#CODE NO.1.Write simple Python program to check entered number is Even or Odd using if else statements

```
# Get input from the user
number = int(input("Enter a number: "))

# Check if the number is even or odd
if number % 2 == 0:
    print(f"{number} is an Even number.")
else:
    print(f"{number} is an Odd number.")
```

#CODE NO.2.Python program to find the area of a triangle using user defined function

```
def triangle_area(base, height):
    return 0.5 * base * height

# Input
base = float(input("Base: "))
height = float(input("Height: "))

# Output
print("Area:", triangle_area(base, height))
```

#CODE NO.3.Python Program to Display the multiplication Table using while loop

```
num = int(input("Enter a number: "))
limit = int(input("Enter limit: "))
```

```
i = 1

while i <= limit:
    print(f"{num} x {i} = {num * i}")
    i += 1
```

#CODE NO.4.Python Program to Print the Factorial number using For loop

```
num = int(input("Enter a number: "))
factorial = 1

for i in range(1, num + 1):
    factorial *= i

print(factorial)
```

#CODE NO.5.Python Program to Make a Simple Calculator using user defined module

```
def add(x, y): return x + y
def subtract(x, y): return x - y
def multiply(x, y): return x * y
def divide(x, y): return x / y if y != 0 else "Error! Division by zero."
```

#CODE NO.6.Write simple Python program using Arithmetic Operators and logical operators

```
# Input two numbers
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

# Arithmetic operations
sum = a + b
diff = a - b
prod = a * b
div = a / b if b != 0 else "undefined"

# Logical operation (example: check if both conditions are True)
is_greater = (a > b) and (sum > 10)

# Display results
print(f"Sum: {sum}, Difference: {diff}, Product: {prod}, Division: {div}")
print("Both conditions are True:", is_greater)
```

#CODE NO.7.Write simple Python program using Relational Operators and Bitwise Operators using if elif else

```
# Input two numbers
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

# Relational operators
if a > b:
    print("a is greater than b")
elif a < b:
    print("a is less than b")
```

```
else:
    print("a is equal to b")

# Bitwise operators
bitwise_and = a & b
bitwise_or = a | b

print(f"Bitwise AND: {bitwise_and}, Bitwise OR: {bitwise_or}")
```

CODE NO.8.Create a simple program to demonstrate use of for loop in Python (e.g : various pattern building)

```
# Simple pattern using for loop
# Pattern: Right-angled triangle of stars
n = int(input("Enter the number of rows: "))

for i in range(1, n+1):
    print("*" * i)
```

CODE NO.9.Write python program to perform following operations on Lists:

a) Create list

b) Update list (Add, Remove)

c) Delete list

```
# a) Create a list
my_list = [10, 20, 30, 40, 50]
print("Original list:", my_list)
```

```
# b) Update list (Add, Remove)
# Add an element to the list
my_list.append(60)
print("After adding 60:", my_list)

# Remove an element from the list
my_list.remove(20)
print("After removing 20:", my_list)

# c) Delete the entire list
del my_list
print("List deleted")
```

```
# CODE NO. 10.Create a tuple and perform different operation on it
a)minimum,maximum,sort,reverse,delete,
# Create a tuple
my_tuple = (10, 20, 30, 40, 50)

# a) Find minimum and maximum values
min_value = min(my_tuple)
max_value = max(my_tuple)
print(f"Minimum value: {min_value}")
print(f"Maximum value: {max_value}")

# b) Sorting the tuple (Tuples are immutable, so we need to convert to list for sorting)
sorted_tuple = tuple(sorted(my_tuple))
print(f"Sorted tuple: {sorted_tuple}")
```

```
# c) Reversing the tuple (Using slicing)
reversed_tuple = my_tuple[::-1]
print(f"Reversed tuple: {reversed_tuple}")

# d) Deleting the tuple
del my_tuple
print("Tuple deleted")
```

#CODE NO .11.Write python program to perform following operations on Tuples:

a) Create Tuple

b) Access Tuple

c) Update Tuple

d) Delete Tuple

```
# a) Create Tuple
my_tuple = (10, 20, 30, 40, 50)
print("Original Tuple:", my_tuple)

# b) Access Tuple (Access elements by index)
print("Element at index 2:", my_tuple[2])  # Accessing 30

# c) Update Tuple
# Tuples are immutable, so we can't update them directly.
# But we can convert to a list, update it, and then convert back to a tuple.

temp_list = list(my_tuple)  # Convert tuple to list
temp_list[1] = 25  # Update second element (index 1)
my_tuple = tuple(temp_list)  # Convert list back to tuple
```

```
print("Updated Tuple:", my_tuple)

# d) Delete Tuple
del my_tuple
print("Tuple deleted.")
```

#CODE NO.12.Write a python program create a set add member in set and remove form set.

```
# a) Create a set
my_set = {10, 20, 30, 40}
print("Original Set:", my_set)

# b) Add a member to the set
my_set.add(50)
print("After adding 50:", my_set)

# c) Remove a member from the set
my_set.remove(20)
print("After removing 20:", my_set)

# Note: If you try to remove an element that is not in the set, it will raise a KeyError.
# You can use discard() to avoid an error if the element is not found.
```

#CODE NO. 13.Write the a python program to perform operation

intersection of set ,Union of set , set diff , symmetric diff ,clear

```
# Create two sets
```

```
set1 = {10, 20, 30, 40}
set2 = {30, 40, 50, 60}

# 1. Intersection of sets (common elements)
intersection = set1 & set2
print("Intersection of set1 and set2:", intersection)

# 2. Union of sets (all unique elements)
union = set1 | set2
print("Union of set1 and set2:", union)

# 3. Set difference (elements in set1 but not in set2)
set_diff = set1 - set2
print("Difference of set1 and set2 (set1 - set2):", set_diff)

# 4. Symmetric difference (elements in either set but not in both)
sym_diff = set1 ^ set2
print("Symmetric difference of set1 and set2:", sym_diff)

# 5. Clear the set (removes all elements from set1)
set1.clear()
print("Set1 after clearing:", set1)
```

#CODE NO. 14.Write python program to perform following operations on Dictionaries:

a) Create Dictionary

b) Update Dictionary

c) Delete Dictionary

d) Print only Keys of the dictionary

```
# a) Create Dictionary
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}
print("Original Dictionary:", my_dict)

# b) Update Dictionary
my_dict['age'] = 26  # Update value of 'age'
my_dict['job'] = 'Engineer'  # Add new key-value pair
print("Updated Dictionary:", my_dict)

# c) Delete Dictionary
del my_dict['city']  # Remove key 'city'
print("Dictionary after deletion:", my_dict)

# d) Print only Keys
print("Keys of the Dictionary:", my_dict.keys())
```

```
#CODE NO. 15.Write Python program to demonstrate any four math built- in functions
import math

# 1. math.sqrt() - Returns the square root of a number
number = 16
print(f"Square root of {number}: {math.sqrt(number)}")

# 2. math.pow() - Returns x raised to the power of y
x, y = 2, 3
print(f"{x} raised to the power of {y}: {math.pow(x, y)}")

# 3. math.factorial() - Returns the factorial of a number
```

```
num = 5
print(f"Factorial of {num}: {math.factorial(num)}")

# 4. math.ceil() - Returns the smallest integer greater than or equal to a given number
decimal_number = 3.7
print(f"Ceiling of {decimal_number}: {math.ceil(decimal_number)}")
```

#CODE NO. 16.Write Python program to create user defined package with two module.

```
# main_program.py

# Import the modules from the package
from my_package.module1 import greet
from my_package.module2 import add

# Use functions from the modules
name = "Alice"
greeting = greet(name)
print(greeting)  # Output: Hello, Alice!

result = add(5, 3)
print(f"Sum: {result}")  # Output: Sum: 8
```

#CODE NO.17.Develop user defined Python function for given problem:

a) Function with minimum 2 arguments

b) Function returning values

```
# Function with minimum 2 arguments and returning values
def calculate_area(length, width):
    """Calculate the area of a rectangle."""
    area = length * width
    return area

# Example usage
length = 10
width = 5
area = calculate_area(length, width)

print(f"The area of the rectangle with length {length} and width {width} is: {area}")
```

#CODE NO.18.Develop Python program to demonstrate use of NumPy packages for creating , accessing and

performing different array operations.

```
import numpy as np

# Create arrays
arr1 = np.array([1, 2, 3, 4, 5])  # 1D array
arr2 = np.array([[1, 2], [3, 4], [5, 6]])  # 2D array

# Access elements
print("First element of arr1:", arr1[0])
print("Element at arr2[1, 1]:", arr2[1, 1])

# Array Operations
print("arr1 + 5:", arr1 + 5)
```

```
print("arr1 * 2:", arr1 * 2)

# Array functions
print("Sum of arr1:", np.sum(arr1))
print("Mean of arr1:", np.mean(arr1))

# Reshape arr1 to 2D (5x1)
reshaped_arr = arr1.reshape(5, 1)
print("Reshaped arr1:", reshaped_arr)

# Slicing arr1
print("First 3 elements of arr1:", arr1[:3])
```

```
#CODE NO. 19.Write a program in Python to demonstrate Single inheritance .
# Parent class
class Animal:
    def __init__(self, name):
        self.name = name  # Attribute of the parent class

    def speak(self):
        print(f"{self.name} makes a sound.")  # Method of the parent class

# Child class inheriting from Animal class
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)  # Calling the constructor of the parent class
        self.breed = breed  # Attribute of the child class
```

```
    def speak(self):
        print(f"{self.name} barks.")  # Overriding the method in the child class

# Creating an object of the Dog class
dog = Dog("Buddy", "Golden Retriever")

# Accessing attributes and methods
print(f"Dog's Name: {dog.name}")
print(f"Dog's Breed: {dog.breed}")
dog.speak()  # This will call the overridden method in the Dog class




# CODE NO.20.Write a program in Python to demonstrate Multiple inheritance .
# Parent class 1
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound.")

# Parent class 2
class Pet:
    def __init__(self, owner):
        self.owner = owner

    def play(self):
        print(f"{self.owner}'s pet is playing.")
```

```
# Child class inheriting from both Animal and Pet
class Dog(Animal, Pet):
    def __init__(self, name, owner, breed):
        Animal.__init__(self, name)
        Pet.__init__(self, owner)
        self.breed = breed

    def speak(self):
        print(f"{self.name} barks.")

# Creating a Dog object
dog = Dog("Buddy", "Alice", "Golden Retriever")
print(f"Dog's Name: {dog.name}, Owner: {dog.owner}, Breed: {dog.breed}")
dog.speak()
dog.play()
```

```
#CODE NO. 21.Write a program in Python to demonstrate Multilevel inheritance .
# Base class
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        print(f"{self.name} makes a sound.")

# Intermediate class
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
```

```
        self.breed = breed
    def speak(self):
        print(f"{self.name} barks.")

# Derived class
class Puppy(Dog):
    def __init__(self, name, breed, age):
        super().__init__(name, breed)
        self.age = age
    def speak(self):
        print(f"{self.name}, the {self.breed} puppy, barks excitedly!")

# Creating an object of Puppy class
puppy = Puppy("Buddy", "Golden Retriever", 1)
print(f"Puppy's Name: {puppy.name}, Breed: {puppy.breed}, Age: {puppy.age}")
puppy.speak()



#CODE NO.22.Write a program in Python to demonstrate Hierarchical inheritance .
# Parent class
class Animal:
    def __init__(self, name):
        self.name = name  # Attribute of the parent class

    def speak(self):
        print(f"{self.name} makes a sound.")  # Method of the parent class

# Child class 1
class Dog(Animal):
```

```
    def speak(self):
        print(f"{self.name} barks.")  # Overriding the method in the Dog class

# Child class 2
class Cat(Animal):
    def speak(self):
        print(f"{self.name} meows.")  # Overriding the method in the Cat class

# Creating objects of the child classes
dog = Dog("Buddy")
cat = Cat("Whiskers")

# Accessing methods of the parent and child classes
dog.speak()  # Calls the speak method in Dog class
cat.speak()  # Calls the speak method in Cat class
```

```
# CODE NO.23.Write a program in Python to handling array to demonstrate following operations:
a) Array declaration

b) Insertion
c) Display of array
import array  # Import the array module

# a) Array declaration
arr = array.array('i', [])  # Create an empty array of integers

# b) Insertion
arr.append(10)  # Insert 10 into the array
```

```
arr.append(20)  # Insert 20 into the array
arr.append(30)  # Insert 30 into the array

# c) Display of array
print("Elements in the array:")
for element in arr:
    print(element)
```

```
#CODE NO.24.Write a program in Python for Creation and Display of Single linked list
    class Node:
def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node
```

```
    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Using the Linked List
ll = LinkedList()
ll.append(10)
ll.append(20)
ll.append(30)
ll.display()
```

```
#CODE NO. 25.Write a program in Python for insertion sort
def insertion_sort(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1

        # Move elements of arr[0..i-1] greater than key one position ahead
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key

# Input array
```

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

# Perform insertion sort
insertion_sort(arr)

# Output sorted array
print("Sorted Array:", arr)
```

```
#CODE NO. 26.Write a program in Python to create and display various operations on queues.
import queue

q = queue.Queue(maxsize=5)

while True:
    print("\n1. Enqueue\n2. Dequeue\n3. Peek\n4. Display\n5. Exit")
    choice = input("Choose an option: ")

    if choice == '1':
        if q.full():
            print("Queue is full!")
        else:
            q.put(input("Enter element: "))
    elif choice == '2':
        print(f"Dequeued: {q.get()}" if not q.empty() else "Queue is empty!")
    elif choice == '3':
        print(f"Front: {q.queue[0]}" if not q.empty() else "Queue is empty!")
    elif choice == '4':
```

```
        print(f"Queue: {list(q.queue)}")
    elif choice == '5':
        break
    else:
        print("Invalid choice!")
```

#CODE NO.27.Write a program in Python to evaluate infix,postfix and prefix expression

```
def evaluate_postfix(expr):
    stack = []
    for ch in expr:
        if ch.isdigit():
            stack.append(int(ch))
        else:
            b, a = stack.pop(), stack.pop()
            stack.append(eval(f"{a}{ch}{b}"))
    return stack[0]

def evaluate_prefix(expr):
    stack = []
    for ch in reversed(expr):
        if ch.isdigit():
            stack.append(int(ch))
        else:
            a, b = stack.pop(), stack.pop()
            stack.append(eval(f"{a}{ch}{b}"))
    return stack[0]

def evaluate_infix(expr):
```

```
    return eval(expr)

# Main program
expr_type = input("Enter type (postfix, prefix, infix): ").strip().lower()
expr = input("Enter expression: ")

if expr_type == "postfix":
    print("Result:", evaluate_postfix(expr))
elif expr_type == "prefix":
    print("Result:", evaluate_prefix(expr))
elif expr_type == "infix":
    print("Result:", evaluate_infix(expr))
else:
    print("Invalid type!")
```

```
#CODE NO.28.Write a program in Python for Bubble sort
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:  # If no two elements were swapped, the array is sorted
            break

# Example usage
```

```
if __name__ == "__main__":
    arr = [64, 34, 25, 12, 22, 11, 90]
    print("Original array:", arr)
    bubble_sort(arr)
    print("Sorted array:", arr)
```

#CODE NO. 29.Write a program in Python for binary search

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

# Example usage
arr = [2, 3, 4, 10, 40]
target = 10
print("Index:", binary_search(arr, target))
```

#CODE NO. 30.Write a program in Python for Linear search

```
def linear_search(arr, target):
    for i in range(len(arr)):
        if arr[i] == target:
            return i  # Target found, return index
    return -1  # Target not found

# Example usage
arr = [5, 3, 8, 6, 2]
target = 8
result = linear_search(arr, target)

if result != -1:
    print(f"Element {target} found at index {result}")
else:
    print(f"Element {target} not found")
```