# Errors and Exceptions

- Error in Python can be of two types:
  a) Syntax errors
  b) Exceptions
- Errors are the **problems in a program due to which the program will stop the execution.**
- Exceptions are **raised when some internal events occur** which changes the normal flow of the program.

**Syntax Error:** As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

**Example 1:**

```
amount = 10000
 # check that You are eligible to purchase Table or not
if(amount > 2999)
print("You are eligible to purchase.")
```

**Output:**

```
if(amount > 2999)
SyntaxError: invalid syntax
```

**Exceptions:** Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program; however, it changes the normal flow of the program.

**Example:**

```
marks = 10000
 # perform division with 0
a = marks / 0
print(a)
```

**Output:**

```
----> 4 a = marks / 0
      5 print(a)
ZeroDivisionError: division by zero
```

# Python Exception Handling

- When a Python code throws an exception, it has two options:
  - ✓ handle the exception immediately or
  - ✓ stop and quit.
- In Python, Try and Except statements are used to handle exceptions within a code.

1) The try block lets you test a block of code for errors.
2) The except block lets you handle the error.
3) The else block lets you execute code when there is no error.
4) The finally block lets you execute code, regardless of the result of the try- and except blocks.

- **Syntax:**

```
try:
   #statements in try block
except:
   #executed when error in try block
else:
   #executed if try block is error-free
finally:
   #executed irrespective of exception occurred or not
```

## 1) Try and Except Statement – Catching Exceptions

- The try block is used to check some code for errors i.e the code inside the try block will execute when there is no error in the program.
- Whereas the code inside the except block will execute whenever the program encounters some error in the preceding try block.

**Syntax:**

```
try:
   # Some Code
except:
   # Executed if error in the
   # try block
```

**Example:**

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

Since the try block raises an error, the except block will be executed. Here, the print(x) statement will raise error because x is not defined.

**Example:**

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

a = ["Python", "Exceptions", "try and except"]
try:
    for i in range(4):
       print( "The index and element is", i, a[i] )
except:
   print ("Index out of range")
```

## 2) 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.
- You can use the else keyword to define a block of code to be executed if no errors were raised.
- **Syntax:**

```
try:
   # Some Code
except:
   # Executed if error in the
   # try block
else:
   # execute if no exception
```

**Example:**

```
# 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 )
```

**Output:**

```
0.25
We cannot divide by zero
```

```
# Program to depict else clause with try-except
```

**Example:**

```
# 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)

AbyB(2.0, 3.0)
AbyB(3.0, 3.0)
```

**Output:**

```
-5.0
a/b result in 0
```

## 3) Finally Keyword

- It is always used after the try and except blocks.
- The finally block, if specified, will be executed regardless if the try block raises an error or not.
- The final block always executes after normal termination of try block or after try block terminates due to some exceptions.
- This can be useful to close objects and clean up resources

**Syntax:**

```
try:
   # Some Code....

except:
   # optional block
   # Handling of exception (if required)

else:
   # execute if no exception

finally:
   # Some code .....(always executed)
```

**Example 1:**

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

**Example 2:**

```
f = open("demofile.txt")
try:
  f.write("Abcd efgh ijkl")
except:
  print("Something went wrong when writing to the file")
finally:
  f.close()
except:
   print("Something went wrong when opening the file")
```

**Example 3:**

```
try:
   k = 5//0 .
   print(k)
 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

**Example: 4**

```
try:
   div = 4 // 0
   print( div )
except ZeroDivisionError:
   print( "Attempting to divide by zero" )
finally:
   print( 'This is code of finally clause' )
```

**Output:**

Attempting to divide by zero

This is code of finally clause

## 5) Raise an exception

- The raise statement allows the programmer to force a specific exception to occur.
- To throw (or raise) an exception, use the `raise` keyword.

**Example 1:**

Raise an error and stop the program if x is lower than 0:

```
x = -1
if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

**Example 2:**

```
#Python code to show how to raise an exception in Python
num = [3, 4, 5, 7]
if len(num) > 3:
   raise Exception("Length of the given list must be less than or equal to 3 " )
```

**Example 3:**

```
try:
   x=int(input('Enter a number upto 100: '))
   if x > 100:
      raise ValueError(x)
except ValueError:
   print(x, "is out of allowed range")
else:
   print(x, "is within the allowed range")
```

**Output**

```
Enter a number upto 100: 200
200 is out of allowed range
Enter a number upto 100: 50
50 is within the allowed range
```

# Additional Points:

### How try except works?

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

- First, the **try** clause is executed i.e. the code between **try** and **except** clause.
- If there is no exception, then only the **try** clause will run.
- If any exception occurs, the **try** clause will be skipped and **except** clause will run.
- A **try** statement can have more than one **except** clause.
- If any exception occurs, but the **except** clause within the code doesn't handle it, it is passed on to the outer **try** statements. If the exception is left unhandled, then the execution stops.

**Example for both error and exception:**

```
string = "Python Exceptions"
for s in string:
   if (s != o:
      print( s )
```

**Output:**

**Syntax error**

**Name Error: Exception 'o' is not defined.**

**Some of the common Exception Errors are:**

- IOError: if the file can't be opened
- KeyboardInterrupt: when an unrequired key is pressed by the user
- ValueError: when built-in function receives a wrong argument
- EOFError: if End-Of-File is hit without reading any data
- ImportError: if it is unable to find the module

PYTHON : Exception hierarchy

The class hierarchy for built-in exceptions is:

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |
      |
      |
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
```

```
    |    +-- SyntaxError
    |    |    +-- IndentationError
    |    |         +-- TabError
    |    +-- SystemError
    |    +-- TypeError
    |    +-- ValueError
    |         +-- UnicodeError
    |              +-- UnicodeDecodeError
    |              +-- UnicodeEncodeError
    |              +-- UnicodeTranslateError
    +-- Warning
         +-- DeprecationWarning
         +-- PendingDeprecationWarning
         +-- RuntimeWarning
         +-- SyntaxWarning
         +-- UserWarning
         +-- FutureWarning
              +-- ImportWarning
              +-- UnicodeWarning
              +-- BytesWarning
```