Python Exception Handling

In this article, you'll learn how to handle exceptions in your Python program using try, except and finally statements. This will motivate you to write clean, readable and efficient code in Python.

What are exceptions in Python?

Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
When these exceptions occur, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash.
For example, if function A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A.
If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

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.

Common Exceptions

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

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

Problem without handling exceptions

As we have already discussed, the exception is an abnormal condition that halts the execution of the program. Consider the following example.

Example

```
1.	a = int(input("Enter a:"))
2.	b = int(input("Enter b:"))
3.	c = a/b;
4.	print("a/b",c)
5.
6.	#other code:
7.	print("Hi I am other part of the program")
```

Output:

```
Enter a:10
Enter b:0
Traceback (most recent call last):
  File "exception-test.py", line 3, in <module>
    c = a/b;
ZeroDivisionError: division by zero
```

Exception handling in python

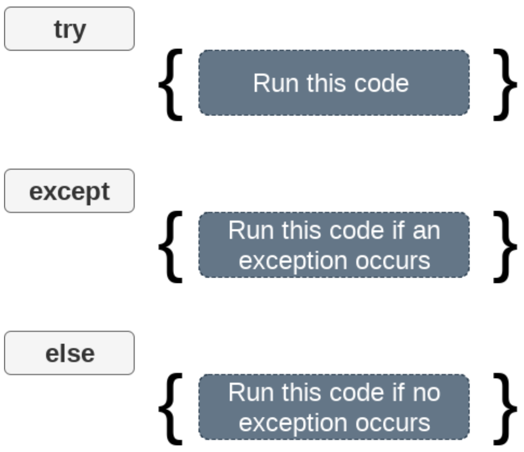
If the python program contains suspicious code that may throw the exception, we must place that code in the try block. The try block must be followed with the except statement which contains a block of code that will be executed if there is some exception in the try block.
We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.
The syntax to use the else statement with the try-except statement is given below.

```
try:
    #block of code

except Exception1:
    #block of code

else:
    #this code executes if no except block is executed

```


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("Hi I am else block")
```

Output:

```
Enter a:10
Enter b:2
a/b = 5
Hi I am else block
```

The except statement with no exception

Python provides the flexibility not to specify the name of exception with the except statement. Consider the following example.

Example

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b;
    print("a/b = %d"%c)
except:
    print("can't divide by zero")
else:
    print("Hi I am else block")
```

Output:

```
Enter a:10
Enter b:0
can't divide by zero
```

Points to remember

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.

Example

```
try:
    #this will throw an exception if the file doesn't exist.
    fileptr = open("file.txt","r")
except IOError:
    print("File not found")
else:
    print("The file opened successfully")
    fileptr.close()
```

Output:

```
File not found
```

Declaring multiple exceptions

The python allows us to declare the multiple exceptions with the except clause. Declaring multiple exceptions is useful in the cases where a try block throws multiple exceptions.

Syntax

```
try:
   #block of code

except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)
```

```
    #block of code

else:
    #block of code
```

Example

```
try:
    a=10/0;
except ArithmeticError,StandardError:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

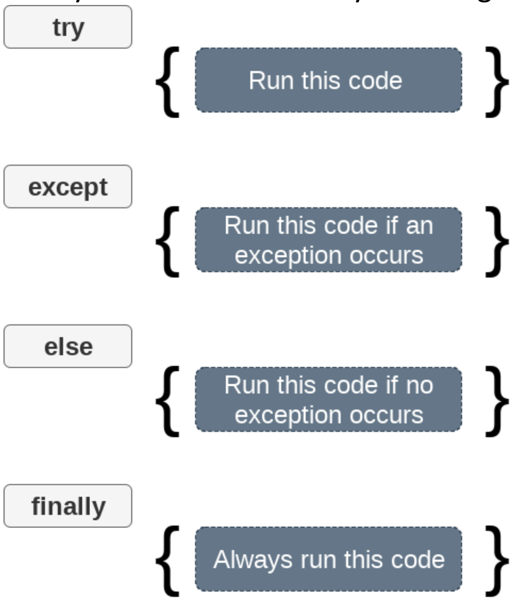
Output:

```
Arithmetic Exception
```

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.

The syntax to use the finally block is given below.


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>
```

Points to remember

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.

Example

```
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
```

Example

```
try:
    a = int(input("Enter a?"))
    b = int(input("Enter b?"))
    if b is 0:
        raise ArithmeticError;
    else:
        print("a/b = ",a/b)
except ArithmeticError:
    print("The value of b can't be 0")
```

Output:

```
Enter a?10
Enter b?0
The value of b can't be 0
```