Exceptions

- Compile Time Error
- Run Time Error
- Logical Error
- Exception
- Exception Handling
- Try Block
- Except Block
- Else Block
- Finally Block
- Assert Statement
- User defined Exception

Errors

Three types of errors:

1. Compile Time Error
2. Run Time Error
3. Logical Error

Compile Time Error

> These are **syntactical errors** found in the code, due to which program fails to compile.

> Such errors are detected by **Python Compiler** and line number along with error description is displayed.

Program 1: A Python program to understand compile time error

```
x = 1
if x==1
  print("Colon is missing.")
```

```
  File "<ipython-input-6-ef810c46740d>", line 2
    if x==1
           ^
SyntaxError: invalid syntax
```

Program 2: A Python program to demonstrate indentation compile time error

```
x=22
if x%2==0:
  print(x, "is divisible by 2")
    print(x,'is even number')
```

```
  File "<ipython-input-7-5221694dbc6d>", line 4
    print(x,'is even number')
    ^
IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

Run Time Error

> When PVM cannot execute the byte code, it flags runtime error.

> Runtime errors are not detected by compiler, they are detected only by **PVM**.

Program 3: A Python program to understand run time error

```
def concat(a,b):
  print(a+b)

concat("Python"," Program")
concat("Python", 3)
```

```
Python Program
--------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-ef85d6547677> in <module>()
```

Program 4: A Python program to demonstrate index out range run time error

```
animal = ['Lion','Tiger','Bear']
print(animal[3])
```

```
--------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-9-7bb501bb3b0a> in <module>()
      1 animal = ['Lion','Tiger','Bear']
----> 2 print(animal[3])

IndexError: list index out of range
```

SEARCH STACK OVERFLOW

Logical Error

> These errors depict flaws in the logic of the program. The programmer might be using wrong formula or design of the program itself is wrong.

> These errors are not detected by compiler or PVM, it is sole responsibility of **Programmer**.

Program 5: A Python program to increment salary of employee by 15%

```
#logical error
def increment(sal):
  sal = sal * 15/100
  #sal = sal + sal * 15/100
  #sal = sal * 115/100
  return sal

print("Your salary is incremented from 1000 to ", increment(1000))
```

```
Your salary is incremented from 1000 to  150.0
```

Exceptions

> The runtime errors that can be handelled by the programmer is called **Exceptions.**

Program 6: A Python program to understand the effect of an exception

```
f = open('file1','w')
a,b = [int(x) for x in input('Enter two numbers - ').split()]
c  = a/b
f.write('Writing %d into file 1' %c)
f.close()
print('File Closed')
```

```
Enter two numbers - 7 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-11-dcd44c284de7> in <module>()
      1 f = open('file1','w')
      2 a,b = [int(x) for x in input('Enter two numbers - ').split()]
----> 3 c  = a/b
      4 f.write('Writing %d into file 1' %c)
      5 f.close()

ZeroDivisionError: division by zero
```

SEARCH STACK OVERFLOW

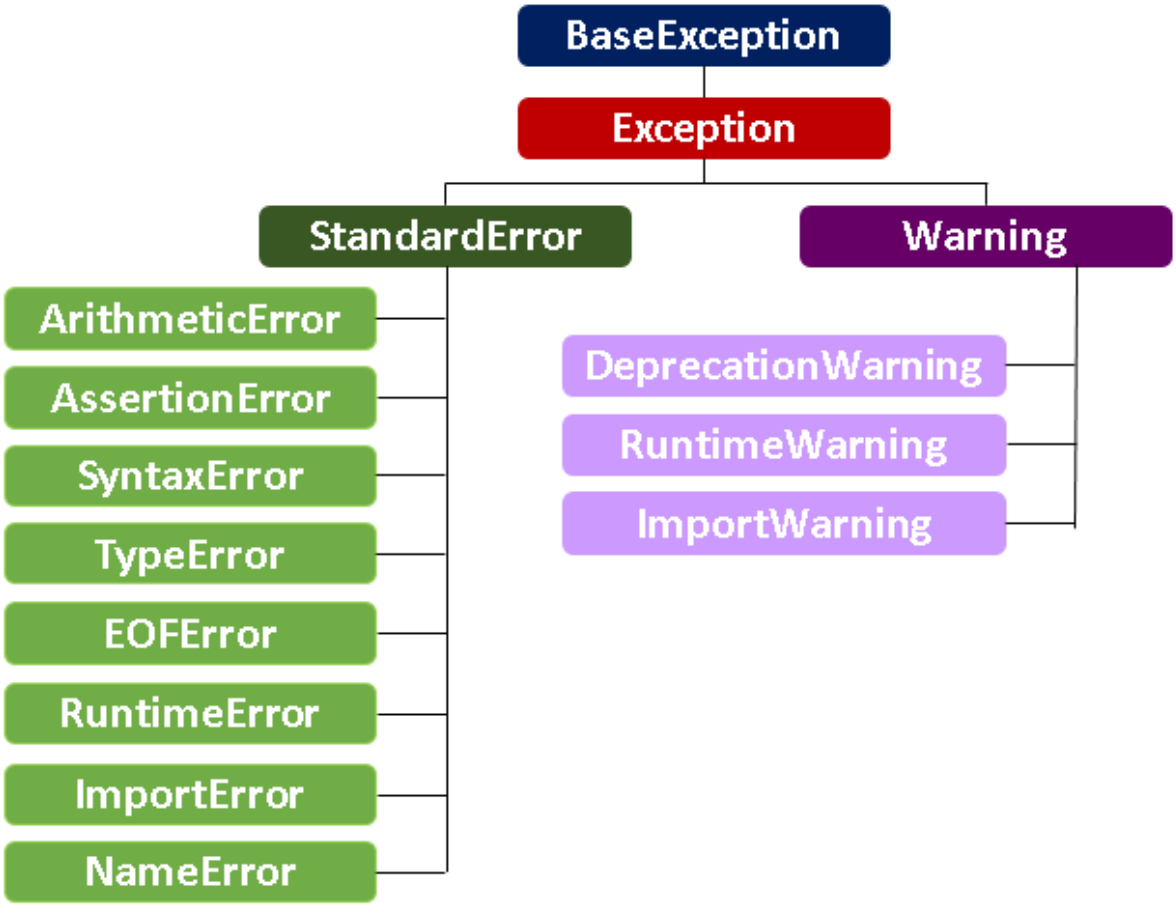
> In the Program 6, suppose user enter second number as zero, then it will raise **ZeroDivisonError**. When such error occurs PVM simply displayed error and terminates the program.

> Due to this abnormal termination, f.close() is not executed, which leads to loss of entire data that is already present in the file.

> An exception is a runtime error which can be handled by the programmer. That means if the programmer can guess an error in the program and he can do something to eliminate the harm caused by that error, then it is called 'Exception'.

> All exceptions are represented as classes in python. The exceptions which are already available in Python are called in **'Built-in'** exceptions.

> The base class for all built in exceptions is **BaseException** class. From BaseException class, the subclass **'Exception'** is derived. From Exception class, the sub classes **'StandardError' and 'Warning'** are derived.



All errors or exceptions are derived as subclasses from StandardError, which should be compulsorily handled otherwise program will not execute.

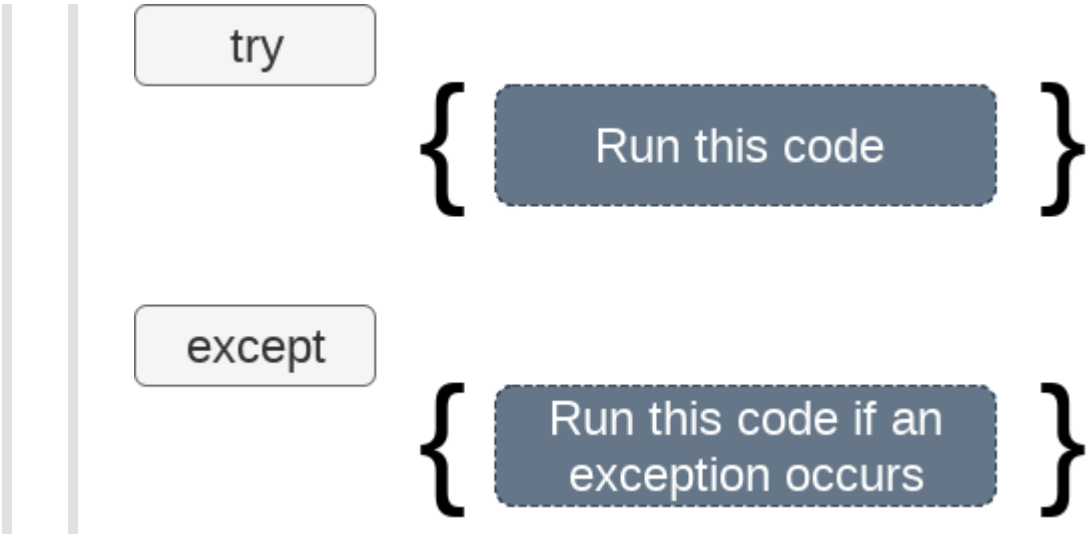
Similarly, all warnings are derived as subclasses from Warning class, which represents cautions and program will execute, even though if it is not handled.

Exception Handling

The purpose of handling errors is to make program robust i.e. does not abruptly terminate in the middle.

The try--except block

- If the Python program contains suspicious code that may throw the exception, we must place that code in the **try block**.
- The try block must be followed with the **except statement**, which contains a block of code that will be executed if there is some exception in the try block.



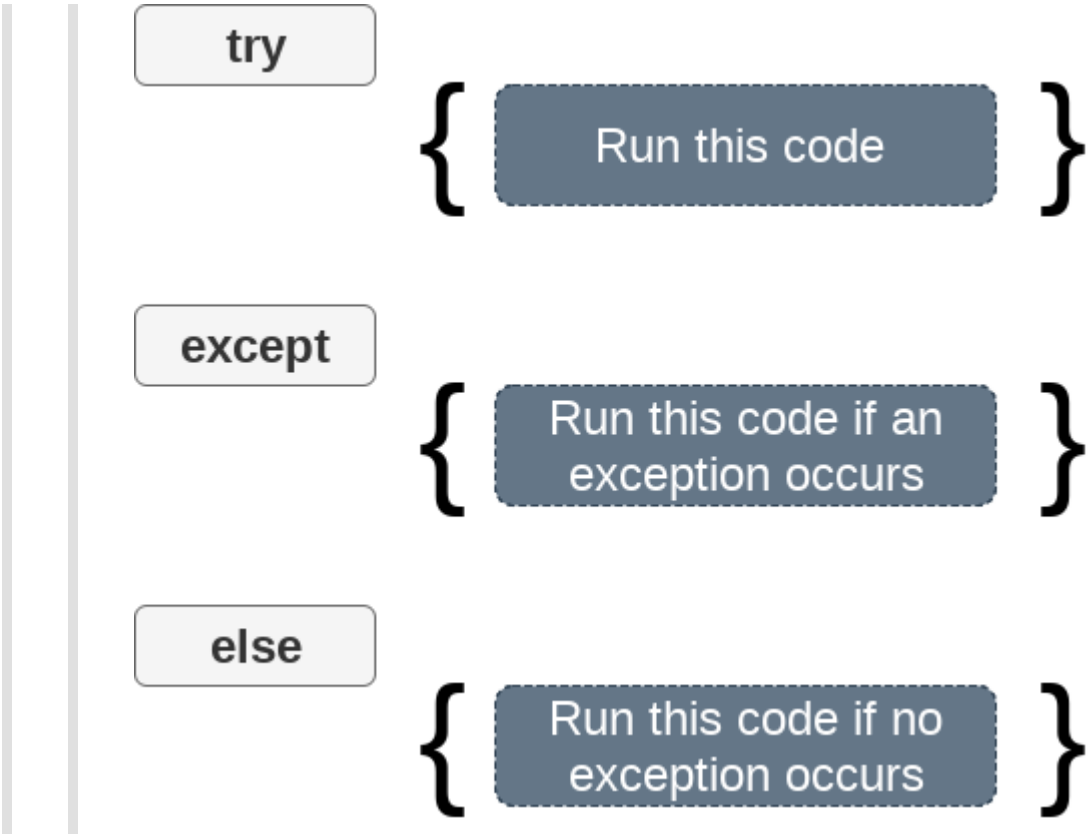
Program 7: A Python program to handle DivisionByZero Error

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b
except:
    print("Can't divide with zero")
```

```
 Enter a:7
 Enter b:0
 Can't divide with zero
```

The try-except-else block

- The else statement can also be used with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.



Program 8:A Python program to handle DivisionByZero Error with esle block.

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b
    print("a/b = %d"%c)
    # Using exception object with the except statement
except Exception as e:
    print("can't divide by zero")
    print(e)
else:
    print("Hi I am else block")
```

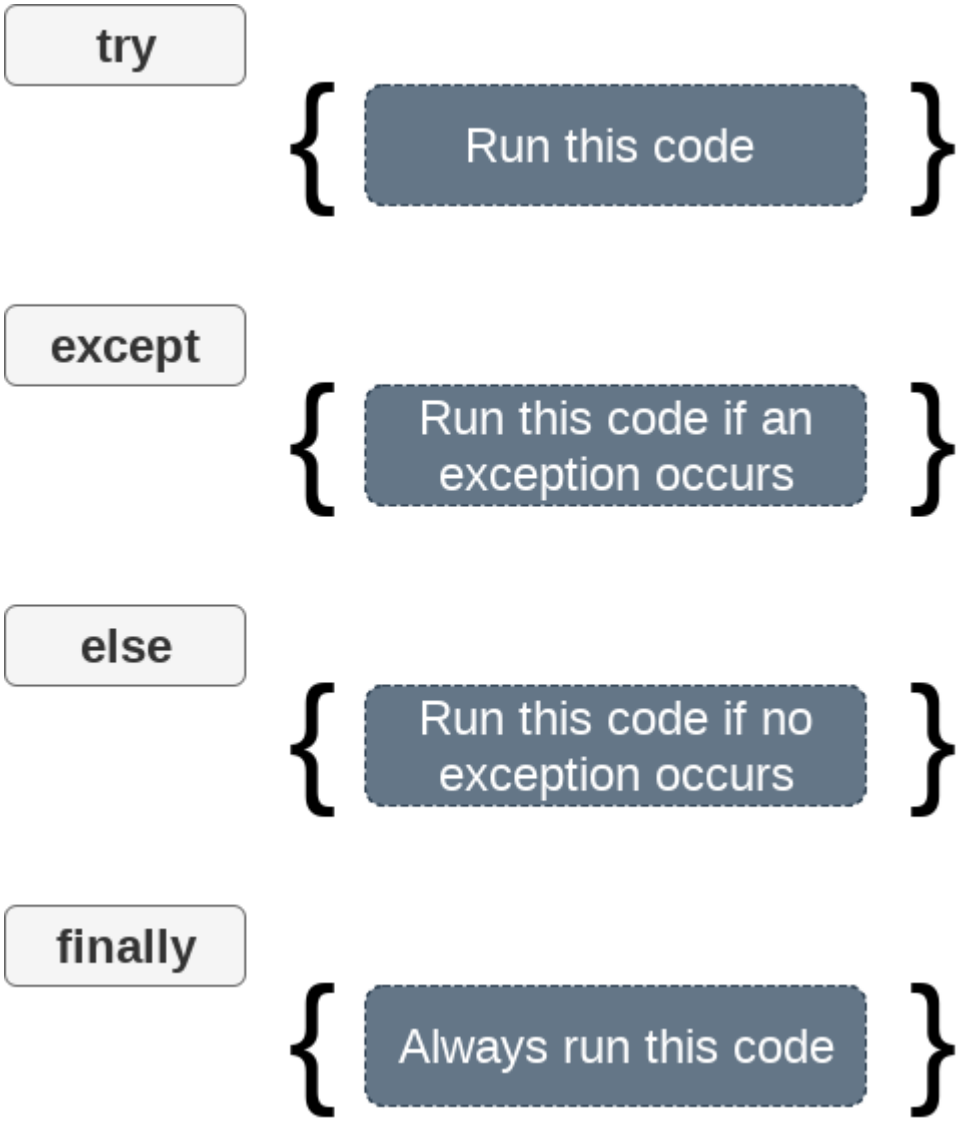
```
 Enter a:7
 Enter b:1
 a/b = 7
 Hi I am else block
```

The try.....finally block

- Python provides the optional finally statement, which is used with the try statement. It is executed no matter what exception occurs and used to release

the external resource. The finally block provides a guarantee of the execution.

- We can use the finally block with the try block in which we can pace the necessary code, which must be executed before the try statement throws an exception.



Program 9:A Python program to handle DivisionByZero Error with finally block.

```
try:
  f = open('file2','w')
  a,b = [int(x) for x in input('Enter two numbers - ').split()]
  c  = a/b
  f.write('Writing %d into file 2' %c)
except ZeroDivisionError:
  print('Division by zero has occured, do not enter second number as zero')

finally:
  f.close()
  print('File Closed')
```

```
Enter two numbers - 8 0
Division by zero has occured, do not enter second number as zero
File Closed
```

> The complete Exception Handling syntax will be in the following format:

```
try:
    statement(s)
except Exception1:
    handler1
except Exception2:
    handler2
else:
    statement(s)
finally:
    statement(s)
```

Important noteworthy points -

> - A single try block can be followed by multiple except block.
> - Multiple except block can be used to handle multiple exceptions.
> - We cannot write an except block without a try block.
> - We can write try block without any except blocks.
> - Else blocks and finally blocks are not compulsory.
> - When there is no exception, else block is executed after try block.
> - Finally block is always executed.

Type of Exception

https://docs.python.org/3/library/exceptions.html#exception-hierarchy

---

Program 10: A Python program to handle IOError.

```
#Example of IOError
try:
  name = input('Enter filename  ')
  f = open(name,'r')

except IOError:
  print('File not found -', name)
else:
  n = len(f.readlines())
```

```
  print(name,'has',n,'lines')
```

```
Enter filename  file1
file1 has 0 lines
```

Program 11: A Python program to handle multiple exception

```
#A function to find total and average of list elements
def averageTotal(list):
  tot = 0
  for x in list:
    tot += x

  avg = tot/len(list)
  return avg, tot

#outside function
#Function call
try:
  a,t = averageTotal([1,2,3,4,5])
  #a,t = averageTotal([1,2,3,4,5,'a'])
  #a,t = averageTotal([])
  print('Total = ',t, 'Average = ',a)
except TypeError:
  print('Type Error....please provide only numbers')
except ZeroDivisionError:
  print('List is empty')
```

The Except Block

It can be written in various formats:

1. Except block with Exceptionclass name

```
except Exceptionclass:
```

2. Exception can be catch in an object that will contain some description about the exception.

```
except Exceptionclass as obj:
```

3. For catching multiple exception, we can write one single except block.

```
except (Exceptionclass1,Exceptionclass2):
```

4. To catch any type of exception, except block can be written without any Exception class.

```
except :
```

The Assert Statement

- It is useful to ensure that the given condition is True.
- If not True, then it raises an AssertionError
- Syntax:

```
assert condition, message
```

- If the condition is False, then Exception by the name AssertionError is raised alongwith the message written in the assert statement.
- If message is not written in the assert statement, and if the condition is false then also AssertionError is raised without message.

Program 12: A Python program using assert statement and catching AssertionError

```
try:
  x = int(input('Enter a number between 5 and 10 - '))
  assert x>=5 and x <=10
  print('Number is',x)
except AssertionError:
  print('Incorrect input')
```

```
Enter a number between 5 and 10 - 6
Number is 6
```

Program 13: A Python program using assert statement with message

```
try:
  x = int(input('Enter a number between 5 and 10 - '))
  assert x>=5 and x <=10, 'Incorrect input'
  print('Number is',x)
except AssertionError as obj: #Whenever there is an assertion error, message is passed to Ass
  print(obj)
```

```
Enter a number between 5 and 10 - 11
Incorrect input
```

User Defined Exceptions

> These are exception which is created by the programmer, also known as **Custom Exception**.

> There are many situations where none of the built-in exceptions are useful. So programmer can create their own exception and raise it.

> **Steps**:

> 1. Create your own **class** to handle exception, which will be derived from base **class Exception**.
> 2. Exception are raised using **raise** statement , syntax is
>
> ```
> raise Classname(Message)
> ```
>
> 3. Lastly, try ... except block is use to catch the exception

---

Program 14: A Python program to create own exception ans raise it

```
#Scenario - There is a bank, which raises exception when the balance is below 2000.
class UserException(Exception):
  def __init__(self, arg):
    self.msg = arg
  def check(customer):
    for k,v in customer.items():
      print('Name = ',k,'Amount = ',v)
      if (v<2000):
          raise UserException('Less Balance')


#outside class
cust = {'ABC':5000,'LMN':1500,'PQR':3000}
try:
  UserException.check(cust)
except UserException as obj:
        print(obj.msg)
```

```
Name =  ABC Amount =  5000
Name =  LMN Amount =  1500
Less Balance
```