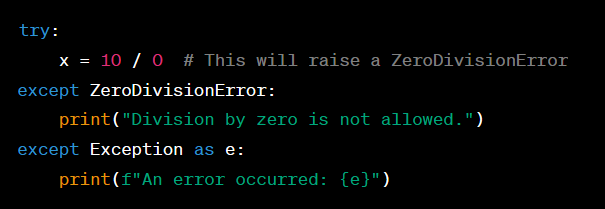
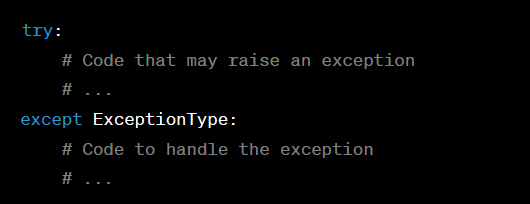
1. **What is the role of try and exception block?**

Ans: The `try` and `except` blocks in Python are used for handling exceptions, which are runtime errors that can occur during the execution of a program. The primary role of the `try` and `except` blocks is to handle exceptions gracefully, preventing the program from crashing and allowing you to respond to unexpected errors in a controlled manner.



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

Ans: The basic syntax for a `try` and `except` block in Python is as follows:



1. **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, and Python will display an unhandled exception error message. This is known as an "unhandled exception," and it causes the program to crash.
* In other words, without a corresponding `except` block for the exception that occurred, there's no error-handling logic to address the exception, and Python will raise an unhandled exception error. The error message typically includes information about the type of exception, a traceback showing where the exception occurred, and, in some cases, a description of the error.
* To handle exceptions gracefully and prevent the program from crashing, it's essential to include an appropriate `except` block for the specific type of exception that may be raised within the `try` block. This allows you to provide error-handling logic or recovery procedures. You can also include a more general `except` block to catch unexpected or unspecified exceptions, but it's generally better to handle exceptions specifically whenever possible to provide more informative error messages and tailored error handling.

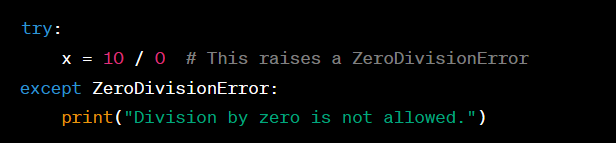
1. What is the difference between using a bare except block and specifying a specific exception type?

Ans: The main difference between using a bare `except` block and specifying a specific exception type in a `try` and `except` block lies in the level of control and precision in handling exceptions:

1. Specific Exception Type (`except ExceptionType`):

* When you specify a specific exception type in an `except` block, you are providing precise error handling for that particular type of exception. It allows you to define custom error-handling logic for that specific exception.
* You can have multiple `except` blocks, each targeting a different type of exception, providing tailored responses to different errors.
* Using specific exception types is considered good practice because it allows you to handle exceptions more effectively, provide meaningful error messages, and take appropriate actions.

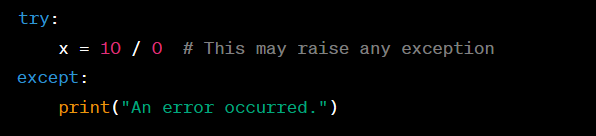
Example:



2. Bare `except` Block (`except` without specifying an exception type):

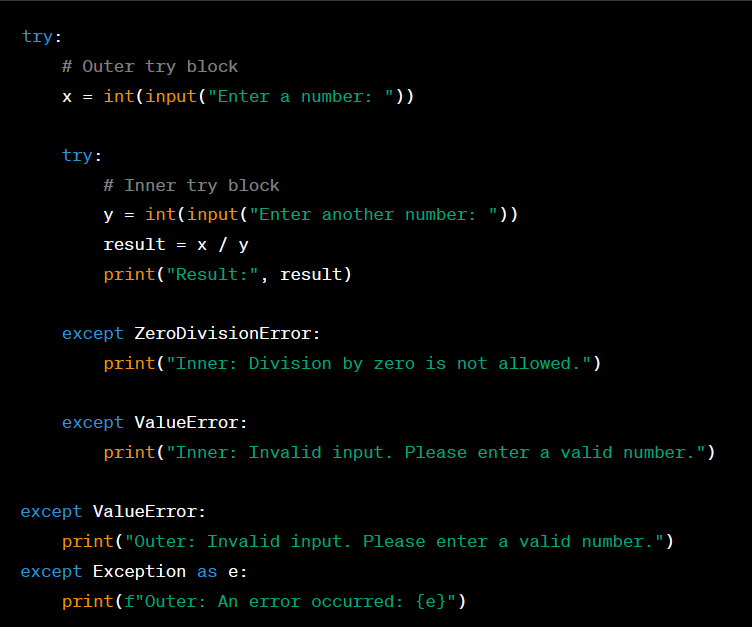
* + A bare `except` block is a more general catch-all mechanism. It catches any exception that may occur within the `try` block.
  + While it can be useful for catching unexpected or unspecified exceptions, it provides less specific information about the error, making it more challenging to determine the cause of the issue.
  + Using a bare `except` block should be avoided in most cases because it can lead to poor error diagnosis and can hide programming mistakes.

Example:



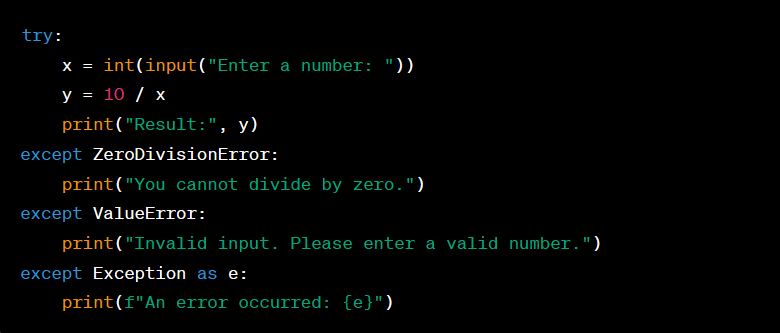
5. **Can you have nested try-except blocks in Python? If yes, then give an example.**

Ans; Yes, you can have nested `try` and `except` blocks in Python. This means that you can place a `try` block inside another `try` block, and each `try` block can have its corresponding `except` blocks. This allows you to handle exceptions at different levels of granularity. Here's an example of nested `try` and `except` blocks:



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

Ans:. Yes, you can use multiple `except` blocks to handle different types of exceptions in a `try` and `except` construct. This allows you to provide specific error-handling logic for various types of exceptions that may occur. Here's an example of using multiple `except` blocks:



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

**b. FloatingPointError c. IndexError**

**d. MemoryError e. OverflowError f. TabError**

**g. ValueError**

Ans: *Here are the reasons why the mentioned errors are raised:*

*a. EOFError:*

* + *An `EOFError` (End of File Error) occurs when an input operation reaches the end of a file or stream unexpectedly. This typically happens when trying to read more data than is available in a file or input stream.*

*b. FloatingPointError:*

* + *A `FloatingPointError` occurs when there is an issue with floating-point arithmetic, such as division by zero or other mathematical errors related to floating-point numbers.*

*c. IndexError:*

* + *An `IndexError` is raised when you try to access an index that is out of range in a sequence, such as a list, tuple, or string. This error indicates that the specified index doesn't exist in the sequence.*

*d. MemoryError:*

* + *A `MemoryError` is raised when an operation cannot be completed due to insufficient memory (RAM). This error occurs when your program consumes more memory than is available.*

*e. OverflowError:*

* + *An `OverflowError` occurs when a mathematical operation produces a result that is too large to be represented in the available memory or data type. For example, trying to calculate a very large factorial.*

*f. TabError:*

* + *A `TabError` is raised when there are inconsistencies in the use of tabs and spaces for indentation in your Python code. Python expects consistent indentation, and mixing tabs and spaces can lead to this error.*

*g. ValueError:*

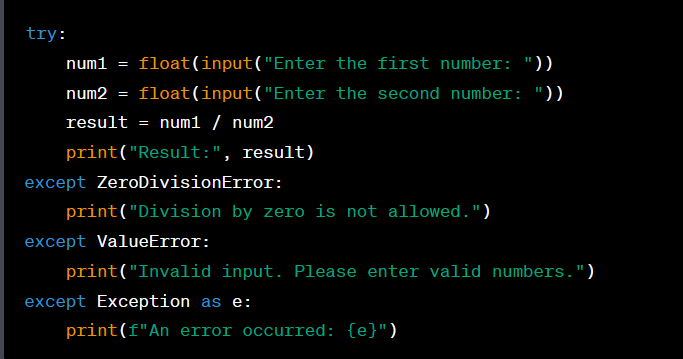
* *A `ValueError` is raised when an operation or function receives an argument of the correct data type but with an inappropriate or invalid value. For example, trying to convert a string to an integer when the string does not represent a valid integer.*

8**. Write code for the following given scenario and add try-exception block to it. a. Program to divide two numbers**

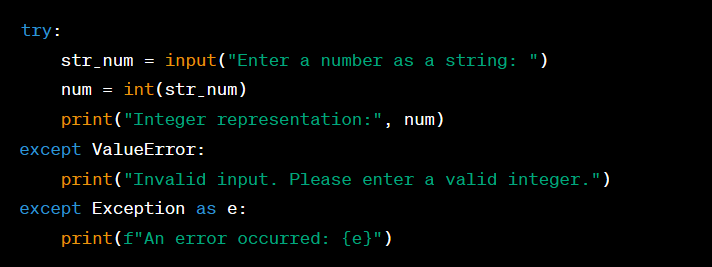
**b. Program to convert a string to an integer c. Program to access an element in a list d. Program to handle a specific exception e. Program to handle any exception**

Ans: *Here are examples of code for the given scenarios with try-except blocks:*

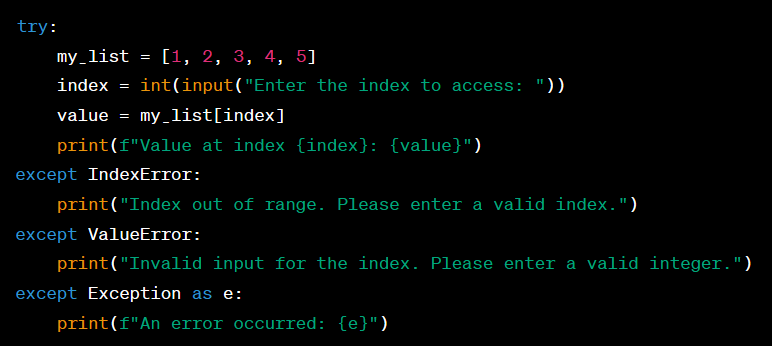
a. Program to divide two numbers:



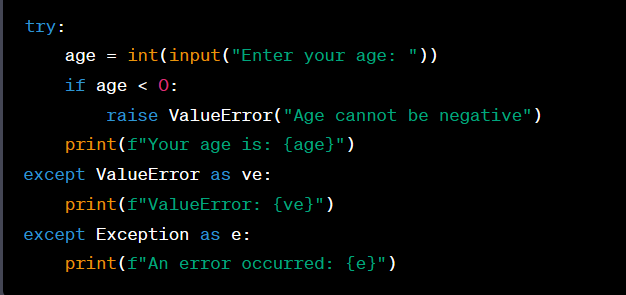
* b. Program to convert a string to an integer:



c. Program to access an element in a list:



d. Program to handle a specific exception:



e. Program to handle any exception:

