**Level 2- Python - Advance Python Programming**

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

In Python, file functions are used to create, read, write, and manipulate files. The key function used is open(), which allows you to interact with files.

Creating and Writing a File

To create and write to a file, you use the open() function with specific modes:

w: Write mode (creates a file if it doesn't exist or overwrites an existing file).

a: Append mode (creates a file if it doesn't exist or adds content to the end of an existing file).

x: Exclusive creation mode (creates a file but raises an error if it already exists).

Example:

# Create and write to a file

file = open("example.txt", "w") # Open the file in write mode

file.write("Hello, World!") # Write content to the file

file.close() #close

1. **Explain Exception handling? What is an Error in Python?**

**Exception Handling in Python**

Exception handling in Python is a way to manage runtime errors and ensure the program does not crash. It allows the program to handle unexpected situations gracefully.

Exceptions are events that occur during execution and disrupt the normal flow of the program.

Exception handling is done using the following keywords:

**try**: Contains the code that might raise an exception.

**except**: Contains the code to handle the exception if it occurs.

**else**: (Optional) Executes if no exception occurs in the try block.

**finally**: (Optional) Executes code regardless of whether an exception occurred or not.

Example:

try:

num = int(input("Enter a number: ")) # Code that might cause an exception

print("You entered:", num)

except ValueError: # Handles a specific exception

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

finally:

print("Execution complete.") # Always executes

**Error in Python**

An Error is a problem in the code that prevents it from running properly.

Errors can be categorized into:

i. Syntax Errors

ii. Exceptions (Runtime Errors)

iii. Logical Errors

1. **How many except statements can a try-except block have? Name Some built-in exception classes:When will the else part of try-except-else be executed?**
2. **Number of except statements**: A try block can have multiple except statements to handle different exceptions.
3. **Built-in Exception Classes**:

ValueError

TypeError

ZeroDivisionError

KeyError

IndexError

FileNotFoundError

AttributeError

1. **When else is executed**: The else block runs only if no exception occurs in the try block.
2. **Can one block of except statements handle multiple exception?**

**S**

Yes, a single except block can handle multiple exceptions by specifying them as a tuple.

1. **When is the finally block executed?**

The **finally** block is **always executed**, regardless of whether an exception occurred or not. It is used for cleanup actions like closing files or releasing resources

1. **What happens when „1‟== 1 is executed?**

When '1' == 1 is executed in Python, the result is **False**.

1. **How Do You Handle Exceptions With Try/Except/Finally In Python? Explain with coding snippets.**

In Python, you handle exceptions using the try-except-finally blocks. Here's how each part works:

**try**: Contains code that might raise an exception.

**except**: Handles specific or general exceptions.

**finally**: Executes code regardless of whether an exception occurred or not (used for cleanup tasks).

**Example:**

try:

num = int(input("Enter a number: ")) # Code that might raise an exception

result = 10 / num

print("Result:", result)

except ValueError: # Handles invalid input (e.g., entering text instead of a number)

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

except ZeroDivisionError: # Handles division by zero

print("Division by zero is not allowed!")

finally:

print("Execution complete. Cleaning up resources if any.")

1. **What are oops concepts? Is multiple inheritance supported in java**

**OOP Concepts:**

1. **Encapsulation**: Bundling data and methods, restricting access.
2. **Inheritance**: Child class inherits parent class properties.
3. **Polymorphism**: One method behaves differently for different classes.
4. **Abstraction**: Hiding implementation, showing essential details.
5. **Classes/Objects**: Blueprints and instances.

**Multiple Inheritance in Java:**

* **Not supported with classes** (avoids ambiguity).
* **Supported with interfaces**. Example:

1. **How to Define a Class in Python? What Is Self? Give An Example Of A Python Class**

**Defining a Class in Python:**

Use the class keyword to define a class.

**self:**

* Represents the **current instance** of the class.
* Used to access instance variables and methods.

**Example:**

class MyClass:

def \_\_init\_\_(self, name): # Constructor

self.name = name # Instance variable

def greet(self):

print(f"Hello, {self.name}!")

obj = MyClass("Alice") # Create object

obj.greet() # Output: Hello, Alice!

1. **Explain Inheritance in Python with an example?What is init? Or What is a Constructor in Python?**

**Inheritance in Python:**

Inheritance allows a class to inherit attributes and methods from another class (parent class).

**Example:**

class Animal: # Parent class

def speak(self):

print("Animal speaks")

class Dog(Animal): # Child class inheriting from Animal

def bark(self):

print("Dog barks")

dog = Dog()

dog.speak() # Inherited method

dog.bark() # Child class method

**\_\_init\_\_ (Constructor in Python):**

* **\_\_init\_\_** is a special method used to initialize object attributes when an object is created.
* It is called automatically when a class is instantiated.

1. **What is Instantiation in terms of OOP terminology?**

**Instantiation** in OOP refers to the process of creating an **instance** (object) of a class. When a class is instantiated, the constructor (\_\_init\_\_ in Python) is called, and memory is allocated for the object.

1. **What is used to check whether an object o is an instance of class A?**

In Python, the **isinstance()** function is used to check whether an object o is an instance of a specified class A.

1. **What relationship is appropriate for Course and Faculty?**

The appropriate relationship between **Course** and **Faculty** is typically a **"one-to-many"** relationship.

1. **What relationship is appropriate for Student and Person?**

The appropriate relationship between **Student** and **Person** is typically a **"is-a"** relationship, which is a form of **inheritance** in object-oriented programming.