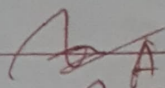
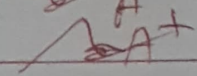
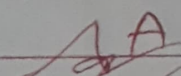
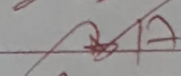
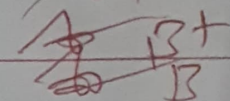
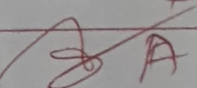
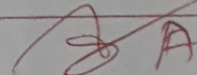


Name	Mukund Kulkreja	Year	2 nd Year
Subject	Python	Class	AIML-1
Semester	III rd	Roll No.	0206A1211059

I N D E X

Sr. No.	Experiment Description	Experiment Date	Submission Date	Remarks / Signature
1	Introduction to python	8/8/22		 A
2	GCD of two numbers	22/8/22		 A+
3	Square root by Newton's method	29/8/22		 A
4	Exponentiation of number	9/9/22		 A
5	Maximum from list	23/9/22		 B+
6	Linear Search	17/10/22		 B
7	Binary Search	7/11/22		 A
8	Selection Sort	28/11/22		
9	Insertion Sort	5/12/22		
10	Merge Sort	12/12/22		

INTRODUCTION TO PYTHON

History of python

Creator - Guido Van Rossum

* He started implementing python at Centrum Wiskunde & Informatica (CWI) Netherlands in 1989.

* He attributes choosing the name "Python" to "being in a slightly irreverent mood (and a big of Monty Python's Flying Circus)".

Features of python

- i) Python is interpreted - It is processed at runtime, you do not need to compile the program before executing which is similar to perl & PHP. You can actually sit at a python prompt and interact with interpreter directly.
 - ii) Python is Object Oriented - It supports object oriented style or technique of programming that encapsulates code.
 - iii) Python is a beginner's level - It is a great language for beginner level programming and supports the development of a wide range of applications from simple text processing to games.
- * It supports Functional & structural programming methods as well as OOP.

- * It can be used as a scripting language or can be compiled to byte code for building large applications.
- * It provides very high level dynamic data types and supports basic type checking.
- * It supports automatic garbage collection.
- * It is easily integrated C, C++, COM, Active X, CORBA, JAVA.

Applications of python

- i) Easy to learn - Few keywords, simple structure & clearly defined syntax, this allows students to pick up the language quickly.
- ii) Easy to read - Python code is more clearly defined & visible eyes.
- iii) Easy to maintain - Python source code is easy to maintain.
- iv) A broad standard library - Its bulk of library is very portable and has cross platform compatibility in Linux, Unix, MacOS, Windows.
- v) Interactive mode - It has support of interactive mode which allows testing & debuggi. debugging of code.
- vi) Portable - Python can run on a wide variety of hardware platforms and has the same interface on all.

- vii) Extensible - You can add low level modules to the python interpreter. These modules enable programmers to add tools or customize their tools to
- viii) Databases - Python provides interfaces to all major commercial databases.
- ix) Scalable - Python provides a better structure and support for large programs than shell scripting.
- x) GUI Programming - Python supports GUI applications that can be created and ported to many systems calls, libraries & windows system such as Windows, MFC, MAC, and x windows system of linux.

Experiment - 1

Write a python program to find GCD of two numbers.

```
import math
```

```
num1 = int(input("Enter first number: "))  
num2 = int(input("Enter second number: "))
```

```
GCD = math.gcd(num1, num2)
```

```
print("The gcd of {} and {} is : {}".format(num1, num2, GCD))
```

Output 1 » Enter first number : 48
Enter second number : 60
The gcd of 48 and 60 is : 12

Experiment - 2

Write a python program to find the square root of a number by Newton's Method.

```
def sqrt(num):
```

```
    assumption = num
```

```
    while (1):
```

```
        root = 0.5 * (assumption + (num/assumption))
```

```
        if (abs(root - assumption) < 0.1):
```

```
            break
```

```
        assumption = root
```

```
    return root
```

```
n = int(input("Enter a number: "))
```

```
print("Square root of {} is : {}".format(n, sqrt(n)))
```

Output 2 ⇒ Enter a number : 327

Square root of 327 is : 18.083166659124

Output 3 >> Enter a number: 4
4 should be raised to the power of?: 2
4 raised to the power of 2 is: 16

Experiment - 3

Write a python program to find exponentiation of number.

```
n = int(input("Enter a number: "))  
p = int(input("{} should be raised to power at?: ".format(n)))  
print("{} raised to the power at {} is: {}".format(n, p, pow(n, p)))
```

Output 4 >> How many numbers will there be in the list

Enter 5 numbers:

12342

2

-312

756

98

Maximum Number in list [12342, 2, -312, 756, 98]

is: 12342

Experiment - 4

Write a python program to find maximum from a list of numbers

```
n = int(input("How many numbers will there be in the list: "))
```

```
numList = []
```

```
print("Enter {} numbers: ".format(n))
```

```
for i in range(n):  
    numList.append(int(input()))
```

```
print("Maximum Number in list {} is: {}".format(numList,  
max(numList)))
```


Output 5 >> Case 1:

Enter a number to search in array [10, 20, 30, 40, 50]
20 is present at index 1 in the array

Case 2:

Enter a number to search in array [10, 20, 30, 40, 50]
72 is not present in the array.

Experiment - 5

Write a python program to perform linear search

```
def linearSearch(arr, key):
```

```
    for index in range(len(arr)):
```

```
        if arr[index] == key:
```

```
            return index
```

```
    return -1
```

```
array = [10, 20, 30, 40, 50]
```

```
k = int(input("Enter a number to search in array {}: ".  
              format(array)))
```

```
searchResult = linearSearch(array, k)
```

```
if searchResult == -1:
```

```
    print(k, "is not present in the array")
```

```
else:
```

```
    print(k, "is present at index", searchResult, "in the  
          array")
```

Output 6 >> Case 1: (Element found)

Enter a number to search in array [10, 20, 30, 40, 50, 60]:
30 is present at index 2 in the array.

Case 2: (Element not found)

Enter a number to search in array [10, 20, 30, 40, 50, 60]: 8
8 is not present in the array.

Experiment - 6

Write python program to perform Binary Search

```
def binarySearch(arr, key):
```

```
    low = 0
```

```
    high = len(arr) - 1
```

```
    while (low <= high):
```

```
        mid = (low + high) // 2
```

```
        if arr[mid] == key:
```

```
            return mid
```

```
        elif arr[mid] > key:
```

```
            high = mid - 1
```

```
        else:
```

```
            low = mid + 1
```

```
    return -1
```

```
array = [10, 20, 30, 40, 50, 60]
```

```
k = int(input("Enter a number to search in array {}: ".format(array)))
```

```
searchResult = binarySearch(array, k)
```

```
if (searchResult == -1):
```

```
    print(k, "is not present in array")
```

```
else:
```

```
    print(k, "is present at index", searchResult, "in the array")
```

Experiment - 7

Write a python program to perform Selection Sort.

```
def getMinIndex(arr, start):
```

```
    index = start
```

```
    for i in range(start, len(arr)):
```

```
        if (arr[i] < arr[index]):
```

```
            index = i
```

```
    return index
```

```
def selectionSort(arr):
```

```
    for i in range(0, len(arr)):
```

```
        minElemIndex = getMinIndex(arr, i)
```

```
        arr[i], arr[minElemIndex] = arr[minElemIndex], arr[i]
```

```
myArr = [15, 30, 25, 10, 35, 20, 45, 40]
```

```
print("Unsorted Array:", myArr)
```

```
selectionSort(myArr)
```

```
print("Sorted Array:", myArr)
```

Output 7>> Unsorted Array: [15, 30, 25, 10, 35, 20, 45, 40]

Sorted Array: [10, 15, 20, 25, 30, 35, 40, 45]

Experiment - 8

Write a python Program to perform Insertion Sort.

```
def insertionSort(arr):
```

```
    for i in range(1, len(arr)):
```

```
        for j in range(i, 0, -1):
```

```
            if (arr[j] < arr[j-1]):
```

```
                arr[j], arr[j-1] = arr[j-1], arr[j]
```

```
            else:
```

```
                break
```

```
myArr = [15, 30, 25, 10, 35, 20, 45, 40]
```

```
print("Unsorted Array:", myArr)
```

```
insertionSort(myArr)
```

```
print("Sorted Array:", myArr)
```

Output 8 » Unsorted Array: [15, 30, 25, 10, 35, 20, 45, 40]

Sorted Array: [10, 15, 20, 25, 30, 35, 40, 45]

Experiment - 10

Write a python program to perform Merge Sort.

def merge(arr, l, m, r):

~~n1 = m - l + 1~~

~~n2 = r - m~~

L = arr[l : m+1] # Copy data to temp lists L[]

R = arr[m+1 : r] # and R[]

i, j = 0, 0

k = l

while i < len(L) and j < len(R):

if L[i] <= R[j]:

arr[k] = L[i]

i += 1

else:

arr[k] = R[j]

j += 1

k += 1

while i < len(L): # Copy remaining elements from L[]

arr[k] = L[i]

i += 1

k += 1

while j < len(R): # Copy remaining elements from R[]

arr[k] = R[j]

j += 1

k += 1

Output 10 >> Unsorted Array: [12, 11, 13, 5, 6, 7]

Sorted Array: [5, 6, 7, 11, 12, 13]

```
def mergeSort (arr, l, r):  
    if l >= r:  
        return
```

```
    m = (l l + r) // 2  
    mergeSort (arr, l, m)  
    mergeSort (arr, m+1, r)  
    merge (arr, l, m, r)
```

```
arr = [12, 11, 13, 5, 6, 7]  
print ("Unsorted Array: ", arr)
```

```
mergeSort (arr, 0, len(arr)-1)
```

```
print ("Sorted Array: ", arr)
```