Types of Errors

In [ ]:
```
Types of Error:
    1.Syntax Error
    2.logical Error
    3.Runtime Error
```

Syntax Error

In [ ]:
```
Syntax Error --> these error are occuring  due to invalid syntaxes.
             --> Example:
                     print "Hello World"
             --> Programmer is responsible for these types of syntax error.
             --> Syntax error are easily handled by programmer.
```

Examples of Syntax Error

In [1]:
```
def add(a,b)
    print(a,b)
    return
```

```
  Input In [1]
    def add(a,b)
                ^
SyntaxError: invalid syntax
```

In [2]:
```
print "hello world"
```

```
  Input In [2]
    print "hello world"
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print("hello world")?
```

Logical Errors

In [ ]:
```
Logical Error --> In case of logical error you will not get any kind of error but the required output is not up
                  to the mark.
              --> If your logic is incorrect then you will get logical error.
              --> 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.
```

Examples of Logical Errors

In [3]:
```
#5! --> 1!*2!*3!*4*5!
def factorial(num):
    fact=1
    for i in range(0,num+1):
        fact=fact*i
    return fact
print(factorial(5))
```

```
0
```

In [4]:
```
def add(a,b):
        return a-b
add(10,5)
```

Out[4]:
```
5
```

Runtime Errors

In [ ]:
```
Runtime Error --> it is also known as Exceptions
              --> While executing a program if something goes wrong because of any end userinput,data. then the
                  error that is coming is known as Runtime Error.
              --> After completing the compilation process if something goes wrong then runtime error will there.
              --> Exceptions will be occuring at the runtime only.
```

Example of Runtime Error

In [28]:
```
n=int(input("ENTER A NUMBER: "))
a=int(input("Enter a number: "))
y=n/a
print(y)
#quotient
```

```
ENTER A NUMBER: 10
Enter a number: 0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [28], in <cell line: 3>()
      1 n=int(input("ENTER A NUMBER: "))
      2 a=int(input("Enter a number: "))
----> 3 y=n/a
      4 print(y)

ZeroDivisionError: division by zero
```

In [30]:
```
n=int(input("Enter a Number"))
print(n)
```

```
Enter a Numberten
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [30], in <cell line: 1>()
----> 1 n=int(input("Enter a Number"))
      2 print(n)

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

Why Exception Handling?

In [ ]:
```
--> It is highly recommeneded to handle the exceptions if you are not handling
    the exception the whole program will terminate abnormally.
--> After Exception No any statement will be executed.
```

In [ ]:
```
while True:
    print("Hello ")

#Abnormal Termination
```

About Exceptions in Python.

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 obejct of the corresponding exception class.
--> All Exceptions are the child class of BaseException class.
--> Each and every expection is a child class of exception.
```

About Exception Handling In Python

In [ ]:
```
Exception Handling -->  if the execption is occuring at the runtime then exception handling will give
                        you a chance to handle the runtime exceptions by providing an alternative way.
                   -->  We can handle exceptions with the help of Try and except block.
```

Try and Except Block

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 Block    --> 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.


Note 1.:
    try:
        risky code
    except:
        Handlingcode/alernative code

Note 2: Except block will be executed if any error is occuring in try block otherwise try block will be executed We can also
        handle multiple exceptions at a time
```

Examples

Without Try Except Block

In [5]:
```
#Without try except block
print("Stmt-1")
print(10/0)
print("Stmt-2")
```

```
Stmt-1
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [5], in <cell line: 3>()
      1 #Without try except block
      2 print("Stmt-1")
----> 3 print(10/0)
      4 print("Stmt-2")

ZeroDivisionError: division by zero
```

With Try Except Block

In [6]:
```
try:
    print(10/2)
except ZeroDivisionError:
    print("Don't give 0 in denominator")
```

```
5.0
```

More Example of Try Except Block

In [7]:
```
try:
    a=int(input("Enter a Number : "))
    b=int(input("Enter a Number : "))
    print(a/b)
except:
    print("Please give integer input ")
```

```
Enter a Number : 10
Enter a Number : 0
Please give integer input
```

Try With Multiple Except Block

In [7]:
```
try:
    a=int(input("Enter a Number : "))
    b=int(input("Enter a Number : "))
    print(a/b)
except ValueError:
    print("Please give integer input ")
except ZeroDivisionError:
    print("Denominator cannot be zero")
```

```
Enter a Number : ten
Please give integer input
```