State Institute of Engineering & Technology, Nilokheri
Department of Computer Science & Engineering (AI &ML)

Lab Manual

IT Workshop (Python) Lab
(B23-CSE213)

PROGRAM NO. 1

Aim: Write a program to compute the GCD of two numbers.

Description: The Greatest Common Divisor (GCD), also known as the Highest Common Factor (HCF), is the largest positive integer that divides two or more numbers without leaving any remainder. One of the most efficient methods to find the GCD is the Euclidean Algorithm, which repeatedly divides the larger number by the smaller one and replaces the larger number with the remainder. This process continues until the remainder becomes zero. The last non-zero divisor is the GCD of the two numbers.

Example

Find the GCD of 48 and 18.

- 48 ÷ 18 = 2, Remainder = 12
- 18 ÷ 12 = 1, Remainder = 6
- 12 ÷ 6 = 2, Remainder = 0

Therefore, GCD = 6.

Algorithm:

1. Start.
2. Read two numbers from the user.
3. Repeat while the second number is not equal to zero:
    - Find the remainder of the first number divided by the second number.
    - Replace the first number with the second number.
    - Replace the second number with the remainder.
4. When the second number becomes zero, the first number is the GCD.
5. Display the GCD.
6. Stop.

Program:

```
a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))

while b != 0:
   a, b = b, a % b

print("GCD =", a)
```

Output:

```
>>> Enter the first number: 48
Enter the second number: 18
 GCD = 6
```

Viva Questions:

Q1. What is the GCD of a number and 0?
Answer: The GCD of a number n and 0 is n (provided $n \neq 0$).

Q2. Which operator is used to find the remainder in Python?
Answer: The modulus operator %.

PROGRAM NO. 2

Aim: Write a program to find the square root of a number

Description: The square root of a number is a value that, when multiplied by itself, gives the original number. For example, the square root of 25 is 5, because 5 × 5 = 25. In Python, the square root of a number can be calculated using the sqrt() function available in the math module. This function returns the positive square root of the given number. It is widely used in mathematics, engineering, scientific calculations, graphics, and data analysis.

Example
Find the square root of 64.

$$8 \times 8 = 64$$

Therefore, $\sqrt{64} = 8$

Algorithm:

1. Start.
2. Import the math module.
3. Read a number from the user.
4. Calculate its square root using the math.sqrt() function.
5. Display the result.
6. Stop.

Program:

```
import math
num = float(input("Enter a number: "))
result = math.sqrt(num)
print("Square Root =", result)
```

Output:

```
>>>
Enter a number: 64

Square Root = 8.0
>>>
```

Viva Questions

Q1. Which module is used to calculate the square root in Python?
Answer: The math module.

Q2. Which function is used to find the square root?
Answer: The math.sqrt() function.

Q3. Why do we import the math module?
Answer: To access mathematical functions such as sqrt(), sin(), cos(), and log().

PROGRAM NO. 3

Aim: Write a program to find the Exponentiation (power of a number)

Description: Exponentiation is a mathematical operation in which a number, called the base, is multiplied by itself a specified number of times, called the exponent or power. In Python, exponentiation can be performed using the ** (double asterisk) operator or the built-in pow() function.

Example
Find the value of $\mathbf{5^3}$.

$$5^3 = 5 \times 5 \times 5 = 125$$

Program:

```
base = float(input("Enter the base number: "))

power = int(input("Enter the exponent: "))

result = base ** power

print("Result =", result)
```

Output:

```
>>> Enter the base number: 5
Enter the exponent: 3
Result = 125.0 >>>
```

Viva Questions:

Q1. Which operator is used for exponentiation in Python?
Answer: The ** (double asterisk) operator.

Q2. Which built-in function can also be used for exponentiation?
Answer: The pow() function.

Q3. Can exponentiation be performed on floating-point numbers?
Answer: Yes. Python supports exponentiation for both integers and floating-point numbers.

Q4. Can the exponent be negative in Python?
Answer: Yes. A negative exponent returns the reciprocal of the positive power. For example, 2 ** -2 returns 0.25.

PROGRAM NO. 4

Aim: Write a program to find the maximum of a list of numbers

Description: A list in Python is a collection of elements stored in a single variable. Finding the maximum element means identifying the largest value present in the list. Python provides a built-in function called max() that returns the largest element from a list.

Example

Consider the following list:

[12, 45, 8, 67, 23]

The largest number in the list is 67.

Therefore, Maximum = 67

Algorithm:

1. Start.
2. Create or input a list of numbers.
3. Use the max() function to find the largest element.
4. Display the maximum value.
5. Stop.

Program:

```
numbers = [ ]
n = int(input("Enter the number of elements: "))
for i in range(n):
   num = int(input("Enter element: "))
   numbers.append(num)
maximum = max(numbers)
print("The maximum number is:", maximum)
```

Output:

```
>>> Enter the number of elements: 4
    Enter element: 12
    Enter element: 45
    Enter element: 8
    Enter element: 67
    The maximum number is: 67
```

Viva Questions

Q1. What is a list in Python?

Answer: A list is an ordered and mutable collection of elements enclosed within square brackets [ ].

Q2. Can the max() function work with floating-point numbers?

Answer: Yes. It works with both integers and floating-point numbers.

PROGRAM NO. 5

Aim: Write a program for Linear search and Binary search.

Description: Linear Search is one of the simplest searching techniques used to find an element in a list. In this method, each element of the list is compared sequentially with the target element until a match is found or the end of the list is reached. If the element is found, its position is displayed; otherwise, the program reports that the element is not present.

Algorithm

1. Start.
2. Read the number of elements.
3. Input the list elements.
4. Read the element to be searched.
5. Compare the search element with each element of the list.
6. If found, display its position.
7. Otherwise, display "Element not found."
8. Stop.

Program:

```
numbers = []
n = int (input ("Enter the number of elements: "))
for i in range(n):
   numbers.append(int (input ("Enter element: ")))
key = int (input ("Enter the element to search: "))
found = False
for i in range(n):
   if numbers[i] == key:
      print("Element found at position", i + 1)
      found = True
      break
if not found:
   print("Element not found")
```

Output:

```
>>>   Enter the number of elements: 5
      Enter element: 12
      Enter element: 25
      Enter element: 18
      Enter element: 40
      Enter element: 35
      Enter the element to search: 40
      Element found at position 4
```

Binary Search:

Description: Binary Search is an efficient searching algorithm used to find an element in a sorted list. Instead of checking every element, it repeatedly divides the search range into two halves. First, it compares the middle element with the target value. If they are equal, the search is successful. If the target is smaller, the search continues in the left half; otherwise, it continues in the right half. This process is repeated until the element is found or the search range becomes empty.

Algorithm:

1. Start.
2. Read the sorted list.
3. Read the element to search.
4. Set low = 0 and high = n - 1.
5. Repeat while low <= high:
    - Find the middle element.
    - If it matches the search element, display its position.
    - If the search element is smaller, update high.
    - Otherwise, update low.
6. If the element is not found, display "Element not found."
7. Stop.

Program:

```
numbers = []
n = int(input("Enter the number of elements: "))
print("Enter elements in sorted order:")
for i in range(n):
    numbers.append(int(input("Enter element: ")))
key = int(input("Enter the element to search: "))
low = 0
high = n - 1
found = False
while low <= high:
    mid = (low + high) // 2
    if numbers[mid] == key:
        print("Element found at position", mid + 1)
        found = True
        break
   elif key < numbers[mid]:
        high = mid - 1
    else:
        low = mid + 1
if not found:
    print("Element not found")
```

Output:

Enter the number of elements: 5

Enter elements in sorted order:
10
20
30
40
50
Enter the element to search: 50
Element found at position 5

Viva Questions:

Q1. What is the main requirement for Binary Search?
Answer: The list must be sorted.

Q2. What is the time complexity of Binary Search?
Answer: $O(\log_2 n)$.

Q3. Why is Binary Search faster than Linear Search?
Answer: Because it reduces the search space by half after each comparison.

Q4. What happens if the list is not sorted?
Answer: Binary Search may produce incorrect results because it relies on the elements being in sorted order.

Q5. What is the major advantage of Binary Search?
Answer: It is highly efficient for searching large sorted datasets.

Q6. What is the worst-case time complexity of Linear Search?
Answer: O(n), when the element is at the last position or not present.

Q7. Where is Linear Search commonly used?
Answer: Small datasets, unsorted lists, and simple applications.

Q8. What is the time complexity of Linear Search?
Answer: O(n).

Q9. What is the best-case time complexity of Linear Search?
Answer: O(1), when the element is found at the first position.

Q10. What is the main disadvantage of Linear Search?
Answer: It is slow for large datasets because each element must be checked.

PROGRAM NO. 6

Aim: Write a program for Selection sort

Description: Selection Sort is a simple comparison-based sorting algorithm used to arrange elements in ascending or descending order. In this algorithm, the smallest element from the unsorted part of the list is selected and swapped with the first unsorted element. This process is repeated until all elements are sorted. After each pass, one element is placed in its correct position, and the sorted portion of the list increases by one element.

Example

Consider the following list:

[64, 25, 12, 22, 11]

Pass 1: Smallest element = **11**

[11, 25, 12, 22, 64]

Pass 2: Smallest element = **12**

[11, 12, 25, 22, 64]

Pass 3: Smallest element = **22**

[11, 12, 22, 25, 64]

Pass 4: Smallest element = **25**

[11, 12, 22, 25, 64]

Final Sorted List:

[11, 12, 22, 25, 64]

Algorithm:

1. Start.
2. Read the number of elements.
3. Input the list elements.
4. Repeat for each position from the first element to the second last element:
    - Assume the current element is the smallest.
    - Compare it with all remaining elements.
    - Find the smallest element.
    - Swap it with the current element.
5. Display the sorted list.
6. Stop.

Program:

```
numbers = []
n = int(input("Enter the number of elements: "))
for i in range(n):
    num = int(input("Enter element: "))
    numbers.append(num)
for i in range(n - 1):
    min_index = i
```

```
    for j in range(i + 1, n):
        if numbers[j] < numbers[min_index]:
            min_index = j
    numbers[i], numbers[min_index] = numbers[min_index], numbers[i]
print("Sorted List:")
print(numbers)
```

Output:

```
Enter the number of elements: 5
Enter element: 64
Enter element: 25
Enter element: 12
Enter element: 22
Enter element: 11
Sorted List:
[11, 12, 22, 25, 64]
```

Viva Questions:

Q1. What is the best-case time complexity of Selection Sort?
Answer: $O(n^2)$.

Q2. What is the average-case time complexity of Selection Sort?
Answer: $O(n^2)$.

Q3. What is the worst-case time complexity of Selection Sort?
Answer: $O(n^2)$.

Q4. What is the space complexity of Selection Sort?
Answer: O(1) because it is an in-place sorting algorithm.

Q5 Is Selection Sort a stable sorting algorithm?
Answer: No. The standard implementation of Selection Sort is not stable because swapping can change the relative order of equal elements.

Q6. Does Selection Sort require extra memory?
Answer: No. It sorts the list in place and requires only a constant amount of extra memory.

Q7. Is Selection Sort suitable for large datasets?
Answer: No. More efficient algorithms such as Merge Sort, Quick Sort, or Heap Sort are preferred for large datasets.

PROGRAM NO. 7

Aim: Write a program for Insertion sort

Description: Insertion Sort is a simple comparison-based sorting algorithm that builds the sorted list one element at a time. It works by taking one element from the unsorted part of the list and inserting it into its correct position in the sorted part. Initially, the first element is considered sorted. Then, each subsequent element is compared with the elements before it and shifted to the correct position if necessary.

Example

Consider the following list:

[64, 25, 12, 22, 11]

Pass 1: Insert **25**

[25, 64, 12, 22, 11]

Pass 2: Insert **12**

[12, 25, 64, 22, 11]

Pass 3: Insert **22**

[12, 22, 25, 64, 11]

Pass 4: Insert **11**

[11, 12, 22, 25, 64]

Final Sorted List:

[11, 12, 22, 25, 64]

Algorithm

1. Start.
2. Read the number of elements.
3. Input the list elements.
4. Consider the first element as already sorted.
5. Select the next element (called the **key**).
6. Compare the key with the elements in the sorted portion.
7. Shift larger elements one position to the right.
8. Insert the key into its correct position.
9. Repeat Steps 5–8 until all elements are sorted.
10. Display the sorted list.
11. Stop.

Program:

```
numbers = [ ]
n = int(input("Enter the number of elements: "))
for i in range(n):
    num = int(input("Enter element: "))
    numbers.append(num)
for i in range(1, n):
    key = numbers[i]
    j = i - 1
    while j >= 0 and numbers[j] > key:
        numbers[j + 1] = numbers[j]
        j = j - 1
```

```
    numbers[j + 1] = key
print("Sorted List:")
print(numbers)
```

Output:

```
Enter the number of elements: 5
Enter element: 64
Enter element: 25
Enter element: 12
Enter element: 22
Enter element: 11
Sorted List:
[11, 12, 22, 25, 64]
```

Viva Questions:

Q1. What is the best-case time complexity of Insertion Sort?
Answer: O(n), when the list is already sorted.

Q2. What is the average-casetime complexity of Insertion Sort?
Answer: $O(n^2)$.

Q3. What is the worst-case time complexity of Insertion Sort?
Answer: $O(n^2)$, when the list is in reverse order.

Q4. What is the space complexity of Insertion Sort?
Answer: O(1) because it sorts the list in place.

Q5. Is Insertion Sort a stable sorting algorithm?
Answer: Yes. It preserves the relative order of equal elements.

Q6. Does Insertion Sort require extra memory?
Answer: No. It is an in-place sorting algorithm and uses only constant extra memory.

Q7. When is Insertion Sort preferred?
Answer: It is preferred for small datasets, nearly sorted data, and situations where elements are added one at a time.

PROGRAM NO. 8

Aim: Write a program to find first n prime numbers

Description: A prime number is a positive integer greater than 1 that has exactly two factors: 1 and itself. Numbers such as 2, 3, 5, 7, 11, and 13 are prime numbers, whereas numbers like 4, 6, 8, and 9 are not prime because they have more than two factors. To find the first *n* prime numbers, the program checks each number starting from 2. For every number, it counts the number of factors. If the number has exactly two factors, it is printed as a prime number. The process continues until the required number of prime numbers is obtained.

Example

Find the first **5** prime numbers.
The prime numbers are:
2, 3, 5, 7, 11
Therefore, the first **5** prime numbers are:
2 3 5 7 11

Algorithm

1. Start.
2. Read the value of **n** from the user.
3. Initialize count = 0 and num = 2.
4. Repeat until count becomes equal to n:
    - Check whether num is prime.
    - If it is prime:
        - Display the number.
        - Increase count by 1.
    - Increase num by 1.
5. Stop.

Program:

```
n = int(input("Enter the value of n: "))
count = 0
num = 2
print("First", n, "prime numbers are:")
while count < n:
   prime = True
   for i in range(2, int(num ** 0.5) + 1):
      if num % i == 0:
         prime = False
         break
   if prime:
      print(num, end=" ")
      count += 1
   num += 1
```

Output:

Enter the value of n: 10
First 10 prime numbers are:
2 3 5 7 11 13 17 19 23 29

Viva Questions:

Q1. Which loop is used in this program?
Answer: A while loop is used to generate the required number of prime numbers, and a for loop is used to check whether each number is prime.

Q2. Why is the condition int(num ** 0.5) + 1 used?
Answer: A number only needs to be checked for divisibility up to its square root, making the program more efficient.

Q3. Can a negative number be a prime number?
Answer: No. Prime numbers are positive integers greater than 1.

Q4. What is the difference between a prime number and a composite number?
Answer: A prime number has exactly two factors, while a composite number has more than two factors.

Q5. What is the time complexity of this prime-checking approach?
Answer: Checking one number takes approximately O(√n) time. Generating the first n prime numbers requires repeated checks, so the overall execution time depends on the value of n.

PROGRAM NO. 9

Aim: Write a program to multiply matrices

Description: Matrix multiplication is a mathematical operation in which two matrices are multiplied to produce a new matrix. Matrix multiplication is possible only when the number of columns in the first matrix is equal to the number of rows in the second matrix. Each element of the resulting matrix is obtained by multiplying the corresponding elements of a row from the first matrix with a column from the second matrix and then adding the products.

Example
Let the two matrices be:
Matrix A

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Matrix B

$$\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

The multiplication is:

$$C_{11} = (1 \times 5) + (2 \times 7) = 19$$
$$C_{12} = (1 \times 6) + (2 \times 8) = 22$$
$$C_{21} = (3 \times 5) + (4 \times 7) = 43$$
$$C_{22} = (3 \times 6) + (4 \times 8) = 50$$

Therefore, the resultant matrix is:

$$\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

Algorithm

1. Start.
2. Read the number of rows and columns of the matrices.
3. Input the elements of the first matrix.
4. Input the elements of the second matrix.
5. Initialize a result matrix with all elements as zero.
6. Multiply the corresponding rows and columns using nested loops.
7. Store the result in the resultant matrix.
8. Display the resultant matrix.
9. Stop.

Program:

```
rows = int(input("Enter number of rows: "))
cols = int(input("Enter number of columns: "))
print("Enter elements of Matrix A:")
A = []
for i in range(rows):
    row = []
```

```
    for j in range(cols):
        row.append(int(input()))
    A.append(row)

print("Enter elements of Matrix B:")
B = []

for i in range(rows):
    row = []
    for j in range(cols):
        row.append(int(input()))
    B.append(row)
result = []
for i in range(rows):
    result.append([0] * cols)
for i in range(rows):
    for j in range(cols):
        for k in range(cols):
            result[i][j] += A[i][k] * B[k][j]
print("\nResultant Matrix:")
for i in range(rows):
    for j in range(cols):
        print(result[i][j], end=" ")
    print()
```

Output:

```
Enter number of rows: 2
Enter number of columns: 2
Enter elements of Matrix A:
1
2
3
4
Enter elements of Matrix B:
5
6
7
8
Resultant Matrix:
19 22
43 50
```

Viva Questions:

Q1. What is the order of the result matrix if Matrix A is of order $m \times n$and Matrix B is of order $n \times p$?
Answer: The resultant matrix is of order $m \times p$.

Q2. Is matrix multiplication commutative?
Answer: No. In general, A × B ≠ B × A.

PROGRAM NO. 10

Aim: Write a program that take command line arguments (word count)

Description: Command line arguments are the values provided to a program when it is executed from the command prompt or terminal. In Python, these arguments are accessed using the sys.argv list from the sys module. The first element (sys.argv[0]) contains the program name, while the remaining elements contain the arguments entered by the user. In this program, the words entered as command line arguments are counted by subtracting one from the total number of elements in sys.argv.

Example
Suppose the program file is named **wordcount.py**.
Command:
python wordcount.py Welcome to Python Programming
Arguments received:

- Welcome
- to
- Python
- Programming

Therefore,
Word Count = 4

Algorithm

1. Start.
2. Import the sys module.
3. Read the command line arguments using sys.argv.
4. Count the number of arguments excluding the program name.
5. Display the total number of words.
6. Stop.

Program:

```
import sys
count = len(sys.argv) - 1
print("Words entered are:")
for i in range(1, len(sys.argv)):
   print(sys.argv[i])
print("Total Number of Words =", count)
```

Output:

```
Words entered are:
Welcome
to
```

Python
Programming

Total Number of Words = 4

Viva Questions:

Q1. What are command line arguments?
Answer: Command line arguments are values passed to a program when it is executed from the command prompt or terminal.

Q2. Which module is used to access command line arguments in Python?
Answer: The sys module.

Q3. What is sys.argv?
Answer: sys.argv is a list that stores the command line arguments passed to a Python program.

Q4. What does sys.argv[0] contain?
Answer: It contains the name (or path) of the Python program.

Q5. Why is len(sys.argv) - 1 used in this program?
Answer: Because sys.argv[0] contains the program name, so subtracting 1 gives the actual number of command line arguments (words).

Q6. What is the output of len(sys.argv) if the command is:
python wordcount.py Hello World
Answer: len(sys.argv) is 3 (wordcount.py, Hello, World).

Q7. Can command line arguments contain numbers?
Answer: Yes. Command line arguments can contain strings, numbers, or any text, but they are received as strings.

Q8. How can numeric command line arguments be used for calculations?
Answer: They must first be converted to int or float using functions like int() or float().

Q9. What happens if no command line arguments are provided?
Answer: Only the program name is stored in sys.argv, so the word count will be 0.

Q10. What is the type of sys.argv?
Answer: sys.argv is a list in Python.

PROGRAM NO. 11

Aim: Write a program to find the most frequent words in a text read from a file

Description: A text file stores data in the form of characters and words. Sometimes, it is useful to know which words occur most frequently in a file for text analysis, document summarization, or data mining. In Python, a file can be opened using the open() function and its contents can be read using the read() method. The text is then converted into a list of words using the split() function. A dictionary is used to count how many times each word appears in the file. Finally, the word with the highest frequency is displayed.

Example

Suppose the file **sample.txt** contains the following text:
Python is easy to learn.
Python is powerful.
Python is popular.
Word frequencies are:

- Python → 3
- is → 3
- easy → 1
- to → 1
- learn → 1
- powerful → 1
- popular → 1

Therefore, the most frequent words are **Python** and **is**, each appearing **3 times**.

Algorithm

1. Start.
2. Open the text file in read mode.
3. Read the contents of the file.
4. Convert the text to lowercase.
5. Split the text into words.
6. Count the frequency of each word using a dictionary.
7. Find the maximum frequency.
8. Display the word(s) having the highest frequency and their count.
9. Close the file.
10. Stop.

Program:

```
file = open("sample.txt", "r")
text = file.read().lower()
words = text.split()
frequency = {}
```

```
for word in words:
    word = word.strip(".,!?;:\"'()[]{}")
    if word in frequency:
        frequency[word] += 1
    else:
        frequency[word] = 1

max_count = max(frequency.values())
print("Most Frequent Word(s):")
for word in frequency:
    if frequency[word] == max_count:
        print(word, "->", frequency[word], "times")

file.close()
```

Output:

```
>>>
Most Frequent Word(s):
python -> 3 times
is -> 3 times
```

Viva Questions

Q1. What is a text file?
Answer: A text file is a file that stores data in the form of characters and words.

Q2. Which function is used to open a file in Python?
Answer: The open() function.

Q3. Which method is used to read the entire contents of a file?
Answer: The read() method.

Q4. Why is the split() function used?
Answer: It splits the text into individual words.

Q5. Why is lower() used in this program?
Answer: It converts all words to lowercase so that words like Python and python are treated as the same word.

Q6. Which data structure is used to count word frequencies?
Answer: A dictionary.

Q7. Why is strip() used in the program?
Answer:
It removes punctuation marks attached to words, such as commas and periods.