Exception Handling

Introduction

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.

Exceptions are raised when some internal events occur that change the normal flow of the program.

Two types of Error occurs in python.

1. Syntax errors
2. Logical errors (Exceptions)

1. Syntax errors :

When the proper syntax of the language is not followed then a syntax error is thrown. # initialize the amount variable

---

Example :

```
amount = 10000

# check that You are eligible to

# purchase Dsa Self Paced or not

if(amount>2999)

        print("You are eligible to purchase Dsa Self Paced")
```

It returns a syntax error message because after the if statement a colon: is missing.

Output :

```
  File "/home/ac35380186f4ca7978956ff46697139b.py", line 4
    if(amount>2999)
                  ^
SyntaxError: invalid syntax
```

2. **logical errors(Exception)**

When in the runtime an error that occurs after passing the syntax test is called exception or logical type.

For example, when we divide any number by zero then the ZeroDivisionError exception is raised, or when we import a module that does not exist then ImportError is raised.

Example :

```
# initialize the amount variable

marks = 10000

# perform division with 0

a = marks / 0

print(a)
```

ZeroDivisionError as we are trying to divide a number by 0.

Output :

```
Traceback (most recent call last):
  File "/home/f3ad05420ab851d4bd106ffb04229907.py", line 4, in <module>
    a=marks/0
ZeroDivisionError: division by zero
```

Built-in Exceptions

Exception	Description
IndexError	When the wrong index of a list is retrieved.
AssertionError	It occurs when the assert statement fails
AttributeError	It occurs when an attribute assignment is failed.
ImportError	It occurs when an imported module is not found.
KeyError	It occurs when the key of the dictionary is not found.
NameError	It occurs when the variable is not defined.
MemoryError	It occurs when a program runs out of memory.
TypeError	It occurs when a function and operation are applied in an incorrect type.

Error Handling

When an error and an exception are raised then we handle that with the help of the Handling method.

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 :

```
a=int(input('Enter a Number:'))
b=int(input('Enter a divisor'))

try:
      c=a/b
      print('The Divison is ',c)
      a=[1,2,3]
      print(a[4])
except:
      print('Error Occured')
```

In this example, the error is handled commonly, The error may be divisible by 0 if the divider value is inputted as 0 or and Index out of range.

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.

The general syntax for adding specific exceptions are –

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

Example :

```
a = int(input('Enter a Number'))
try:
    if a<4:
        #throws ZeroDivisionError for a = 3
        b=a/(a-3)

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

except ZeroDivisionError:
    print("ZeroDivisionError occured and handeled")
except NameError:
    print("NameError occured and handeled")
```

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.

```
a = int(input('Enter first Number:'))
b = int(input('Enter second Number:'))

try:
    c = ((a+b)/(a-b))
except:
    print('ZeroDivisionError Occured')
else:
    print('C = ',c)
```

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 :

```
a = int(input('Enter first Number:'))
b = int(input('Enter second Number:'))

try:
    c=a/b
except ZeroDivisionError:
    print('Error Occured : Number can no be divided by 0')
else:
    print('C=',c)
finally:
    print('Finally block is always executed')
```

Raising Exception

The raise statement allows the programmer to force a specific exception to occur.

The sole argument in raise indicates the exception to be raised.

This must be either an exception instance or an exception class (a class that derives from Exception).

Example :

```
a = int(input('Enter first Number:'))
b = int(input('Enter second Number:'))

try:
    if b==0:
        raise ZeroDivisionError('Divide by 0 error occured')
    c=a/b
except ZeroDivisionError as e:
    print('Error Occured :', e)
    raise
else:
    print('C=',c)
finally:
    print('Finally block executed')
```