Exception Handling in Python

An exception is an error that occurs during program execution.
When an exception occurs in an application, the system throws an error. The error is handled through the process of exception handling. The errors in a program are called as "bugs", and the process of removing them are called as "debugging"

Types of Errors

The three types of errors that can occur in an application are:

- Compile-time errors (Syntax errors)
- Run-time errors
- Logical errors

Syntax errors

We write many program statements using code constructs.
Sometimes we may miss colon or any other punctuation in syntax. This is a syntax error.

Such errors are detected by compiler and line number along with the description is displayed by the Python compiler.

Run-time errors

When PVM cannot execute a byte code, it flags run-time error.
Example:
Insufficient memory to store,
Inability to execute some statements

Runtime errors are not detected by Python compiler. They are detected by PVM only at runtime

```
mylist=[12,13,14,15]
print(mylist)
print(mylist[4])
```

Division by zero is a run-time error.

```
num1 = int(input("Enter Number1:\n"))
num2 = int(input("Enter Number2:\n"))
num3=num1/num2
print("num1/num2 = " , num3)
```

Logical errors

These errors depict the flaws in the logic of the program.
These errors may occur when a programmer wrongly defined the logical statements of the program.

Logical errors are not detected by compiler or PVM The programmer is solely responsible for them.

Objective of the program is to divide the first number by second number. The programmer written as second / first, hence

```
num1 = int(input("Enter Number1:\n"))
num2 = int(input("Enter Number2:\n"))
num3=num2/num1
print("num1/num2 = " , num3)
```

This is a logical error. The correct statement should have been:
Num3=Num1/Num2;

```
# Program to increment the salary of an employee by 15%

def increSal(sal):
sal=sal*15/100
    return sal

salary=increSal(5000)
print("Incremented Salary ",salary)
```

The correct statement is

```
 sal=sal+(sal*15/100)
```

Exception Handling

An exception is a run time error which can be handled by the programmer.
The programmer should guess the error causes in the program and need to be eliminate the error. All exceptions are represented as classes in Python
The Exceptions which are already available in Python are called as built in Exceptions.
The base class for all built in exception is "BaseException" class.
From BaseException, the sub class "Exception" is derived. from Exception class, the sub classes "StandardError' and "Warning" are derived

Exceptions in a program are handled by using exception handlers. To implement exception handling, a program is divided into blocks of code.

The blocks for exception handling can be implemented using the following keywords:

Python **try except**
Block
The try...except block is used to handle exceptions in Python.

Syntax:

```
 try:
    # code that may cause exception except:
    # code to run when exception occurs
```

Here, we have placed the code that might generate an exception inside the try block.
Every try block is followed by an except block.

When an exception occurs, it is caught by the except block. The except block cannot be used without the try block.

Catching Specific Exceptions in Python
For each try block, there can be zero or more except blocks.

Example:

```
 try:
    num1 = int(input("Enter Number1:\n"))
    num2=  int(input("Enter Number1:\n"))
    num3=num1/num2
    print("num1/num2 =” ,num3)
except:
   print("Divisible by Zero attempted ")
```

Note: Here the except block has no specific class to hold the exception

Example -2:

```
 try:
    num1 = int(input("Enter Number1:\n"))
    num2 = int(input("Enter Number2:\n"))
    num3=num1/num2
   print("num1/num2 = " , num3)
except ZeroDivisionError as e:
  print(e)
```

Multiple except blocks allow us to handle each exception differently.
The argument type of each except block indicates the type of exception that can be handled by it.

```
num1=int(input("Enter Number1"))
num2=int(input("Enter Number2"))
try:
          print(num1/num2)
          print("No print statement or message")
          print('10'+num2)
except TypeError:
          print("Attempted to add incompatible types")
except ZeroDivisionError:
          print("Attempted to divided by 0")
except ZeroDivisionError:
          print("Cannot divide by 0")
except:
         print("Something went wrong")
```

Finally Block in Python

To execute a block in all circumstances, irrespective to the exception,
You can include a finally exception block after the last except block

```
num1=int(input("Enter Number1 \n"))
num2=int(input("Enter Number2 \n"))
try:
        print(num1/num2)
        print("No exception")
except ZeroDivisionError as e:
        print(e)
        print("Attempted to divided by 0")
 finally:
        print("Block executes no matter Exception raised or not .")
```