PRACTICAL-1

AIM: Write a program to find the roots of a quadratic equation.

CODE:

```
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))

d = b**2 - 4*a*c

if d >= 0:
    root1 = (-b + d**0.5) / (2*a)
    root2 = (-b - d**0.5) / (2*a)
else:
    root1 = (-b / (2*a)) + ((-d)**0.5 / (2*a)) * 1j
    root2 = (-b / (2*a)) - ((-d)**0.5 / (2*a)) * 1j

print("Roots are:", root1, "and", root2)
```

OUTPUT:

```
Enter coefficient a:  1
Enter coefficient b:  -3
Enter coefficient c:  2
Roots are: 2.0 and 1.0
```

PRACTICAL-2

AIM: Write a program to accept a number ‘n’.

a. Check if ’n’ is prime

b. Generate all prime numbers till ‘n’

c. Generate first ‘n’ prime numbers This program may be done using functions.

CODE:

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

def primes_up_to_n(n):
    primes = []
    for i in range(2, n + 1):
        if is_prime(i):
            primes.append(i)
    return primes

def first_n_primes(n):
    primes = []
    i = 2
    while len(primes) < n:
        if is_prime(i):
            primes.append(i)
        i += 1
    return primes

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

if is_prime(n):
    print(f"{n} is a prime number.")
else:
    print(f"{n} is not a prime number.")

print(f"All prime numbers up to {n}:", primes_up_to_n(n))
print(f"The first {n} prime numbers:", first_n_primes(n))
```

OUTPUT:

```
Enter a number:  10
10 is not a prime number.
All prime numbers up to 10: [2, 3, 5, 7]
The first 10 prime numbers: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

PRACTICAL-3

AIM: Write a program to create a pyramid of the character '*' and a reverse pyramid.

CODE:

```
def pyramid(n):
    for i in range(1, n+1):
        print(' ' * (n - i) + '*' * (2 * i - 1))

def reverse_pyramid(n):
    for i in range(n, 0, -1):
        print(' ' * (n - i) + '*' * (2 * i - 1))

n = int(input("Enter the number of rows: "))

print("Pyramid:")
pyramid(n)

print("Reverse Pyramid:")
reverse_pyramid(n)
```

OUTPUT:

```
Enter the number of rows:  6
Pyramid:
     *
    ***
   *****
  *******
 *********
***********
Reverse Pyramid:
***********
 *********
  *******
   *****
    ***
     *
```

PRACTICAL-4

AIM: Write a program that accepts a character and performs the following:

a. print whether the character is a letter, numeric digit, or a special character.

b. if the character is a letter, print whether the letter is uppercase or lowercase

c. if the character is a numeric digit, prints its name in text (e.g., if input is 9, output is NINE)

CODE:

```
def print_digit_name(n):
    digit_names = ["ZERO", "ONE", "TWO", "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT", "NINE"]
    return digit_names[n]

char = input("Enter a character: ")

if char.isalpha():
    print(f"{char} is a letter.")
    if char.isupper():
        print("It is uppercase.")
    else:
        print("It is lowercase.")
elif char.isdigit():
    print(f"{char} is a numeric digit.")
    print(f"Its name in text is {print_digit_name(int(char))}.")
else:
    print(f"{char} is a special character.")
```

OUTPUT:

```
Enter a character:  A
A is a letter.
It is uppercase.

Enter a character:  n
n is a letter.
It is lowercase.

Enter a character:  4
4 is a numeric digit.
Its name in text is FOUR.

Enter a character:  @
@ is a special character.
```

PRACTICAL-5

AIM: Write a program to perform the following operations on a string:
a. Find the frequency of a character in a string.
b. Replace a character by another character in a string.
c. Remove the first occurrence of a character from a string.
d. Remove all occurrences of a character from a string.

CODE:

```
def find_frequency(s, char):
    return s.count(char)

def replace_char(s, old_char, new_char):
    return s.replace(old_char, new_char)

def remove_first_occurrence(s, char):
    return s.replace(char, '', 1)

def remove_all_occurrences(s, char):
    return s.replace(char, '')

s = input("Enter a string: ")

char = input("Enter the character to find its frequency: ")
print(find_frequency(s, char))

old_char = input("Enter the character to replace: ")
new_char = input("Enter the new character: ")
print(replace_char(s, old_char, new_char))

char_to_remove = input("Enter the character to remove the first occurrence: ")
print(remove_first_occurrence(s, char_to_remove))

char_to_remove_all = input("Enter the character to remove all occurrences: ")
print(remove_all_occurrences(s, char_to_remove_all))
```

OUTPUT:

```
Enter a string:  flibbertigibbet
Enter the character to find its frequency:  b
4
Enter the character to replace:  b
Enter the new character:  t
flittertigittet
Enter the character to remove the first occurrence:  b
flibertigibbet
Enter the character to remove all occurrences:  b
fliertigiet
```

PRACTICAL-6

AIM: Write a program to swap the first n characters of two strings.

CODE:

```
def swap_strings(str1, str2, n):
    return str2[:n] + str1[n:], str1[:n] + str2[n:]

str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")
n = int(input("Enter the number of characters to swap: "))

new_str1, new_str2 = swap_strings(str1, str2, n)
print("New first string:", new_str1)
print("New second string:", new_str2)
```

OUTPUT:

```
Enter the first string:  hello
Enter the second string:  world
Enter the number of characters to swap:  2
New first string: wollo
New second string: herld
```

PRACTICAL-7

AIM: Write a function that accepts two strings and returns the indices of all the occurrences of the second string in the first string as a list. If the second string is not present in the first string then it should return -1.

CODE:

```
def find_occurrences(str1, str2):
    indices = []
    start = 0
    while start < len(str1):
        start = str1.find(str2, start)
        if start == -1:
            break
        indices.append(start)
        start += 1
    return indices if indices else -1

str1 = input("Enter the first string: ")
str2 = input("Enter the second string: ")

print(find_occurrences(str1, str2))
```

OUTPUT:

```
Enter the first string:  hellohello
Enter the second string:  lo
[3, 8]
```

PRACTICAL-8

AIM: Write a program to create a list of the cubes of only the even integers appearing in the input list (may have elements of other types also) using the following:

a. 'for' loop;

b. list comprehension

CODE:

```
def cubes_with_loop(lst):
    result = []
    for x in lst:
        if isinstance(x, int) and x % 2 == 0:
            result.append(x ** 3)
    return result

def cubes_with_comprehension(lst):
    return [x ** 3 for x in lst if isinstance(x, int) and x % 2 == 0]

lst = eval(input("Enter a list of elements: "))
print("Using 'for' loop:", cubes_with_loop(lst))
print("Using list comprehension:", cubes_with_comprehension(lst))
```

OUTPUT:

```
Enter a list of elements:  [1, 2, 'a', 4, 5.5, 6, 'hello', 8]
Using 'for' loop: [8, 64, 216, 512]
Using list comprehension: [8, 64, 216, 512]
```

PRACTICAL-9

AIM: Write a program to read a file and

a. Print the total number of characters, words, and lines in the file.
b. Calculate the frequency of each character in the file. Use a variable dictionary type to maintain the count.
c. Print the words in reverse order.
d. Copy even lines of the file to a file named ‘File1’ and odd lines to another file named ‘File2’.

CODE:

```
def file_operations():
    filename = input("Enter the filename: ")
    with open(filename, 'r') as f:
        lines = f.readlines()

    total_characters = sum(len(line) for line in lines)
    total_words = sum(len(line.split()) for line in lines)
    total_lines = len(lines)
    print("Total characters:", total_characters)
    print("Total words:", total_words)
    print("Total lines:", total_lines)

    char_frequency = {}
    for line in lines:
        for char in line:
            char_frequency[char] = char_frequency.get(char, 0) + 1
    print("Character frequencies:", char_frequency)

    reversed_words = [word[::-1] for line in lines for word in line.split()]
    print("Words in reverse order:", reversed_words)

    with open('File1.txt', 'w') as f1, open('File2.txt', 'w') as f2:
        for i, line in enumerate(lines):
            if i % 2 == 0:
                f1.write(line)
            else:
                f2.write(line)

file_operations()
```

OUTPUT:

```
Enter the filename:  example.txt
Total characters: 104
Total words: 21
Total lines: 3
Character frequencies: {'H': 2, 'e': 9, 'l': 5, 'o': 8, ' ': 18, 'g': 5, 'u': 3, 'y': 3, 's': 2, ',': 2, '\n': 2, 'T': 1, 'd': 2, 'a': 8, 'w': 3, 'r':
3, 'i': 5, 'n': 9, 't': 5, 'b': 1, 'h': 3, 'f': 2, 'p': 1, 'v': 1, '.': 1}
Words in reverse order: ['olleH', ',syug', 'yadoT', 'ew', 'era', 'gniog', 'ot', 'nrael', 'tuoba', 'woh', 'ot', 'daer', 'a', 'elif', 'ni', ',nohtyp', 'ev
aH', 'nuf', 'gninael', 'wen', '.sgniht']
```

PRACTICAL-10

AIM: Write a program to define a class Point with coordinates x and y as attributes. Create relevant methods and print the objects. Also define a method distance to calculate the distance between any two point objects.

CODE:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f"Point({self.x}, {self.y})"

    def distance(self, other):
        return ((self.x - other.x) ** 2 + (self.y - other.y) ** 2) ** 0.5

x1, y1 = map(int, input("Enter coordinates for Point 1 (x y): ").split())
x2, y2 = map(int, input("Enter coordinates for Point 2 (x y): ").split())

p1 = Point(x1, y1)
p2 = Point(x2, y2)

print(p1)
print(p2)
print("Distance between points:", p1.distance(p2))
```

OUTPUT:

```
Enter coordinates for Point 1 (x y):  1 2
Enter coordinates for Point 2 (x y):  4 6
Point(1, 2)
Point(4, 6)
Distance between points: 5.0
```

PRACTICAL-11

AIM: Write a function that prints a dictionary where the keys are numbers between 1 and 5, and the values are cubes of the keys.

CODE:

```
def cube_dict():
    result = {}
    for i in range(1, 6):
        result[i] = i ** 3
    print(result)

cube_dict()
```

OUTPUT:

```
{1: 1, 2: 8, 3: 27, 4: 64, 5: 125}
```

PRACTICAL-12

AIM: Consider a tuple t1=(1, 2, 5, 7, 9, 2, 4, 6, 8, 10). Write a program to perform the following operations:

a. Print half the values of the tuple in one line and the other half in the next line.
b. Print another tuple whose values are even numbers in the given tuple.
c. Concatenate a tuple t2=(11,13,15) with t1.
d. Return maximum and minimum value from this tuple

CODE:

```
t1 = (1, 2, 5, 7, 9, 2, 4, 6, 8, 10)
t2 = (11, 13, 15)

half = len(t1) // 2
print(t1[:half])
print(t1[half:])

even_tuple = tuple(x for x in t1 if x % 2 == 0)
print(even_tuple)

concatenated_tuple = t1 + t2
print(concatenated_tuple)

print("Max:", max(t1))
print("Min:", min(t1))
```

OUTPUT:

```
(1, 2, 5, 7, 9)
(2, 4, 6, 8, 10)
(2, 2, 4, 6, 8, 10)
(1, 2, 5, 7, 9, 2, 4, 6, 8, 10, 11, 13, 15)
Max: 10
Min: 1
```

PRACTICAL-13

AIM: Write a program to accept a name from a user. Raise and handle appropriate exception(s) if the text entered by the user contains digits and/or special characters.

CODE:

```
def accept_name():
    name = input("Enter your name: ")
    try:
        if any(char.isdigit() for char in name) or not name.isalpha():
            raise ValueError("Name should not contain digits or special characters")
        print("Name accepted:", name)
    except ValueError as e:
        print(e)

accept_name()
```

OUTPUT:

```
Enter your name:  Avijit
Name accepted: Avijit

Enter your name:  Avi123
Name should not contain digits or special characters

Enter your name:  @vi123
Name should not contain digits or special characters
```