EXCEPTION HANDLING

Exception can be said to be any abnormal condition in a program resulting to the disruption in the flow of the program.

Whenever an exception occurs the program halts the execution and thus further code is not executed. Thus exception is that error which python script is unable to tackle with.

Exception in a code can also be handled. In case it is not handled, then the code is not executed further and hence execution stops when exception occurs.

**Hierarchy Of Exception:**

1. ZeroDivisionError: Occurs when a number is divided by zero.
2. NameError: It occurs when a name is not found. It may be local or global.
3. IndentationError: If incorrect indentation is given.
4. IOError: It occurs when Input Output operation fails.
5. EOFError: It occurs when end of file is reached and yet operations are being performed

etc..

Exception Handling:

The suspicious code can be handled by using the try block. Enclose the code which raises an exception inside the try block. The try block is followed except statement. It is then further followed by statements which are executed during exception and in case if exception does not occur.

**Syntax:**

**try**:

    malicious code

**except** Exception1:

    execute code

**except** Exception2:

    execute code

....

....

**except** ExceptionN:

    execute code

**else**:

    In case of no exception, execute the **else** block code.

## Finally Block:

**In case if there is any code which the user want to be executed, whether exception occurs or not then that code can be placed inside the finally block. Finally block will always be executed irrespective of the exception.**

Declaring Multiple Exception

Multiple Exceptions can be declared using the same except statement:

**Syntax:**

**try**:

    code

**except** Exception1, Exception2 , Exception3,..,ExceptionN

    execute this code **in** case any Exception of these occur.

**else**:

    execute code **in** case no exception occurred.

Raise an Exception:

You can explicitly throw an exception in Python using ?raise? statement. raise will cause an exception to occur and thus execution control will stop in case it is not handled.

**Syntax:**

**raise** Exception\_class,<value>

## Custom Exception:

Creating your own Exception class or User Defined Exceptions are known as Custom Exception.

List of Standard Exceptions −

|  |  |
| --- | --- |
| **EXCEPTION NAME** | **DESCRIPTION** |
| Exception | Base class for all exceptions |
| StopIteration | Raised when the next() method of an iterator does not point to any object. |
| SystemExit | Raised by the sys.exit() function. |
| StandardError | Base class for all built-in exceptions except StopIteration and SystemExit. |
| ArithmeticError | Base class for all errors that occur for numeric calculation. |
| OverflowError | Raised when a calculation exceeds maximum limit for a numeric type. |
| FloatingPointError | Raised when a floating point calculation fails. |
| ZeroDivisionError | Raised when division or modulo by zero takes place for all numeric types. |
| AssertionError | Raised in case of failure of the Assert statement. |
| AttributeError | Raised in case of failure of attribute reference or assignment. |
| EOFError | Raised when there is no input from either the raw\_input() or input() function and the end of file is reached. |
| ImportError | Raised when an import statement fails. |
| KeyboardInterrupt | Raised when the user interrupts program execution, usually by pressing Ctrl+c. |
| LookupError | Base class for all lookup errors. |
| IndexError  KeyError | Raised when an index is not found in a sequence.  Raised when the specified key is not found in the dictionary. |
| NameError | Raised when an identifier is not found in the local or global namespace. |
| UnboundLocalError  EnvironmentError | Raised when trying to access a local variable in a function or method but no value has been assigned to it.  Base class for all exceptions that occur outside the Python environment. |
| IOError  IOError | Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.  Raised for operating system-related errors. |
| SyntaxError  IndentationError | Raised when there is an error in Python syntax.  Raised when indentation is not specified properly. |
| SystemError | Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. |
| SystemExit | Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. |
| TypeError | Raised when an operation or function is attempted that is invalid for the specified data type. |
| ValueError | Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. |
| RuntimeError | Raised when a generated error does not fall into any category. |
| NotImplementedError | Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |

### **Assertions in Python**

An assertion is a sanity-check that you can turn on or turn off when you are done with your testing of the program.

The easiest way to think of an assertion is to liken it to a **raise-if** statement (or to be more accurate, a raise-if-not statement). An expression is tested, and if the result comes up false, an exception is raised.

Assertions are carried out by the assert statement, the newest keyword to Python, introduced in version 1.5.

Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

### **The *assert* Statement**

When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true. If the expression is false, Python raises an *AssertionError* exception.

The syntax for assert is −

assert Expression[, Arguments]

If the assertion fails, Python uses ArgumentExpression as the argument for the AssertionError. AssertionError exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback.