**Module – 15**

**Advance Python Programming**

1. **Theory:**
2. **Introduction to the print() function in Python.**

**Ans. The print() function is a built-in Python function used to display messages, variables, or any other data to the console. It’s often the first function beginners learn—famously used in the classic "Hello, World!" example.**

1. **Formatting outputs using f-strings and format().**

**Ans. Introduced in Python 3.6, f-strings are the most modern and preferred way to format strings.**

**name = "Bhagirath"**

**age = 30**

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

1. **Using the input() function to read user input from the keyboard.**

**Ans. The input() function is a built-in Python function that pauses program execution and waits for the user to type something into the console. Once the user presses Enter, the function returns the input as a string.**

**name = input("What is your name? ")**

**print("Hello,", name)**

1. **Converting user input into different data types (e.g., int, float, etc.).**

**Ans. The input() function always returns a string, even if the user types a number. To perform calculations or logical operations, you need to convert that string into the appropriate data type.**

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

**print("Next year, you'll be", age + 1)**

1. **Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').**

**Ans. Here’s a detailed guide to opening files in different modes in Python using the open() function. Each mode determines how the file will be accessed—whether for reading, writing, appending, or updating.**

**'r' – Read Mode:**

* **Opens the file for reading.**
* **Fails if the file doesn’t exist.**

**'w' – Write Mode**

* **Opens the file for writing.**
* **Overwrites the file if it exists; creates a new one if it doesn’t.**

**'a' – Append Mode**

* **Opens the file for writing.**
* **Adds content to the end without deleting existing data.**

**'r+' – Read and Write**

* **Opens the file for both reading and writing.**
* **Fails if the file doesn’t exist. Doesn’t truncate the file**

**'w+' – Write and Read**

* **Opens the file for reading and writing.**
* **Overwrites the file if it exists; creates a new one if not**

1. **Using the open() function to create and access files.**

**Ans. file = open("filename.txt", "mode")**

* **"filename.txt": Name of the file you want to work with.**
* **"mode": Specifies the operation (read, write, append, etc.)**

1. **Closing files using close().**

**Ans. f = open("example.txt", "r")**

**content = f.read()**

**f.close()**

1. **Reading from a file using read(), readline(), readlines().**

**Ans.**

**Read() : Reads the entire file as a single string.**

**with open("example.txt", "r") as f: content = f.read() print(content)**

**readline() : Reads one line at a time.**

**with open("example.txt", "r") as f: line1 = f.readline() line2 = f.readline() print(line1, line2)**

**readlines() : Reads all lines into a list.**

**with open("example.txt", "r") as f: lines = f.readlines() print(lines)**

* **Each line is a string element in the list.**
* **Useful for looping through lines.**

1. **Writing to a file using write() and writelines().**

**Ans.**

**write() : Writes a single string to the file.**

**with open("example.txt", "w") as f: f.write("Hello Bhagirath!\**

**writelines() : Writes a list of strings to the file.**

**lines = ["Line 1\n", "Line 2\n", "Line 3\n"] with open("example.txt", "w") as f: f.writelines(lines)**

* **Doesn’t add newlines automatically—include \n in each string.**

1. **Introduction to exceptions and how to handle them using try, except, and finally.**

**Ans.**

* **ZeroDivisionError: Dividing by zero**
* **FileNotFoundError: Trying to open a file that doesn’t exist**
* **ValueError: Invalid value passed to a function**
* **TypeError: Operation on incompatible types**
* **try Block**

**This is where you write code that might raise an exception.**

* **except Block**

**This is where you handle the exception if one occurs.**

* **finally Block**

**This block always runs, whether an exception occurred or not. It’s useful for clean-up actions like closing files or releasing res**

1. **Understanding multiple exceptions and custom exceptions.**

**Ans. Multiple Exception Handling: You can catch different types of exceptions using multiple except blocks, or group them in a single block.**

**try:**

**value = int(input("Enter a number: "))**

**result = 10 / value**

**except ValueError:**

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

**except ZeroDivisionError:**

**print("Cannot divide by zero.")**

**Custom Exception Handling: Sometimes built-in exceptions aren’t descriptive enough for your use case. You can define your own by subclassing Exception.**

**Defining a Custom Exception**

**class NegativeValueError(Exception): """Raised when a negative value is not allowed.""" pass**

**def process\_value(x): if x < 0: raise NegativeValueError("Negative values are not allowed.") return x \* 2 try: num = int(input("Enter a positive number: ")) result = process\_value(num) print(f"Processed result: {result}") except NegativeValueError as e: print(f"Custom Error: {e}") except ValueError: print("Please enter a valid integer.")**

1. **Understanding the concepts of classes, objects, attributes, and methods in Python.**

**Ans. Class: A blueprint for creating objects. It defines attributes and methods.**

**class Car: color = "red" # Attribute def drive(self): # Method print("Driving...")**

**Object: An instance of a class. It has its own copy of attributes and can use methods.**

**my\_car = Car() # Object**

**Attributes: Variables that belong to a class or object. They describe the object's state.**

**print(my\_car.color) # Output: red**

**Methods: Functions defined inside a class. They define behavior.**

**my\_car.drive() # Output: Driving...**

1. **Difference between local and global variables.**

**Ans. Global Variables**

* **Declared outside any function.**
* **Accessible throughout the program, including inside functions (unless shadowed).**
* **Can be modified inside a function using the global keyword.**

**x = 10 # Global variable**

**def show():**

**print(x) # Accessing global variable**

**show() # Output: 10**

**Local Variables**

* **Declared inside a function.**
* **Accessible only within that function.**
* **Created when the function is called, destroyed when it ends.**

**def show():**

**y = 5 # Local variable**

**print(y)**

**show() # Output: 5**

**# print(y) → Error: y is not defined outside the function**

1. **Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.**

**Ans.**

**Single Inheritance: Imagine you have a basic class called Vehicle, and then you create a Car class that inherits everything from Vehicle. That’s single inheritance—one parent, one child. The child gets all the features of the parent.**

**Multilevel Inheritance: Now think of a chain: Animal → Mammal → Dog. Each class builds on the previous one. So Dog inherits from Mammal, and Mammal inherits from Animal. It’s like passing traits down through generations.**

**Multiple Inheritance: Here, a class gets features from more than one parent. For example, a Duck might inherit from both a Flyer class and a Swimmer class. So it can fly and swim. Python lets you combine traits from multiple sources.**

**Hierarchical Inheritance: This is like one parent having multiple children. Say you have a Person class, and both Student and Teacher inherit from it. They each get the common features of Person, but can also have their own unique traits.**

**Hybrid Inheritance: This is a mix of the above types. For example, you might have a class that inherits from two classes, and those two classes themselves inherit from a common base. It’s like combining multiple paths into one—Python handles this using something called the Method Resolution Order (MRO) to keep things organized.**

1. **Using the super() function to access properties of the parent class.**

**Ans. super() is a built-in function that gives you access to methods and attributes of a parent class from within a child class. It’s commonly used to:**

* **Call the parent class’s constructor (\_\_init\_\_)**
* **Extend or override parent methods while still using their logic**

1. **Method overloading: defining multiple methods with the same name but different parameters.**

**Ans. Method overloading means defining multiple methods with the same name but different parameters—like giving a function more than one way to be called depending on what you pass in.**

**Technically, Python doesn’t support method overloading the way some other languages do. If you define multiple methods with the same name, only the last one will be used.**

**But you can simulate overloading using default arguments or \*args and \*\*kwargs.**

**class Calculator:**

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

**return a + b**

**calc = Calculator()**

**print(calc.add(5)) # Output: 5**

**print(calc.add(5, 3)) # Output: 8**

1. **Method overriding: redefining a parent class method in the child class.**

**Ans. Method overriding means redefining a method from the parent class in the child class to change or customize its behavior.**

**It lets the child class replace or extend the functionality of the parent method—perfect for tailoring behavior while keeping the same method name.**

**class Animal:**

**def speak(self):**

**print("Animal speaks")**

**class Dog(Animal):**

**def speak(self): # Overriding the parent method**

**print("Dog barks")**

**pet = Dog()**

**pet.speak() # Output: Dog barks**

1. **Introduction to SQLite3 and PyMySQL for database connectivity.**

**Ans.**

**SQLite3**

* **Type: Built-in, file-based database.**
* **Use Case: Great for small apps, local storage, and prototyping.**
* **Python Support: Comes with Python (sqlite3 module).**
* **Setup: No installation needed.**

**PyMySQL**

* **Type: Connects Python to MySQL server.**
* **Use Case: Ideal for web apps and larger systems.**
* **Python Support: Requires pymysql package.**
* **Setup: Install PyMySQL and MySQL server.**

1. **Creating and executing SQL queries from Python using these connectors.**

**Ans.**

**Local database with python:**

**Steps to Execute SQL Queries**

1. **Connect to Database  
   Use sqlite3.connect('file.db') to open or create a database.**
2. **Create Cursor & Execute Queries  
   Use cursor.execute("SQL QUERY") to run commands like CREATE, INSERT, SELECT.**
3. **Commit & Close  
   Use conn.commit() and conn.close() to save changes and release resources.**

**Connect to my sql query:**

**Steps to Execute SQL Queries**

1. **Connect to MySQL  
   Use pymysql.connect() with host, user, password, and database.**
2. **Create Cursor & Run SQL  
   Use cursor.execute("SQL QUERY") for operations like SELECT, INSERT, UPDATE.**
3. **Fetch Results & Commit  
   Use cursor.fetchall() to retrieve data and conn.commit() to save changes.**
4. **Using re.search() and re.match() functions in Python’s re module for pattern matching.**

**Ans.**

**re.match()**

* **Checks only at the beginning of the string.**
* **Returns a match object if the pattern starts right at index 0.**
* **If the pattern is elsewhere, it returns None.**

**re.search()**

* **Scans the entire string for the first match.**
* **Returns a match object if the pattern is found anywhere.**
* **More flexible than re.match() for general pattern detection.**

1. **Difference between search and match.**

**Ans.**

**re.match()**

**It checks only the beginning of the string.  
If the pattern starts right at the start, it matches.  
If the pattern is anywhere else, it won’t work.**

**re.search()**

**It scans the entire string for the pattern.  
Even if the match is in the middle or end, it’ll find it**