Module-3

Q-1 What is File function in python? What is keywords to create and write file.

Ans: file handling involves working with external files (like text or binary files) for reading, writing, or manipulating data. To interact with files, you use built-in file functions.

File Opening Modes:

- **'r'**: Read mode (default) - Opens the file for reading. The file must exist.
- **'w'**: Write mode - Opens the file for writing. Creates a new file if it doesn't exist or truncates the existing file.
- **'a'**: Append mode - Opens the file for appending data. Creates the file if it doesn't exist.
- **'x'**: Exclusive creation mode - Creates a new file. Fails if the file exists.

Q-2 Explain Exception handling? What is an Error in Python?

Ans: Exception handling in Python is a way to manage runtime errors gracefully, allowing the program to continue execution or terminate without crashing. It is done using **try-except** blocks to handle potential errors and provide alternative solutions when something goes wrong.

An **Error** is an issue in a program that causes the program to crash or terminate abnormally. Errors can be of two types:

1. **Syntax Errors**: Occur when there's a problem with the syntax of the code.
2. **Exceptions (Runtime Errors)**: Occur during the execution of the program. Exceptions can be handled with the try-except blocks, allowing the program to recover or fail gracefully

Q-3 How many except statements can a try-except block have? Name Some built-in exception classes:

Ans: A try-except block can have **multiple except statements** to handle different types of exceptions. You can specify as many except statements as needed, each handling a different exception type. This allows you to catch and handle various errors that might occur during the execution of the try block.

A try-except block can have **multiple except statements** to handle different exceptions. There is no strict limit to how many except blocks you can have, as long as each one is meant to catch a specific exception or a group of exceptions.

Some built-in exception classes in Python:

1. **Exception** – Base class for all exceptions.
2. **ArithmeticError** – Base class for arithmetic errors (e.g., division by zero).
3. **ZeroDivisionError** – Raised when dividing by zero.

4. **ValueError** – Raised when a function gets an argument of the right type but inappropriate value.
5. **IndexError** – Raised when a sequence index is out of range.
6. **KeyError** – Raised when a dictionary key is not found.
7. **TypeError** – Raised when an operation is applied to an object of inappropriate type.
8. **FileNotFoundError** – Raised when a file or directory is requested but doesn't exist.
9. **IOError** – Raised when an I/O operation fails.
10. **NameError** – Raised when a local or global name is not found.

Q-4 When will the else part of try-except-else be executed?

Ans: The else part of a try-except-else block is executed **only if no exceptions are raised** in the try block. In other words, it runs after the try block completes successfully without encountering any errors.

Q-5 Can one block of except statements handle multiple exception?

Ans: Yes, a single except block can handle multiple exceptions by specifying them as a tuple within the except statement. If any of the listed exceptions occur, the corresponding except block will be executed.

Q-6 When is the finally block executed?

Ans: The finally block is always executed, regardless of whether an exception occurs or not. It is typically used to ensure that important cleanup actions (like closing files or releasing resources) are performed, even if an error happens in the try or except blocks.

Q-7 What happens when „1"== 1 is executed?

Ans: When the expression "1" == 1 is executed in Python, it evaluates to **False**.

Reason:

- "1" is a string, and 1 is an integer. In Python, the == operator checks for equality of **values** and **types**.
- Since the two operands are of different types (string vs integer), the comparison returns False, even though they may represent similar values in human-readable form

Q-8 How Do You Handle Exceptions With Try/Except/Finally In Python? Explain with coding snippets.

Ans: **try block**: The code that might raise an exception is placed here.

except block: This block catches and handles the exceptions if any are raised during the execution of the try block.

finally block: This block executes **no matter what**, whether an exception occurred or not. It's typically used for cleanup actions like closing files or releasing resources

```
try:
    # Code that may raise an exception
    num1 = int(input("Enter a number: "))
    num2 = int(input("Enter another number: "))
    result = num1 / num2
    print(f"Result: {result}")
except ZeroDivisionError:
    # Handle specific ZeroDivisionError
    print("Error: You cannot divide by zero!")
except ValueError:
    # Handle invalid input (non-integer)
    print("Error: Please enter a valid integer!")
finally:
    # This block will always execute
    print("Execution complete.")
```

Q-9 What are oops concepts? Is multiple inheritance supported in java

Ans: Object-Oriented Programming (OOP) is a programming paradigm centered around the concept of objects, which are instances of classes. OOP focuses on using objects to represent real-world entities, combining both data (attributes) and behavior (methods) into a single structure. The primary goal of OOP is to promote better organization, modularity, and reusability of code by utilizing four key principles: encapsulation, inheritance, polymorphism, and abstraction.

No, Java does **not support multiple inheritance** through classes, meaning a class cannot inherit from more than one class directly. This limitation is set to avoid complexity and ambiguity issues, like the **Diamond Problem**.

Q-10 How to Define a Class in Python? What Is Self? Give An Example Of A Python Class

Ans : In Python, a class is a blueprint for creating objects (instances). It defines a set of attributes and methods that the objects created from the class will have.

The keyword class is used to define a class, and the self parameter refers to the instance of the class itself, allowing access to the attributes and methods of the class in instance methods.

Example:

```
class ClassName:
    def __init__(self, parameters):
        # Constructor to initialize instance attributes
```

```
        self.attribute = value


    def method(self):
        # Method to perform some action
        pass
```

Q-11 Explain Inheritance in Python with an example? What is init? Or What Is A Constructor In Python?

Ans: Inheritance is one of the key features of Object-Oriented Programming (OOP) that allows a class to inherit properties and behavior (methods) from another class. It enables code reusability and the creation of a hierarchical relationship between classes.

- The class that is being inherited from is called the parent class or base class.
- The class that inherits from the parent is called the child class or derived class.

Example: # Parent class

```
class Animal:
    def sound(self):
        print("This animal makes a sound")


# Child class inheriting from Animal class
class Dog(Animal):
    def sound(self):
        print("The dog barks")


# Creating an object of Dog class
dog = Dog()
dog.sound()  # Output: The dog barks
```

What is `__init__`?

In Python, `__init__` is a special method, also known as a **constructor**. It is automatically called when a new object is created from a class and allows the class to initialize the object's attributes.

Q-12 What is Instantiation in terms of OOP terminology?

Ans: Instantiation in Object-Oriented Programming (OOP) refers to the process of creating an instance (object) of a class. When you instantiate a class, you create an object that has the properties and behaviors defined by that class.

Q-13 What is used to check whether an object o is an instance of class A?

Ans: To check whether an object o is an instance of a class A, you can use Python's built-in function isinstance().

Syntax: isinstance(object, class)

Example: class A:

```
    pass

class B:

    pass

# Instantiate an object of class A

o = A()

# Check if o is an instance of class A

print(isinstance(o, A))  # Output: True

# Check if o is an instance of class B

print(isinstance(o, B))  # Output: False
```

Q-14 What relationship is appropriate for Course and Faculty?

Ans: The appropriate relationship between **Course** and **Faculty** in object-oriented programming (OOP) is generally an **association** or specifically an **aggregation**.

Explanation:

1. **Association**: This is a general relationship where one class (e.g., Faculty) can be related to another class (e.g., Course), but neither depends entirely on the other.
    - **Example**: A faculty member can teach multiple courses, and a course can have multiple faculty members teaching it. Both Course and Faculty are independent of each other.
2. **Aggregation** (a type of association): This implies a "whole-part" relationship, where the Course and Faculty are related, but each can exist independently.
    - **Example**: A Course can have multiple faculty members assigned to teach it, but the Faculty and Course can exist independently (i.e., removing a faculty member does not delete the course).

Q-15 What relationship is appropriate for Student and Person?

Ans: The appropriate relationship between Student and Person in object-oriented programming (OOP) is inheritance.

Explanation:

- Inheritance represents an "is-a" relationship. In this case, a Student "is a" Person, meaning that the Student class can inherit common properties and behaviors from the Person class, while also having its own specific attributes and methods.
- A Student is a more specific version of a Person, with additional attributes like student ID, grade, or enrollment status.