**Exception Handling in Python**

Errors in Python can be of two types

1. [Syntax errors - Errors are the problems in a program due to which the program will stop the execution.](https://www.geeksforgeeks.org/errors-and-exceptions-in-python/)
2. [Exceptions](https://www.geeksforgeeks.org/errors-and-exceptions-in-python/) - Exceptions are raised when some internal events occur which changes the normal flow of the program.

**Syntax Errors :**

# initialize the Salary variable

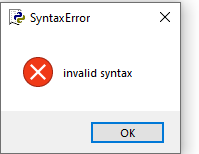
salary = 10000

# check that You are eligible to

#  purchase or not

if(salary>2999)

    print("You are eligible to purchase")



It returns a syntax error message because after the if statement a colon: is missing. We can fix this by writing the correct syntax.

**Correct program :**

# initialize the Salary variable

salary = 10000

# check that You are eligible to

#  purchase or not

if(salary>2999):

    print("You are eligible to purchase")



**Exceptions :**

* When in the runtime an error that occurs after passing the syntax test is called exception or logical type.
* For example, when we divide any number by zero then the ZeroDivisionError exception is raised, or when we import a module that does not exist then ImportError is raised.

**Example :**

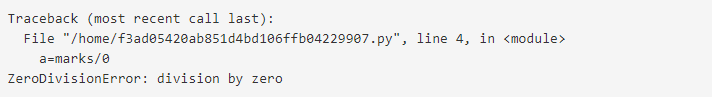
# initialize the amount variable

marks = 10000

# perform division with 0

a = marks / 0

print(a)



| **Exception** | **Description** |
| --- | --- |
| IndexError | When the wrong index of a list is retrieved. |
| AssertionError | It occurs when the assert statement fails |
| AttributeError | It occurs when an attribute assignment is failed. |
| ImportError | It occurs when an imported module is not found. |
| KeyError | It occurs when the key of the dictionary is not found. |
| NameError | It occurs when the variable is not defined. |
| MemoryError | It occurs when a program runs out of memory. |
| TypeError | It occurs when a function and operation are applied in an incorrect type. |

**Error Handling**

When an error and an exception are raised then we handle that with the help of the Handling method.

* **Handling Exceptions with Try/Except/Finally**   
  We can handle errors by the Try/Except/Finally method. we write unsafe code in the try, fall back code in except and final code in finally block.
* **Raising exceptions for a predefined condition**   
  When we want to code for the limitation of certain conditions then we can raise an exception.

## Try and Except Statement – Catching Exceptions

Try and except statements are used to catch and handle exceptions in Python.

Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

**Syntax:**

try:

# Some Code

except:

# Executed if error in the

# try block

**Example :**

# Python program to handle simple runtime error

#Python 3

a = [1, 2, 3]

try:

    print ("Second element = %d" %(a[1]))

    # Throws error since there are only 3 elements in array

    print ("Fourth element = %d" %(a[3]))

except:

    print ("An error occurred")

**Output :**

Second element = 2

An error occurred

## Catching Specific Exception

A try statement can have more than one except clause, to specify handlers for different exceptions.

At most one handler will be executed.

The general syntax for adding specific exceptions are –

try:

# statement(s)

except IndexError:

# statement(s)

except ValueError:

# statement(s)

**Example :**

# Program to handle multiple errors with one

# except statement

# Python 3

def fun(a):

    if a < 4:

        # throws ZeroDivisionError for a = 3

        b = a/(a-3)

    # throws NameError if a >= 4

    print("Value of b = ", b)

try:

    fun(3)

    fun(5)

# note that braces () are necessary here for

# multiple exceptions

except ZeroDivisionError:

    print("ZeroDivisionError Occurred and Handled")

except NameError:

    print("NameError Occurred and Handled")

**Output :**

ZeroDivision Error Occurred and Handled

## Try with Else Clause

In python, you can also use the else clause on the try-except block which must be present after all the except clauses.

The code enters the else block only if the try clause does not raise an exception.

**Example :**

# Program to depict else clause with try-except

# Python 3

# Function which returns a/b

def AbyB(a , b):

    try:

        c = ((a+b) / (a-b))

    except ZeroDivisionError:

        print ("a/b result in 0")

    else:

        print (c)

# Driver program to test above function

AbyB(2.0, 3.0)

AbyB(3.0, 3.0)

**Output :**

-5.0

a/b result in 0

## Finally Keyword in Python

* Python provides a keyword finally, which is always executed after the try and except blocks.
* The final block always executes after normal termination of try block or after try block terminates due to some exception.

**Syntax:**

try:

# Some Code....

except:

# optional block

# Handling of exception (if required)

else:

# execute if no exception

finally:

# Some code .....(always executed)

**Example :**

# Python program to demonstrate finally

# No exception Exception raised in try block

try:

    k = 5//0  # raises divide by zero exception.

    print(k)

# handles zerodivision exception

except ZeroDivisionError:

    print("Can't divide by zero")

finally:

    # this block is always executed

    # regardless of exception generation.

    print('This is always executed')

**Output :**

Can't divide by zero

This is always executed

## Raising Exception

The raise statement allows the programmer to force a specific exception to occur.

The sole argument in raise indicates the exception to be raised.

This must be either an exception instance or an exception class (a class that derives from Exception).

**Example :**

# Program to depict Raising Exception

try:

    raise NameError("Hi there")  # Raise Error

except NameError:

    print ("An exception")

    raise  # To determine whether the exception was raised or not

**Output :**

