Python

Python

Exception Handling

try :

{RUN THIS CODES}

except ....

{RUN THIS CODE IF AN EXCEPTION OCCURS}

What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

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.

Python try...except Block

The `try...except` block is used to handle exceptions in Python. Here's the syntax of `try...except` block:

```
try:
    # code that may cause exception
except:
    # code to run when exception occurs
```

Here, we have placed the code that might generate an exception inside the `try` block. Every `try` block is followed by an `except` block.

When an exception occurs, it is caught by the `except` block. The `except` block cannot be used without the try block.

Example: Exception Handling Using try...except

```
try:
    numerator = 10
    denominator = 0

    result = numerator/denominator

    print(result)
except:
    print("Error: Denominator cannot be 0.")

# Output: Error: Denominator cannot be 0.
```

In the example, we are trying to divide a number by **0**. Here, this code generates an exception.

To handle the exception, we have put the code, `result = numerator/denominator` inside the `try` block. Now when an exception occurs, the rest of the code inside the `try` block is skipped.

The `except` block catches the exception and statements inside the `except` block are executed.

If none of the statements in the `try` block generates an exception, the `except` block is skipped.

Catching Specific Exceptions in Python

For each `try` block, there can be zero or more `except` blocks. Multiple `except` blocks allow us to handle each exception differently.

The argument type of each `except` block indicates the type of exception that can be handled by it. For example,

```
try:

    even_numbers = [2,4,6,8]
    print(even_numbers[5])

except ZeroDivisionError:
    print("Denominator cannot be 0.")

except IndexError:
    print("Index Out of Bound.")

# Output: Index Out of Bound
```

In this example, we have created a list named `even_numbers` .

Since the list index starts from **0**, the last element of the list is at index **3**. Notice the statement,

```
print(even_numbers[5])
```

Here, we are trying to access a value to the index **5**. Hence, `IndexError` exception occurs.

When the `IndexError` exception occurs in the `try` block,

- The `ZeroDivisionError` exception is skipped.
- The set of code inside the `IndexError` exception is executed.

Python try with else clause

In some situations, we might want to run a certain block of code if the code block inside `try` runs without any errors.

For these cases, you can use the optional `else` keyword with the `try` statement.

Let's look at an example:

```
# program to print the reciprocal of even numbers

try:
    num = int(input("Enter a number: "))
    assert num % 2 == 0
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

Run Code »

Output

If we pass an odd number:

```
Enter a number: 1
Not an even number!
```

If we pass an even number, the reciprocal is computed and displayed.

```
Enter a number: 4
0.25
```

However, if we pass **0**, we get `ZeroDivisionError` as the code block inside `else` is not handled by preceding `except`.

```
Enter a number: 0
Traceback (most recent call last):
  File "<string>", line 7, in <module>
    reciprocal = 1/num
ZeroDivisionError: division by zero
```

> **Note**: Exceptions in the `else` clause are not handled by the preceding except clauses.

Python try...finally

In Python, the `finally` block is always executed no matter whether there is an exception or not.

The `finally` block is optional. And, for each `try` block, there can be only one `finally` block.

Let's see an example,

```
try:
    numerator = 10
    denominator = 0

    result = numerator/denominator

    print(result)
except:
    print("Error: Denominator cannot be 0.")

finally:
    print("This is finally block.")
```

Output

```
Error: Denominator cannot be 0.
This is finally block.
```

In the above example, we are dividing a number by **0** inside the `try` block. Here, this code generates an exception.

The exception is caught by the `except` block. And, then the `finally` block is executed.