

TEAM ID	PNT2022TMID41762
TOPIC	Exception Handling

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them –

- **Exception Handling** – This would be covered in this tutorial. Here is a list of standard Exceptions available in Python: [Standard Exceptions](#).
- **Assertions** – This would be covered in [Assertions in Python](#) tutorial.

List of Standard Exceptions –

S.No	Exception Name & Description
1	Exception Base class for all exceptions
2	StopIteration Raised when the next() method of an iterator does not point to any object.
3	SystemExit Raised by the sys.exit() function.
4	StandardError Base class for all built-in exceptions except StopIteration and SystemExit.
5	ArithmeticError Base class for all errors that occur for numeric calculation.
6	OverflowError Raised when a calculation exceeds maximum limit for a numeric type.
7	FloatingPointError Raised when a floating point calculation fails.

8	ZeroDivisionError Raised when division or modulo by zero takes place for all numeric types.
9	AssertionError Raised in case of failure of the Assert statement.
10	AttributeError Raised in case of failure of attribute reference or assignment.
11	EOFError Raised when there is no input from either the raw_input() or input() function and the endfile is reached.
12	ImportError Raised when an import statement fails.
13	KeyboardInterrupt Raised when the user interrupts program execution, usually by pressing Ctrl+c.
14	LookupError Base class for all lookup errors.
15	IndexError Raised when an index is not found in a sequence.
16	KeyError Raised when the specified key is not found in the dictionary.
17	NameError Raised when an identifier is not found in the local or global namespace.

N

18	UnboundLocalError Raised when trying to access a local variable in a function or method but no value has been assigned to it.
19	EnvironmentError Base class for all exceptions that occur outside the Python environment.
20	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.
21	IOError Raised for operating system-related errors.
22	SyntaxError Raised when there is an error in Python syntax.
23	IndentationError Raised when indentation is not specified properly.
24	SystemError Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
25	SystemExit Raised when Python interpreter is quit by using the sys.exit() function. If not handled in code, causes the interpreter to exit.
26	TypeError Raised when an operation or function is attempted that is invalid for the specified data type.

27	ValueError Raised when the built-in function for a data type has the valid type of arguments, but arguments have invalid values specified.
28	RuntimeError Raised when a generated error does not fall into any category.
29	NotImplementedError Raised when an abstract method that needs to be implemented in an inherited class is 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.

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.

What is 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.

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.