Exception Handling in Python

Error in Python can be of two types i.e. Syntax errors and Exceptions. Errors are the problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occur which changes the normal flow of the program.

Difference between Syntax Error and Exceptions

Syntax Error: As the name suggests this error is caused by the wrong syntax in the code. It leads to the termination of the program.

Example:

```
# initialize the amount variable
amount = 10000

#check eligibility whether amount is above 3000 or not
if(amount > 3000)
print("You are eligible")
```

Exceptions: Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program; however, it changes the normal flow of the program.

Example:

```
# initialize the amount variable
marks = 10000

# perform division with 0
a = marks / 0
print(a)
```

In the above example raised the ZeroDivisionError as we are trying to divide a number by 0

Try and Except Statement – Catching Exceptions

Try and except statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause.

Example: Let us try to access the array element whose index is out of bound and handle the corresponding exception.

```
# Python program to handle simple runtime error

a = [1, 2, 3]
try:
    print ("Second element = %d" %(a[1]))
```

```
    # Throws error since there are only 3 elements in array
    print ("Fourth element = %d" %(a[3]))

except:
    print ("An error occurred")
```

In the above example, the statements that can cause the error are placed inside the try statement. The second print statement tries to access the fourth element of the list which is not there and this throws an exception. This exception is then caught by the except statement.

Catching Specific Exception

A try statement can have more than one except clause, to specify handlers for different exceptions. Please note that at most one handler will be executed. For example, we can add IndexError in the above code. The general syntax for adding specific exceptions is –

Syntax:

```
try:
    # statement(s)
except IndexError:
    # statement(s)
except ValueError:
    # statement(s)
```

Example:

```
# Program to handle multiple errors with one
# except statement

def fun(a):
    if a < 4:

        # throws ZeroDivisionError for a = 3
        b = a/(a-3)

    # throws NameError if a >= 4
    print("Value of b = ", b)

try:
    fun(3)
    fun(5)

# note that braces () are necessary here for
# multiple exceptions
except ZeroDivisionError:
    print("ZeroDivisionError Occurred and Handled")
except NameError:
    print("NameError Occurred and Handled")
```

Try with Else Clause

In python, you can also use the else clause on the try-except block which must be present after all the except clauses. The code enters the else block only if the try clause does not raise an exception.

Example:

```
# Program to depict else clause with try-except
# Function which returns a/b
def new(a , b):
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print ("a/b result in 0")
    else:
        print (c)

# Driver program to test above function
new(2.0, 3.0)
new(3.0, 3.0)
```

Finally Keyword in Python

Python provides a keyword finally, which is always executed after the try and except blocks. The final block always executes after normal termination of try block or after try block terminates due to some exception.

Syntax:

```
try:
    # Some Code....
except:
    # optional block
    # Handling of exception (if required)
else:
    # execute if no exception
finally:
    # Some code .... (Always executed)
```

Example:

```
try:
    k = 5//0 # raises divide by zero exception.
    print(k)

# handles zerodivision exception
except ZeroDivisionError:
    print("Can't divide by zero")

finally:
    # this block is always executed regardless of exception generation.
    print('This is always executed')
```