Exception Handling in Python

Created by :- The Easylearn Academy
9662512857

What is exception in python

- Exception means run time error in python program. Exception are type of error which may or may not occur during execution of program.
- Exception depends upon user input or environment in which program runs.
- For example in program which divides one from another number given by user, if user give correct give both number correctly then error will not come and program will finish sucessfully
- But if one or both number are not valid then program will suddenly stop (which is called crash) now such situation is called exception.
- If any type of exception occur in program then program can not complete task for which it is developed.
- So must develop program in way that if exception occur in program then program should recover itself from it and try to finish given task.
- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.

Catching Exceptions in Python

- In Python, exceptions can be handled using a try statement.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.

It is up to us, what operations we perform once we have caught the exception.

Syntax

Here is simple syntax of try....except...else blocks −

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Important points

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

Example

```
# import module sys to get the type of exception
import sys
randomList = ['a', 0, 2]
for entry in randomList:
    try:
        print("The entry is", entry)
        r = 1/int(entry)
        break
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Next entry.")
        print()
print( "Good Bye")
```

The *except* Clause with No Exceptions

```
try:
   You do your operations here;
   ......................
except:
   If there is any exception, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

- This kind of a **try-except** statement catches all the exceptions that occur.
- Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

The *except* Clause with Multiple Exceptions

```
you can also use the same except statement to handle multiple
   exceptions as follows –

try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

The try-finally Clause

- You can use a finally: block along with a try: block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this −

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be skipped.
finally:
   This would always be executed.
   ......................
```

Example 1

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

Example 2

```
def getAge():
        try:
                val = int(input("what is your age? "))
                print val
        except ValueError as e:
                print ("Error - ",e)
                return “invalid date"
        finally:
                print “program ends"
return val

getAge()
```

Example 3

```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Else

- You can use the else keyword to define a block of code to be executed if no errors were raised:
- Example

- In this example, the try block does not generate any error:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Raising an Exceptions

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise.
- We can also optionally pass in value to the exception to clarify why that exception was raised.

```
try:
        age = int(input("Enter the age?"))
        if age<18:
            raise ValueError;
        else:
            print("the age is valid")
    except ValueError:
        print("The age is not valid")
```

Custom Exception / user defined exception

- python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.
- In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived form this class.
- We will learn this topic after we learn object and classes in python