HANDLING EXCEPTION

Exception

An Exception is an event which occurs during the execution of a program that disrupts the normal flow of the program's instructions. If a python script encounters a situation it cannot cope up, it raises an exception

HOW TO HANDLE EXCEPTION?

There are four blocks which help to handle the exception. They are

- try block
- except statement
- else block
- finally block

i) *try block*

- In the try block a programmer will write the suspicious code that may raise an exception. One can defend their program from a run time error by placing the codes inside the try block.

Syntax:

```
try:
   #The operations here
```

ii) except statement

- Except statement should be followed by the try block.
- A single try block can have multiple except statement.
- The except statement which handles the exception.
- Multiple except statements require multiple exception names to handle the exception separately.

```
except Exception 1:
   #Handle Exception 1
except Exception 2:
   #Handle Exception 2
```

iii) else block

If there is no exception in the program the else block will get executed.

Syntax:

```
else:
   #If no Exception, it will execute
```

iv) finally block:

A finally block will always execute whether an exception happens or not the block will always execute.

```
finally:
   #Always Execute
```

Syntax:

```
try:
   #The operations here
except Exception 1:
   #Handle Exception 1
except Exception 2:
   #Handle Exception 2
else:
   #If no Exception, it will execute
finally:
   #Always Execute
```

(i) Write a python program to write a file with exception handling.

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    f.seek(0)
    print(f.read())
except IOError:
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

(ii) Write a python program to read a file which raises an exception

Assume test.txt is not created in the computer and the following program is executed which raises an exception.

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    f.seek(o)
    print(f.read())
except IOError:
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

a) Except Clause with no exception

An except statement without the exception name, the python interpreter will consider as default 'Exception' and it will catch all exceptions.

Syntax:

```
try:
   #The operations here
except:
    #Handles Exception
else:
   #If no Exception, it will execute
finally:
   #Always Execute
```

b) Except clause with multiple Exception

An except statement can have multiple exceptions, We call it by exception name

Syntax:

```
try:
   #The operations here
except (Exception 1, Exception 2):
    #Handles Exception
else:
   #If no Exception, it will execute
finally:
   #Always Execute
```

(i) Write a python program to read a file with multiple exceptions

```
try:
    f=open("test.txt", 'w+')
    f=write("My First File")
    print(f.read())
except (IOError, ValueError,
ZeroDivisionError):
    print("File not Found")
else:
    print("File not Found")
    f.close()
finally:
     print("Close a file")
```

c) Argument of an Exception

An exception can have an argument which is a value that gives additional information about the problem. The content of an argument will vary by the exception.

```
try:
   #The operations here
except Exception Type, Argument:
     #Handles Exception with Argument
else:
   #If no Exception, it will execute
finally:
   #Always Execute
```

d) Raising an Exception

You can raise exceptions in several ways by using raise statement

Syntax:

```
raise Exception, Argument:
```

Example:

```
def fun(level):
        if level<10:
                raise "Invalid Level", level
fun(5)   #raise an Exception
fun(11)
```

TYPES OF EXCEPTION

There are two types of Exception:

- Built-in Exception
- User Defined Exception

i) Built-in Exception

There are some built-in exceptions which should not be changed.

The Syntax for all Built-in Exception

```
except Exception_Name
```

S.No	Exception Name	Description
1	Exception	It is the Base class for all Exceptions
2	ArithmeticError	It is the Base class for all Errors that occur on numeric calculations.
3	ZeroDivisionError	Raised when a number is divided by zero
4	IOError	Raised when an Input or Output operation fails such as open() function. When a file is not exist in the folder
5	TypeError	Raised when operation or function is invalid for a specified data type.
6	ValueError	Raised when built-in function for a data type has valid arguments and it has invalid values.
7	RuntimeError	Raised when a generated error does not fall into any category.
8	KeyboardInterrupt	Raised when the user interrupts program execution by pressing Ctrl+C
9	FloatingPointError	Raised when floating calculation Fails.

S.No	Exception Name	Description
10	AssertionError	Raised in case of failure of assert statement.
11	OverFlowError	Raised when a calculation exceeds maximum limit for a numeric types.
12	StandardError	Base class for all built-in exception except 'StopIteration and SystemExit'
13	StopIteration	Raised when next() method of an iteration does not exist
14	SystemExit	Raised by the sys.exit() function
15	SyntaxError	Raised when there is an error in Python Syntax
16	IndentationError	Raised when Indentation is not specified properly
17	AssertionError	Raised in case of failure of assert statement.

USER DEFINED EXCEPTION

In Python a user can create their own exception by deriving classes from standard Exceptions. There are two steps to create a user defined exception.

Step-1

A User Defined Exception should be derived from standard Built-in Exceptions.

Step-2

After referring base class the user defined exception can be used in the program

```
class NetworkError(RuntimeError):
def __init__ (self,arg):
                self.args=arg
try:
   raise NetworkError("Bad host name")
except NetworkError,e:
   print(e.args)
```

ASSERTION

An assertion is a sanity check which can turn on (or) turn off when the program is in testing mode.

- The assertion can be used by assert keyword. It will check whether the input is valid and after the operation it will check for valid output.

Syntax:

```
assert(Expression)
```

Example

```
def add(x):
        assert(x<0)
        return(x)
add(10)
```