

**INSTITUTE OF TECHNOLOGY
& MANAGEMENT**
GWALIOR • MP • INDIA

PRACTICAL – FILE

MCA 3rd Semester

MCA – 307

Python Lab

Submitted To:

Mr. Mayank Kurchaniya

Asst.Professor

Dept. of Computer Science

Submitted By:

Shivank sahu

0905CA221007

Date :

INDEX

S.No.	TITLE	PAGE NO.	SIGNATURE
1.	To write a python program to find GCD of two numbers.	3	
2.	To write a python program to find the square root of a number by newton's method.	4	
3.	To write a python program to find the exponential of a number.	5	
4.	To write a python program to find the maximum from a list of numbers.	6	
5.	To write a python program to perform Linear Search.	7	
6.	To write a python program to perform Binary Search.	8	
7.	To write a python program to perform Selection Sort.	9	
8.	To write a python program to perform insertion sort.	10	
9.	To write a python program Merge Sort.	11-12	
10.	To write a python program to find first n prime numbers.	13	

1) To write a python program to find GCD of two numbers.

Algorithm :

Step 1: Initialize GCD = 1

Step 2: Run a loop in the iteration of (i) between [1, min(num1, num2)]

Step 3: Note down the highest number that divides both num1 & num2

Step 4: If i satisfies (num1 % i == 0 and num2 % i == 0) then new value of GCD is i

Step 5: Print value of GCD

Code :

main.py



Run

```
1 num1 = 36
2 num2 = 60
3 gcd = 1
4
5 for i in range(1, min(num1, num2)):
6     if num1 % i == 0 and num2 % i == 0:
7         gcd = i
8 print("GCD of", num1, "and", num2, "is", gcd)
```

Output :

Shell

GCD of 36 and 60 is 12

>

2) To write a python program to find the square root of a number by newton's method.

Algorithm :

Step 1: Newton_sqrt is a function that takes the number x for which you want to find the square root and an optional epsilon parameter, which defines the acceptable level of error in the approximation.

Step 2: It starts with an initial guess for the square root, and then it iteratively refines the guess using the Newton's method formula until the difference between the new guess and the previous guess is smaller than epsilon.

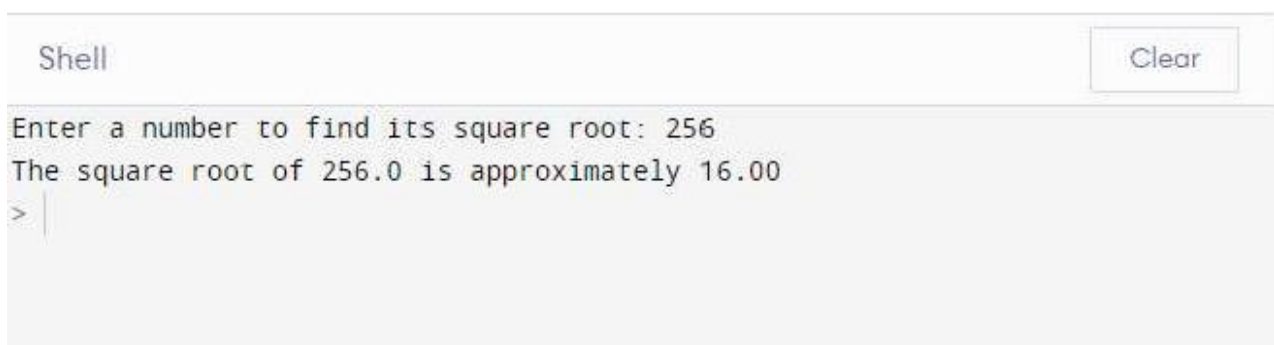
Step 3: The program takes user input for the number, uses the newton_sqrt function to calculate the square root, and then prints the result.

Code :



```
main.py
1 def newton_sqrt(x, epsilon=1e-6):
2     guess = x
3     while True:
4         new_guess = 0.5 * (guess + x / guess)
5         if abs(new_guess - guess) < epsilon:
6             return new_guess
7         guess = new_guess
8
9 if __name__ == "__main__":
10     num = float(input("Enter a number to find its square root: "))
11     result = newton_sqrt(num)
12     print(f"The square root of {num} is approximately {result:.2f}")
13
```

Output :



```
Shell
Enter a number to find its square root: 256
The square root of 256.0 is approximately 16.00
> |
```

3) To write a python program to find the exponential of a number.

Algorithm :

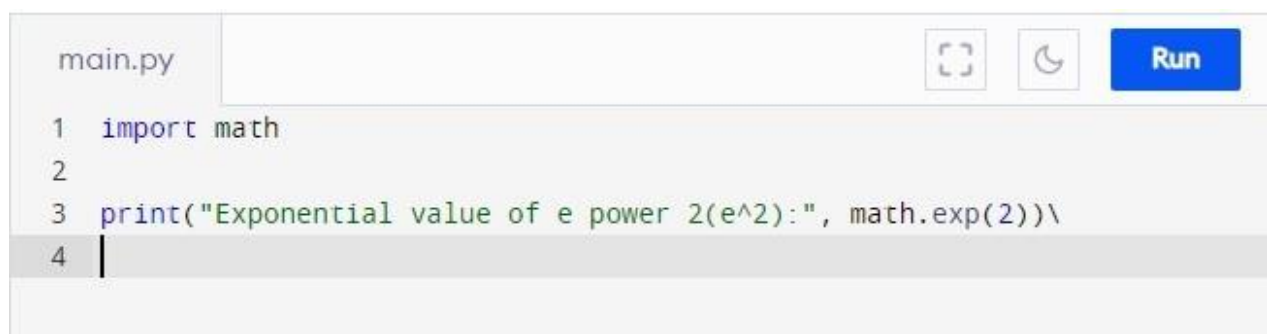
Step 1: Start

Step 2: Import Math module

Step 3: Use the exp() function of the math module to get the exponential value of a positive number passed as an argument to it.

Step 4: Print the output.

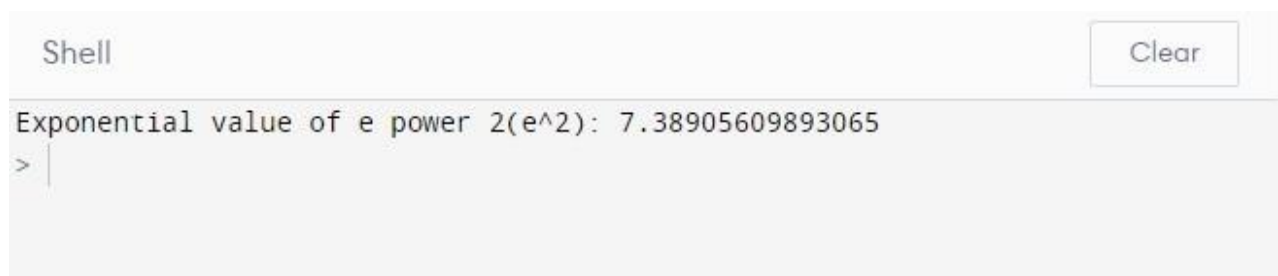
Code :

A screenshot of a Python IDE window titled 'main.py'. The code is as follows:

```
1 import math
2
3 print("Exponential value of e power 2(e^2):", math.exp(2))\
4 |
```

On the right side of the editor, there are three icons: a square with four arrows pointing outwards, a circular arrow, and a blue button labeled 'Run'.

Output :

A screenshot of a shell window titled 'Shell'. It shows the output of the program:

```
Exponential value of e power 2(e^2): 7.38905609893065
> |
```

On the right side of the shell window, there is a button labeled 'Clear'.

4) To write a python program to find the maximum from a list of numbers.

Algorithm :

Step 1: Start

Step 2: Import the heapq module.




Step 3: Create a list of numbers.

Step 4: Use the heapq.nlargest() function to find the largest element.


Step 5: Retrieve the largest element from the list of largest elements returned by heapq.nlargest() function.

Step 6: Print the largest element.

Code :

```
main.py     
1 import heapq  
2  
3 list1 = [10, 20, 4, 45, 99]  
4  
5 largest_element = heapq.nlargest(1, list1)[0]  
6  
7 print("Largest element is:", largest_element)  
8
```

Output :

```
Shell   
Largest element is: 99  
>
```

5) To write a python program to perform Linear Search.

Algorithm :

Step 1: .Import the re module

Step 2: Initialize the input list and element to search for

Step 3: Convert the list to a comma-separated string

Step 4: Use regular expression to search for the element in the string

Step 5: If the element is found, calculate the index of the element in the list by counting the number of commas before the match

Step 6: If the element is not found, display a message.

Step 7: Output the result

Code :

```
main.py ⌂ 🔄 Run
1 import re
2 arr = [10, 20, 80, 30, 60, 50, 110, 100, 130, 170]
3 x = 110
4 arr_str = ','.join(str(i) for i in arr)
5 match = re.search(r'\b{}\b'.format(x), arr_str)
6
7 if match:
8     result = arr_str[:match.start()].count(',')
9     print(f"Element {x} is present at index {result}")
10 else:
11     print(f"Element {x} is not present in the array")
12
```

Output :

```
Shell Clear
Element 110 is present at index 6
> |
```

6) To write a python program to perform Binary Search.

Algorithm :

Step 1: Start

Step 2: Find the middle element of the sorted list.




Step 3: If the middle element is equal to the target value, return its index.

Step 4: If the middle element is greater than the target value, search the left half of the list.


Step 5: If the middle element is less than the target value, search the right half of the list.

Step 6: Repeat steps 2-5 until the target value is found or the search space is empty.

Code :

```
main.py   
1 def binary_search(arr, target):
2     left, right = 0, len(arr) - 1
3     while left <= right:
4         mid = (left + right) // 2
5
6         if arr[mid] == target:
7             return mid
8         elif arr[mid] < target:
9             left = mid + 1
10        else:
11            right = mid - 1
12    return -1
13 if __name__ == "__main__":
14     num_list = [2, 4, 7, 10, 15, 20, 23, 26]
15     target = 15
16     result = binary_search(num_list, target)
17
18     if result != -1:
19         print(f"Element {target} found at index {result}.")
20     else:
21         print(f"Element {target} not found in the list.")
22
```

Output :

```
Shell 
Element 15 found at index 4.
> |
```


7) To write a python program to perform Selection Sort.

Algorithm :

Step 1: Start with the first element as the minimum

Step 2: Compare the current minimum with the elements in the unsorted part of the list.

Step 3: If an element in the unsorted part is smaller (or larger) than the current minimum, update the minimum.

Step 4: After comparing all elements in the unsorted part, swap the minimum (or maximum) element with the first element in the unsorted part.

Step 5: Increase the size of the sorted part by one, and decrease the size of the unsorted part by one.

Step 6: Repeat the process from step 1 until the entire list is sorted.

Code :

main.py

```
1 def selection_sort(arr):
2     n = len(arr)
3     for i in range(n):
4         min_index = i
5
6         for j in range(i + 1, n):
7             if arr[j] < arr[min_index]:
8                 min_index = j
9         if min_index != i:
10            arr[i], arr[min_index] = arr[min_index], arr[i]
11
12 if __name__ == "__main__":
13     num_list = [64, 25, 12, 22, 11]
14     print("Original list:", num_list)
15     selection_sort(num_list)
16     print("Sorted list:", num_list)
```

Output :

Shell

Clear

```
Original list: [64, 25, 12, 22, 11]
Sorted list: [11, 12, 22, 25, 64]
>
```

8) To write a python program to perform insertion sort.

Algorithm :

Step 1: Start with the second element (index 1) and consider it as the key.

Step 2: Compare the key with the element to its left

Step 3: If the key is smaller, move the left element to the right.



Step 4: Repeat step 3 until you find an element that is not greater than the key.

Step 5: Insert the key into the correct position.

Step 6: Move to the next unsorted element (the next element to the right) and repeat steps 2-5.

Step 7: Continue this process until the entire list is sorted.

Code :

```
main.py   
1 def insertion_sort(arr):
2     for i in range(1, len(arr)):
3         key = arr[i]
4         j = i - 1
5         while j >= 0 and key < arr[j]:
6             arr[j + 1] = arr[j]
7             j -= 1
8         arr[j + 1] = key
9 if __name__ == "__main__":
10     num_list = [12, 11, 13, 5, 6]
11     print("Original list:", num_list)
12     insertion_sort(num_list)
13     print("Sorted list:", num_list)
```

Output :

```
Shell 
Original list: [12, 11, 13, 5, 6]
Sorted list: [5, 6, 11, 12, 13]
> |
```

9) To write a python program Merge Sort.




Algorithm :

Step 1: Divide the unsorted list into two equal halves.

Step 2: Recursively sort both halves.

Step 3 : Merge the two sorted halves into a single sorted list.

Code :

```
main.py   
1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr) // 2
4         left_half = arr[:mid]
5         right_half = arr[mid:]
6         merge_sort(left_half)
7         merge_sort(right_half)
8         i = j = k = 0
9         while i < len(left_half) and j < len(right_half):
10            if left_half[i] < right_half[j]:
11                arr[k] = left_half[i]
12                i += 1
13            else:
14                arr[k] = right_half[j]
15                j += 1
16            k += 1
17        while i < len(left_half):
18            arr[k] = left_half[i]
19            i += 1
20            k += 1
21        while j < len(right_half):
22            arr[k] = right_half[j]
23            j += 1
24            k += 1
25
26 if __name__ == "__main__":
27     num_list = [38, 27, 43, 3, 9, 82, 10]
28     print("Original list:", num_list)
29     merge_sort(num_list)
30     print("Sorted list:", num_list)
```

Output :

```
Shell Clear  
Original list: [38, 27, 43, 3, 9, 82, 10]  
Sorted list: [3, 9, 10, 27, 38, 43, 82]  
> |
```

10) To write a python program to find first n prime numbers.

Algorithm :

Step 1: Create a counter variable (say **X = 0**) to keep count of primes found till now and an iterator (say **i**) to iterate through the positive integers starting from 2.



Step 2: Iterate till **X** becomes **N**

Step 3: Check if **i** is a prime or not.

Step 4: If it is a prime, print **i** and increase the value of **X**, otherwise, keep **X** unchanged.

Step 5: Increment the value of **i** by 1.

Code :

```
main.py   Run

1 import math
2 def generatePrime(n):
3     X = 0
4     i = 2
5     flag = False
6     while(X < n):
7         flag = True
8         for j in range(2, math.floor(math.sqrt(i)) + 1):
9             if (i%j == 0):
10                 flag = False
11                 break
12         if(flag):
13             print(i, end=" ")
14             X+=1
15         i+=1
16     print()
17 N = 10
18
19 print ( "First 10 Primes Numbers")
20 generatePrime(N)
```

Output :

```
Shell Clear

First 10 Primes Numbers
2 3 5 7 11 13 17 19 23 29
> |
```