DR. B.R. AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY, JALANDHAR

AI – 523

Programming with Python Laboratory

Practical File

Submitted To:

Dr. Diksha Kumari

And

Dr. Ruchika Arora

Submitted By:

Atul Dhiman

25901325

A.I. (M.Tech)

INDEX

1. Installation and Basic Programs (Variables and Data Types, Arithmetic, Comparison, Assignment, Logical Operators)

a. Write a program to display data of different types using variables and literal constants.

Objective: To write a Python program that demonstrates different data types using variables and literal constants

Code:

```python
# Variables with different data types
integer_var = 10
float_var = 25.5
string_var = "Python Programming"
boolean_var = True

# Literal constants
print("Integer value:", integer_var)
print("Float value:", float_var)
print("String value:", string_var)
print("Boolean value:", boolean_var)

print("Integer literal:", 100)
print("Float literal:", 3.14)
print("String literal:", "Hello World")
print("Boolean literal:", False)
```

```
Integer value: 10
Float value: 25.5
String value: Python Programming
Boolean value: True
Integer literal: 100
Float literal: 3.14
String literal: Hello World
Boolean literal: False
```

Code Explanation

- Variables are created to store integer, float, string, and boolean values.
- Literal constants are values written directly in the program.
- The print() function displays both variable values and literals.

b. Program to show the application of the += operator on str

Objective: To demonstrate the use of the += assignment operator with string variables.

Code:

```python
# String concatenation using += operator
text = "Hello"
text += " World"

print(text)
```

```
Hello World
```

Code Explanation

- The variable text initially stores "Hello".
- The += operator appends " World" to the existing string.
- This performs string concatenation.

c. Program that performs addition and multiplication on string variables

Objective: To demonstrate addition and multiplication operations on string variables in Python.

Code:

```
# String addition (concatenation)
str1 = "Data"
str2 = "Science"
result_add = str1 + str2

# String multiplication (repetition)
result_mul = str1 * 3

print("String Addition:", result_add)
print("String Multiplication:", result_mul)
```

```
String Addition: DataScience
String Multiplication: DataDataData
```

Code Explanation

- The + operator concatenates two strings.
- The * operator repeats a string a specified number of times.
- These operations are valid only when operands are strings and integers (for *).

2. Operators based programs

a. Write a program to calculate area of a triangle using Heron's Formula.

Objective: To write a Python program to calculate the area of a triangle using Heron's Formula based on the lengths of its three sides.

Code:

```
import math

# Input sides of the triangle
a = float(input("Enter side a: "))
b = float(input("Enter side b: "))
c = float(input("Enter side c: "))

# Calculate semi-perimeter
s = (a + b + c) / 2

# Calculate area using Heron's formula
area = math.sqrt(s * (s - a) * (s - b) * (s - c))

print("Area of the triangle is:", area)
```

```
Area of the triangle is: 194.97692171126306
```

Code Explanation

- The math module is used to calculate the square root.
- The three sides of the triangle are taken as input.
- Semi-perimeter s is calculated using arithmetic operators.
- Heron's formula is applied to compute the area.

b. Program to calculate the total amount of money in the piggy bank

Objective: To write a Python program to calculate the total amount of money in a piggy bank given the number of coins of Rs.10, Rs.5, Rs.2, and Rs.1.

Code:

```
# Input number of coins
rs10 = int(input("Enter number of Rs.10 coins: "))
rs5  = int(input("Enter number of Rs.5 coins: "))
rs2  = int(input("Enter number of Rs.2 coins: "))
rs1  = int(input("Enter number of Rs.1 coins: "))

# Calculate total amount
total_amount = (rs10 * 10) + (rs5 * 5) + (rs2 * 2) + (rs1 * 1)

print("Total amount in piggy bank = Rs.", total_amount)
```

```
Total amount in piggy bank = Rs. 228
```

Code Explanation:

- The number of coins for each denomination is taken as input.
- Multiplication (*) is used to calculate the value of each coin type.
- Addition (+) operator sums all values to get the total amount.

3. String Manipulation, Number System and Conversions

a. Program to convert a decimal number into binary, octal, and hexadecimal

Objective: To write a Python program that accepts a decimal (base-10) integer from the user and displays its binary, octal, and hexadecimal equivalents using built-in functions.

Code:

```
# Input decimal number
decimal_number = int(input("Enter a decimal number: "))

# Conversions using built-in functions
binary = bin(decimal_number)
octal = oct(decimal_number)
hexadecimal = hex(decimal_number)

# Display results
print("Binary equivalent:", binary)
print("Octal equivalent:", octal)
print("Hexadecimal equivalent:", hexadecimal)
```

```
Binary equivalent: 0b11100
Octal equivalent: 0o34
Hexadecimal equivalent: 0x1c
```

Code Explanation:

- The input is read as an integer using int().
- bin() converts the number to binary.
- oct() converts the number to octal.
- hex() converts the number to hexadecimal.
- The converted values are printed.

b. Program to convert a string to float, add 10.5, and display the result

Objective: To write a Python program that reads a string representing a floating-point number, converts it to a float, adds 10.5, and prints the final result.

Code:

```
# Input string representing a floating-point number
num_str = input("Enter a floating-point number: ")

# Convert string to float
num = float(num_str)

# Add 10.5
result = num + 10.5
```

```python
# Display result
print("Result after adding 10.5:", result)
```

```
Result after adding 10.5: 39.1
```

Code Explanation:

- The input is taken as a string.
- float() converts the string into a floating-point number.
- Arithmetic addition is performed.
- The final result is displayed using print().

4. Operator Precedence, Conditional Statements, and Loops

a. Program to solve the expression and show each calculation

```
# Step-by-step evaluation
step1 = 3 ** 2          # Exponentiation
step2 = 2 * step1      # Multiplication
step3 = 8 / 4          # Division
x = 10 + step2 - step3 # Addition and subtraction

# Display steps
print("3 ** 2 =", step1)
print("2 * 9 =", step2)
print("8 / 4 =", step3)
print("Final value of x =", x)
```

```
3 ** 2 = 9
2 * 9 = 18
8 / 4 = 2.0
Final value of x = 26.0
```

Code Explanation:

- Exponentiation (**) is evaluated first.
- Multiplication and division are evaluated next.
- Addition and subtraction are evaluated last.
- Intermediate results are stored in variables for clarity.

b. Program using dictionary to store student names and marks

Objective: To create a dictionary storing student names and marks, print the dictionary, and display marks for a given student using conditional statements.

Code:

```
# Create dictionary
students = {
    "Alice": 85,
    "Bob": 90,
    "Charlie": 78
}

# Print dictionary
print("Student Marks Dictionary:")
print(students)

# Input student name
name = input("Enter student name to view marks: ")

# Conditional check
```

```
if name in students:
    print("Marks of", name, ":", students[name])
else:
    print("Student not found.")
```

```
Student Marks Dictionary:
{'Alice': 85, 'Bob': 90, 'Charlie': 78}
Marks of Charlie : 78
```

Code Explanation

- A dictionary stores names as keys and marks as values.
- User input is checked using if condition.
- Marks are displayed if the student exists.

c. Program to print the reverse of a number

Objective: To write a Python program to reverse a given number using a while loop.

```
# Input number
num = int(input("Enter a number: "))
reverse = 0

# Reverse logic
while num > 0:
    digit = num % 10
    reverse = reverse * 10 + digit
    num = num // 10

print("Reversed number:", reverse)
```

```
Reversed number: 6082
```

Code Explanation:

- % extracts the last digit.
- // removes the last digit.
- Loop continues until the number becomes zero.
- Digits are rebuilt in reverse order.

d. Program to print multiplication table of a number

Objective: To write a Python program that prints the multiplication table of a given number using a for loop.

Code:

```
# Input number
n = int(input("Enter a number: "))
```

```python
# Print multiplication table
print("Multiplication Table of", n)
for i in range(1, 11):
    print(n, "x", i, "=", n * i)
```

```
Multiplication Table of 28
28 x 1 = 28
28 x 2 = 56
28 x 3 = 84
28 x 4 = 112
28 x 5 = 140
28 x 6 = 168
28 x 7 = 196
28 x 8 = 224
28 x 9 = 252
28 x 10 = 280
```

Code Explanation:

- The number is taken as input.
- for loop iterates from 1 to 10.
- Each iteration prints one row of the multiplication table.

5. Nested Loops, String Multiplication, Keyword Argument

a. Program to print the given number pattern

Objective:To write a Python program using nested loops to print the following pattern.

```
    1
   12
  123
 1234
12345
```

Code:

```python
rows = 5

for i in range(1, rows + 1):
    print(" " * (rows - i), end="")
    for j in range(1, i + 1):
        print(j, end="")
    print()
```

```
    1
   12
  123
 1234
12345
```

Code Explanation:

- The outer loop controls the number of rows.
- The inner loop prints numbers from 1 to the current row number.
- end="" prevents a new line after each number.
- print() moves to the next line after each row.

b. Program to print multiplication table of n

Objective: To write a Python program that prints the multiplication table of a given number using a loop.

Code:

```python
# Input number
n = int(input("Enter a number: "))

# Print multiplication table
for i in range(1, 11):
    print(n, "x", i, "=", n * i)
```

```
28 x 1 = 28
28 x 2 = 56
```

```
28 x 3 = 84
28 x 4 = 112
28 x 5 = 140
28 x 6 = 168
28 x 7 = 196
28 x 8 = 224
28 x 9 = 252
28 x 10 = 280
```

Code Explanation:

- The number n is taken from the user.
- A for loop runs from 1 to 10.
- Each iteration prints one row of the multiplication table.

c. Program using keyword arguments (student information)

Objective: To write a Python function that accepts keyword arguments and prints student information in a readable format.

Code:

```python
# Function definition with keyword arguments
def student_info(name, age, course):
    print("Student Name :", name)
    print("Age          :", age)
    print("Course       :", course)

# Function call using keyword arguments in any order
student_info(course="Data Science", name="Rahul", age=20)
```

```
Student Name : Rahul
Age          : 20
Course       : Data Science
```

Code Explanation:

- The function accepts three parameters: name, age, and course.
- Keyword arguments allow passing values in any order.
- The information is displayed neatly.

d. Program to print absolute value, square root, and cube of a number

Objective: To write a Python program that calculates and prints the absolute value, square root, and cube of a given number.

Code:

```python
import math

# Input number
```

```
num = float(input("Enter a number: "))

# Calculations
absolute_value = abs(num)
square_root = math.sqrt(absolute_value)
cube = num ** 3

# Display results
print("Absolute Value:", absolute_value)
print("Square Root    :", square_root)
print("Cube           :", cube)
```

```
Absolute Value: 28.0
Square Root    : 5.291502622129181
Cube           : 21952.0
```

Code Explanation:

- abs() gives the absolute value.
- math.sqrt() calculates the square root.
- ** operator is used to calculate the cube.

6. Program Based on Functions

a. Program using a lambda function to square a number

Objective: To write a Python program that takes a number as input, uses a lambda function to calculate its square, and prints the result.

Code:

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

# Lambda function to find square
square = lambda x: x * x

# Print result
print("Square of the number:", square(num))
```

```
Square of the number: 36
```

Code Explanation:

- Input is taken from the user.
- A lambda function is defined to compute the square of a number.
- The function is called by passing the input number.

b. Recursive function to count number of digits

Objective: To write a recursive function that counts and returns the number of digits in a positive integer.

Code:

```
def count_digits(n):
    if n == 0:
        return 0
    return 1 + count_digits(n // 10)

# Input number
num = int(input("Enter a positive integer: "))

print("Number of digits:", count_digits(num))
```

```
Number of digits: 4
```

Code Explanation:

- The function divides the number by 10 in each recursive call.
- Each call counts one digit.
- Recursion stops when the number becomes 0.

c. Program to print list in original and reverse order

Objective: To write a Python program that:
1) Takes n integers into a list 2) Prints the list in original order 3) Prints the list in reverse order without using [::-1] or reverse()

Code:

```python
# Input number of elements
n = int(input("Enter number of elements: "))
lst = []

# Input elements
for i in range(n):
    element = int(input("Enter element: "))
    lst.append(element)

# Print original list
print("Original list:", lst)

# Print reverse list without using reverse methods
print("Reversed list:", end=" ")
for i in range(len(lst) - 1, -1, -1):
    print(lst[i], end=" ")
```

```
Original list: [28, 1, 6, 12]
Reversed list: 12 6 1 28
```

Code Explanation:

- Elements are added to the list using append().
- The original list is printed directly.
- A for loop prints elements from the last index to the first.

7. Modules and Packages

a. Program using the random module with alias

Objective: To write a Python program that imports the random module using an alias and prints 5 random integers between 1 and 100.

Code:

```python
import random as rnd

# Print 5 random integers between 1 and 100
for i in range(5):
    print(rnd.randint(1, 100))
```

```
43
75
45
86
57
```

Code Explanation:

- The random module is imported with the alias rnd.
- randint(1, 100) generates a random integer between 1 and 100 (inclusive).
- A for loop runs 5 times to print 5 random numbers

b. Program to plot a simple line graph using matplotlib

Objective: To write a Python program that imports pyplot from the matplotlib package and plots a simple line graph.

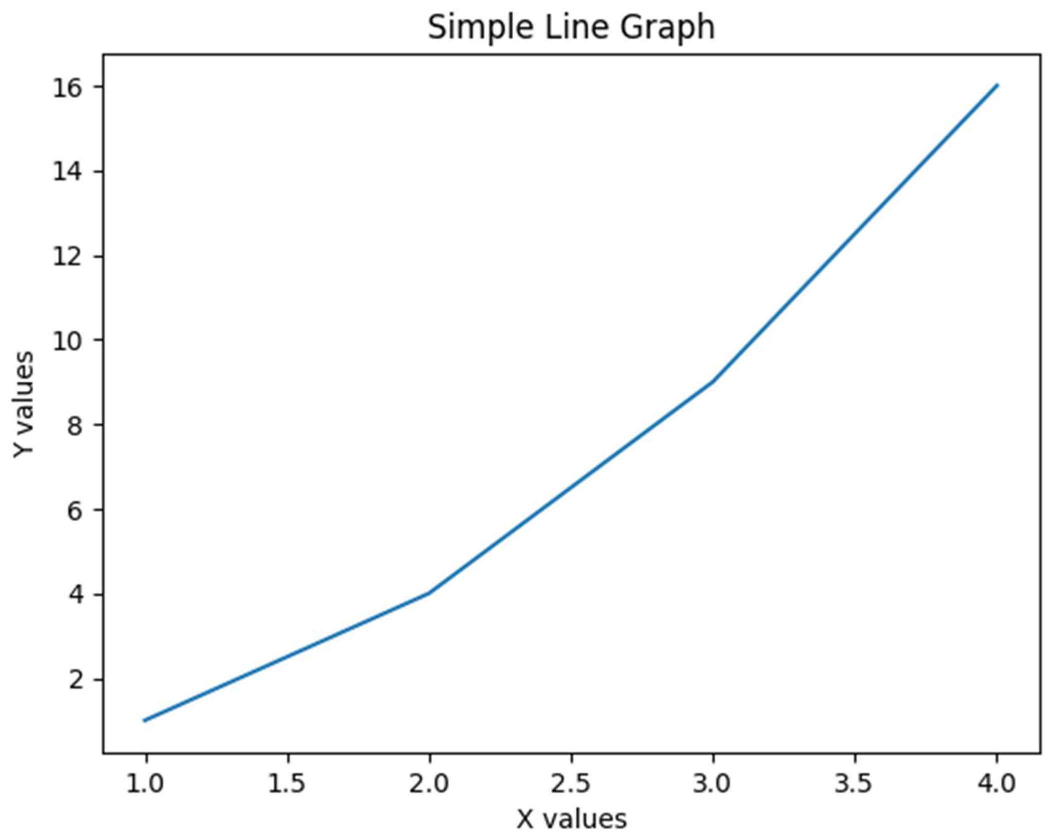
Code:

```python
import matplotlib.pyplot as plt

# Data points
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

# Plot line graph
plt.plot(x, y)
plt.xlabel("X values")
plt.ylabel("Y values")
plt.title("Simple Line Graph")

# Display graph
plt.show()
```



Code Explanation:

- pyplot is imported from matplotlib as plt.
- Lists x and y store the data points.
- plt.plot() draws the line graph.
- Labels and title improve readability.
- plt.show() displays the graph.

8. String and Its Operations

a. Program to remove vowels from a string

Objective: To write a Python program that takes a string as input and prints a new string with all vowels removed, while preserving the order of remaining characters.

Code:

```
# Input string
text = input("Enter a string: ")

# Vowels set
vowels = "aeiouAEIOU"

# Remove vowels
result = ""
for ch in text:
    if ch not in vowels:
        result += ch

print("String after removing vowels:", result)
```

```
String after removing vowels: tl Dhmn
```

Code Explanation:

- The input string is taken from the user.
- A string containing all vowels (both cases) is defined.
- Each character is checked and added to the result only if it is not a vowel.
- Order of characters is preserved.

b. Program to check whether a string is a palindrome

Objective: To write a Python program that checks whether a string is a palindrome, ignoring case and non-alphanumeric characters.

Code:

```
# Input string
text = input("Enter a string: ")

# Keep only alphanumeric characters and convert to lowercase
cleaned = ""
for ch in text:
    if ch.isalnum():
        cleaned += ch.lower()

# Check palindrome
if cleaned == cleaned[::-1]:
    print(f"The string {text} is a palindrome")
else:
    print(f"The string {text} is not a palindrome")
```

```
The string Atul is not a palindrome
```

Code Explanation:

- isalnum() removes spaces and special characters.
- lower() ignores case differences.
- The cleaned string is compared with its reverse.
- Result is printed accordingly.

c. Program to reverse the order of words in a sentence

Objective: To write a Python program that prints the words in reverse order, while keeping the characters inside each word unchanged.

Code:

```
# Input sentence
sentence = input("Enter a sentence: ")

# Split sentence into words
words = sentence.split()

# Print words in reverse order
print("Sentence with reversed word order:")
for i in range(len(words) - 1, -1, -1):
    print(words[i], end=" ")
```

```
Sentence with reversed word order:
Dhiman Atul
```

Code Explanation:

- split() divides the sentence into words.
- A loop prints words from the last to the first.
- Characters inside each word remain unchanged.

9. File Handling

a. Program to count tabs, spaces, and newline characters in a file

Objective: To write a Python program that reads a text file and counts the number of tabs (\t), spaces (' '), and newline characters (\n) present in the file.

Code:

```python
# Open file in read mode
file = open("exp9.txt", "r")

# Initialize counters
spaces = 0
tabs = 0
newlines = 0

# Read file content
content = file.read()

# Count characters
for ch in content:
    if ch == ' ':
        spaces += 1
    elif ch == '\t':
        tabs += 1
    elif ch == '\n':
        newlines += 1

file.close()

# Display results
print("Number of spaces:", spaces)
print("Number of tabs:", tabs)
print("Number of newlines:", newlines)
```

```
Number of spaces: 2
Number of tabs: 0
Number of newlines: 1
```

Code Explanation:

- The file is opened in read mode.
- The entire content is read using read().
- Each character is checked:

- ' ' → space
- '\t' → tab
- '\n' → newline

- Counters are incremented accordingly.
- The file is closed after reading.

b. Program to copy first 10 bytes of a binary file into another file

Objective: To write a Python program that copies the first 10 bytes from a binary file into another binary file.

Code:

```python
# Open source file in binary read mode
source = open("source.bin", "rb")

# Open destination file in binary write mode
destination = open("destination.bin", "wb")

# Read first 10 bytes
data = source.read(10)

# Write data to destination file
destination.write(data)

# Close files
source.close()
destination.close()

print("First 10 bytes copied successfully.")
```

```
First 10 bytes copied successfully.
```

Code Explanation:

- "rb" opens the source file in binary read mode.
- "wb" opens the destination file in binary write mode.
- read(10) reads the first 10 bytes.
- write() copies the bytes to the new file.
- Both files are closed properly.

10. Data Structure

a. Program to remove duplicates from a list

Objective: To write a Python program that:

- Takes n integers from the user and stores them in a list
- Creates a new list without duplicate elements
- Prints both the original list and the new list

Code:

```python
# Input number of elements
n = int(input("Enter number of elements: "))
lst = []

# Input elements
for i in range(n):
    element = int(input("Enter element: "))
    lst.append(element)

# Remove duplicates while preserving order
unique_list = []
for item in lst:
    if item not in unique_list:
        unique_list.append(item)

# Display lists
print("Original list:", lst)
print("List without duplicates:", unique_list)
```

```
Original list: [28, 6, 28, 1]
List without duplicates: [28, 6, 1]
```

Code Explanation:

- Elements are stored in a list using append().
- A new list unique_list is created.
- Each element is added only if it is not already present.
- Order of elements is preserved.

b. Program to find index of an element in a tuple

Objective:

- To write a Python program that:
- Creates a tuple
- Accepts a number from the user

Code:
```python
# Create tuple
nums = (10, 20, 30, 40, 50)

# Input number
num = int(input("Enter a number to search: "))

# Check presence and print index
if num in nums:
    print("Index of", num, "is", nums.index(num))
else:
    print("Number not present in the tuple")
```

```
Index of 20 is 1
```

Code Explanation:

- A tuple nums is predefined.
- User input is checked using the in operator.
- index() returns the position of the element if found.
- If not found, a suitable message is printed.

c. Common Elements and Symmetric Difference of Two Sequences

Objective: To write a Python program that:

- Reads two sequences of integers (duplicates allowed)
- Prints: 1) The set of elements common to both sequences 2) The set of elements appearing in exactly one sequence (symmetric difference)
- Displays both results in sorted order without duplicates

Code:
```python
# Read first sequence
n1 = int(input("Enter number of elements in first sequence: "))
seq1 = []
for i in range(n1):
    seq1.append(int(input("Enter element: ")))

# Read second sequence
n2 = int(input("Enter number of elements in second sequence: "))
seq2 = []
for i in range(n2):
    seq2.append(int(input("Enter element: ")))

# Convert lists to sets
set1 = set(seq1)
set2 = set(seq2)
```

```
# Common elements
common_elements = sorted(set1 & set2)

# Symmetric difference
symmetric_diff = sorted(set1 ^ set2)

# Display results
print("Common elements:", common_elements)
print("Elements appearing in exactly one sequence:", symmetric_diff)
```

```
Common elements: [1, 6, 28]
Elements appearing in exactly one sequence: [12, 18, 27]
```

Code Explanation:

- The sequences are stored in lists (duplicates allowed).
- Lists are converted into sets to remove duplicates.
- & finds common elements (intersection).
- ^ finds symmetric difference (elements in exactly one set).
- sorted() arranges the output in ascending order.

11. Programming Exercises with Classes and Objects, Inheritance

a. Program with Employee class to track employees

Objective: To write a Python program using a class Employee that:

- Stores employee name, designation, and salary
- Keeps track of the total number of employees in an organization

Code:

```
class Employee:
    # Class variable to track number of employees
    emp_count = 0

    def __init__(self, name, designation, salary):
        self.name = name
        self.designation = designation
        self.salary = salary
        Employee.emp_count += 1

    def display_details(self):
        print("Name       :", self.name)
        print("Designation:", self.designation)
        print("Salary     :", self.salary)
        print()

    @classmethod
    def total_employees(cls):
        print("Total number of employees:", cls.emp_count)


# Create employee objects
e1 = Employee("Rahul", "Manager", 50000)
e2 = Employee("Anita", "Engineer", 40000)
e3 = Employee("Karan", "Clerk", 25000)

# Display employee details
e1.display_details()
e2.display_details()
e3.display_details()

# Display total employees
Employee.total_employees()
```

```
Name       : Rahul
Designation: Manager
Salary     : 50000
```

```
Name       : Anita
Designation: Engineer
Salary     : 40000

Name       : Karan
Designation: Clerk
Salary     : 25000

Total number of employees: 3
```

Code Explanation:

- emp_count is a class variable shared by all objects.
- Each time an employee object is created, the count increases.
- display_details() shows employee information.
- @classmethod is used to access class-level data

b. Program using Student, Course, and Department classes

Objective: To write a Python program that:

- Defines classes Student, Course, and Department
- Enrolls a student in a course of a particular department
- Displays enrollment details

Code:

```python
class Department:
    def __init__(self, name):
        self.name = name


class Course:
    def __init__(self, name, year):
        self.name = name
        self.year = year


class Student:
    def __init__(self, name, rollno, course, year):
        self.name = name
        self.rollno = rollno
        self.course = course
        self.year = year

    def display_info(self, department):
        print("Student Name :", self.name)
        print("Roll Number  :", self.rollno)
        print("Course       :", self.course.name)
        print("Year         :", self.year)
        print("Department   :", department.name)


# Create department
dept = Department("Computer Science")

# Create course
course = Course("Data Science", 2)

# Enroll student
student = Student("Amit", 101, course, 2)

# Display enrollment details
student.display_info(dept)
```

```
Student Name : Amit
Roll Number  : 101
Course       : Data Science
Year         : 2
Department   : Computer Science
```

Code Explanation:

- Department stores department name.
- Course stores course name and year.
- Student stores student details and course enrolled.
- Objects are linked to represent enrollment.

12. Operator Overloading, Error Handling

a. Program to overload the + operator for a Student class

Objective: To write a Python program that overloads the + operator for a Student class having attributes name and marks, such that adding two students adds their marks.

Code:

```python
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    # Overload + operator
    def __add__(self, other):
        return self.marks + other.marks


# Create student objects
s1 = Student("Amit", 85)
s2 = Student("Ravi", 90)

# Add marks using overloaded + operator
total_marks = s1 + s2

print("Marks of", s1.name, ":", s1.marks)
print("Marks of", s2.name, ":", s2.marks)
print("Total Marks:", total_marks)
```

```
Marks of Amit : 85
Marks of Ravi : 90
Total Marks: 175
```

Code Explanation:

- __ add __() is a special method used to overload the + operator.
- When s1 + s2 is executed, Python internally calls s1.__ add __(s2).
- The method returns the sum of marks of both students.

b. Program to validate name and age for voting eligibility

Objective: To write a Python program that validates name and age entered by the user and checks whether the person is eligible to vote.

Code:

```python
name = input("Enter your name: ")
```

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

if age >= 18:
    print(name, "is eligible to vote.")
else:
    print(name, "is not eligible to vote.")
```

```
Atul Dhiman is eligible to vote.
```

Code Explanation:

- Name is taken as a string.
- Age is taken as an integer.
- If age is 18 or above, the person can vote.
- Otherwise, the person is not eligible.

c. Program to handle invalid integer input using exception handling

Objective: To write a Python program that:

- Repeatedly asks the user to enter an integer
- Handles invalid input using exception handling
- Continues until a valid integer is entered

```
while True:
    try:
        num = int(input("Enter an integer: "))
        print("You entered:", num)
        break
    except ValueError:
        print("Invalid input, please enter an integer.")
```

```
Invalid input, please enter an integer.
You entered: 1
```

Code Explanation:

- try block attempts to convert input into an integer.
- If input is invalid (abc, 3.5), a ValueError occurs.
- except block catches the error and prints a message.
- Loop continues until valid input is provided.