Python Exceptions

Python Exceptions, Handling an Exception, The finally block, Raising exceptions, Custom Exception

Gaurav Kr. suman
4/14/20
DSPMU_MCA/MSIT

Python Exceptions

An exception can be defined as an abnormal condition in a program resulting in the disruption in the flow of the program.

Whenever an exception occurs, the program halts the execution, and thus the further code is not executed. Therefore, an exception is the error which python script is unable to tackle with.

Python provides us with the way to handle the Exception so that the other part of the code can be executed without any disruption. However, if we do not handle the exception, the interpreter doesn't execute all the code that exists after the that.

Common Exceptions

A list of common exceptions that can be thrown from a normal python program is given below.

1. **ZeroDivisionError:** Occurs when a number is divided by zero.
2. **NameError:** It occurs when a name is not found. It may be local or global.
3. **IndentationError:** If incorrect indentation is given.
4. **IOError:** It occurs when Input Output operation fails.
5. **EOFError:** It occurs when the end of the file is reached, and yet operations are being performed.

Problem without handling exceptions

```
a = int(input("Enter a:"))
b = int(input("Enter b:"))
c = a/b;
print("a/b = %d"%c)
```

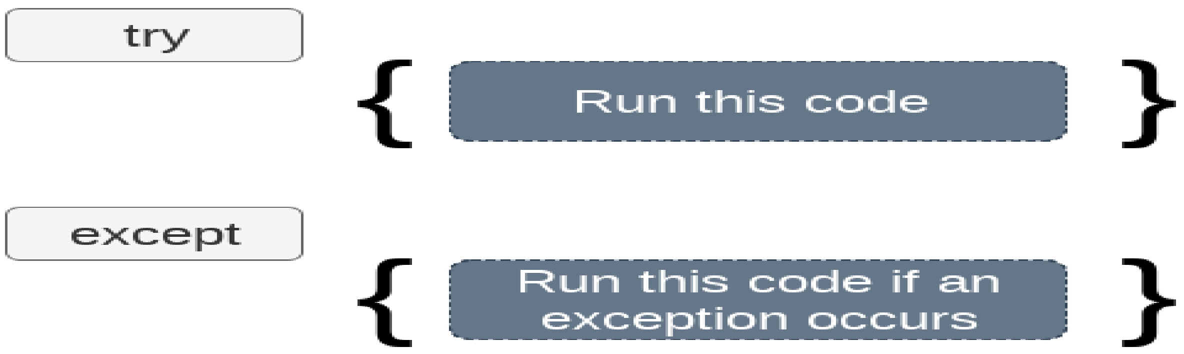
Output:

```
Enter a:5
Enter b:0
Traceback (most recent call last):
  File "C:/Exception.py", line 3, in <module>
    c = a/b;
.ZeroDivisionError: division by zero
```

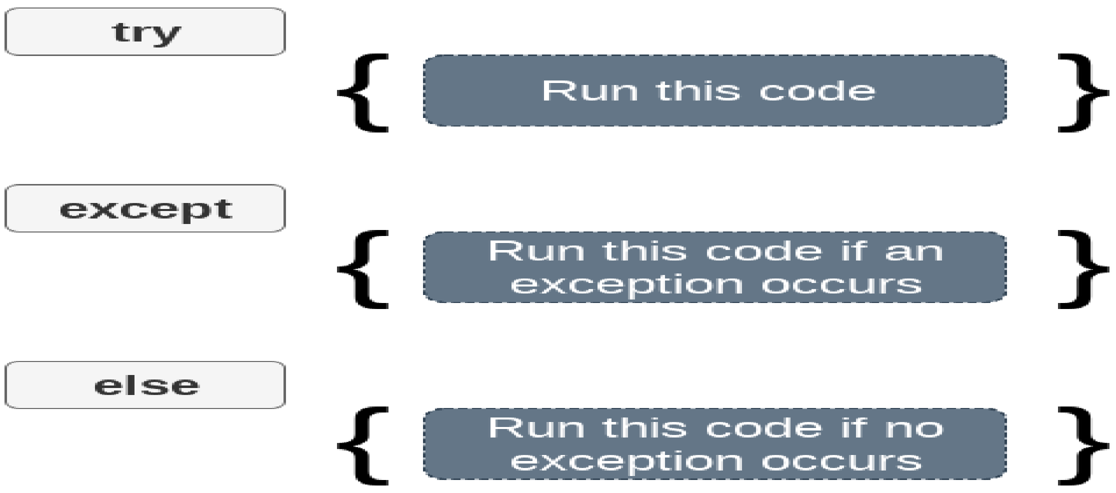
Handling an Exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax 1



Syntax 2(with else):


try
{ Run this code }
except
{ Run this code if an exception occurs }
else
{ Run this code if no exception occurs }

Example

```
try:
   a = int(input("Enter a:"))
   b = int(input("Enter b:"))
   c = a/b;
   print("a/b = %d"%c)
except Exception:
   print("can't divide by zero")
else:
   print("Else execute")
```

Output:

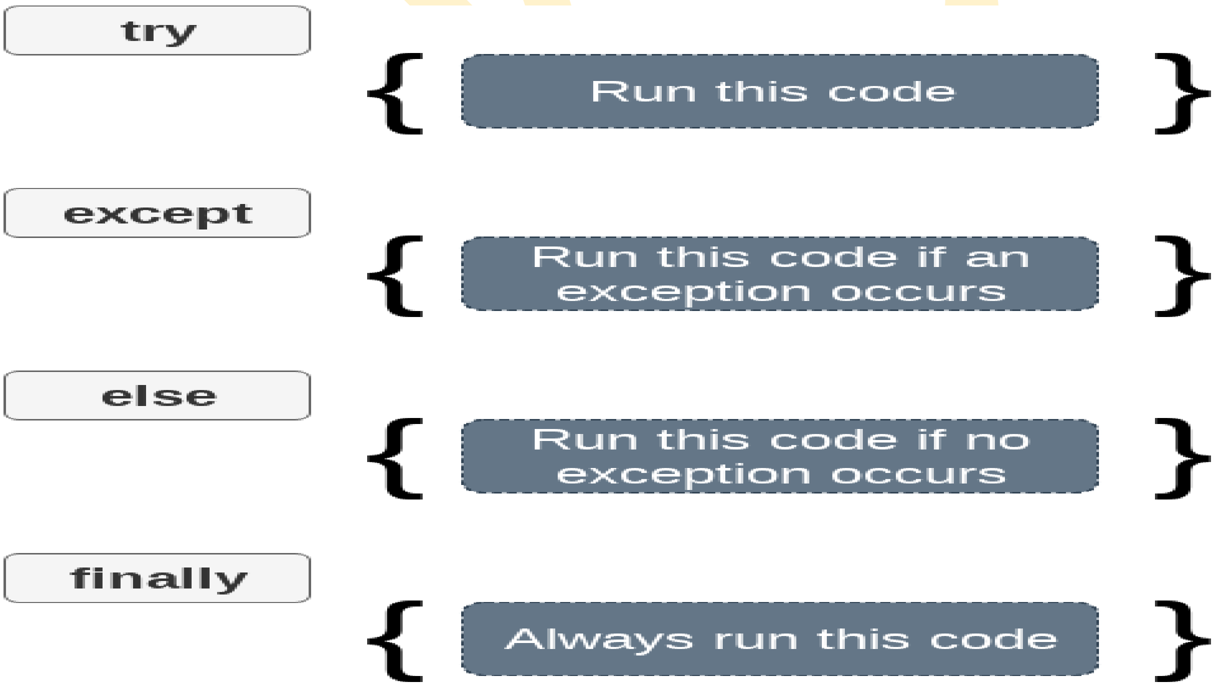
```
Enter a:5
Enter b:0
can't divide by zero
```

Note:

1. Python facilitates us to not specify the exception with the except statement.
2. We can declare multiple exceptions in the except statement since the try block may contain the statements which throw the different type of exceptions.
3. We can also specify an else block along with the try-except statement which will be executed if no exception is raised in the try block.
4. The statements that don't throw the exception should be placed inside the else block.

The finally block

We can use the finally block with the try block in which, we can pace the important code which must be executed before the try statement throws an exception.


try
{ Run this code }
except
{ Run this code if an exception occurs }
else
{ Run this code if no exception occurs }
finally
{ Always run this code }

Example:

```
try:
  fileptr = open("file.txt","r")
  try:
    fileptr.write("Hi I am good")
  finally:
    fileptr.close()
    print("file closed")
except:
  print("Error")
```

Output:

```
file closed

Error
```

Raising exceptions

An exception can be raised by using the raise clause in python. The syntax to use the raise statement is given below.

Syntax

```
raise Exception_class,<value>
```

Note:

1. To raise an exception, raise statement is used. The exception class name follows it.
2. An exception can be provided with a value that can be given in the parenthesis.
3. To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

```
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")
```

Output:

```
Enter the age?17

The age is not valid
```

Custom Exception

The python allows us to create our exceptions that can be raised from the program and caught using the except clause. However, we suggest you read this section after visiting the Python object and classes.

```
class ErrorInCode(Exception):
    def __init__(self, data):
        self.data = data
    def __str__(self):
        return repr(self.data)

try:
    raise ErrorInCode(2000)
except ErrorInCode as ae:
    print("Received error:", ae.data)
```

Output:

```
Received error: 2000
```