**Advance Python Programming**

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

**Ans :** In Python, the term "File function" is not a specific function or concept. However, there are built-in functions and methods in Python's standard library that are used for working with files. These functions and methods allow you to perform various operations on files, such as creating, reading, writing, and managing file contents.

Here are some of the key functions and methods related to creating and writing to files:

**1. ‘open()’:** This built-in function is used to open a file. It takes a filename and a mode as arguments and returns a file object. The mode specifies the purpose of opening the file (read, write, append, etc.).

**2. ‘file.write()’:** This method is used to write data to a file. It takes a string as an argument and writes the content to the file. It can be used with a file object obtained from the `open()` function.

**3. ‘file.writelines()’:** This method is used to write a list of strings to a file. Each string in the list is written as a separate line in the file.

**4. ‘file.close()’:** This method is used to close an opened file. It's important to close files after you're done working with them to release system resources.

**Que-2:** **Explain Exception handling? What is an Error in Python?**

**Ans :** Exception handling is a programming concept used to manage and respond to unexpected situations or errors that may occur during the execution of a program. It provides a structured way to deal with these errors and prevent the program from abruptly terminating. Exception handling allows programmers to gracefully handle errors, communicate error messages, and take appropriate actions without disrupting the entire program flow.

In Python, errors are issues that prevent a program from executing correctly. Errors can be categorized into three main types:

**1. Syntax Errors:** These errors occur due to violations of the syntax rules of the programming language. They are detected by the Python interpreter during the parsing phase before the program runs.

**2. Runtime Errors (Exceptions):** These errors occur during the execution of a program when unexpected conditions or inputs are encountered. They disrupt the normal flow of the program and need to be handled to prevent the program from crashing.

**3. Logical Errors:** Also known as semantic errors, these errors occur when the program produces incorrect results due to flaws in the algorithm or incorrect implementation. The program runs without crashing, but the output is not what was intended.

Exception handling in Python involves using the **try, except, else,** and **finally** blocks. The **try** block contains the code that might raise an exception. If an exception occurs, the corresponding **except** block handles the error and defines how the program should respond. The optional **else** block runs if no exceptions occur in the **try** block. The **finally** block, if used, always runs, regardless of whether an exception occurred or not. It is typically used for cleanup operations.

In summary, exception handling is a programming practice that enables programs to respond to errors gracefully.

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

**Ans :** A **try** block in a **try-except** block can have multiple **except** statements to handle different types of exceptions. There is no strict limit to the number of **except** statements you can have, but keep in mind that each **except** statement will catch a specific type of exception, so it's a good practice to keep the number of **except** statements reasonable to avoid making the code too complex.

Here are some built-in exception classes in Python:

**1. ZeroDivisionError:** Raised when division or modulo operation is performed with zero as the divisor.

**2. ValueError:** Raised when a function receives an argument of the correct type but an invalid value.

**3. TypeError:** Raised when an operation or function is applied to an object of inappropriate type.

**4. IndexError:** Raised when an index is not found in a sequence (e.g., list, tuple, string).

**5. KeyError:** Raised when a dictionary key is not found.

**6. FileNotFoundError:** Raised when an attempt is made to open a file that does not exist.

**7. ImportError:** Raised when an imported module or name cannot be found.

**8. NameError:** Raised when a local or global name is not found.

**9. AttributeError:** Raised when an attribute reference or assignment fails.

**10. RuntimeError:** Raised when an error occurs that does not belong to any specific category.

**Que-4:** **When will the else part of try-except-else be executed?**

**Ans :** In a try-except-else block in Python, the **else** part is executed when no exceptions are raised within the **try** block.

**Que-5:** **Can one block of except statements handle multiple exception?**

**Ans :** Yes, a single block of **except** statements in Python can handle multiple exceptions. You can list multiple exception types inside a single **except** block, separated by commas. This allows you to handle different exceptions in a concise way using the same error-handling code.

**Exaple:**

**try:**

# Some code that might raise exceptions

**except (ValueError, KeyError):**

# Code to handle ValueError or KeyError

**Que-6:** **When is the finally block executed?**

**Ans :** The **finally** block in Python always runs, no matter if there was an exception or not. It's used for cleanup tasks that should happen regardless of the program's flow.

**Que-7:** **What happens when ‘’1’’== 1 is executed?**

**Ans :** When "1" == 1 is executed in Python, it will evaluate to ‘False’.

The reason is that "1" is a string literal containing the character '1', while ‘1’ is an integer. In Python, the ‘==’ operator checks for equality between values, but it also checks for type compatibility. Since the types of the operands are different (string and integer), the comparison will result in ‘False’.

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

**Ans :** In Python, you can handle exceptions using the try, except, and finally blocks. The try block contains the code that might raise an exception, the except block handles the exception, and the finally block allows you to perform cleanup tasks that should always run, regardless of whether an exception occurred or not. Here's an explanation with coding snippets:

**Example:**

**try:**

# Code that might raise an exception

num = int(input("Enter a number: "))

result = 10 / num

**except ZeroDivisionError:**

# Handling a specific exception (division by zero)

print("Error: Cannot divide by zero")

**except ValueError:**

# Handling another specific exception (invalid input)

print("Error: Invalid input, please enter a number")

**else:**

# Code to run if no exceptions occurred

print("Result:", result)

**finally**

# Cleanup code that always runs

print("Cleaning up...")

In this example:

- The **try** block contains code that takes user input, converts it to an integer, and performs a division operation.

- The **except ZeroDivisionError** block handles the specific exception when the user enters 0, causing a division by zero.

- The **except ValueError** block handles the exception when the user enters a non-integer value.

- The **else** block contains code that executes if no exceptions were raised (e.g., the user entered a valid number).

- The **finally** block contains code that always runs, regardless of whether an exception occurred or not.

This structure allows you to gracefully handle exceptions and ensure that necessary cleanup tasks are performed, promoting better program reliability.

**Que-9:** **How to Define a Class in Python? What Is Self? Give An Example Of A Python Class.**

**Ans :** In Python, a class is defined using the `class` keyword. A class is a blueprint for creating objects that share common attributes and methods. It encapsulates data and behavior into a single unit.

The keyword `self` is used inside methods within a class to refer to the instance of the class itself. It's a way to access and manipulate the instance's attributes and methods.

**Here's an example of how to define and use a simple Python class:**

class Dog:

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

def bark(self):

print(f"{self.name} barks!")

# Creating an instance of the Dog class

my\_dog = Dog("Buddy", 3)

# Accessing attributes and calling methods

print(f"My dog's name is {my\_dog.name}.")

print(f"My dog is {my\_dog.age} years old.")

my\_dog.bark()

**Que-10:** **Explain Inheritance in Python with an example? What is init? Or What Is A Constructor In Python?**

**Ans :** **Inheritance in Python:**

Inheritance is a fundamental concept in object-oriented programming (OOP) that allows you to create a new class by inheriting attributes and methods from an existing class. The new class, called the derived or subclass, can extend or modify the behavior of the existing class, called the base or superclass. Inheritance promotes code reuse and supports the hierarchy of classes.

**Here's an example of inheritance in Python:**

class Animal:

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

self.name = name

def speak(self):

pass

class Dog(Animal):

def speak(self):

return f"{self.name} barks!"

class Cat(Animal):

def speak(self):

return f"{self.name} meows!"

dog = Dog("Buddy")

cat = Cat("Whiskers")

print(dog.speak()) # Output: Buddy barks!

print(cat.speak()) # Output: Whiskers meows!

**In this example:**

- The ‘Animal’ class serves as the base class with an ‘\_\_init\_\_’ method that initializes the ‘name’ attribute. It also has a ‘speak’ method that's not implemented (indicated by ‘pass’).

- The ‘Dog’ and ‘Cat’ classes are subclasses of ‘Animal’. They inherit the ‘\_\_init\_\_’ method and can override the ‘speak’ method to provide their own implementations.

- The ‘Dog’ class overrides the ‘speak’ method to make the dog bark.

- The ‘Cat’ class overrides the ‘speak’ method to make the cat meow.

- Instances of ‘Dog’ and ‘Cat’ can access the ‘name’ attribute inherited from `Animal` and call the overridden ‘speak’ method.

**Constructor in Python:**

In Python, the ‘\_\_init\_\_’ method is a special method, also known as a constructor. It's automatically called when an object of a class is created. The purpose of the constructor is to initialize the attributes of the object. It's used to set up the initial state of the object.

**Here's a brief explanation of the ‘\_\_init\_\_’ method with an example:**

class MyClass:

def \_\_init\_\_(self, attribute):

self.attribute = attribute

obj = MyClass("Hello")

print(obj.attribute) # Output: Hello

**In this example:**

- The ‘MyClass’ class has an ‘\_\_init\_\_’ method that takes an ‘attribute’ parameter and assigns it to the instance variable ‘self.attribute’.

- When an instance of ‘MyClass’ is created (‘obj = MyClass("Hello")’), the ‘\_\_init\_\_’method is automatically called, initializing the ‘attribute’ of the object.

- You can access the `attribute` using the object (‘obj.attribute’).

**Que-11:** **What is Instantiation in terms of OOP terminology?**

**Ans :** Instantiation in terms of object-oriented programming (OOP) refers to the creation of an instance (object) of a class. It involves creating a specific object that follows the structure and behavior defined by the class blueprint.

**Que-12:** **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 class ‘A’, you can use the ‘isinstance()’ function in Python. The ‘isinstance()’ function returns ‘True’ if the object is an instance of the specified class or a subclass thereof, and ‘False’ otherwise.

**Que-13: What relationship is appropriate for Course and Faculty?**

**Ans :** The appropriate relationship between the "Course" and "Faculty" entities can be described as a “many-to-many” relationship. This is because a single course can be taught by multiple faculty members, and a faculty member can teach multiple courses.

In a many-to-many relationship, each instance of one entity (e.g., a course) can be associated with multiple instances of another entity (e.g., faculty), and vice versa. To implement this relationship, you would typically use an intermediary entity, often referred to as a “junction table” or “association table”.

Here's a simple representation:

- **Course:** Represents a course offered at the institution.

- **Faculty:** Represents a faculty member.

- **TeachingAssignment:** Junction table that stores the assignments of faculty to courses.

This design allows you to handle the complexity of a many-to-many relationship while maintaining data integrity. Each teaching assignment entry in the junction table would link a specific faculty member to a specific course they are teaching.

In a more detailed implementation, each record in the junction table might also include additional information such as the semester, year, or any other relevant details.

Overall, a many-to-many relationship with a junction table is an appropriate approach for modeling the relationship between courses and faculty members in an educational context.

**Que-14:** **What relationship is appropriate for Student and Person?**

**Ans :** The appropriate relationship between "Student" and "Person" is “inheritance”. This means that "Student" can inherit the attributes and methods of the broader "Person" class. In other words, a "Student" is a specialized form of "Person" sharing common characteristics while also having additional attributes or behaviors unique to students.