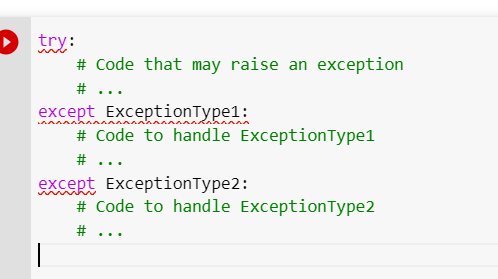
**1. What is the role of try and exception block?**

Ans.) The role of a try-except block, also known as an exception handling block, is to handle potential errors or exceptions that may occur during the execution of a program. In python, errors can occur while executing certain statements or blocks of code. These errors can be caused by various factors, such as invalid input, unexpected conditions, or external dependencies. When an error or exception occurs, it disrupts the normal flow of the program and can lead to crashes or unexpected behavior.

The try-except block allows you to anticipate and handle these errors gracefully. The structure of a try-except block typically looks like this:



**2. What is the syntax for a basic try-except block?**

Ans.) In Python, the syntax for a basic try-except block is as follows:

A screenshot of a computer code

Description automatically generated

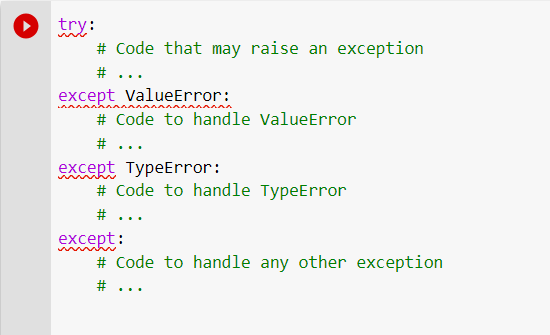
Where:

The **try** keyword marks the start of the try block, where you place the code that you want to monitor for exceptions.

If an exception occurs within the try block, the program will immediately jump to the except block that matches the type of the raised exception.

The **except** keyword is followed by the specific **ExceptionType** that you want to catch and handle. This can be a built-in exception type (e.g., **ValueError, TypeError**) or a custom exception type.

It's important to note that you can have multiple except blocks to handle different types of exceptions. Here's an example that demonstrates this:



**3. What happens if an exception occurs inside a try block and there is no matching except block?**

Ans.)   
If an exception occurs inside a try block and there is no matching except block to handle that specific type of exception, the program will terminate abruptly, and an error message will be displayed. This error message will include information about the unhandled exception, such as the exception type and a traceback, which shows the sequence of function calls that led to the exception.

Ex.)

A white background with black text

Description automatically generated

In this example, if an exception occurs within the try block and it is a ValueError, the program will execute the code inside the corresponding except block. However, if a different type of exception, such as a TypeError, occurs, and there is no except block to handle it, the program will terminate and display an error message.

To handle exceptions gracefully and prevent program crashes, it is good practice to have a catch-all except block at the end, without specifying any exception type. This allows you to handle any unexpected exceptions that might occur and perform appropriate actions, such as logging the error or displaying a friendly error message to the user.

Ex.)

A screenshot of a computer error message

Description automatically generated

**4. What is the difference between using a bare except block and specifying a specific exception type?**

|  |  |
| --- | --- |
| **Bare Except Block** | **Specifying a Specific Exception Type** |
| A bare except block, written as except:, catches any type of exception that occurs within the try block, regardless of its specific type. | When you specify a specific exception type in an except block, such as **except ValueError**:, you are explicitly catching and handling only that particular type of exception. |
| It may lead to potentially catching and handling exceptions that you didn't intend to handle, which can make debugging more challenging. | This allows you to handle different exceptions in different ways, tailoring your error handling logic based on the specific type of exception. |
| This can be useful for handling unexpected or unknown exceptions that you may not have anticipated or explicitly handled. | It helps in debugging and understanding the specific type of error that occurred, as you have explicit information about the exception being caught. |

**5. Can you have nested try-except blocks in Python? If yes, then give an example.**

Ans.) Yes, you can have nested try-except blocks in Python. Nesting try-except blocks allows you to handle exceptions at different levels of code execution, providing more fine-grained control over exception handling. Inner try-except blocks can handle specific exceptions locally, while outer try-except blocks can handle exceptions that are not handled within the inner blocks.

Format of nested try-except blocks: A screenshot of a computer error

Description automatically generated

Example of nested try-except:

A screenshot of a computer program

Description automatically generated

**6. Can we use multiple exception blocks, if yes then give an example.**

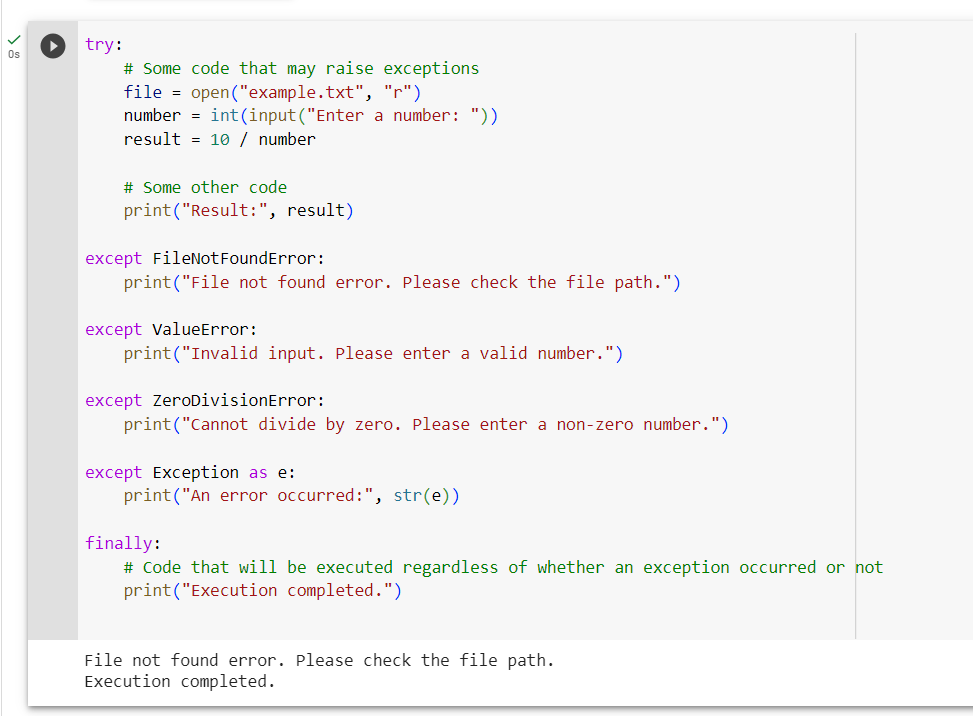
Ans.) Yes, we can use multiple exception blocks to handle different types of exceptions. This allows us to specify different exception handling logic for each type of exception.

Format of Multiple exception blocks:

A screenshot of a computer code

Description automatically generated

Example of Multiple exception blocks:

****

**7. Write the reason due to which following errors are raised:**

Ans.)

**a. EOFError:** The **EOFError** in Python stands for "End of File Error". It is raised when the built-in **input()** function hits the end of a file (EOF) while trying to read input from the user.

**b. FloatingPointError:** In Python, a FloatingPointError is raised when an error occurs during floating-point arithmetic operations.

**c. IndexError:** The **IndexError** is raised in Python when you try to access an index that is outside the range of valid indices for a sequence (such as a list, tuple, or string). It occurs when you attempt to access an element using an index that is either negative or greater than or equal to the length of the sequence.

**d.** **MemoryError**: The MemoryError in Python is raised when an operation cannot be completed due to insufficient memory allocation. This error occurs when the program attempts to allocate more memory than is available in the system.

**e. OverflowError:** The **OverflowError** in Python is raised when the result of an arithmetic operation exceeds the maximum representable value for a numeric type. This error occurs when performing calculations that result in a value that is too large to be stored in the chosen data type.

**f. TabError:** A TabError in Python is raised when there is an issue with the indentation of your code. It typically occurs when you mix spaces and tabs inconsistently within the same block of code. Python relies on consistent indentation to define blocks of code (such as loops, conditionals, and function definitions), and using a mixture of spaces and tabs can lead to indentation errors.

**g. ValueError:** The **ValueError** is a common error raised in Python when a function receives an argument of the correct type, but the value of the argument is inappropriate or out of range for the operation being performed.

**8. Write code for the following given scenario and add try-exception block to it.**

a. Program to divide two numbers:

A screenshot of a computer program

Description automatically generated

b. Program to convert a string to an integer

**A screenshot of a computer program

Description automatically generated**

c. Program to access an element in a list:

A screenshot of a computer code

Description automatically generated

d. Program to handle a specific exception:

program to handle **ZeroDivisionError:**

**A screenshot of a computer code

Description automatically generated**

e. Program to handle any exception:

