Exception Handling Class 12 Important Questions

Exception Handling Class 12 Important Questions

1. “Every syntax error is an exception but every exception cannot be a syntax error.” Justify the statement.

Answer: A syntax error occurs when code does not follow the rules of a programming language, but every exception cannot be a syntax error because there are several types of errors that can be encountered during the execution of the program, such as division by zero or file not found errors. These errors are not syntax errors.

2. When are the following built-in exceptions raised? Give examples to support your answers.
a) ImportError
b) IOError
c) NameError
d) ZeroDivisionError

Answer:

- **ImportError:** It is raised when the requested module definition is not found.
- **IOError:** It is raised when the file specified in a program statement cannot be opened.
- **NameError:** It is raised when a local or global variable name is not defined.
- **ZeroDivisionError:** It is raised when the denominator in a division operation is zero.

3. What is the use of a raise statement? Write a code to accept two numbers and display the quotient. Appropriate exception should be raised if the user enters the second number (denominator) as zero (0).

Answer: This raise statement is used to manually trigger an exception. The raise statement in Python is used when a program triggers an exception for a specific error, which helps to prevent unexpected crashes.

Write a code to accept two numbers and display the quotient.

```
try:
    num1 = float(input("Enter numerator: "))
    num2 = float(input("Enter denominator: "))

    if num2 == 0:
        raise ZeroDivisionError("Denominator cannot be zero.")

    print("Quotient:", num1 / num2)

except ZeroDivisionError as e:
    print("Error:", e)
```

4. Use assert statement in Question No. 3 to test the division expression in the program.

Answer: The assert statement takes a boolean expression and checks if a condition in your code returns True. If not, then the program will raise an AssertionError. We can use an assert statement in the division program to check whether the denominator is not zero before performing the division.

```
try:
    num1 = float(input("Enter numerator: "))
    num2 = float(input("Enter denominator: "))
```

```
        assert num2 != 0, "Denominator cannot be zero."

        print("Quotient:", num1 / num2)

    except AssertionError as e:
        print("Error:", e)
```

5. Define the following:
a) Exception Handling
b) Throwing an exception c) Catching an exception

Answer:

- **Exception Handling:** Exception handling is a method of managing errors or unexpected conditions which occur during the program execution. It prevents the program from crashing using try-except in Python.
- **Throwing an exception:** Throwing an exception refers to explicitly triggering an error using a raise statement.
- **Catching an exception:** Catching an exception means handling an error using a try-except block. It prevents the program from crashing.

6. Explain catching exceptions using try and except block.

Answer: The try clause contains statements that can raise exceptions, and the except clause contains statements that handle the exception. The try block is always followed by a catch block, which can handle the exception. The try block contains code that may produce an error, and if the try block finds any exception happens, then Python jumps to the except block and executes the error.