1. **Write program to illustrate variables, constants, datatypes and type conversion.**

PI = 3.14159

age = 20 # integer

height = 5.9 # float

name = "Ghanashyam" # string

is\_student = True # boolean

print("Name:", name)

print("Age:", age)

print("Height:", height)

print("Student:", is\_student)

print("Constant PI:", PI)

age\_float = float(age) # int → float

number\_str = "100"

number\_int = int(number\_str) # string → int

print("Age as float:", age\_float)

print("Number string:", number\_str)

print("Number as integer:", number\_int)

1. **Write program to demonstrate different types of operators available in python and perform calculations.**

a = 10

b = 3

print("=== Arithmetic Operators ===")

print("a + b =", a + b) # Addition

print("a - b =", a - b) # Subtraction

print("a \* b =", a \* b) # Multiplication

print("a / b =", a / b) # Division

print("a % b =", a % b) # Modulus (remainder)

print("a // b =", a // b) # Floor Division

print("a \*\* b =", a \*\* b) # Exponent (power)

print("\n=== Comparison Operators ===")

print("a == b:", a == b)

print("a != b:", a != b)

print("a > b:", a > b)

print("a < b:", a < b)

print("a >= b:", a >= b)

print("a <= b:", a <= b)

print("\n=== Logical Operators ===")

x = True

y = False

print("x and y:", x and y)

print("x or y:", x or y)

print("not x:", not x)

print("\n=== Assignment Operators ===")

c = 5

print("c =", c)

c += 2 # c = c + 2

print("c += 2:", c)

c \*= 3 # c = c \* 3

print("c \*= 3:", c)

print("\n=== Bitwise Operators ===")

print("a & b =", a & b) # AND

print("a | b =", a | b) # OR

print("a ^ b =", a ^ b) # XOR

print("~a =", ~a) # NOT

print("a << 1 =", a << 1) # Left shift

print("a >> 1 =", a >> 1) # Right shift

print("\n=== Membership Operators ===")

numbers = [1, 2, 3, 4, 5]

print("3 in numbers:", 3 in numbers)

print("7 not in numbers:", 7 not in numbers)

print("\n=== Identity Operators ===")

p = [1, 2, 3]

q = [1, 2, 3]

r = p

print("p is q:", p is q) # False (different objects)

print("p is r:", p is r) # True (same object)

print("p is not q:", p is not q)

1. **Write program to make use of I/O functions.**

name = input("Enter your name: ")

age = int(input("Enter your age: ")) # input by default string हुन्छ, त्यसलाई int मा convert गरियो

marks = float(input("Enter your marks: "))

# Displaying output

print("\n=== Student Information ===")

print("Name:", name)

print("Age:", age)

print("Marks:", marks)

# Formatting output

print(f"\nHello {name}, you are {age} years old and you scored {marks} marks.")

1. **Write program to apply if, match, break and continue statements for decision making.**

age = int(input("Enter your age: "))

if age >= 18:

print("You are eligible to vote.")

else:

print("You are not eligible to vote.")

# --- MATCH STATEMENT

day = input("\nEnter a day (Mon/Tue/Wed): ")

match day:

case "Mon":

print("Start of the week!")

case "Tue":

print("Second day of the week!")

case "Wed":

print("Mid of the week!")

case \_:

print("Other day")

# --- BREAK AND CONTINUE ---

print("\nNumbers from 1 to 10 (skip 5, stop at 8):")

for i in range(1, 11):

if i == 5:

continue # skip 5

if i == 8:

break # stop loop when 8

print(i)

1. **Write program to utilize different loop statements to solve meaningful problems.**

sum\_numbers = 0

for i in range(1, 11):

sum\_numbers += i

print("Sum of numbers from 1 to 10 =", sum\_numbers)

# --- WHILE LOOP Example: Factorial of a number ---

n = int(input("\nEnter a number to find factorial: "))

fact = 1

i = 1

while i <= n:

fact \*= i

i += 1

print(f"Factorial of {n} = {fact}")

# --- NESTED LOOP Example: Multiplication table ---

print("\nMultiplication Table (1 to 5):")

for i in range(1, 6): # row

for j in range(1, 6): # column

print(i \* j, end="\t")

print()

1. **Write program to demonstrate input validation using loop.**

while True:

age = input("Enter your age (1-120): ")

# check if input is digit

if age.isdigit():

age = int(age)

if 1 <= age <= 120:

print("Valid age entered:", age)

break # exit loop if valid

else:

print("Age must be between 1 and 120. Try again.")

else:

print("Invalid input! Please enter a number.")

**7. Write program to create different patterns using nested loop**.

**(a) Right Triangle Star Pattern**

# Right Triangle Star Pattern

rows = 5

for i in range(1, rows + 1):

for j in range(i):

print("\*", end=" ")

print()

**(b) Inverted Triangle Star Pattern**

# Inverted Triangle Star Pattern

rows = 5

for i in range(rows, 0, -1):

for j in range(i):

print("\*", end=" ")

print()

**(c) Pyramid Star Pattern**

# Pyramid Star Pattern

rows = 5

for i in range(1, rows + 1):

print(" " \* (rows - i), end="") # spaces

print("\* " \* i)

1. **Write program to make use of infinite loop.**

while True: # infinite loop

choice = input("Enter 'exit' to stop, or any text to repeat: ")

if choice.lower() == "exit":

print("Loop stopped by user.")

break # exit infinite loop

else:

print("You entered:", choice)

1. **Write program to create list, add elements in list, remove elements from list and display list items.**

fruits = []

# Add elements

fruits.append("Apple")

fruits.append("Banana")

fruits.append("Mango")

print("After adding elements:", fruits)

# Insert element at specific position

fruits.insert(1, "Orange")

print("After inserting Orange at index 1:", fruits)

# Remove element by value

fruits.remove("Banana")

print("After removing Banana:", fruits)

# Remove element by index

fruits.pop(0) # removes first element

print("After removing first element:", fruits)

# Display list using loop

print("\nDisplaying list items:")

for item in fruits:

print(item)

1. **Write program to make use of list slicing concept to display elements of list.**

numbers = [10, 20, 30, 40, 50, 60, 70, 80]

print("Original List:", numbers)

# Slice from index 2 to 5

slice1 = numbers[2:6] # 2,3,4,5 index elements

print("Slice from index 2 to 5:", slice1)

# Slice from start to index 4

slice2 = numbers[:5] # 0 to 4

print("Slice from start to index 4:", slice2)

# Slice from index 3 to end

slice3 = numbers[3:] # 3 to last

print("Slice from index 3 to end:", slice3)

# Slice with step (every 2nd element)

slice4 = numbers[::2]

print("Every 2nd element in list:", slice4)

# Reverse the list using slicing

reverse\_list = numbers[::-1]

print("Reversed List:", reverse\_list)

1. **Write program to elaborate different list methods.**

fruits = ["Apple", "Banana", "Mango"]

print("Original List:", fruits)

# append() - add element at end

fruits.append("Orange")

print("After append Orange:", fruits)

# insert() - add element at specific position

fruits.insert(1, "Grapes")

print("After insert Grapes at index 1:", fruits)

# extend() - merge another list

fruits.extend(["Kiwi", "Pineapple"])

print("After extend with ['Kiwi','Pineapple']:", fruits)

# remove() - remove element by value

fruits.remove("Banana")

print("After removing Banana:", fruits)

# pop() - remove last element

last\_item = fruits.pop()

print("After pop():", fruits, "| Popped item:", last\_item)

# index() - find index of element

index\_mango = fruits.index("Mango")

print("Index of Mango:", index\_mango)

# count() - count occurrences

fruits.append("Apple")

apple\_count = fruits.count("Apple")

print("After adding another Apple:", fruits, "| Apple count:", apple\_count)

# sort() - sort list

fruits.sort()

print("After sort():", fruits)

# reverse() - reverse list

fruits.reverse()

print("After reverse():", fruits)

# copy() - copy list

fruits\_copy = fruits.copy()

print("Copy of list:", fruits\_copy)

# clear() - empty the list

fruits.clear()

print("After clear():", fruits)

1. **Write program to apply list comprehension.**
2. Create a list of squares from 1 to 10

squares = [x\*\*2 for x in range(1, 11)]

print("Squares from 1 to 10:", squares)

1. Create a list of even numbers from 1 to 20

evens = [x for x in range(1, 21) if x % 2 == 0]

print("Even numbers from 1 to 20:", evens)

1. Create a new list by converting strings to uppercase

fruits = ["apple", "banana", "mango"]

upper\_fruits = [fruit.upper() for fruit in fruits]

print("Uppercase fruits:", upper\_fruits)

1. Create a list of tuples (number, square)

num\_square = [(x, x\*\*2) for x in range(1, 6)]

print("Number and its square:", num\_square)

1. **Write program to illustrate two-dimensional list.**

matrix = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

# Display the 2D list

print("2D List (Matrix):")

for row in matrix:

print(row)

# Accessing individual elements

print("\nAccessing elements:")

print("Element at row 1, column 2:", matrix[0][1]) # 2

print("Element at row 3, column 3:", matrix[2][2]) # 9

# Modifying element

matrix[1][1] = 50

print("\nAfter modifying element at row 2, column 2:")

for row in matrix:

print(row)

# Sum of all elements

total = 0

for row in matrix:

for item in row:

total += item

print("\nSum of all elements:", total)

1. **Write program to create tuple, add elements in tuple, remove elements from tuple and display tuple items.**

fruits = ("Apple", "Banana", "Mango")

print("Original Tuple:", fruits)

# --- Add element to tuple ---

# Convert to list

temp\_list = list(fruits)

temp\_list.append("Orange") # add element

fruits = tuple(temp\_list) # convert back to tuple

print("After adding Orange:", fruits)

# --- Remove element from tuple ---

temp\_list = list(fruits)

temp\_list.remove("Banana") # remove element

fruits = tuple(temp\_list)

print("After removing Banana:", fruits)

# --- Display tuple items using loop ---

print("\nTuple items:")

for item in fruits:

print(item)

1. **Write program to create dictionary, add elements in dictionary, remove elements from dictionary and display dictionary items.**

student = {

"name": "Ghanashyam",

"age": 30,

"marks": 85

}

print("Original Dictionary:", student)

# --- Add elements ---

student["grade"] = "A"

print("After adding grade:", student)

# --- Remove elements ---

student.pop("age") # remove by key

print("After removing age:", student)

# OR using del

# del student["marks"]

# --- Display dictionary items ---

print("\nDictionary Items:")

for key, value in student.items():

print(key, ":", value)

1. **Write program to create set, add elements in set, remove elements from set and display set items.**

# Create a set

fruits = {"Apple", "Banana", "Mango"}

print("Original Set:", fruits)

# --- Add elements ---

fruits.add("Orange")

print("After adding Orange:", fruits)

# --- Remove elements ---

fruits.remove("Banana") # remove Banana

print("After removing Banana:", fruits)

# discard() example (does not give error if element not found)

fruits.discard("Pineapple") # no error

print("After discarding Pineapple (not in set):", fruits)

# --- Display set items using loop ---

print("\nSet Items:")

for item in fruits:

print(item)

1. **Write program to perform set operations.**

# Program to perform set operations

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

setB = {4, 5, 6, 7, 8}

print("Set A:", setA)

print("Set B:", setB)

# Union

union\_set = setA | setB

print("Union of A and B:", union\_set)

# Intersection

intersection\_set = setA & setB

print("Intersection of A and B:", intersection\_set)

# Difference

diff\_set = setA - setB

print("Difference A - B:", diff\_set)

# Symmetric Difference

sym\_diff\_set = setA ^ setB

print("Symmetric Difference of A and B:", sym\_diff\_set)

# Subset check

print("Is A subset of B?", setA.issubset(setB))

# Superset check

print("Is A superset of B?", setA.issuperset(setB))

1. **Write program to make use of string manipulation methods and also perform different string operations.**

# Program to demonstrate string manipulation and operations

text = " hello Python World! "

print("Original String:", text)

# Length

print("Length of string:", len(text))

# Strip spaces

text = text.strip()

print("After strip():", text)

# Uppercase

print("Uppercase:", text.upper())

# Lowercase

print("Lowercase:", text.lower())

# Capitalize first letter

print("Capitalize:", text.capitalize())

# Title case

print("Title Case:", text.title())

# Replace substring

new\_text = text.replace("Python", "Programming")

print("After replace:", new\_text)

# Split string into list

words = text.split()

print("After split():", words)

# Join list into string

joined\_text = "-".join(words)

print("After join():", joined\_text)

# Find index of substring

index\_world = text.find("World")

print("Index of 'World':", index\_world)

# Check if substring exists

print("'Python' in text?", "Python" in text)

# String concatenation

concat\_text = text + " Enjoy coding!"

print("After concatenation:", concat\_text)

# String repetition

repeat\_text = "Hi! " \* 3

print("After repetition:", repeat\_text)

1. **Write program to elaborate object oriented concept with simple examples.**

# Example of OOP concepts

# --- Class and Object ---

class Student:

def \_\_init\_\_(self, name, age): # constructor

self.name = name

self.age = age

def display(self): # method

print(f"Name: {self.name}, Age: {self.age}")

# Create objects

student1 = Student("Ghanashyam", 20)

student2 = Student("Sita", 19)

# Display object information

student1.display()

student2.display()

# --- Inheritance ---

class Person:

def \_\_init\_\_(self, name):

self.name = name

def greet(self):

print(f"Hello, my name is {self.name}")

class Teacher(Person): # Teacher inherits Person

def teach(self):

print(f"{self.name} is teaching Python")

teacher1 = Teacher("Mr. Sharma")

teacher1.greet()

teacher1.teach()

# --- Polymorphism ---

class Calculator:

def add(self, a, b):

return a + b

class AdvancedCalculator(Calculator):

def add(self, a, b, c=0): # same method name, different behavior

return a + b + c

calc1 = Calculator()

calc2 = AdvancedCalculator()

print("Simple Add:", calc1.add(5, 10))

print("Advanced Add:", calc2.add(5, 10, 15))

1. **Write program to make use of initmethod to initialize objects.**

# Program to demonstrate \_\_init\_\_ method

class Student:

def \_\_init\_\_(self, name, age, marks):

self.name = name

self.age = age

self.marks = marks

def display(self):

print(f"Name: {self.name}")

print(f"Age: {self.age}")

print(f"Marks: {self.marks}")

# Create objects with initialization

student1 = Student("Ghanashyam", 20, 85)

student2 = Student("Sita", 19, 90)

# Display object information

print("Student 1 Details:")

student1.display()

print("\nStudent 2 Details:")

student2.display()

1. **Write program to apply different types of inheritance.**

# --- Single Inheritance ---

print("----- Single Inheritance -----")

class Person:

def greet(self):

print("Hello! I am a Person.")

class Student(Person): # Student inherits Person

def study(self):

print("I am studying.")

student1 = Student()

student1.greet()

student1.study()

# --- Multilevel Inheritance ---

print("\n----- Multilevel Inheritance -----")

class Animal:

def eat(self):

print("Animal is eating.")

class Mammal(Animal):

def walk(self):

print("Mammal is walking.")

class Dog(Mammal):

def bark(self):

print("Dog is barking.")

dog1 = Dog()

dog1.eat()

dog1.walk()

dog1.bark()

# --- Multiple Inheritance ----

print("\n----- Multiple Inheritance -----")

class Father:

def skills\_father(self):

print("Father has programming skills.")

class Mother:

def skills\_mother(self):

print("Mother has painting skills.")

class Child(Father, Mother):

def skills\_child(self):

print("Child has both skills.")

child1 = Child()

child1.skills\_father()

child1.skills\_mother()

child1.skills\_child()

# --- Hierarchical Inheritance ----

print("\n----- Hierarchical Inheritance -----")

class Vehicle:

def type(self):

print("This is a vehicle.")

class Car(Vehicle):

def wheels(self):

print("Car has 4 wheels.")

class Bike(Vehicle):

def wheels(self):

print("Bike has 2 wheels.")

car1 = Car()

bike1 = Bike()

car1.type()

car1.wheels()

bike1.type()

bike1.wheels()

1. **Write program to elaborate polymorphism and data hiding concept.**

# --- Polymorphism Example ---

# Method Overriding (Run-time Polymorphism)

class Animal:

def sound(self):

print("Animal makes a sound")

class Dog(Animal):

def sound(self): # override

print("Dog barks")

class Cat(Animal):

def sound(self):

print("Cat meows")

# Objects

a = Animal()

d = Dog()

c = Cat()

a.sound()

d.sound()

c.sound()

# Method Overloading (Polymorphism via default arguments)

class Calculator:

def add(self, a, b, c=0):

return a + b + c

calc = Calculator()

print("Add 2 numbers:", calc.add(5, 10))

print("Add 3 numbers:", calc.add(5, 10, 15))

# --- Data Hiding / Encapsulation Example ---

class BankAccount:

def \_\_init\_\_(self, name, balance):

self.name = name

self.\_\_balance = balance # private attribute

# Getter method

def get\_balance(self):

return self.\_\_balance

# Setter method

def deposit(self, amount):

if amount > 0:

self.\_\_balance += amount

print(f"Deposited {amount}, New Balance: {self.\_\_balance}")

else:

print("Invalid deposit amount")

def withdraw(self, amount):

if 0 < amount <= self.\_\_balance:

self.\_\_balance -= amount

print(f"Withdrawn {amount}, New Balance: {self.\_\_balance}")

else:

print("Insufficient balance or invalid amount")

# Create object

account = BankAccount("Ghanashyam", 1000)

print("\nAccount Balance (using getter):", account.get\_balance())

account.deposit(500)

account.withdraw(300)

# Trying to access private attribute directly

# print(account.\_\_balance) # This will give error

1. **Write program to divide work in functions.**

# Program to divide work in functions

# Function to add two numbers

def add(a, b):

return a + b

# Function to subtract two numbers

def subtract(a, b):

return a - b

# Function to multiply two numbers

def multiply(a, b):

return a \* b

# Function to divide two numbers

def divide(a, b):

if b != 0:

return a / b

else:

return "Cannot divide by zero"

# Main program

num1 = float(input("Enter first number: "))

num2 = float(input("Enter second number: "))

print("Addition:", add(num1, num2))

print("Subtraction:", subtract(num1, num2))

print("Multiplication:", multiply(num1, num2))

print("Division:", divide(num1, num2))

1. **Write different variety of functions: function with arguments, value returning function, function without arguments.**

# --- 1. Function without arguments & without return ---

def greet():

print("Hello! Welcome to Python functions.")

# --- 2. Function with arguments & without return ---

def display\_name(name):

print(f"Hello, {name}! Have a nice day.")

# --- 3. Function with arguments & with return ---

def add\_numbers(a, b):

return a + b

# --- Main Program ---

greet() # call function without arguments

display\_name("Ghanashyam") # call function with argument

result = add\_numbers(10, 20) # call function with return value

print("Sum of numbers:", result)

1. **Write program to store output in file.**

# Program to store output in a file

# Data to write

output\_data = "Hello! This is Python file handling example.\n"

output\_data += "We are storing this output in a file."

# Open file in write mode

file = open("output.txt", "w") # 'output.txt' name file create/overwrite

file.write(output\_data)

file.close() # Close the file

print("Output successfully written to 'output.txt'")

# Read the file to verify

file = open("output.txt", "r")

content = file.read()

print("\nContent of file:")

print(content)

file.close()

1. **Write program to read input from file.**

# Program to read input from file

# Assume file 'input.txt' exists with some content

# If not, let's create one first

with open("input.txt", "w") as f:

f.write("Hello Python!\n")

f.write("This is a file reading example.\n")

f.write("We are learning Python file handling.\n")

# Open file in read mode

file = open("input.txt", "r")

# Read entire content

content = file.read()

print("Reading entire file content:\n", content)

# Move cursor to beginning

file.seek(0)

# Read file line by line

print("Reading file line by line:")

for line in file:

print(line.strip()) # strip removes \n

file.close()

1. **Write program to handle different types of exception.**

# Program to handle different types of exceptions

# --- ZeroDivisionError ---

try:

a = 10

b = 0

result = a / b

except ZeroDivisionError:

print("Error: Cannot divide by zero!")

# --- ValueError ---

try:

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

print("You entered:", num)

except ValueError:

print("Error: Invalid input! Please enter an integer.")

# --- IndexError ---

try:

lst = [1, 2, 3]

print(lst[5])

except IndexError:

print("Error: Index out of range!")

# --- FileNotFoundError ---

try:

file = open("nonexistent.txt", "r")

except FileNotFoundError:

print("Error: File does not exist!")

# --- Using else and finally ---

try:

x = 5

y = 2

print("Division:", x / y)

except ZeroDivisionError:

print("Division by zero error")

else:

print("No exception occurred")

finally:

print("This block always executes")