In [ ]:
```
#types of Error
1.Syntax Error --> those error which are coming due to the programmers fault
2.Logical Error
3.RunTime Error
```

In [ ]:
```
Syntax Error -> We are responsible to correct the syntax error
Example:
    print "hello world"
```

In [1]:
```
logical error--> the errrors that are coming due to the incorrect6 logic
Example:
    def add(a,b):
        return a-b
The error is coming because we have written wrong logic.In logical error the syntax of the code
of the code is correct but the required output is not according to our need
```

```
  Input In [1]
    print "hello world"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello world")?
```

In [2]:
```
def fact(num):
    fact=0
    for i in range(0,num+1):
        fact=fact*1
    print(fact)
fact(10)
```

```
0
```

In [ ]:
```
#Runtime Error
are also known as exception
#while executing any program if something goes wrong because of the user input.Once you will not
get any syntax error then there is a chanceof getting run time error.
Exception handling can only be done with runtime error
```

In [7]:
```
n=int(input("Enter n"))
x=int(input("Enter x"))
z=n/x
print(z)
```

```
Enter n5
Enter x0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [7], in <cell line: 3>()
      1 n=int(input("Enter n"))
      2 x=int(input("Enter x"))
----> 3 z=n/x
      4 print(z)

ZeroDivisionError: division by zero
```

In [8]:
```
n=int(input("Enter a number"))
print(n)
```

```
Enter a numberten
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [8], in <cell line: 1>()
----> 1 n=int(input("Enter a number"))
      2 print(n)

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

In [ ]:
```
#Note It is highly recommeneded to handle the exceptions if you are not handling
the exception the whole program will terminate abnormally
```

In [9]:
```
n=int(input("Enter n"))
x=int(input("Enter x"))
z=n/x
print(z)  #
print(n)
print(x)
```

```
Enter n10
Enter x0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [9], in <cell line: 3>()
      1 n=int(input("Enter n"))
      2 x=int(input("Enter x"))
----> 3 z=n/x
      4 print(z)  #
      5 print(n)

ZeroDivisionError: division by zero
```

In [ ]:
```
In python each and every exception is an object. for each object there is one seperate
class are also available.
Whenever pvm faces runtime error or exception then PVM will create the objcct of the correcposing
exception class.
The rest program will not executed
All Exceptions are the child class of BaseException class
```

In [ ]:
```
Each and every expection is a child class of exception
```

In [ ]:
```
Exception Handling --> if the exception is encountered then exception handling gives you a chance
to handle that exception by providing the alternative way to perform a task.


How we can handle exceptions:
    With the help of try except
```

In [ ]:
```
Try Block : In try block we always write risky code. (Risky code means that code beacuse od that
                                                      a certain exception is coming)
    The codes that may lead to exceptions are always write inside try block

Except : In except block always write the code that is used to handled the try block error.
    The code that is used to handle the error that are occured inside the try block are
    generally written in except block.
```

In [ ]:
```
Example:
    try:
        Risky code
    except ZeroDivisionError:
        Handling code/Alternative code
```

In [10]:
```
#Without using try except:
print("Stmt-1")
print(10/0)
print("Stmt-2")
```

```
Stmt-1
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      1 #Without using try except:
      2 print("Stmt-1")
----> 3 print(10/0)
      4 print("Stmt-2")

ZeroDivisionError: division by zero
```

In [16]:
```
#With Try Block
print("Stmt-1")
try:
    print(10/0)
except ZeroDivisionError:
    print("Zero Division is not possible")
except ValueError:
    print("Value error")
print("Stmt-2")
#Note: if try block is not getting any error then except block never be executed
#whenever you are using try execept block your program will terminate normalle
```

```
Stmt-1
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [16], in <cell line: 3>()
      2 print("Stmt-1")
      3 try:
----> 4     print(10/0)
      5 except ValueError:
      6     print("Zero Division is not possible")

ZeroDivisionError: division by zero
```

In [19]:
```
#With Try Block
print("Stmt-1")
try:
    print(10/0)
except ZeroDivisionError:
    try:
        print(5/0)
    except:
        print("zero division")
print("Stmt-2")
#Note: if try block is not getting any error then except block never be executed
#whenever you are using try execept block your program will terminate normalle
#Nested Try except block is also possible but for each try block there must be an except block
```

```
  Input In [19]
    print("Stmt-2")
    ^
IndentationError: unexpected unindent
```

In [27]:
```
#Try with multiple except block
try:
    n=int(input("eNTER A NUMBER"))
    n1=int(input("enter a number"))
    print(n/n1)
except ZeroDivisionError:
    print("Please enter n1 value otehr than 0")
except:
    print("Error")
#the except block must be the last block of the code if you are using except block in between
the blocks
then it will give you an error you cannot use except block in between any block.
```

```
  Input In [27]
    print(n/n1)
    ^
SyntaxError: default 'except:' must be last
```

In [30]:
```
#finally block
Sometimes we need something to exccute in our program weather the exception is occured or not,
finally block is always be excuted weather the exception is occured or not . Basically finally block
are used to perform cleanup activities(DB connection closinh , resource allocation , gc)
Example:
    try:
        risky code
    except:
        alternative code/handling code
    finally:
        cleanup code
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [30], in <cell line: 1>()
----> 1 print(i)

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

In [31]:
```
Case1:
try:
    print("hello")
except:
    print("world")
finally:
    print("Hello")
```

```
hello
Hello
```

In [33]:
```
#Case2:
try:
    print("inside Try")
    print(10/0)
except:
    print("Except")
finally:
    print("Finally")
```

```
inside Try
Except
Finally
```

In [34]:
```
#Case3:
try:
    print("inside Try")
    print(10/0)
except NameError:
    print("Except")
finally:
    print("Finally")
```

```
inside Try
Finally
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [34], in <cell line: 2>()
      2 try:
      3     print("inside Try")
----> 4     print(10/0)
      5 except NameError:
      6     print("Except")

ZeroDivisionError: division by zero
```

In [ ]:
```
Two Types of Exceptions :
Predefined Exceptions--> for each exception a seprease class is given we can use that class
Example: ZerodivisionError, nameerror,valueerror,eof error etc
Userdefined Exceptions--> Customised exceptions or programmatic
example:
    too young exception
    toooldexception
    insufiicientfund
```

In [ ]:
```
#Creation of user defined exceptions : Tooyoungexception
                                       Toooldexception
```

In [41]:
```
class TooYoungExceptions(Exception):
    def __init__(self,str):
        self.str=str


class TooOldExceptions(Exception):
    def __init__(self,str):
        self.str=str


age=int(input())

if age>60:
    raise TooOldExceptions("Your are retired personlity")
elif age<18:
    raise TooYoungExceptions()
else:
    print("you are perfect")
```

```
11
---------------------------------------------------------------------------
TooYoungExceptions                        Traceback (most recent call last)
Input In [41], in <cell line: 12>()
     13     raise TooOldExceptions("Your are retired personlity")
     14 elif age<18:
---> 15     raise TooYoungExceptions()
     16 else:
     17     print("you are perfect")

TooYoungExceptions:
```

In [ ]:
```
#ternary operator  --> Conditional expression in the form of operator.
ternary operator are generally used as if else statement in asingle line
Syntax of ternary operator in python:
    [on_true] if expression else [on_false]
```

In [43]:
```
a=30
b=20
m=a if a<b else b
m
```

Out[43]:
```
20
```

In [47]:
```
#Enumerate Function--> it deals with the iterators if you want to count the
number of iterators than you can use enumerate number. Enumerate adds an additonal counter
that will count the index and return it in the form of enumerate object
Synatx:
    enumerate(iterable,start=0)
```

```
100 10
101 20
102 30
103 40
104 50
```

In [50]:
```
#Example
s1="Sujit"
print(enumerate(s1,100))
```

```
[(100, 'S'), (101, 'u'), (102, 'j'), (103, 'i'), (104, 't')]
```