Module 9: Exception Handling

CHAPTER 1: What is an Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Error in Python can be of two types i.e., Syntax errors and Exceptions. Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.

In Python, there are several built-in exceptions that can be raised when an error occurs during the execution of a program. Here are some of the most common types of exceptions in Python:

- **SyntaxError:** This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.
- **TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
- **NameError**: This exception is raised when a variable or function name is not found in the current scope.
- **IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence type.
- **KeyError**: This exception is raised when a key is not found in a dictionary.
- **ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.
- **AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.

- **IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.
- **ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.
- **ImportError**: This exception is raised when an import statement fails to find or load a module.

CHAPTER 2: Handling an Exception.

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

TRY_EXCEPT: To handle possible exceptions, we use a try-except block. The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped. If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message. You can specify the type of exception after the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate. The rest of the statements after the except block will continue to be executed, regardless if the exception is encountered or not. You can mention a specific type of exception in front of the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

ELSE_FINALLY: There are two other clauses that we can add to a try-except block: else and finally. else will be executed only if the try clause doesn’t raise an exception. In Python, keywords else and finally can also be used along with the try and except clauses. While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free. The finally block consists of statements which should be processed regardless of an exception occurring in the try block or not. As a consequence, the error-free try block skips the except clause and enters the finally block before going on to execute the rest of the code. If, however, there's an exception in the try block, the appropriate except block will be processed, and the statements in the finally block will be processed before proceeding to the rest of the code. If you can follow this as well as you would like to, don't worry I will demonstrate these practically soon.

RAISE: Python also provides the raise keyword to be used in the context of exception handling. It causes an exception to be generated explicitly. Built-in errors are raised implicitly. However,

a built-in or custom exception can be forced during execution.you can define your custom exception type to be raised. What i mean by this is. If a condition does not meet our criteria but is correct according to the Python interpreter, we can intentionally raise an exception using the raise keyword. We can use a customized exception in conjunction with the statement.

ASSERT: In Python, the assert statement is used to continue the execute if the given condition evaluates to True. If the assert condition evaluates to False, then it raises the Assertion Error exception with the specified error message. The syntax is assert condition [, Error Message]. The assert statement can optionally include an error message string, which gets displayed along with the Assertion Error. When we're finished verifying the program, an assertion is a consistency test that we can switch on or off.

CHAPTER 3: User-defined Exception

Programs may name their own exceptions by creating a new exception class. Exceptions should typically be derived from the Exception class, either directly or indirectly.

Exception classes can be defined which do anything any other class can do, but are usually kept simple, often only offering a few attributes that allow information about the error to be extracted by handlers for the exception.

To create a User-defined Exception, we have to create a class that implements the Exception class. Exceptions need to be derived from the Exception class, either directly or indirectly. Although not mandatory, most of the exceptions are named as names that end in “Error” similar to the naming of the standard exceptions in python.

Our Exception class should Implement Exceptions to raise exceptions. In this exception class, we can either put pass or give an implementation. we can define init function. class JustException(Exception):

```
def _init_(self, message):
    print(message)
```

To raise a User-defined Exception, we can do the following:

```
raise JustException("Raise an Exception")
```

The raise keyword raises the exception mentioned after it. In this case, JustException is raised. In the output Raise an Exception will be there with a Traceback (most recent call last) and the message we passed to the _init_ class will be displayed in the exception and output.