x,y)].

```
import random

x = []
y = []

for i in range(1, 1000):
    x.append(i)
    r = random.uniform(0, 6)
    if r < 1:
        print("One")
        y.append(1)
    elif r < 2:
        print("Two")
        y.append(2)
    elif r < 3:
        print("Three")
        y.append(3)
    elif r < 4:
        print("Four")
        y.append(4)
    elif r < 5:
        print("Five")
        y.append(5)
    else:
        print("Six")
        y.append(6)
        break


print(x)
print(y)
```

```
import matplotlib.pyplot as plt
plt.text(x[0], y[0]-0.5, "start") #prints "start" at coordinates (x[0], y[0]-0.5)
plt.text(x[-1], y[-1]-0.5, "six") #print "six" at the last throw
plt.plot(x, y, 'o:r')
plt.xlabel("throws")
plt.ylabel("number on dice")
#plt.show()
plt.savefig("dice.png")
```



8. Given the plot of land in the figure below, use the formula of Rectangular rule to approximate the area (shown in black). Do the same using both the original formulas and scipy implementations of Trapezoidal rule and Simpson's rule. Compute the actual area (hint: area between an ellipse and a circle) and compare the approximate values for this area by printing the percentage accuracy (abs(approx-actual)/actual)*100) for all the above approximate values.




```
import numpy as np
N = 10
#circle
xs, h = np.linspace(0, 4, N, endpoint=False, retstep=True)

#print(xs)
#print(h)
```

```
ys = np.sqrt(16 - xs*xs)
#print(ys)
#Rectangular
Irect_c = h*np.sum(ys)

#ellipse
xs, h = np.linspace(0, 5, N, endpoint=False, retstep=True)

#print(xs)
#print(h)

ys = 6*np.sqrt(1 - xs*xs/25)
#print(ys)
#Rectangular
Irect_e = h*np.sum(ys)

print("Rectangular=%.5f" %(Irect_e-Irect_c))

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

import scipy.integrate as spi

#Trapezoidal rule

#print("Actual=%f" %(1/11.))
#circle
xs, h = np.linspace(0, 4, N, endpoint=True, retstep=True)
ys = np.sqrt(16 - xs*xs)
#print(ys)
Itrap_c = h*(0.5*(ys[0] + ys[-1]) + np.sum(ys[1:-1]))
#print("Trapezoidal=%.5f" %(Itrap))

Itrap1_c = spi.trapz(ys, xs)
#ellipse
xs, h = np.linspace(0, 5, N, endpoint=True, retstep=True)
ys = 6*np.sqrt(1 - xs*xs/25)
#print(ys)
Itrap_e = h*(0.5*(ys[0] + ys[-1]) + np.sum(ys[1:-1]))
#print("Trapezoidal=%.5f" %(Itrap))

Itrap1_e = spi.trapz(ys, xs)

print("Trapezoidal ori=%f" %(Itrap_e - Itrap_c))
print("Trapezoidal scipy=%f" %(Itrap1_e - Itrap1_c))
```

Set 7

1. For the overall betterment of institute performance at the national level, the Football Club of IISER Kolkata has decided to apply data science to the player performance statistics. The first application is to segregate the strikers based on their goal-scoring performance and expose them to customized training. To be more specific, the strikers with similar goal-scoring performance in the similar number of matches, should be trained together.

   To this end, consider a pool of n (>=10) strikers with (num_of_matches, number_of_goals_scores) for each player and apply k-means clustering algorithm (implemented in sklearn) to meaningfully cluster them. Use the elbow method to determine the best choice for k and finally plot the clusters with the chosen value of k. You should plot the data before and after clustering as well as the elbow plot.

```
import matplotlib.pyplot as plt

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [10, 2, 14, 4, 9, 5, 4, 10, 10, 2]

plt.scatter(x, y)
plt.show()

from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)
```

```
plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

kmeans = KMeans(n_clusters=4)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```

2. The course instructor decides to apply a binary classification scheme to classify students into two classes: “developed” (who are already expert coders) and “developing” (who need separate doubt-clearing sessions) based on their CS1101 total percentage marks and CS2201 (so far) total percentage marks. Assume that some ‘n’ students are manually assigned classes 1 (developing class) and 0 (developed class). The CS1101 percentage marks and CS2201 percentage marks respectively of these n students are stored in arrays x and y (same order), with class labels (1 or 0) stored (in the same order as in x and y) in another array “labels”.

   Now, given a new student (whose class label is unknown) and his/her marks in CS1101 and CS2201, implement (from scratch, without using any module like k-means in question 1) k-nearest neighbour classification algorithm to classify the student into one of the two classes. That is, find the k-nearest neighbours of the new student from the information stored in arrays x and y and assign the new student the majority class label of the k nearest neighbours given by the array “labels”. Please choose the value of k judiciously.

   Finally, plot the the labelled points as well as the new point on a scatter-plot (in three different colours). Note, that the code for plotting a single point (a,b) is:
   plt.scatter([a], [b], color="magenta")

```
import matplotlib.pyplot as plt

import math

x = [100, 90, 95, 89, 55, 70, 65, 70] #CS1101 marks
y = [100, 90, 80, 80, 65, 70, 70, 65] #CS2201 marks

labels = [0, 0, 0, 0, 1, 1, 1, 1] #labels
```

```
plt.scatter(x, y, c=labels)

def distance(x, y, point):
    return math.sqrt((x-point[0])**2 + (y-point[1])**2)


def majority(labels):

    label_freq = {}
    for lb in labels:
        if lb not in label_freq:
            label_freq[lb] = 1
        else:
            label_freq[lb] = label_freq[lb] + 1

    max_freq = -1
    max_freq_label = -1

    for lb in label_freq:
        if label_freq[lb] > max_freq:
            max_freq = label_freq[lb]
            max_freq_label = lb

    print(label_freq)
    print("max label = %d max label freq = %d" %(max_freq,
max_freq_label))

    return max_freq_label

def kNN(x, y, labels, k, point):
    x_temp = x
    y_temp = y

    l = len(x_temp)

    knn_labels = []

    for j in range(1, k+1):
        min_distance = 1000
        x_coord_min = -1
        y_coord_min = -1
        min_label = -1
```

```
        for i in range(0, l):
            if x_temp[i] != -1 and distance(x_temp[i], y_temp[i], point) <
min_distance:
                min_distance = distance(x_temp[i], y_temp[i], point)
                x_coord_min = x_temp[i]
                y_coord_min = y_temp[i]
                min_label = i
                x_temp[i] = -1 #considered

        print("%d %d" %(x_coord_min, y_coord_min))
        knn_labels.append(labels[min_label])
    print(knn_labels)
    knn_class = majority(knn_labels)
    return knn_class


point = (85, 90)

class_label = kNN(x, y, labels, 3, point)

if class_label == 1:
    print("No worries, the instructor and the TAs will help you!")
else:
    print("Great to know you are doing well!")


plt.scatter([point[0]], [point[1]], color="magenta")
```

3. In question 2, the manually labelled data is termed as the “training set” in machine learning parlance. Now consider a “test set” containing new student scores but also with the correct class labels. A format for a test data point can be (50, 60, 1) where 50 is the CS1101 score, 60 is the CS2201 score and 1 is the true label i.e. “developing”. Assume that we have a test set with m (>= 3) test data points (each point containing two scores and the true label). Run k-nearest neighbour classifier (as in question 2) on this test set and compute the True Positives, True Negatives, False Positives, False Negatives, Precision, Recall and F-score for the test set. Note that you have to find the class label (called the “predicted labels”) for each test set point (e.g. (50, 60)) using the k-nearest neighbour algorithm and tally the class label produced by this algorithm with the true test label for the same test set point.

```
import matplotlib.pyplot as plt
```

```
import math

x = [100, 90, 95, 89, 55, 70, 65, 70]
y = [100, 90, 80, 80, 65, 70, 70, 65]

labels = [0, 0, 0, 0, 1, 1, 1, 1]


#plt.scatter(x, y, c=labels)

def distance(x, y, point):
    return math.sqrt((x-point[0])**2 + (y-point[1])**2)


def majority(labels):

    label_freq = {}
    for lb in labels:
        if lb not in label_freq:
            label_freq[lb] = 1
        else:
            label_freq[lb] = label_freq[lb] + 1

    max_freq = -1
    max_freq_label = -1

    for lb in label_freq:
        if label_freq[lb] > max_freq:
            max_freq = label_freq[lb]
            max_freq_label = lb

    print(label_freq)
    print("max label = %d max label freq = %d" %(max_freq,
max_freq_label))

    return max_freq_label

def kNN(x, y, labels, k, point):
    x_temp = x
    y_temp = y

    l = len(x_temp)

    knn_labels = []
```

```
    for j in range(1, k+1):
        min_distance = 1000
        x_coord_min = -1
        y_coord_min = -1
        min_label = -1
        indices_covered = []

        for i in range(0, l):
            if i not in indices_covered and distance(x_temp[i], y_temp[i], point)
< min_distance:
                min_distance = distance(x_temp[i], y_temp[i], point)
                x_coord_min = x_temp[i]
                y_coord_min = y_temp[i]
                min_label = i
                indices_covered.append(i) #considered

        print("%d %d" %(x_coord_min, y_coord_min))
        knn_labels.append(labels[min_label])
    print(knn_labels)
    knn_class = majority(knn_labels)
    return knn_class


test_set = [(90, 90, 0), (50, 60, 1), (85, 80, 1), (20, 20, 0)]

TP = 0
TN = 0
FP = 0
FN = 0

for t in test_set: #1 -- positive class
    point = (t[0], t[1])

    class_label = kNN(x, y, labels, 3, point)

    print(class_label)

    if t[2] == 1 and class_label == 1:
        TP = TP + 1
    elif t[2] == 0 and class_label == 0:
        TN = TN + 1
    elif t[2] == 1 and class_label == 0:
        FN = FN + 1
```

```
    else:
        FP = FP + 1


print("TP = %d" %TP)
print("FP = %d" %FP)
print("TN = %d" %TN)
print("FN = %d" %FN)


Accuracy =  (TP + TN)/len(test_set)
Precision = TP/(TP+FP)
Recall = TP/(TP+FN)

Fscore = 2*((Recall*Precision)/(Recall+Precision))

print("Accuracy = %f" %(Accuracy))
print("Precision = %f" %(Recall))
print("Recall = %f" %(Precision))
print("F-score = %f" %Fscore)
```

4. Consider the following table showing the number of mangoes produced in a place in West Bengal in different years.

Year	2000	2005	2010	2015	2020
Mango production (‘000 MT)	600	650	665	600	585

Using Newton’s forward Interpolation, estimate the mango growth in the year 2001.

```
# Python3 Program to interpolate using
# newton forward interpolation


# calculating u mentioned in the formula
def u_cal(u, n):

        temp = u;
```

```
        for i in range(1, n):
                temp = temp * (u - i);
        return temp;

# calculating factorial of given number n
def fact(n):
        f = 1;
        for i in range(2, n + 1):
                f *= i;
        return f;

# Number of values given
n = 5;
x = [ 2000, 2005, 2010, 2015, 2020 ];

# y[][] is used for difference table
# with y[][0] used for input
y = [[0 for i in range(n)]
                for j in range(n)];

import math


# Displaying the forward difference table

y[0][0] = 600;
y[1][0] = 650;
y[2][0] = 665;
y[3][0] = 600;
y[4][0] = 585;



# Calculating the forward difference
# table

for i in range(1, n):
        for j in range(n - i):
                y[j][i] = y[j + 1][i - 1] - y[j][i - 1];


# Displaying the forward difference table
for i in range(n):
        print(x[i], end = "\t");
```

```
        for j in range(n - i):
                print(y[i][j], end = "\t");
        print("");


# Value to interpolate at
value = 2001;

# initializing u and sum
yout = []
sum = y[0][0];
yout.append(sum)
u = (value - x[0]) / (x[1] - x[0]);
for i in range(1,n):
        sum = sum + (u_cal(u, i) * y[0][i]) / fact(i);

print("\nValue at", value,
        "is", round(sum, 6));
```

5. India is one of the largest producers of dairy milk in the world. The following table shows milk production in India in million tonnes for five years:

Year	2015	2017	2019	2021	2023
Production (Million Tonnes)	146.3	165.4	187.7	210.0	230.6

Use an appropriate Interpolation scheme to calculate the production in 2016 (the actual value is 155.5, shown for 2015-16 here https://www.nddb.coop/information/stats/milkprodindia).

```
# calculating u mentioned in the formula
def u_cal(u, n):

        temp = u;
        for i in range(1, n):
                temp = temp * (u - i);
        return temp;

# calculating factorial of given number n
```

```
def fact(n):
	f = 1;
	for i in range(2, n + 1):
		f *= i;
	return f;

# Driver Code

# Number of values given
n = 5;
x = [ 2015, 2017, 2019, 2021, 2023 ];

# y[][] is used for difference table
# with y[][0] used for input
y = [[0 for i in range(n)]
		for j in range(n)];

import math

"""
i = 0
for v in x:
    y[i][0] = math.sin(math.radians(v))
    i = i + 1

"""
# Displaying the forward difference table

y[0][0] = 146.3;
y[1][0] = 165.4;
y[2][0] = 187.7;
y[3][0] = 210.0;
y[4][0] = 230.6;

"""
for i in range(n):
	print(x[i], end = "\t");
	for j in range(n - i):
		print(y[i][j], end = "\t");
	print("");

"""
```

```
# Calculating the forward difference
# table

for i in range(1, n):
        for j in range(n - i):
                y[j][i] = y[j + 1][i - 1] - y[j][i - 1];


# Displaying the forward difference table
for i in range(n):
        print(x[i], end = "\t");
        for j in range(n - i):
                print(y[i][j], end = "\t");
        print("");


# Value to interpolate at
value = 2016;

# initializing u and sum
yout = []
sum = y[0][0];
yout.append(sum)
u = (value - x[0]) / (x[1] - x[0]);
for i in range(1,n):
        sum = sum + (u_cal(u, i) * y[0][i]) / fact(i);

print("\nValue at", value,
        "is", round(sum, 6));
```

6. The following table shows the number of poaching cases in a forest at different days starting from a start point in time:

Day	5	15	30	42	55
No. of cases	250	200	150	102	75

Use Lagrange's interpolation formula to estimate the number of cases on day 50.

```
n = 5;
```

```
x = [ 5, 15, 30, 42, 55 ];

y = [250, 200, 150, 102, 75]



value = 50

result = 0.0
for i in range(n):

        # Compute individual terms of above formula
        term = y[i]
        for j in range(n):
                if j != i:
                        term = term * (value - x[j]) / (x[i] - x[j])

        # Add current term to result
        result += term

print("\nValue at", value,
        "is", round(result, 6))
```

Miscellaneous Practice problems

1. Write a Python program to find uncommon words (as a list) from two strings. E.g. if the two strings are “I hate coding” and “I love coding”, the output should be [“hate”, “love”].

```
# Function to return all uncommon words
def UncommonWords(A, B):
        A=A.split()
        B=B.split()
        x=[]
        for i in A:
                if i not in B:
                        x.append(i)
        for i in B:
                if i not in A:
                        x.append(i)
        x=list(set(x))
```

```
        return x


A = "I hate coding"
B = "I love coding"

# Print required answer
print(UncommonWords(A, B))
```

2. Write a Python program to find the first non-repeated element in a list. E.g. if the list is [1, 2, **3**, 1, 2, 4, 5, 6, 7, 8], the first non-repeated element is 3.

```
# Define a function to find the first non-repeated element in a list
def first_non_repeated_el(lst):
    # Create a dictionary 'ctr' to count the occurrences of each element
    ctr = {}
    # Iterate through the elements in the list
    for i in lst:
        # If the element is already in the dictionary, increment its count
        if i in ctr:
            ctr[i] += 1
        # If the element is not in the dictionary, add it with a count of 1
        else:
            ctr[i] = 1
    # Iterate through the elements in the list again
    for i in lst:
        # Find the first element with a count of 1 (non-repeated)
        if ctr[i] == 1:
            return i
    # If no non-repeated element is found, return None

# Create a list of numbers
nums =  [1, 2, 3, 4, 5, 6, 7, 8]

# Print the original list of numbers
print("Original list:")
print(nums)
```

```
# Call the first_non_repeated_el function with the list and store the result in
'result'
result = first_non_repeated_el(nums)

# Print the result, which is the first non-repeated element in the list
print("First non-repeated element in the said list:")
print(result)

# Create another list of numbers with some repeated elements
nums =  [1, 2, 3, 1, 2, 4, 5, 6, 7, 8]

# Print the original list of numbers
print("\nOriginal list:")
print(nums)

# Call the first_non_repeated_el function with the second list and store the result
in 'result'
result = first_non_repeated_el(nums)

# Print the result, which is the first non-repeated element in the list
print("First non-repeated element in the said list:")
print(result)

# Create another list of numbers with more repeated elements
nums =  [1, 1, 2, 3, 1, 2, 3, 8, 8]

# Print the original list of numbers
print("\nOriginal list:")
print(nums)

# Call the first_non_repeated_el function with the third list and store the result in
'result'
result = first_non_repeated_el(nums)

# Print the result, which is the first non-repeated element in the list
print("First non-repeated element in the said list:")
print(result)
```

3. Write a program to check if two strings are Rotationally Equivalent (that is one can be reached by rotating the other). E.g. “code” and “odec” are rotationally equivalent.

```
test_str1 = 'code'
test_str2 = 'odec'

# printing original strings
print("The original string 1 is : " + str(test_str1))
print("The original string 2 is : " + str(test_str2))

# Check if two strings are Rotationally Equivalent
# Using loop + string slicing
res = False
for idx in range(len(test_str1)):
        if test_str1[idx: ] + test_str1[ :idx] == test_str2:
            res = True
            break

# printing result
print("Are two strings Rotationally equivalent ? : " + str(res))
```

4. Write a Python program to check the validity of passwords input by users as per the conditions below :

   i) At least 1 letter between [a-z] and 1 letter between [A-Z].
   ii) At least 1 number between [0-9].
   iii) At least 1 character from [$#@].
   iv) Minimum length 6 characters.
   v) Maximum length 16 characters.