Module-15) Advance python programming

1. Printing on Screen

Theory

- The print() function in Python is used to display output on the screen.
- It can print text, numbers, and variables.
- Formatted output can be created using:

  f-strings (f"{...}") – modern and easy to read.

Lab Program

Python program to print a formatted string using print() and f-string:

```
name = "Alice"

age = 20

print(f"My name is {name} and I am {age} years old.")
```

Practical Example 1

Python program to print “Hello, World!” on the screen:

```
print("Hello, World!")
```

2. Reading Data from Keyboard

Theory

- The input() function is used to read data entered by the user from the keyboard.
- By default, input() returns data as a string.
- Data type conversion is required when working with numbers:
  - int() → converts input to integer
  - float() → converts input to floating-point number

Lab Program

Python program to read a name and age from the user and print a formatted output:

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

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

print(f"Name: {name}")

print(f"Age: {age}")
```

Practical Example 2

Python program to read a string, an integer, and a float from the keyboard and display them:

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

number = int(input("Enter an integer: "))

decimal = float(input("Enter a float number: "))

print("String:", text)

print("Integer:", number)

print("Float:", decimal)
```

3. Opening and Closing Files

Theory

- Files in Python are handled using the open() function.
- A file can be opened in different modes:
    - 'r' → Read mode (default)
    - 'w' → Write mode (creates a new file or overwrites existing file)
    - 'a' → Append mode (adds data at the end of file)
    - 'r+' → Read and write mode
    - 'w+' → Write and read mode (overwrites existing content)
- After file operations are completed, the file should be closed using the close() method to free system resources.

Lab Program

Python program to open a file in write mode, write some text, and then close it:

```
file = open("sample.txt", "w")

file.write("This is a sample text written to the file.")
file.close()
```

Practical Example 3

Python program to create a file and write a string into it:

```
file = open("example.txt", "w")

file.write("Hello! This string is written into the file.")

file.close()
```

4. Reading and Writing Files

Theory

- Files can be read using:
  - read() → Reads the entire file
  - readline() → Reads one line at a time
  - readlines() → Reads all lines and returns a list
- Files can be written using:
  - write() → Writes a single string
  - writelines() → Writes multiple strings
- The tell() function returns the current position of the file cursor.

Lab Program 1

Python program to read the contents of a file and print them on the console:

```
file = open("example.txt", "r")

content = file.read()

print(content)

file.close()
```

Lab Program 2

Python program to write multiple strings into a file:

```
file = open("multi.txt", "w")

lines = ["First line\n", "Second line\n", "Third line\n"]

file.writelines(lines)

file.close()
```

Practical Example 4

Python program to create a file and print a string into the file:

```
file = open("printfile.txt", "w")

file.write("Printing this string into the file.")

file.close()
```

Practical Example 5

Python program to read a file and print the data on the console:

```
file = open("printfile.txt", "r")

data = file.read()

print(data)

file.close()
```

Practical Example 6

Python program to check the current position of the file cursor using tell():

```
file = open("printfile.txt", "r")

file.read(10)

position = file.tell()

print("Current file cursor position:", position)

file.close()
```

5. Exception Handling

Theory

- An **exception** is an error that occurs during the execution of a program.
- Python handles exceptions using:
    - try → Code that may cause an exception
    - except → Handles the exception
    - finally → Executes whether an exception occurs or not
- **Multiple exceptions** can be handled using multiple except blocks.
- **Custom exceptions** are user-defined exceptions created using the Exception class.

Lab Program 1

Python program to handle exceptions in a simple calculator (division by zero, invalid input):

```
try:
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    result = a / b
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
except ValueError:
    print("Error: Invalid input. Please enter numbers only.")
```

Lab Program 2

Python program to demonstrate handling multiple exceptions:

```
try:
    x = int(input("Enter a number: "))
    y = int(input("Enter another number: "))
    print(x / y)
except ValueError:
    print("Error: Invalid input.")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
```

Practical Example 7

Python program to handle exceptions in a calculator:

```
try:
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    print("Division Result:", num1 / num2)
except ZeroDivisionError:
    print("Error: Division by zero.")
```

```
except ValueError:
    print("Error: Invalid input.")
```

Practical Example 8

Python program to handle multiple exceptions (file not found, division by zero):

```
try:
    file = open("data.txt", "r")
    a = int(input("Enter a number: "))
    print(10 / a)
except FileNotFoundError:
    print("Error: File not found.")
except ZeroDivisionError:
    print("Error: Division by zero.")
finally:
    print("Program execution completed.")
```

Practical Example 9

Python program to handle file exceptions and use the finally block for closing the file:

```
try:
    file = open("sample.txt", "r")
    print(file.read())
except FileNotFoundError:
    print("Error: File does not exist.")
finally:
    print("Closing file.")
    try:
        file.close()
    except:
        pass
```

Practical Example 10

Python program to print custom exceptions:

```
class AgeError(Exception):
    pass
try:
    age = int(input("Enter age: "))
    if age < 18:
        raise AgeError("Age must be 18 or above.")
    print("Access granted.")
except AgeError as e:
    print("Custom Exception:", e)
```

6. Class and Object (OOP Concepts)

Theory

- A **class** is a blueprint for creating objects.
- An **object** is an instance of a class.
- **Attributes** store data, and **methods** define behavior.
- **Local variables** are defined inside methods.
- **Global variables** are defined outside the class and accessible throughout the program.

Lab Program

Python program to create a class and access its properties using an object:

```
class Student:
    name = "John"
    age = 21
obj = Student()
print("Name:", obj.name)
print("Age:", obj.age)
```

Practical Example 11

Python program to create a class and access the properties of the class using an object:

```
class Employee:
    emp_id = 101
    emp_name = "Alice"
e = Employee()
print("Employee ID:", e.emp_id)
print("Employee Name:", e.emp_name)
```

Practical Example 12

Python program to demonstrate the use of local and global variables in a class:

```
global_var = "I am a global variable"
class Demo:
    def show(self):
    local_var = "I am a local variable"
    print(local_var)
    print(global_var)
obj = Demo()
obj.show()
```

7. Inheritance

Theory

- **Inheritance** allows a class (child) to acquire properties and methods of another class (parent).
- Types of inheritance in Python:
    - **Single Inheritance** – One child, one parent
    - **Multilevel Inheritance** – Child derived from another child class
    - **Multiple Inheritance** – One child inherits from multiple parents
    - **Hierarchical Inheritance** – Multiple children inherit from a single parent
    - **Hybrid Inheritance** – Combination of two or more types
- The super() function is used to access parent class methods and variables.

Lab Programs

Python programs to demonstrate different types of inheritance are shown below.

Practical Example 13

Single Inheritance

```
class Parent:
    def show(self):
        print("This is Parent class")


class Child(Parent):
    def display(self):
        print("This is Child class")


obj = Child()
obj.show()
obj.display()
```

Practical Example 14

Multilevel Inheritance

```
class Grandparent:
    def gshow(self):
        print("Grandparent class")


class Parent(Grandparent):
    def pshow(self):
        print("Parent class")


class Child(Parent):
    def cshow(self):
        print("Child class")


obj = Child()
obj.gshow()
obj.pshow()
obj.cshow()
```

Practical Example 15

Multiple Inheritance

```
class Father:
    def fshow(self):
        print("Father class")

class Mother:
    def mshow(self):
        print("Mother class")

class Child(Father, Mother):
    def cshow(self):
        print("Child class")

obj = Child()
obj.fshow()
obj.mshow()
obj.cshow()
```

Practical Example 16

Hierarchical Inheritance

```
class Parent:
    def show(self):
        print("Parent class")

class Child1(Parent):
    def display1(self):
        print("Child1 class")
```

```
class Child2(Parent):
    def display2(self):
        print("Child2 class")

obj1 = Child1()
obj2 = Child2()

obj1.show()
obj1.display1()
obj2.show()
obj2.display2()
```

Practical Example 17

Hybrid Inheritance

```
class A:
    def showA(self):
        print("Class A")

class B(A):
    def showB(self):
        print("Class B")

class C(A):
    def showC(self):
        print("Class C")

class D(B, C):
    def showD(self):
        print("Class D")

obj = D()
```

```
obj.showA()
obj.showB()
obj.showC()
obj.showD()
```

Practical Example 18

Use of super() in Inheritance

```
class Parent:
    def __init__(self):
        print("Parent constructor")


class Child(Parent):
    def __init__(self):
        super().__init__()
        print("Child constructor")
obj = Child()
```

8. Method Overloading and Overriding

Theory

- **Method Overloading**:
    - Python does not support traditional method overloading.
    - Achieved using default arguments or variable-length arguments.
- **Method Overriding**:
    - Child class redefines a method already defined in the parent class.

Lab Program 1

Method Overloading

```
class Calculator:
    def add(self, a=0, b=0, c=0):
        print("Sum:", a + b + c)


obj = Calculator()
obj.add(10)
obj.add(10, 20)
obj.add(10, 20, 30)
```

Lab Program 2

Method Overriding

```
class Parent:
    def show(self):
        print("Parent class method")

class Child(Parent):
    def show(self):
        print("Child class method")

obj = Child()
obj.show()
```

Practical Example 19

Python program to show method overloading

```
class Demo:
    def display(self, a=None, b=None):
        if a is not None and b is not None:
            print("Sum:", a + b)
        elif a is not None:
            print("Value:", a)
        else:
            print("No values")

obj = Demo()
obj.display()
obj.display(10)
obj.display(10, 20)
```

Practical Example 20

Python program to show method overriding

```
class Animal:
    def sound(self):
        print("Animal makes a sound")
class Dog(Animal):
    def sound(self):
        print("Dog barks")
obj = Dog()
obj.sound()
```

9. SQLite3 and PyMySQL (Database Connectors)

Theory

- **SQLite3** is a lightweight, file-based database included with Python.
- **PyMySQL** is used to connect Python programs with MySQL databases.
- Python provides database connectivity using:
    - sqlite3 module for SQLite databases
    - pymysql module for MySQL databases
- SQL queries such as CREATE, INSERT, SELECT, UPDATE, and DELETE can be executed using Python programs.

Lab Program

Python program to connect to an SQLite3 database, create a table, insert data, and fetch data

```
import sqlite3
conn = sqlite3.connect("student.db")
cur = conn.cursor()
cur.execute("CREATE TABLE IF NOT EXISTS student(id INTEGER, name TEXT)")
cur.execute("INSERT INTO student VALUES (1, 'John')")
cur.execute("INSERT INTO student VALUES (2, 'Alice')")

conn.commit()
```

```
cur.execute("SELECT * FROM student")
rows = cur.fetchall()
for row in rows:
    print(row)
conn.close()
```

Practical Example 21

Python program to create a database and a table using SQLite3

```
import sqlite3

conn = sqlite3.connect("college.db")
cur = conn.cursor()

cur.execute("""
CREATE TABLE IF NOT EXISTS course (
    cid INTEGER,
    cname TEXT
)
""")
conn.commit()
conn.close()
```

Practical Example 22

Python program to insert data into an SQLite3 database and fetch it

```
import sqlite3
conn = sqlite3.connect("college.db")
cur = conn.cursor()
cur.execute("INSERT INTO course VALUES (101, 'Python')")
cur.execute("INSERT INTO course VALUES (102, 'Database')")

conn.commit()
cur.execute("SELECT * FROM course")
```

```
data = cur.fetchall()
for row in data:
   print(row)
conn.close()
```

10. Search and Match Functions

Theory

- Python provides the re module for **regular expression pattern matching**.
- re.search():
    - Searches for a pattern **anywhere** in the string.
- re.match():
    - Matches a pattern **only at the beginning** of the string.
- Key difference:
    - search() scans the entire string
    - match() checks only the start of the string

Lab Program 1

Python program to search for a word in a string using re.search()

```
import re
text = "Python programming is easy"
pattern = "programming"
result = re.search(pattern, text)
if result:
   print("Word found")
else:
   print("Word not found")
```

Lab Program 2

Python program to match a word in a string using re.match()

```
import re

text = "Python programming"
pattern = "Python"
```

```
result = re.match(pattern, text)
if result:
    print("Match found at the beginning")
else:
    print("No match found")
```

Practical Example 23

Python program to search for a word in a string using re.search()

```
import re
string = "Learning Python is fun"
word = "Python"
if re.search(word, string):
    print("Word found in string")
else:
    print("Word not found")
```

Practical Example 24

Python program to match a word in a string using re.match()

```
import re
string = "Hello World"
word = "Hello"
if re.match(word, string):
    print("Word matches at the beginning")
else:
    print("Word does not match at the beginning")
```