1. **Define different data types in Python**

 an\_integer =35

print("Integer:", an\_integer)

**output:** 35

 a\_float = 7.14

print("Float:", a\_float)

**output:** 7.14

 a\_string = "Hello"

print("String:", a\_string)

**output:** Hello

 a\_boolean = false

print("Boolean:", a\_boolean)

**output:** false

 a\_list = [1, 2, 3, 5, 6]

print("List:", a\_list)

**output:** [1, 2, 3, 4, 5, 6]

 a\_tuple = (1, 2, 3, 4, 5)

print("Tuple:", a\_tuple)

**output:** (1, 2, 3, 4, 5)

 a\_set = {1, 2, 3, 4, 5, 6, 7}

print("Set:", a\_set)

**output:** {1, 2, 3, 4, 5, 6, 7}

**2.program to makes use of  following operators**

 Arithmetic Operations

a =35

b =45

addition = a + b

subtraction = a - b

multiplication = a \* b

division = a / b

modulus = a % b

print("Arithmetic Operations:")

print("Addition:", addition)

print("Subtraction:", subtraction)

print("Multiplication:", multiplication)

print("Division:", division)

print("Modulus:", modulus)

print("\n")

**output:**

80

-10

1575

0.7777777777777778

35

**Logical Operations**

x = True

y = False

logical\_and = x and y

logical\_or = x or y

logical\_not = not x

print("Logical Operations:")

print("AND:", logical\_and)

print("OR:", logical\_or)

print("NOT:", logical\_not)

print("\n")

**output:**

False

True

False

**3. Program to make use of if-else and nested if-else**

def find\_biggest(a, b, c):

    if a >= b and a >= c:

        return a

    elif b >= a and b >= c:

        return b

    else:

        return c

def even\_or\_odd(n):

    if n % 2 == 0:

        return "Even"

    else:

        return "Odd"

def is\_prime(n):

    if n <= 1:

        return False

    elif n == 2:

        return True

    else:

        for i in range(2, int(n\*\*0.5) + 1):

            if n % i == 0:

                return False

        return True

def find\_grade(marks):

    if marks >= 95:

     return "A"

    elif marks >= 85:

        return "B"

    elif marks >= 75:

        return "C"

    elif marks >= 65:

        return "D"

    else:

        return "F"

# Testing the functions

a, b, c = 25, 35, 15

print(f"Largest among {a}, {b}, and {c} is {find\_biggest(a, b, c)}")

n = 18

print(f"{n} is {even\_or\_odd(n)}")

num = 30

print(f"{num} is {'Prime' if is\_prime(num) else 'Not Prime'}")

marks = 95

print(f"Grade for {marks} marks is {find\_grade(marks)}")

**Output:**

Largest among 25, 35, and 15 is 35

18 is Even

30 is Not Prime

Grade for 95 marks is A

**4. Program to make use of a while loop**

# a. Print the series from 1 to n

def print\_series(n):

    i = 1

    while i <= n:

        print(i, end=" ")

        i += 1

    print()

# b. Print the even and odd series

def print\_even\_odd\_series(n):

    i = 1

    even\_numbers, odd\_numbers = [], []

    while i <= n:

        if i % 2 == 0:

            even\_numbers.append(i)

        else:

            odd\_numbers.append(i)

        i += 1

    print(f"Even numbers: {even\_numbers}")

    print(f"Odd numbers: {odd\_numbers}")

# c. Sum of natural numbers

def sum\_natural\_numbers(n):

    total = 0

    i = 1

    while i <= n:

        total += i

        i += 1

    return total

# d. Armstrong number

def is\_armstrong(num):

    sum\_digits = 0

    temp = num

    length = len(str(num))

    while temp > 0:

        digit = temp % 10

        sum\_digits += digit \*\* length

        temp //= 10

    return sum\_digits == num

# e. Palindrome check

def is\_palindrome(num):

    original = str(num)

    reversed\_num = ""

    i = len(original) - 1

    while i >= 0:

        reversed\_num += original[i]

        i -= 1

    return original == reversed\_num

# Testing the functions

n = 10

print\_series(n)

print\_even\_odd\_series(n)

print(f"Sum of first {n} natural numbers: {sum\_natural\_numbers(n)}")

num\_armstrong = 153

print(f"{num\_armstrong} is {'an Armstrong number' if is\_armstrong(num\_armstrong) else 'not an Armstrong number'}")

num\_palindrome = 121

print(f"{num\_palindrome} is {'a Palindrome' if is\_palindrome(num\_palindrome) else 'not a Palindrome'}")

**Output:** 1 2 3 4 5 6 7 8 9 10

Even numbers: [2, 4, 6, 8, 10]

Odd numbers: [1, 3, 5, 7, 9]

Sum of first 10 natural numbers: 55

153 is an Armstrong number

121 is a Palindrome

**6.  program which demonstrates the use of functions**

# a. No values passing and no parameters returned

def greet():

    print("Hello! Welcome to the Python function demonstration.")

# b. Passing values and no parameters returned

def print\_message(message):

    print(f"Message: {message}")

# c. No passing values but return types

def get\_greeting():

    return "Hello! Have a great day!"

# d. Parameter passing and return types

def add\_numbers(a, b):

    return a + b

# Testing the functions

greet()

print\_message("This is an example of passing values.")

greeting = get\_greeting()

print(greeting)

result = add\_numbers(10, 20)

print(f"The sum of 10 and 20 is {result}.")

**Output:** Hello! Welcome to the Python function demonstration.

Message: This is an example of passing values.

Hello! Have a great day!

The sum of 10 and 20 is 30.

**7. Use of classes and objects**

# Creating an initial list

numbers = [1, 2, 3, 4, 11]

# append(): Adds an element at the end

numbers.append(13)

print(f"After append: {numbers}")

# insert(): Inserts an element at a specific position

numbers.insert(2, 6)  # Insert 6 at index 2

print(f"After insert: {numbers}")

# extend(): Extends the list with another list

numbers.extend([12, 13, 14])

print(f"After extend: {numbers}")

# remove(): Removes the first occurrence of a specific element

numbers.remove(4)

print(f"After remove: {numbers}")

# pop(): Removes and returns the last element (or a specific index)

popped\_element = numbers.pop()

print(f"After pop: {numbers}, Popped element: {popped\_element}")

# clear(): Removes all elements from the list

copy\_numbers = numbers.copy()  # Creating a copy before clearing

copy\_numbers.clear()

print(f"After clear: {copy\_numbers}")

# index(): Finds the first occurrence index of an element

index\_of\_11 = numbers.index(11)

print(f"Index of 11: {index\_of\_11}")

# count(): Counts occurrences of a specific element

count\_of\_5 = numbers.count(5)

print(f"Count of 5 in the list: {count\_of\_5}")

# sort(): Sorts the list in ascending order

sorted\_numbers = sorted(numbers)

print(f"Sorted list: {sorted\_numbers}")

# reverse(): Reverses the list order

reversed\_numbers = numbers[::-1]

print(f"Reversed list: {reversed\_numbers}")

# copy(): Creates a duplicate of the list

copied\_numbers = numbers.copy()

print(f"Copied list: {copied\_numbers}")

# len(): Finds the length of the list

length\_of\_list = len(numbers)

print(f"Length of list: {length\_of\_list}")

# max(): Finds the maximum element

max\_value = max(numbers)

print(f"Maximum value in the list: {max\_value}")

# min(): Finds the minimum element

min\_value = min(numbers)

print(f"Minimum value in the list: {min\_value}")

**Output:**

After append: [1, 2, 3, 4, 11, 13]

After insert: [1, 2, 6, 3, 4, 11, 13]

After extend: [1, 2, 6, 3, 4, 11, 13, 12, 13, 14]

After remove: [1, 2, 6, 3, 11, 13, 12, 13, 14]

After pop: [1, 2, 6, 3, 11, 13, 12, 13], Popped element: 14

After clear: []

Index of 11: 4

Count of 5 in the list: 0

Sorted list: [1, 2, 3, 6, 11, 12, 13, 13]

Reversed list: [13, 12, 13, 11, 3, 6, 2, 1]

Copied list: [1, 2, 6, 3, 11, 13, 12, 13]

Length of list: 8

Maximum value in the list: 13

Minimum value in the list: 1

**8. Dictionaries and their methods**

# Define a dictionary with some initial key-value pairs

person = {

'name': 'yash',

'age': 19,

'city': 'hyderabad' }

 # Print the original dictionary

print("Original Dictionary:", person)

**output**: {'name': 'yash', 'age': 19, 'city': 'hyderabad'}

 # Access a value by its key

name = person['name']

print("Name:", name)

**output:** yash

 # Add a new key-value pair to the dictionary

person['occupation'] = 'student'

print("After adding occupation:", person)

**output:** {'name': 'yash', 'age': 19, 'city': 'hyderabad', 'occupation': 'student'}

 # Update the value of an existing key

person['age'] = 21

print("After updating age:", person)

**output**: {'name': 'yash', 'age': 21, 'city': 'hyderabad', 'occupation': 'student'}

 # Remove a key-value pair from the dictionary using pop method

removed\_value = person.pop('city')

print("After removing city:", person)

print("Removed value:", removed\_value)

**output:** {'name': 'yash', 'age': 21, 'occupation': 'student'}

 # Get a list of all values in the dictionary

values = person.values()

print("Values:", values)

**output:** dict\_values(['yash', 21, 'student'])

**9. Program to make use of tuples and their methods**

# Define a tuple with some initial elements

fruits = ('apple', 'banana', 'muskmelon')

 # Print the original tuple

print("Original Tuple:", fruits)

**output:** ('apple', 'banana', 'muskmelon')

 # Access elements in a tuple

second\_fruit = fruits[0]

print("second fruit:", second\_fruit)

**output:** banana

 # Find the index of an item in the tuple

index\_of\_banana = fruits.index('banana')

print("Index of 'banana':", index\_of\_banana)

**output:**1

 # Concatenate two tuples

more\_fruits = ('watermelon', 'pineapple')

combined\_fruits = fruits + more\_fruits

print("Combined Tuple:", combined\_fruits)

**output:**  ('apple', 'banana', 'muskmelon', 'watermelon', 'pineapple')

 # Get the length of the tuple

length\_of\_tuple = len(fruits)

print("Length of the tuple:", length\_of\_tuple)

**output:** 3

**10. Program to work with sets and their operations**

# Creating sets

set1 = {1, 2, 3, 4, 5}

set2 = set([6, 7, 8, 9, 10])

print("Set 1:", set1)

print("Set 2:", set2)

# Adding elements

set1.add(6)

print("After adding 6:", set1)

# Updating set with multiple elements

set1.update([7, 8, 9])

print("After updating with [7, 8, 9]:", set1)

# Removing elements safely

set1.discard(9)  # Won't throw error if element is absent

print("After discarding 9:", set1)

set1.remove(8)  # Throws error if element is absent

print("After removing 8:", set1)

# Popping a random element

popped\_element = set1.pop()

print(f"After popping an element ({popped\_element}):", set1)

# Clearing the set

set3 = set1.copy()

set3.clear()

print("After clearing set3:", set3)

# Set operations

print("Union:", set1.union(set2))

print("Intersection:", set1 & set2)

print("Difference (set1 - set2):", set1 - set2)

print("Symmetric Difference:", set1 ^ set2)

# Subset and Superset checks

print("Is set1 a subset of set2?", set1 <= set2)

print("Is set1 a superset of set2?", set1 >= set2)

print("Are set1 and set2 disjoint?", set1.isdisjoint(set2))

# Copying a set

set4 = set1.copy()

print("Copied set:", set4)

# Length and membership

print("Length of set1:", len(set1))

print("Is 5 in set1?", 5 in set1)

**Output:**

Set 1: {1, 2, 3, 4, 5}

Set 2: {6, 7, 8, 9, 10}

After adding 6: {1, 2, 3, 4, 5, 6}

After updating with [7, 8, 9]: {1, 2, 3, 4, 5, 6, 7, 8, 9}

After discarding 9: {1, 2, 3, 4, 5, 6, 7, 8}

After removing 8: {1, 2, 3, 4, 5, 6, 7}

After popping an element (1): {2, 3, 4, 5, 6, 7}

After clearing set3: set()

Union: {2, 3, 4, 5, 6, 7, 8, 9, 10}

Intersection: {6, 7}

Difference (set1 - set2): {2, 3, 4, 5}

Symmetric Difference: {2, 3, 4, 5, 8, 9, 10}

Is set1 a subset of set2? False

Is set1 a superset of set2? False

Are set1 and set2 disjoint? False

Copied set: {2, 3, 4, 5, 6, 7}

Length of set1: 6

Is 5 in set1? True

**1. Array creation & properties**

import numpy as np

 # Creating a sample array

array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

# Shape of the array

print("Shape:", array.shape)

op:(3, 3)

# Size of the array

print("Size:", array.size)

op:9

# Data type of the elements

print("Data type:", array.dtype)

op:int64

# Number of dimensions

print("Number of dimensions:", array.ndim)

Op:2