Exception Handling in Python

Python has many **built-in exceptions** that are raised when your program encounters an error (something in the program goes wrong).

When these exceptions occur, the Python interpreter stops the current process and passes it to the calling process until it is handled. If not handled, the program will crash.

For example, let us consider a program where we have a function **A** that calls function **B**, which in turn call's function **C**. If an exception occurs in function **C** but is not handled in **C**, the exception passes to B and then to **A**.

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.

Catching exceptions in Python

In Python, exceptions can be handled using a try statement.

The critical operation which can raise an exception is placed inside the try clause. The code that handles the exceptions is written in the except clause.

We can thus choose what operations to perform once we have caught the exception. Here is a simple example of showing number when divided by zero.

1. **Running program without try-except**

```
a = 10
b = 0
answer = a / b
print(f"Answer is {answer}")
```

Output

```
Traceback (most recent call last):
  File "C:\Users\aniru\Desktop\Python Programming\try-except.py", line 3, in <module>
    answer = a / b
ZeroDivisionError: division by zero
```

You can see our code stopped in between and will not execute any further. If not handled properly, your code will stop in between and thus resulting in bad user experience.

2. **Running program with try-except**

```
try:
    a = 10
    b = 0
    answer = a / b
    print(f"Answer is {answer}")
except:
    print("Some error has occurred")

print("This is another statement")
```

Output

```
Some error has occurred
This is another statement
```

In this case, we handled our code error properly and program did not stop in between thus resulting in better user experience.

Another example

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

```
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")
```

Output

```
Second element = 2
An error occurred
```

In the above example, the statements that can cause the error are placed inside the try statement (second print statement in our case). 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.

A try clause can have any number of except clauses to handle different exceptions, however, only one will be executed in case an exception occurs.

```
try:
    a = 10
    b = 5
    answer = a / b
    print(f"Answer is {answer}")

    # This will give us NameError because name is not defined
anywhere
    print(f"Hello {name}")
except ZeroDivisionError:
    print("Cannot divide by zero")
except NameError:
    print("Some variable is not defined")
except:
    print("Some error has occurred")
```

Output

```
Answer is 2.0
Some variable is not defined
```

The output above is so because as soon as python tries to access the value of **name**, NameError occurs.

Try with Else statement

In some situations, you might want to run a certain block of code if the code block inside **try** ran without any errors. For these cases, you can use the optional **else** keyword with the **try** statement.

Example

```
try:
    a = 10
    b = 5
    name = "Elon"
    answer = a / b
    print(f"Answer is {answer}")

    # This will give us NameError because name is not defined
anywhere
    print(f"Hello {name}")
except ZeroDivisionError:
    print("Cannot divide by zero")
except NameError:
    print("Some variable is not defined")
except:
    print("Some error has occurred")
else:
    print("Everything worked fine")
```

Output

```
Answer is 2.0
Hello Elon
Everything worked fine
```

Note: Else part will not execute if there are any errors.

Try-Finally Statement

The **try** statement in Python can have an optional **finally** clause. This clause is executed no matter what, and is generally used to release external resources.

For example, we may be connected to a remote data center through the network or working with a file or a Graphical User Interface (GUI).

In all these circumstances, we must clean up the resource before the program comes to a halt whether it successfully ran or not. These actions (closing a file, GUI or disconnecting from network) are performed in the **finally** clause to guarantee the execution.

Example

```
try:
    a = 10
    b = 0
    name = "Elon"
    answer = a / b
    print(f"Answer is {answer}")

    # This will give us NameError because name is not defined
anywhere
    print(f"Hello {name}")
except ZeroDivisionError:
    print("Cannot divide by zero")
except NameError:
    print("Some variable is not defined")
except:
    print("Some error has occurred")
finally:
    print("This is a finally statement")
```

Output

```
Cannot divide by zero
This is a finally statement
```