

Module 8 – Advance Python Programming

THEORY ASSIGNMENT

1. Printing on Screen

- Q: Introduction to the `print()` function in Python.

A: The `print()` function in Python is used to display output on the screen. It can print strings, numbers, variables, or expressions. By default, `print()` adds a newline character after each call, but this can be changed using the `'end'` parameter. You can print multiple items separated by commas, and Python automatically adds spaces between them. For example: `print('Hello', 'World')` will print 'Hello World'. This function is essential for debugging, displaying results, and creating user-friendly outputs in Python programs.

- Q: Formatting outputs using f-strings and `format()`.

A: Python provides two major methods for formatting outputs: f-strings (introduced in Python 3.6) and `format()`. F-strings allow embedding expressions inside string literals using curly braces, such as `f'The sum is {a + b}'`. The `format()` method replaces placeholders `'{}'` in a string with values: `'My name is {}'.format('Harsh')`. F-strings are faster and more readable, making them the preferred method for modern Python code.

2. Reading Data from Keyboard

- Q: Using the `input()` function to read user input from the keyboard.

A: The `input()` function is used to read data from the user as a string. It temporarily halts program execution until the user provides input. For example, `name = input('Enter your name: ')` reads a name from the user. Input is always read as a string, even if the user enters numbers.

- Q: Converting user input into different data types.

A: Since `input()` always returns a string, it must be converted into the desired data type using functions like `int()`, `float()`, or `bool()`. For example: `age = int(input('Enter your age: '))` converts input into an integer. This is crucial when performing arithmetic or logical operations.

3. Opening and Closing Files

- Q: Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').

A: Python uses the `open()` function to handle files, which takes two arguments: filename and mode. 'r' opens a file for reading, 'w' creates a new file for writing (overwriting if it exists), 'a' appends data, 'r+' allows reading and writing, and 'w+' opens a file for reading and writing (overwriting contents).

- Q: Using the `open()` function to create and access files.

A: Syntax: `file = open('data.txt', 'w')` creates a writable file object. You can then write data using `write()` or read data using `read()`. File handling is crucial for data persistence and communication between programs.

- Q: Closing files using `close()`.

A: After file operations, always close the file using `file.close()`. This releases system resources and ensures data integrity. Alternatively, use the 'with open()' as' syntax, which automatically closes the file.

4. Reading and Writing Files

- Q: Reading from a file using `read()`, `readline()`, `readlines()`.

A: The `read()` method reads the entire file as a string, `readline()` reads one line at a time, and `readlines()` returns all lines as a list. Example: `file = open('data.txt', 'r'); print(file.read())`. These methods provide flexibility for different reading needs.

- Q: Writing to a file using `write()` and `writelines()`.

A: The `write()` function writes strings to a file, while `writelines()` writes a list of strings. Example: `file.write('Hello World!')`. Always ensure the file is opened in write ('w') or append ('a') mode before writing.

5. Exception Handling

- Q: Introduction to exceptions and how to handle them using `try`, `except`, and `finally`.

A: Exceptions are runtime errors that can interrupt normal program flow. Python provides `try-except` blocks to handle such errors gracefully. For example:

`try:`

```
x = 10/0
```

`except ZeroDivisionError:`

```
    print('Division by zero not allowed')
```

`finally:`

`print('Execution complete')`. The `finally` block executes regardless of whether an exception occurs.

- Q: Understanding multiple exceptions and custom exceptions.

A: Multiple exceptions can be handled using multiple `except` blocks. Custom exceptions are created using user-defined classes that inherit from `Exception`. This allows handling application-specific errors more effectively.

6. Class and Object (OOP Concepts)

- Q: Understanding the concepts of classes, objects, attributes, and methods in Python.

A: A class is a blueprint for creating objects. It defines attributes (data) and methods (functions). An object is an instance of a class. Example:

`class Car:`

```
    def __init__(self, brand):
```

```
        self.brand = brand
```

`mycar = Car('Tesla')`. Here, `mycar` is an object of the `Car` class.

- Q: Difference between local and global variables.

A: A local variable is declared inside a function or method and accessible only there, while a global variable is declared outside and accessible throughout the program. Use the 'global' keyword to modify global variables inside functions.

7. Inheritance

- Q: Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.

A: Inheritance allows a child class to reuse properties and methods from a parent class. Single inheritance: one base and one derived class. Multilevel: class C inherits from class B which inherits from class A. Multiple: derived class inherits from multiple base classes. Hierarchical: multiple child classes inherit from one base class. Hybrid: combination of the above.

- Q: Using the `super()` function to access properties of the parent class.

A: `super()` is used to call methods from the parent class inside a derived class.

Example:

```
class A:
```

```
    def show(self): print('A')
```

```
class B(A):
```

```
    def show(self):
```

```
        super().show(); print('B'). This ensures the parent class's methods are not overridden completely.
```

8. Method Overloading and Overriding

- Q: Method overloading: defining multiple methods with the same name but different parameters.

A: In Python, true method overloading is not directly supported like in Java. However, it can be achieved using default or variable-length arguments. Example: `def add(a=None,b=None): return a+b if a!=None and b!=None else a`. This allows flexible method calls.

- Q: Method overriding: redefining a parent class method in the child class.

A: Overriding allows a child class to modify the behavior of a parent class method.

Example:

```
class Parent:
```

```
    def show(self): print('Parent')
```

```
class Child(Parent):
```

```
    def show(self): print('Child'). Here, Child overrides Parent's method.
```

9. SQLite3 and PyMySQL (Database Connectors)

- Q: Introduction to SQLite3 and PyMySQL for database connectivity.

A: SQLite3 and PyMySQL allow Python to interact with relational databases. SQLite3 is lightweight and built-in, while PyMySQL is used to connect to MySQL servers. They enable data storage, retrieval, and manipulation through SQL queries.

- Q: Creating and executing SQL queries from Python using these connectors.

A: Steps: Import the connector, establish connection, create cursor, execute queries, commit changes, and close connection. Example:

```
import sqlite3
```

```
conn=sqlite3.connect('test.db')
```

```
cursor=conn.cursor()
```

```
cursor.execute('CREATE TABLE users(name TEXT)')
```

```
conn.commit(); conn.close().
```

10. Search and Match Functions

- Q: Using re.search() and re.match() functions in Python's re module for pattern matching.

A: The re module is used for regular expressions. re.search() scans the entire string for a pattern, while re.match() checks only at the beginning. Example:

```
import re
```

re.search('cat','concatenate') returns a match; re.match('cat','concatenate') returns None.

- Q: Difference between search and match.

A: `re.match()` only matches at the start of a string, while `re.search()` looks anywhere in the string. This difference is crucial when searching for patterns in long strings or text files.