The **try** block lets you test a block of code for errors.

The **except** block lets you handle the error.

In Python, we catch exceptions and handle them using try and except code blocks. The try clause contains the code that can raise an exception, while the except clause contains the code lines that handle the exception.

```
print(a)
```

```
5
```

NameError:

```
try:
  print(x)
except:
  print("An exception occurred"+ Exception)
```

```
frozenset({'banana', 'apple', 'cherry'})
```

```
try:
   print(x)
except:
  a=1+9
  print(a)
```

TypeError: This exception is raised when an operation or function is applied to an object of the wrong type.

```
x = 5
y = "hello"
try:
  z = x + y # Raises a TypeError: unsupported operand type(s) for +: 'int' and 'str'
except Exception as err:
  print(type(err).__name__)
```

```
TypeError
```

```
x = 5
y = "hello"
try:
    z = x + y
except TypeError:
    print("Error: cannot add an int and a str")
```

Catching Specific Exception

try: # statement(s) except IndexError: # statement(s) except ValueError: # statement(s)

```
try:
  a='hello'
  b=100
  z=a+100+b
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

```
# 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")
```

```
# Python code to catch an exception and handle it using try and except code blocks

a = ["Python", "Exceptions", "try and except"]
try:
    #looping through the elements of the array a, choosing a range that goes beyond th
     for i in range( 4 ):
        print( "The index and element from the array is", i, a[i] )
#if an error occurs in the try block, then except block will be executed by the Pythor
except:
    print (Exception)
```

```
The index and element from the array is 0 Python
The index and element from the array is 1 Exceptions
The index and element from the array is 2 try and except
<class 'Exception'>
```

```
Exception.args
```

```
<attribute 'args' of 'BaseException' objects>
```

How to Raise an Exception

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.

If we wish to use raise to generate an exception when a given condition happens, we may do so as follows:

```
num = [3, 4, 5, 7]
if len(num) > 3:
    raise Exception( "Length of the given list must be less than or equal to 3 but is"
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-10-efb629f58d47> in <module>
      1 num = [3, 4, 5, 7]
      2 if len(num) > 3:
----> 3     raise Exception( "Length of the given list must be less than or equal to 3 but is" +str( len(num)) )

Exception: Length of the given list must be less than or equal to 3 but is4
```

```
name="palak"
a=len(name)
```

```
    if a<10:
      raise Exception("NAme should be more than 10 letters")
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-11-40c82270d91a> in <module>
      3
      4 if a<10:
----> 5   raise Exception("NAme should be more than 10 letters")

Exception: NAme should be more than 10 letters
```

Try with Else Clause

Python also supports the else clause, which should come after every except clause, in the try, and except blocks. Only when the try clause fails to throw an exception the Python interpreter goes on to the else block.

```
# Python program to show how to use else clause with try and except clauses

# Defining a function which returns reciprocal of a number
def reciprocal( num1 ):
    try:
        reci = 1 / num1
    except ZeroDivisionError:
        print( "We cannot divide by zero" )
    else:
        print ( reci )
# Calling the function and passing values

reciprocal( 4 )
#reciprocal( 0 )
```

```
0.25
```

Double-click (or enter) to edit

[Finally Keyword in Python]

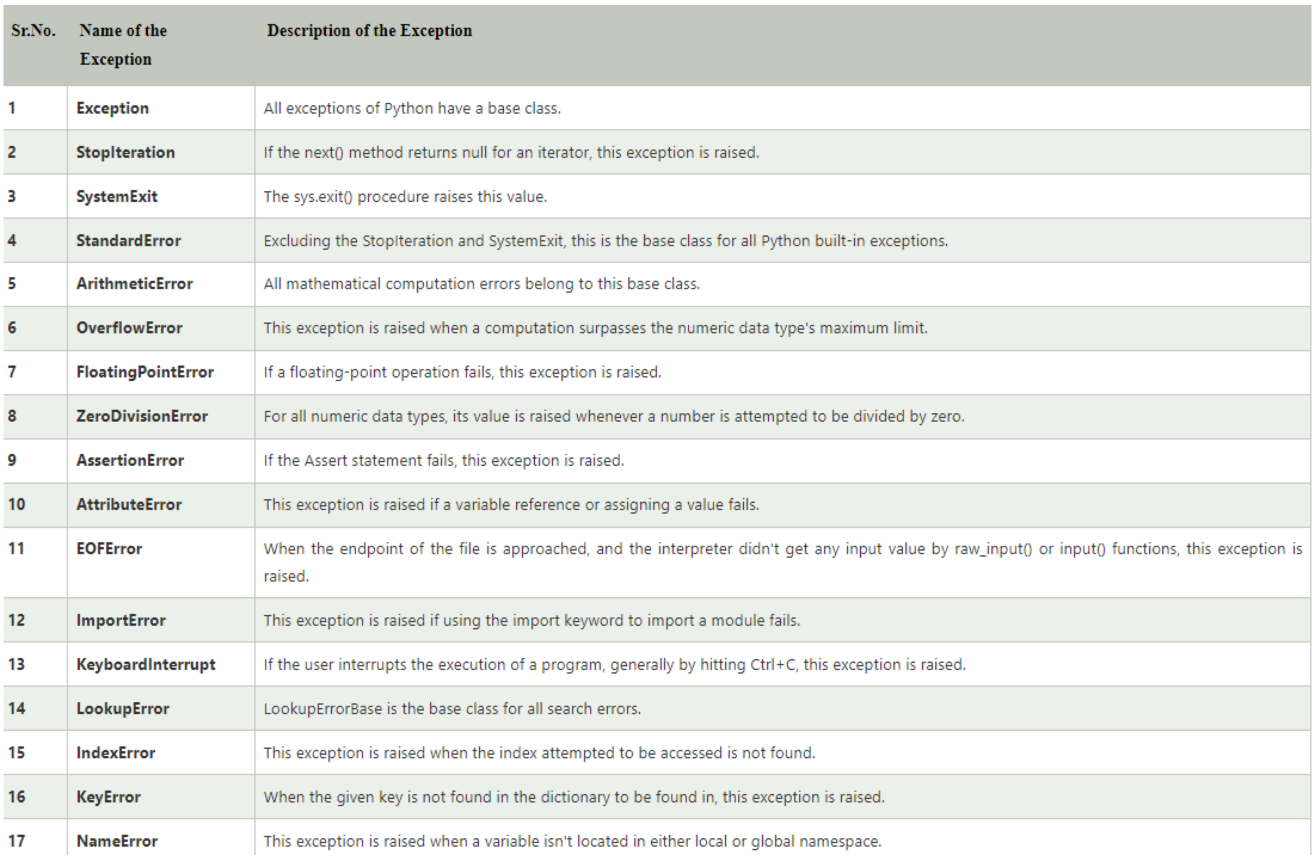
The finally keyword is available in Python, and it is always used after the try-except block. The finally code block is always executed after the try block has terminated normally or after the try block has terminated for some other reason.

```
# Python code to show the use of finally clause

# Raising an exception in try block
try:
    div = 4 / 0
    print( div )
# this block will handle the exception raised
except ZeroDivisionError:
    print( "Atepting to divide by zero" )
# this will always be executed no matter exception is raised or not
finally:
    print( 'This is code of finally clause' )
    #1/0
```

```
Atepting to divide by zero
This is code of finally clause
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-32-7583bf1e4064> in <module>
     11 finally:
     12     print( 'This is code of finally clause' )
---> 13     1/0

ZeroDivisionError: division by zero
```



Sr.No.	Name of the Exception	Description of the Exception
1	**Exception**	All exceptions of Python have a base class.
2	**StopIteration**	If the next() method returns null for an iterator, this exception is raised.
3	**SystemExit**	The sys.exit() procedure raises this value.
4	**StandardError**	Excluding the StopIteration and SystemExit, this is the base class for all Python built-in exceptions.
5	**ArithmeticError**	All mathematical computation errors belong to this base class.
6	**OverflowError**	This exception is raised when a computation surpasses the numeric data type's maximum limit.
7	**FloatingPointError**	If a floating-point operation fails, this exception is raised.
8	**ZeroDivisionError**	For all numeric data types, its value is raised whenever a number is attempted to be divided by zero.
9	**AssertionError**	If the Assert statement fails, this exception is raised.
10	**AttributeError**	This exception is raised if a variable reference or assigning a value fails.
11	**EOFError**	When the endpoint of the file is approached, and the interpreter didn't get any input value by raw_input() or input() functions, this exception is raised.
12	**ImportError**	This exception is raised if using the import keyword to import a module fails.
13	**KeyboardInterrupt**	If the user interrupts the execution of a program, generally by hitting Ctrl+C, this exception is raised.
14	**LookupError**	LookupErrorBase is the base class for all search errors.
15	**IndexError**	This exception is raised when the index attempted to be accessed is not found.
16	**KeyError**	When the given key is not found in the dictionary to be found in, this exception is raised.
17	**NameError**	This exception is raised when a variable isn't located in either local or global namespace.

```
Start coding or generate with AI.
```

Frozen set

Freeze a set(list/tuple) and makes it unchangeable

Def: frozenset() function returns an unchangeable frozen object

Syntax:

frozenset(iterable)

An iterable object like set,list,tuple,etc.

```
mylist=['apple','banana','cherry']
x=frozenset(mylist)
x[1]='straberry' #error
print(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-97de36b602eb> in <module>
      1 mylist=['apple','banana','cherry']
      2 x=frozenset(mylist)
----> 3 x[1]='straberry' #error
      4 print(x)

TypeError: 'frozenset' object does not support item assignment
```