Exception Handling

Session 11

Session Overview

In this session, you will be able to:

- Describe the importance of exception handling
- Identify the built-in exceptions in Python
- Explain the procedure of raising and catching exceptions in Python
- Explain the procedure of creating and raising user-defined exceptions in Python
- Describe the use of assertions in Python

Exception Handling Overview

Abnormality that occurs at the time of execution in a program.

Exception has to be handled with a dedicated construct.

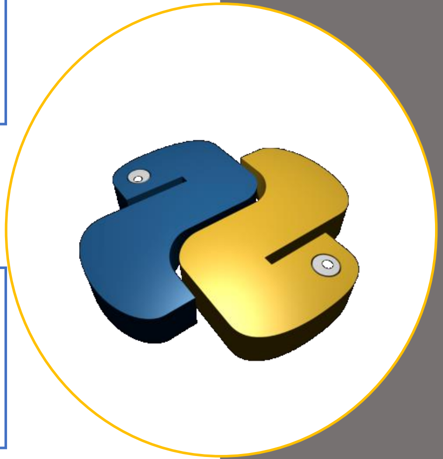
In case it is not handled, it does not allow to proceed the application and it is interrupted.

Need of Exception Handling

Handling an error or exception becomes essential while dealing with a significant number of variables.

Developer does not know in advance what will be the user input to the executing code.

In this case, the developer acts rationally by implementing exceptions and error handling wherever applicable.



Basics of Exception Handling in Python

In Python and Java, handling exceptions is analogous.

The part of code that is likely to raise an exception is put inside the `try` block, which is associated with the exception handler.

Unlike Java in which the `catch` clause catches an exception; in Python, the `except` keyword is used for doing so.

Roles of Exceptions in Python (1-2)

Exceptions in Python are frequently used for different purposes.

- Error Handling
- Special-case Handling
- Event Notifications
- Termination Actions
- Unusual Control Flows

Roles of Exceptions in Python (2-2)

- Python automatically raises exceptions when errors occur.
- Exceptions can be intercepted and handled through the codes.

try/except

try/finally

raise

assert

with/as

Default Exception Handler

In Python, exceptions are relatively lightweight as compared to a few core languages.

The language invokes the default exception handler in case a developer does not code a dedicated exception handling construct.

When a runtime error occurs, Python generates an exception object.

Built-in Exceptions (1-2)

Exceptions as handled by Python:

Exception	StopIterat ion	SystemExit	StandardEr ror
Arithmetic Error	OverflowEr ror	FloatingPo intError	ZeroDiviso nError
AssertionE rror	AttributeE rror	EOFError	ImportErro r

Built-in Exceptions (2-2)

KeyboardInterrupt	LookupError	IndexError	KeyError
NameError	UnboundLocalError	EnvironmentError	IOError
SyntaxError	IndentationError	SystemError	SystemExit
TypeError	ValueError	RuntimeError	NotImplementedError

Catching Exceptions

Invoking the default exception handler is not preferred by developers.

In case of server programs, it is essential for them to run even though an internal error has occurred.

Using `try`, `except`, and `else` Statements (1-2)

- Following is the syntax of all three statements applicable to Python 3.x:

```
Syntax
try: statements # Run this main action first
except name1: statements # Run if name1 is raised during try
block
except (name2, name3):statements # Run if any of these
exceptions occur
except name4 as var: statements # Run if name4 is raised,
assign instance raised to var
except: statements # Run for all other exceptions raised
else:
statements # Run if no exception occured during try block
```

Using `try`, `except`, and `else` Statements (2-2)

- Following are important points about the `try/except/else` syntax:

Multiple `except` clauses are useful for one `try` statement when `try` block contains code that is likely to throw different exceptions.

A generic `except` clause can handle any kind of exception.

The `else` clause always follows the `except` clause.

The `else` block should contain code that does not require `try` block's protection.

Multiple `except` Clauses

- An exception must be specified after the `except` keyword while using the `except` clause.
- This is because it will catch any exception and handle it in the same way.
- `ValueError` exception cannot be handled in the same way as `IOError` exception.
- Thus, it is better to specify which exception the clause shall handle.
- If a code is likely to trigger more than one exception, it is ideal to have two `except` clauses.

`finally` Clause

- It is not necessary to pair a try statement with one or more except clauses.
- This can also be done by using `finally` clause.
- This clause is a clean-up or termination clause, as it executes under all circumstances.
- This clause executes irrespective of whether an exception occurred or not.

Raising Exceptions Using `raise` Statements

- The syntax of raise keyword is followed optionally by the class or its instance to be raised. Following is its syntax:

Syntax

```
raise instance  # Raise instance of class
raise class # Make and raise instance of class
raise    # Re-raise the most recent exception
raise [Exception [, args [, traceback]]]
#General syntax
```

User-defined Exceptions

Developers can define their own exceptions by writing a new exception class.

This new class has to be a child class of `Exception` class, either directly or indirectly.

Although not compulsory, most exceptions have a name that ends with ‘Error’, just as those built-in exceptions in Python.

Deriving Error from Super Class Exceptions

Super class exception is useful for enabling a module to handle multiple, but unique errors.

To do so, define a base class for exceptions that the module defines.

Also, define different subclasses for creating specific exception classes for diverse errors.

Python Assertions

A lucid check that a developer can opt to turn ON when program testing is over is called as assertion.

It helps in testing an expression.

Assertion raises an exception in case the outcome is false.

Assertions are executed using the `assert` statement in Python 1.5.

Summary (1-4)

- Errors that take place during runtime are termed as exceptions.
- An exception refers to an event acting as an abnormal condition in a program at runtime and abruptly stops it execution.
- The part of code that is likely to raise an exception is put inside the `try` block, which is associated with the exception handler, `except`.

Summary (2-4)

- When a runtime error occurs, Python generates an exception object corresponding to its built-in exception class.
- The `try` and `finally` statements allow taking the desired closing-time actions, irrespective of whether exceptions exist or not in the program.
- It is useful to have multiple `except` clauses for a single `try` statement when the `try` block contains code that is likely to throw different exceptions.

Summary (3-4)

- The `else` clause always follows the `except` clause. It executes only when the `try` block does not trigger any exception.
- The optional `finally` clause is a clean-up or termination clause, as it executes under all circumstances and is useful for releasing resources.
- It is possible to raise an exception explicitly using the `raise` keyword.

Summary (4-4)

- Python allows defining one's own exceptions by writing a new exception class that is derived from the `Exception` class, either directly or indirectly.