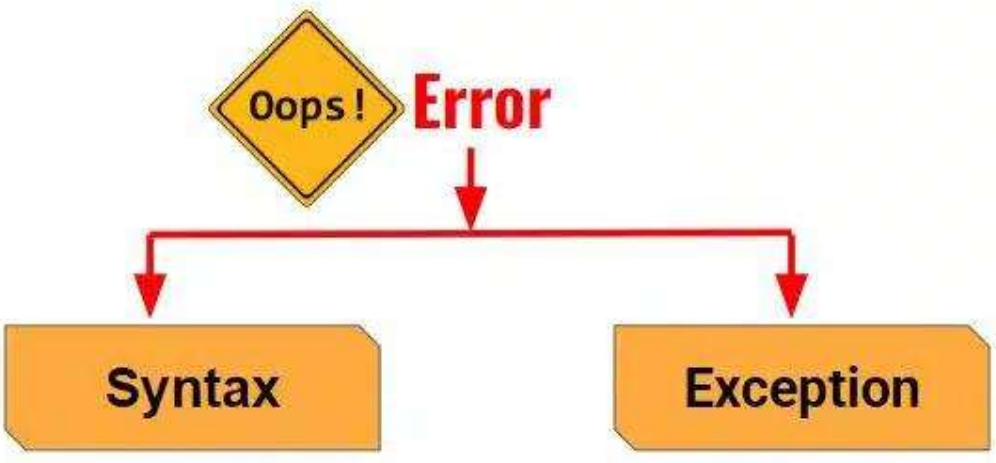
Exception Handling in Python Class 12 Notes

Exception Handling in Python Class 12 Notes

What is error?

Errors are problems that come during the execution of the program. There are two types of error, syntax error and runtime error. Runtime error is handled through exception handling; each of the errors impacts the program's behavior differently.



a. Syntax Error: Syntax errors are detected when we have not followed the rules of the particular programming language while writing a program. These errors are also known as parsing errors. If a syntax error occurs, then without rectifying the error, the program will not execute.

b. Exceptions: Even if a statement or expression is syntactically correct, there might arise an error during its execution. For example, trying to open a file that does not exist, division by zero, and so on. Such types of errors can be handled in the time of execution using exception handling.

i. Run time error without exceptions

ii. Run time error handled with exceptions

What is exception handling?

Exception handling is a method in Python that helps to handle runtime errors that come during the execution of a program. In Python, exception handling is done by using a try-except block. It involves identifying the potential errors, handling them

properly, and allowing the program to execute even if any exception is encountered.

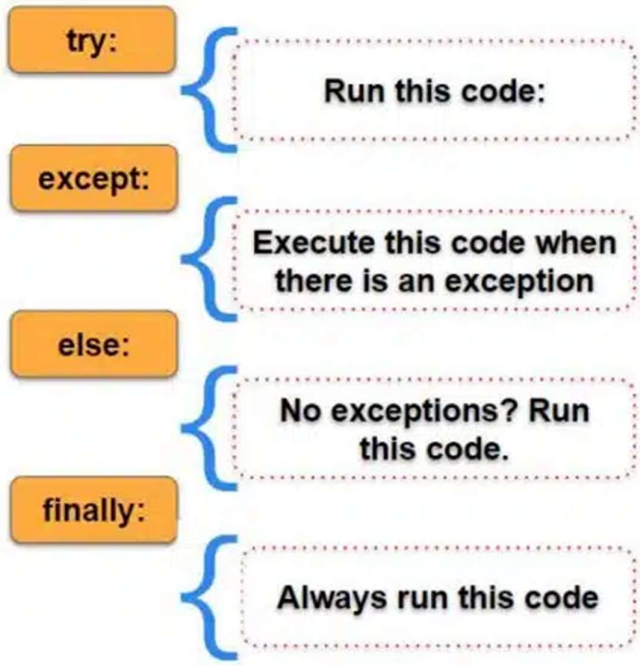
Need for Exception Handling

Exception handling is a useful technique that helps in capturing runtime errors and handling them so as to avoid the program crashing. Following is some of the important points regarding exceptions and their handling:

- Python provides a wide range of built-in exceptions like ValueError, ZeroDivisionError, etc.
- Python provides a separate block for logic and error handling.
- Python helps the developers quickly debug issues and identify problem areas.

How to use exception handling using try and except Block

When an error occurs, the Python interpreter creates an object called the exception object. This object contains information about the error, like its type, file name, and position in the program where the error has occurred. To handle this error, the following exception handling code is required:

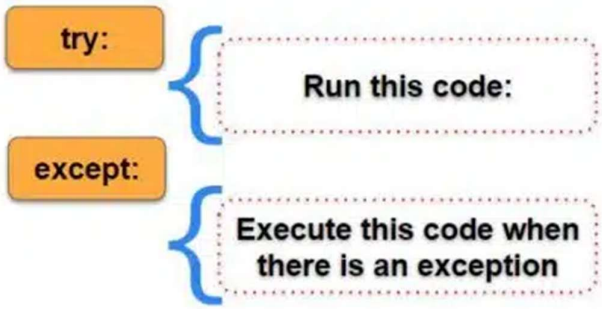


Built-in exceptions in Python

Name of the Built in Exception	Explanation
SyntaxError	It is raised when there is an error in the syntax of the Python code.
ValueError	It is raised when a built-in method or operation receives an argument that has the right data type but mismatched or inappropriate values.
IOError	It is raised when the file specified in a program statement cannot be opened.
KeyboardInterrupt	It is raised when the user accidentally hits the Delete or Esc key while executing a program due to which the normal flow of the program is interrupted.
ImportError	It is raised when the requested module definition is not found.
EOFError	It is raised when the end of file condition is reached without reading any data by input().
ZeroDivisionError	It is raised when the denominator in a division operation is zero.
IndexError	It is raised when the index or subscript in a sequence is out of range.
NameError	It is raised when a local or global variable name is not defined.
IndentationError	It is raised due to incorrect indentation in the program code.
TypeError	It is raised when an operator is supplied with a value of incorrect data type.
OverFlowError	It is raised when the result of a calculation exceeds the maximum limit for numeric data type.

Catching Exceptions

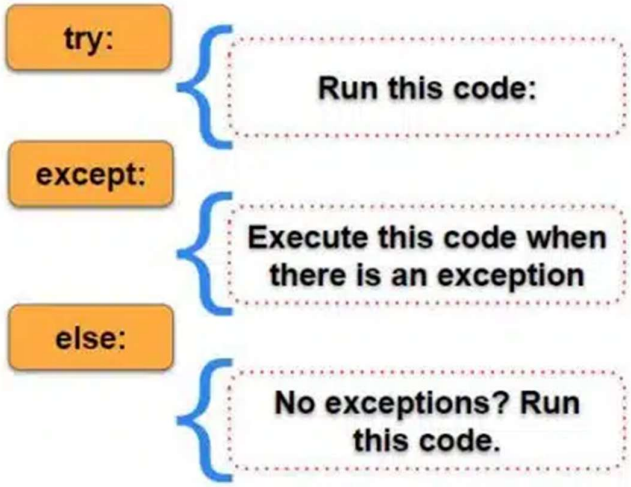
The try block is handled in the except block. If an exception occurs within the try block, the execution immediately jumps to the except block. The except block specifies how to handle the exception. If no exception occurs in the try block, the except block is skipped.



Example of simple program with multiple exception handling blocks:

try...except...else clause

We can put an optional else clause along with the try…except clause. An except block will be executed only if some exception is raised in the try block. But if there is no error then none of the except blocks will be executed. In this case, the statements inside the else clause will be executed.



Example of simple program with multiple exception handling blocks:

main.py

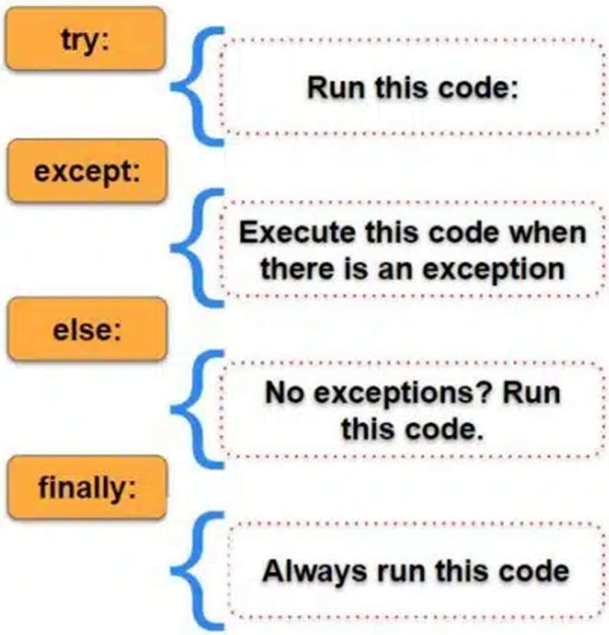
Share Run Output

```
try:
    num1 = int(input("Enter the Number1: "))
    num2 = int(input("Enter the Number2: "))

    result = num1 / num2

except ZeroDivisionError:
    print("Error: Number2 cannot be zero")
except ValueError:
    print("Error: Please enter numeric values only")
else:
    print(f"The result of {num1} divided by {num2} is {result}")

```

```
Enter the Number1: 20
Enter the Number2: 5
The result of 20 divided by 5 is 4.0

=== Code Execution Successful ===
```

Finally clause

The try statement in Python can also have an optional finally clause. The statements inside the finally block are always executed regardless of whether an exception has occurred in the try block or not. It is a common practice to use finally clause while working with files to ensure that the file object is closed.



Example of simple program with multiple exception handling blocks:

main.py

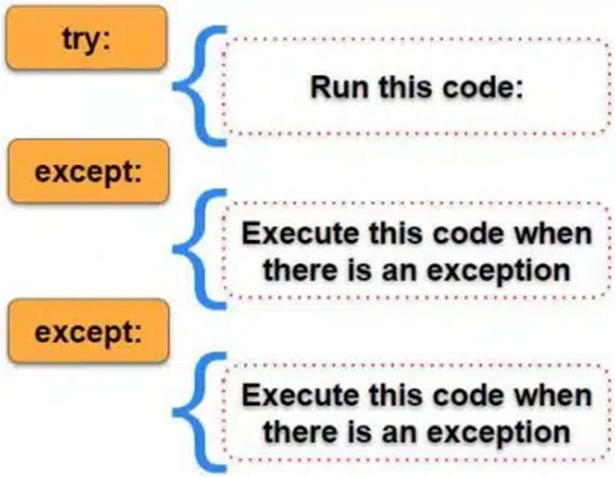
```
try:
    num1 = int(input("Enter the Number1: "))
    num2 = int(input("Enter the Number2: "))

    result = num1 / num2

except ZeroDivisionError:
    print("Error: Number2 cannot be zero")
except ValueError:
    print("Error: Please enter numeric values only")
else:
    print(f"The result of {num1} divided by {num2} is {result}")
finally:
    print("Thank you for using the program!")
```

Output

```
Enter the Number1: 20
Enter the Number2: 5
The result of 20 divided by 5 is 4.0
Thank you for using the program!

=== Code Execution Successful ===
```

How to handle Multiple Errors

To handle multiple exceptions in Python, you can use a single try and multiple except blocks in Python. This helps to handle different types of errors that may occur during the runtime. Every except block is associated with a specific exception type.



Example of simple program with multiple exception handling blocks:

main.py

```
try:
    num1 = int(input("Enter the Number1: "))
    num2 = int(input("Enter the Number2: "))

    result = num1 / num2

except ZeroDivisionError:
    print("Error: Number2 cannot be zero")
except ValueError:
    print("Error: Please enter numeric values only")
```

Output

```
Enter the Number1: 20
Enter the Number2: 0
Error: Number2 cannot be zero

=== Code Execution Successful ===
```