Exception-handling

An exception is a runtime error which can be handled by the programmer. All exceptions are represented as classes in Python.

Type of Exception:-

Built-in Exception – Exceptions which are already available in Python Language. The base class for all built-in exceptions is BaseException class.

User Defined Exception – A programmer can create his own exceptions, called user-defined exceptions.

All exceptions are represented as classes in Python.

There are 2 stages where error may happen in a program

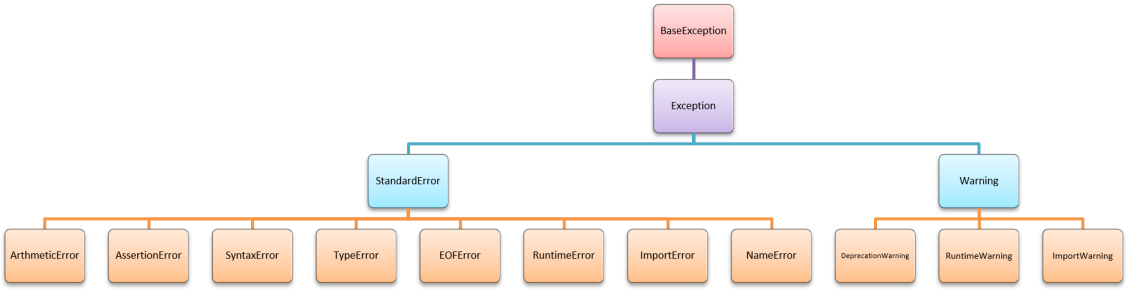
During compilation -> Syntax Error

- Something in the program is not written according to the program grammar.
- Error is raised by the interpreter/compiler
- You can solve it by rectifying the program

Other examples of syntax error

- Leaving symbols like colon,brackets
- Misspelling a keyword
- Incorrect indentation
- empty if/else/loops/class/functions

During execution -> Exceptions



Exception Description

ArithmeticError: Raised when an error occurs in numeric calculations

AttributeError: Raised when attribute reference or assignment fails Exception Base class for all exceptions

EOFError: Raised when the input() method hits an "end of file" condition (EOF)

ImportError: Raised when an imported module does not exist

IndentationError: Raised when indentation is not correct

IndexError: Raised when an index of a sequence does not exist

KeyError: Raised when a key does not exist in a dictionary

NameError: Raised when a variable does not exist

SyntaxError: Raised when a syntax error occurs

TypeError: Raised when two different types are combined

ValueError: Raised when there is a wrong value in a specified data type

ZeroDivisionError Raised when the second operator in a division is zero

In [29]:
```
1 h = input("this is"
```

```
  File "<ipython-input-29-e00974fe9829>", line 1
    h = input("this is"
                      ^
SyntaxError: unexpected EOF while parsing
```

In [28]:
```
1 student = {
2   "name": "John",
3   "level": "400",
4   "faculty": "Engineering and Technology"
5
```

```
  File "<ipython-input-28-dc74c25d4bb5>", line 4
    "faculty": "Engineering and Technology"
                                           ^
SyntaxError: unexpected EOF while parsing
```

In [30]:
```
1 for i in sample.txt:
```

```
  File "<ipython-input-30-60a88ca51960>", line 1
    for i in sample.txt:
                        ^
SyntaxError: unexpected EOF while parsing
```

In [31]:
```
class P:
```

```
  File "<ipython-input-31-de9ce877e47c>", line 1
    class P:
            ^
SyntaxError: unexpected EOF while parsing
```

In [1]:
```
# Examples of syntax error
print 'hello world'
```

```
  File "<ipython-input-1-4655b84ba7b7>", line 2
    print 'hello world'
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print('h
ello world')?
```

In [2]:
```
a = 5
if a==3
  print('hello')
```

```
  File "<ipython-input-2-efc58c10458d>", line 2
    if a==3
          ^
SyntaxError: invalid syntax
```

In [3]:
```
a = 5
iff a==3:
  print('hello')
```

```
  File "<ipython-input-3-d1e6fae154d5>", line 2
    iff a==3:
        ^
SyntaxError: invalid syntax
```

In [4]:
```
var = 5
if var==3:
print('hello')
```

```
  File "<ipython-input-4-e9da9a582f84>", line 3
    print('hello')
    ^
IndentationError: expected an indented block
```

In [5]:
```
# IndexError
# The IndexError is thrown when trying to access an item at an invalid
L = [1,2,3]
L[100]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-5-c90668d2b194> in <module>
      2 # The IndexError is thrown when trying to access an item at an invalid index.
      3 L = [1,2,3]
----> 4 L[100]

IndexError: list index out of range
```

In [6]:
```
# ModuleNotFoundError
# The ModuleNotFoundError is thrown when a module could not be found.
import mathi
math.floor(5.3)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-6-cbdaf00191df> in <module>
      1 # ModuleNotFoundError
      2 # The ModuleNotFoundError is thrown when a module could not be found.
----> 3 import mathi
      4 math.floor(5.3)

ModuleNotFoundError: No module named 'mathi'
```

In [7]:
```
# KeyError
# The KeyError is thrown when a key is not found

d = {'name':'nitish'}
d['age']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-453afa1c9765> in <module>
      3
      4 d = {'name':'nitish'}
----> 5 d['age']

KeyError: 'age'
```

In [8]:
```
# TypeError
# The TypeError is thrown when an operation or function is applied to a
1 + 'a'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
t)
<ipython-input-8-2a3eb3f5bb0a> in <module>
      1 # TypeError
      2 # The TypeError is thrown when an operation or function is applied
to an object of an inappropriate type.
----> 3 1 + 'a'

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [9]:
```
# ValueError
# The ValueError is thrown when a function's argument is of an inapprop
int('a')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last
t)
<ipython-input-9-e419d2a084b4> in <module>
      1 # ValueError
      2 # The ValueError is thrown when a function's argument is of an ina
ppropriate type.
----> 3 int('a')

ValueError: invalid literal for int() with base 10: 'a'
```

In [10]:
```
# NameError
# The NameError is thrown when an object could not be found.
print(k)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last
t)
<ipython-input-10-e3e8aaa4ec45> in <module>
      1 # NameError
      2 # The NameError is thrown when an object could not be found.
----> 3 print(k)

NameError: name 'k' is not defined
```

In [11]:
```
# AttributeError
L = [1,2,3]
L.upper()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-dd5a29625ddc> in <module>
      1 # AttributeError
      2 L = [1,2,3]
----> 3 L.upper()

AttributeError: 'list' object has no attribute 'upper'
```


ArithmeticError is simply an error that occurs during numeric calculations.

ArithmeticError types in Python include:

OverFlowError

ZeroDivisionError

FloatingPointError

In [13]:
```
print("Simple program for showing overflow error")
print("\n")
import math
print("The exponential value is")
print(math.exp(1000))
```

```
Simple program for showing overflow error


The exponential value is
---------------------------------------------------------------------------
OverflowError                             Traceback (most recent call last)
<ipython-input-13-a3e7e3f1c25a> in <module>
      3 import math
      4 print("The exponential value is")
----> 5 print(math.exp(1000))

OverflowError: math range error
```

In [12]:
```
1 arithmetic = 5/0
2 print(arithmetic)
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-12-888908747809> in <module>
----> 1 arithmetic = 5/0
      2 print(arithmetic)

ZeroDivisionError: division by zero
```

Why is it important to handle exceptions

When an exception occurs, the program terminates suddenly.

Suddenly termination of program may corrupt the program.

Exception may cause data loss from the database or a file.

how to handle exceptions

Try – The try block contains code which may cause exceptions.

Syntax-

try:

```
    statements
```

Except – The except block is used to catch an exception that is raised in the try block. There can be multiple except block for try block.

Syntax-

except ExceptionName:

```
      statements
```

Else – This block will get executed when no exception is raised. Else block is executed after try block.

Syntax-

else:

```
    statements
```

Finally – This block will get executed irrespective of whether there is an exception or not.

Syntax-

finally:

```
    statements
```

We can write several except blocks for a single try block.

We can write multiple except blocks to handle multiple exceptions.

We can write try block without any except blocks.

We can not write except block without a try block.

Finally block is always executed irrespective of whether there is an exception or not.

Else block is optional.

Finally block is optional.

#example 1

try:

```
    Statement
```

except ExceptionClassName:

```
    Statement
```

#example2

try:

```
    Statement
```

except ExceptionClassName:

```
    Statement
```

else:

```
    Statement
```

finally:

```
    Statement
```

#example 3

try:

```
    Statement
```

except ExceptionClassName1:

```
    Statement
```

except ExceptionClassName2:

```
    Statement
```

finally:

```
    Statement
```

#example4

try:

```
    Statement
```


how to write except block

1.With the Exception Class Name

```
except ExceptionClassName:

       Statement
```

2.Exception as an object

```
except ExceptionClassName as obj:

      Statement
```

3.Multiple Exception within tuple

```
except (ExceptionClassName1, ExceptionClassName2, Exceptio
nClassName3, …… ):

     Statement
```

4.Catch any Type of Exception

```
except:

   Statement
```

In [17]:
```
# let's create a file
with open('sample.txt','w') as f:
  f.write('hello world')
```

In [18]:
```
# try catch demo
try:
  with open('sample1.txt','r') as f:
    print(f.read())
except:
  print('sorry file not found')
```

```
sorry file not found
```

In [20]:
```
a = 10
b = 0
try:
    d = a/b
    print(d)

except:
    print('Exception Handler')

print('Rest of the Code')
```

```
Exception Handler
Rest of the Code
```

In [19]:
```
# catching specific exception
try:
  m=5
  f = open('sample1.txt','r')
  print(f.read())
  print(m)
  print(5/2)
  L = [1,2,3]
  L[100]
except FileNotFoundError:
  print('file not found')
except NameError:
  print('variable not defined')
except ZeroDivisionError:
  print("can't divide by 0")
except Exception as e:
  print(e)
```

```
file not found
```

In [1]:
```
# catching specific exception
try:
  #m=5
  f = open('sample.txt','r')
  print(f.read())
  print(m)
  print(5/2)
  L = [1,2,3]
  L[100]
except FileNotFoundError:
  print('file not found')
except NameError:
  print('variable not defined')
except ZeroDivisionError:
  print("can't divide by 0")
except Exception as e:
  print(e)
```

```
hello world
variable not defined
```

In [2]:
```
# catching specific exception
try:
  m=5
  f = open('sample.txt','r')
  print(f.read())
  print(m)
  print(5/0)
  L = [1,2,3]
  L[100]
except FileNotFoundError:
  print('file not found')
except NameError:
  print('variable not defined')
except ZeroDivisionError:
  print("can't divide by 0")
except Exception as e:
  print(e)
```

```
hello world
5
can't divide by 0
```

In [3]:
```
# catching specific exception
try:
  m=5
  f = open('sample.txt','r')
  print(f.read())
  print(m)
  print(5/2)
  L = [1,2,3]
  L[100]
except FileNotFoundError:
  print('file not found')
except NameError:
  print('variable not defined')
except ZeroDivisionError:
  print("can't divide by 0")
except Exception as e:
  print(e)
```

```
hello world
5
2.5
list index out of range
```

In [4]:
```
# else
try:
  f = open('sample1.txt','r')
except FileNotFoundError:
  print('file nai mili')
except Exception:
  print('kuch to lafda hai')
else:
  print(f.read())
```

```
file nai mili
```

In [5]:
```
# else
try:
  f = open('sample.txt','r')
except FileNotFoundError:
  print('file nai mili')
except Exception:
  print('kuch to lafda hai')
else:
  print(f.read())
```

```
hello world
```

In [6]:
```
# finally
# else
try:
  f = open('sample1.txt','r')
except FileNotFoundError:
  print('file nai mili')
except Exception:
  print('kuch to lafda hai')
else:
  print(f.read())
finally:
  print('ye to print hoga hi')
```

```
file nai mili
ye to print hoga hi
```

In [7]:
```
# finally
# else
try:
  f = open('sample.txt','r')
except FileNotFoundError:
  print('file nai mili')
except Exception:
  print('kuch to lafda hai')
else:
  print(f.read())
finally:
  print('ye to print hoga hi')
```

```
hello world
ye to print hoga hi
```

In [21]:
```
a = 10
b = 0
try:
    d = a/b
    print(d)
    print('Inside Try')

except ZeroDivisionError:
    print('Division by Zero Not allowed')

print('Rest of the Code')
```

```
Division by Zero Not allowed
Rest of the Code
```

In [22]:
```
a = 10
b = 5
try:
    d = a/b
    print(d)
    print('Inside Try')

except ZeroDivisionError:
    print('Division by Zero Not allowed')

else:
    print('Inside Else')

print('Rest of the Code')
```

```
2.0
Inside Try
Inside Else
Rest of the Code
```

In [23]:
```
a = 10
b = 0
try:
    d = a/b
    print(d)
    print('Inside Try')

except ZeroDivisionError:
    print('Division by Zero Not allowed')

else:
    print('Inside Else')

finally:
    print('Inside Finally')

print('Rest of the Code')
```

```
Division by Zero Not allowed
Inside Finally
Rest of the Code
```

In [24]:
```
a = 10
b = 0
try:
    d = a/b
    print(d)
    print('Inside Try')

except ZeroDivisionError as obj:
    print(obj)

print('Rest of the Code')
```

```
division by zero
Rest of the Code
```

In [25]:
```
a = 10
b = 0
try:
    d = a/g
    print(d)

except ZeroDivisionError as obj:
    print(obj)

except NameError as ob:
    print(ob)

print('Rest of the Code')
```

```
name 'g' is not defined
Rest of the Code
```

In [26]:
```
a = 10
b = 0
try:
    d = a/g
    print(d)

except (NameError, ZeroDivisionError) as obj:
    print(obj)

print('Rest of the Code')
```

```
name 'g' is not defined
Rest of the Code
```


Assert Statement

The assert Statement is useful to ensure that a given condition is True. If it is not true, it raises AssertionError.

Syntax:- assert condition, error_message

If the condition is False then the exception by the name AssertionError is raised along with the message.

If message is not given and the condition is False then also AssertionError is raised without message.

In [10]:
```
1 n=int(input("enter int"))
2 assert n>5,"enter valid number"
```

```
enter int4

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-10-fa83f5a7736c> in <module>
      1 n=int(input("enter int"))
----> 2 assert n>5,"enter valid number"

AssertionError: enter valid number
```


User Defined Exception

A programmer can create his own exceptions, called user-defined exceptions or Custom Exception.

1.Creating Exception Class using Exception Class as a Base Class

2.Raising Exception

3.Handling Exception

Creating Exception

#We can create our own exception by creating a sub class to built-in Exception class.

```
class MyException(Exception):

        pass
```

```
class MyException(Exception):

        def __init__(self, arg):

                self.msg = arg
```

In [11]:
```
class BalanceException (Exception):
    #pass
    def __init__(self, arg):
        self.msg = arg

def checkbalance():
    money = 10000
    withdraw = 9000
    try:
        balance = money - withdraw
        if(balance<=2000):
            raise BalanceException('Insufficient Balance')
        print(balance)
    except BalanceException as be:
        print(be)

checkbalance()
```

```
Insufficient Balance
```

Raising Exception

raise statement is used to raise the user defined exception.

raise MyException('message')

In [16]:
```
# raise Exception
# In Python programming, exceptions are raised when errors occur at run
# We can also manually raise exceptions using the raise keyword.

# We can optionally pass values to the exception to clarify why that ex

raise ZeroDivisionError('aise hi ')
# Java
# try -> try
# except -> catch
# raise -> throw

```

```
---------------------------------------------------------------------------
-
ZeroDivisionError                         Traceback (most recent call las
t)
<ipython-input-16-97b461be5f2b> in <module>
      5 # We can optionally pass values to the exception to clarify why th
at exception was raised
      6
----> 7 raise ZeroDivisionError('aise hi ')
      8 # Java
      9 # try -> try

ZeroDivisionError: aise hi
```

In [18]:
```
money = 10000
withdraw = 9000
try:
        balance = money - withdraw
        if(balance<=2000):
            raise BalanceException('Insufficient Balance')
        print(balance)
except BalanceException as be:
        print(be)
```

```
Insufficient Balance
```

In [12]:
```
class Bank:

  def __init__(self,balance):
    self.balance = balance

  def withdraw(self,amount):
    if amount < 0:
      raise Exception('amount cannot be -ve')
    if self.balance < amount:
      raise Exception('paise nai hai tere paas')
    self.balance = self.balance - amount

obj = Bank(10000)
try:
  obj.withdraw(15000)
except Exception as e:
  print(e)
else:
  print(obj.balance)
```

```
paise nai hai tere paas
```

In [13]:

```
class MyException(Exception):
  def __init__(self,message):
    print(message)

class Bank:

  def __init__(self,balance):
    self.balance = balance

  def withdraw(self,amount):
    if amount < 0:
      raise MyException('amount cannot be -ve')
    if self.balance < amount:
      raise MyException('paise nai hai tere paas')
    self.balance = self.balance - amount

obj = Bank(10000)
try:
  obj.withdraw(5000)
except MyException as e:
  pass
else:
  print(obj.balance)
```

```
5000
```

In [14]:

```
class SecurityError(Exception):

  def __init__(self,message):
    print(message)

  def logout(self):
    print('logout')

class Google:

  def __init__(self,name,email,password,device):
    self.name = name
    self.email = email
    self.password = password
    self.device = device

  def login(self,email,password,device):
    if device != self.device:
      raise SecurityError('bhai teri to lag gayi')
    if email == self.email and password == self.password:
      print('welcome')
    else:
      print('login error')



obj = Google('nitish','nitish@gmail.com','1234','android')

try:
  obj.login('nitish@gmail.com','1234','windows')
except SecurityError as e:
  e.logout()
else:
  print(obj.name)
finally:
  print('database connection closed')
```

```
bhai teri to lag gayi
logout
database connection closed
```

Handling Exception

Using try and except block Programmer can handle exceptions.

try:

    statement

except MyException as mye:

    statement

In [19]:
```
# catching specific exception
try:
  m=5
  f = open('sample.txt','r')
  print(f.read())
  print(m)
  print(5/2)
  L = [1,2,3]
  L[100]

except Exception as e:
  print(e)
```

```
hello world
5
2.5
list index out of range
```


Error vs Exception

An exception is an error that can be handled by a programmer.

An exception which are not handled by programmer, becomes an error.

All exceptions occur only at runtime.

Error may occur at compile time or runtime.

Error vs Warning

It is compulsory to handle all error otherwise program will not execute, while warning represents a caution and even though it is not handled, the program will execute.

Errors are derived as sub class of StandardError, while warning derived as sub class from Warning class.