Module – 3

(Advance python programming)

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

🡪 In python, the file function generally refers to the functions and methods used to work with files, allowing for reading from and writing to files on the filesystem.

1.Opening file:

Use the open() function.

Syntax:

file = open(‘filename’, ’mode’)

2.File modes:

‘r’ – read mode

‘w’ – write mode

‘a’ – append mode

3.Writing to a file

with open(‘filename’, ’w’) as file:

file.write(“Hello world”)

4.Closing a file

file.close()

(2) Explain Exception handling? What is an Error in Python?

🡪 When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

🡪 By using try, except, else, and finally blocks, you can control how your program responds to these errors.

Structure:

try:

# Code that might raise an exception

except ExceptionType1:

# Code to handle ExceptionType1

else:

#Code to execute if  no exceptions occur

finally:

# Code that always executes, regardless of exceptions

How it works:

1.try: Encloses the code that might raise an exception.

2. except :Define specific exception types to catch.

3. else: Executes if no exceptions occur in the try block.

4. finally: Always executes, regardless of whether an exception occurs or not.

Errors in python:

1.Syntax Errors:

* Occur when the code violates Python's grammar rules.
* Examples: missing parentheses, incorrect indentation, misspelled keywords.

2.Runtime Errors (Exceptions):

* Occur while the program is running.
* Examples:
* ZeroDivisionError: Division by zero
* TypeError: Incorrect data type operation
* ValueError: Invalid value for a certain operation
* IndexError: Index out of range FileNotFoundError: File not found

(3) How many except statements can a try-except block have? Name Some built-in exception classes.

🡪 A try-except block can have multiple except statements.

🡪 Each except statement can handle a specific exception type or a general exception. This allows you to handle different types of errors differently.

Some Built-in Exception Classes:

**1.ZeroDivisionError:** Raised when you try to divide a number by zero.

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

**3.ValueError:** Raised when a function or operation receives an argument with an inappropriate value.

**4.IndexError:** Raised when you try to access an index of a sequence (like a list or tuple) that is out of range.

**5.ImportError:** Raised when an import statement fails.

**6.NameError:** Raised when you try to access a variable that has not been defined.

**7.SyntaxError:** Raised when the Python interpreter encounters invalid syntax.

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

🡪 The else part of a try-except-else block will be executed only if no exceptions are raised within the try block.

def divide(x, y):

try: result = x / y

except ZeroDivisionError:

print("Error: Division by zero")

else:

print("Result:", result)

divide(10,2) #Output : Result : 5.0

divide(10,0) #Output : Error: Division by zero

(5) Can one block of except statements handle multiple exception?

🡪 Yes, a single except block can handle multiple exceptions. You can list the exception types you want to catch within parentheses:

try:

#code

except(TypeError, ValueError, ZeroDivisionError):

print(“Error: An exception occurred”)

In this example, the except block will handle TypeError, ValueError, and ZeroDivisionError.

(6) When is the finally block executed?

🡪 The finally block in Python is always executed at the end of a try-except structure, no matter what happens in the try or except blocks.

(7) What happens when ‘’1’’== 1 is executed?

🡪 Data type mismatch:

'1' is a string, representing a character sequence.

1 is an integer, representing a numerical value.

**Inequality:** Since the two operands are of different data types, they are considered unequal, and the comparison returns False.

🡪 "1" == 1 will always result in False.

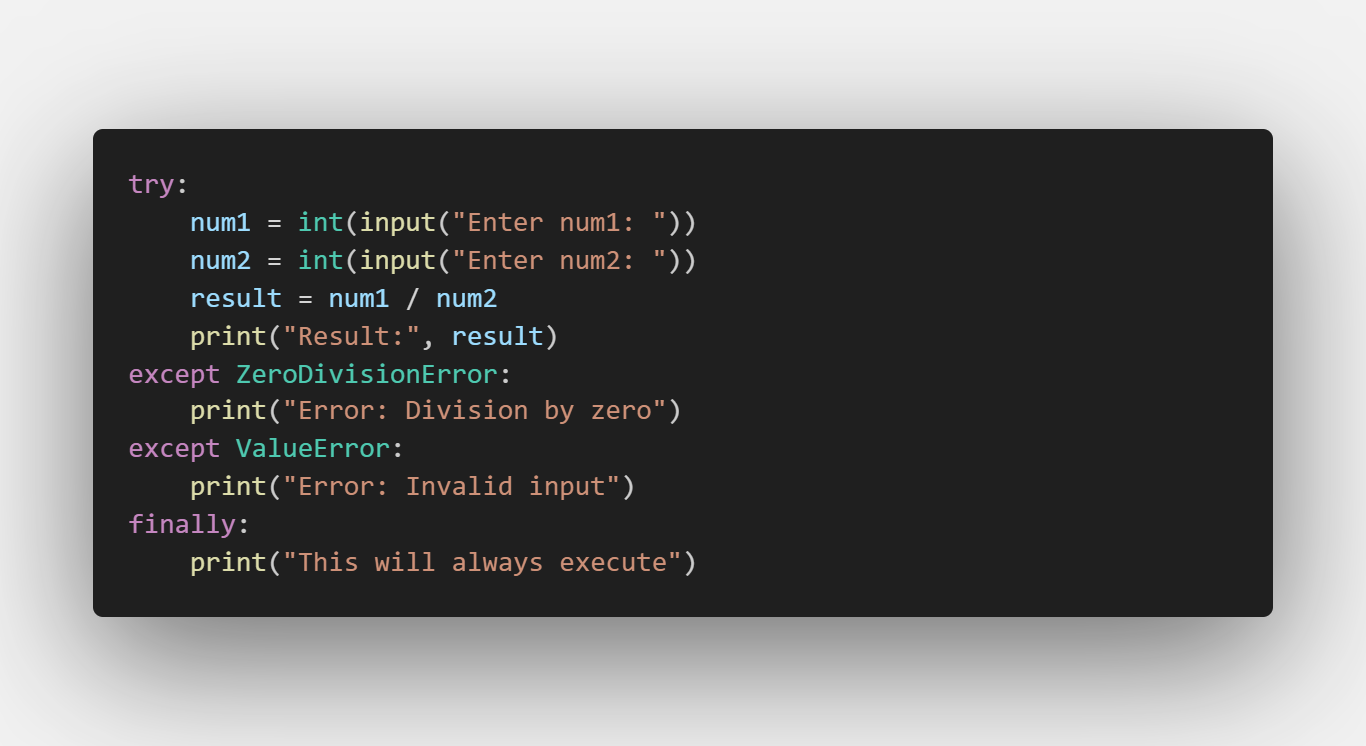
(8) How Do You Handle Exceptions with Try/Except/Finally in python? Explain with coding snippets.

🡪 In Python, exceptions are handled using the try, except, and finally blocks. Here's how these blocks work together:

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

**2.except Block**: Catches and handles exceptions raised in the try block.

**3.finally Block**: Always executes, regardless of whether an exception occurred or was handled, making it useful for cleanup operations.



**Explanation:**

**try Block:**

Prompts the user for two numbers and attempts to divide them.

**except Blocks:**

The first except block handles ZeroDivisionError if the user enters 0 as the divisor.

The second except block handles ValueError if the user enters non-numeric input.

**finally Block:**

Prints a message regardless of whether an exception occurs or not.

This is useful for cleanup operations, but in this case, it simply prints a message.

(9) What are oops concepts? Is multiple inheritance supported in java?

🡪 Object-oriented programming (OOP) is a computer programming model that organizes software design around objects, rather than functions and logic.

It promotes reusability, scalability, and modularity in code development.

*Is multiple inheritance supported in java?*

🡪 No, Java does not directly support multiple inheritance.

This means a class cannot inherit from multiple parent classes.

**Why?**

1.**Diamond Problem:** The diamond problem arises when a class inherits from two parent classes, both of which have a method with the same name.

This causes ambiguity in which method the child class should inherit.

2.**Complexity:** Multiple inheritance can make code more complex and harder to maintain.

(10) How to define a class in Python? What is self? Give an example of a python class.

🡪 In Python, a class is defined using the class keyword followed by the class name.

Example:

class A:

    def \_\_init\_\_(self, name):

        self.name = name

    def print\_name(self):

        print(f"{self.name}")

obj = A("Mihir")

obj.print\_name()

*What is self?*

🡪 self is a reference to the current instance of the class.

🡪 It is used to access attributes and methods of the class in Python.

🡪 self must be the first parameter of instance methods.

(11) Explain inheritance in python with an example? What is init? Or What is a constructor in python?

🡪 Inheritance allows us to define a class that inherits all the methods and properties from another class.

**Types of Inheritance in Python**

**1.Single Inheritance**: A child class inherits from one parent class.

**2.Multiple Inheritance**: A child class inherits from more than one parent class.

**3.Multilevel Inheritance**: A class inherits from a child class, forming a chain.

**4.Hierarchical Inheritance**: Multiple child classes inherit from one parent class.

**5.Hybrid Inheritance**: A mix of two or more types of inheritance.

**Ex**:

# Parent class

class A:

    def \_\_init\_\_(self, name):

        self.name = name

        print("Base class")

# Child class

class B(A):

    def name\_print(self):

        print(f"{self.name}")

# Create an object of the child class

obj = B("Kheshav")

obj.name\_print()

**Explanation:**

1.Base class (A):The ‘A’ class defines a common constructer.

2.Derived Class (B):

The B class inherits from the A class.

It inherits the name attribute.

It also defines its own specific method name\_print().

*What is init?*

🡪 \_\_init\_\_ is a constructor in Python.

🡪 It is a special method automatically invoked when a new object of a class is created.

🡪 Used to initialize the object's attributes with specific values.

(12) What is Instantiation in terms of OOP terminology?

🡪 In Object Oriented Programming, instantiation is the process of creating an object from a class.

🡪 A class serves as a blueprint that defines the attributes and methods of objects

Syntax of Instantiation:

**object\_name = className(arguments)**

* object\_name: The variable that references the new object.
* className: The name of the class.
* arguments: Optional parameters to initialize the object, passed to the \_\_init\_\_ method.

Example:

class Car:

def \_\_init\_\_(self, color, model):

self.color = color

self.model = model

def start(self):

print("Car started")

def stop(self):

print("Car stopped")

# object

obj = Car("red", "volkswagen")

(13) What is used to check whether an object o is an instance of class A?

🡪 Python provides the built-in function isinstance().

**How isinstance() Works**

🡪 Returns True if object is an instance of the specified class or any subclass of it.

🡪Returns False otherwise.

class A:

    pass

class B(A):

    pass

o = B()

# True, because B is a subclass of A

print(isinstance(o, A))

# True, because o is an instance of B

print(isinstance(o, B))

# True, because everything in Python is an instance of object

print(isinstance(o, object))

class C:

pass

# True, because o is an instance of A (or its subclass)

print(isinstance(o, (A, C)))

d = C()

print(isinstance(d, A))

# False, because d is not an instance of A or its subclass

(14) What relationship is appropriate for Course and Faculty?

🡪 Composition is a stronger form of association where one object owns another.

🡪 In this case, a Course could be composed of a Faculty member, meaning the Course cannot exist without a Faculty member.

class Faculty:

    def \_\_init\_\_(self, name, id):

        self.name = name

        self.id = id

class Course(Faculty):

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

        self.name = name

        self.code = code

        self.faculty = faculty

(15) What relationship is appropriate for Student and Person?

🡪 one class (subclass or child class) inherits the properties and methods of another class (superclass or parent class).

In this case, a Student is a specific type of Person, so it makes sense to inherit the common attributes and behaviors of a Person.

class Person:

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

        self.name = name

        self.age = age

class Student(Person):

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

        super().\_\_init\_\_(name, age)

        self.student\_id = student\_id

    def display(self):

print(f"Name : {self.name}, Age : {self.age}, Student id : {self.student\_id}")

obj = Student("Mihir",20,1)

obj.display()

The Person class is the base class, defining common attributes like name and age.

The Student class inherits from the Person class, inheriting the name and age attributes.

The Student class also adds its own specific attribute, student\_id, and a method, display().