**Assignment – 10**

**17th June 2023**

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

**Ans:** The try and except blocks in Python are used for exception handling, which allows you to handle errors and exceptions that may occur during the execution of a program. The primary role of the “try” and “except” blocks is to gracefully handle exceptions and prevent the program from crashing due to unexpected errors.

* **try block:** The code that may raise an exception or error is placed inside the “try” block. Python will attempt to execute the code inside this block, and if an exception occurs during the execution, the flow of control immediately jumps to the corresponding except block.
* **except block:** If an exception is raised in the try block, Python searches for an appropriate except block that can handle the specific type of exception raised. If a matching except block is found, the code inside that block will be executed to handle the exception. This way, the program can gracefully recover from the error and continue executing the rest of the code.

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

**Ans:** The syntax for a basic try-except block in Python is as follows:

**try:**

#code that may raise exception

….

…..

**except** ExceptionType:

#Code to handle exception

…….

……..

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 with an unhandled exception error. Python will display an error message, known as a traceback, indicating the type of exception that occurred and the line where the exception was raised.

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

**Ans:**

* **Bare “except” block:**

Using a bare except block means catching all exceptions without specifying a particular exception type. It is written as just except: without mentioning any specific exception class. While a bare except block can catch any type of exception, it is generally discouraged in most scenarios because it can lead to less informative error handling and potentially hide critical issues.

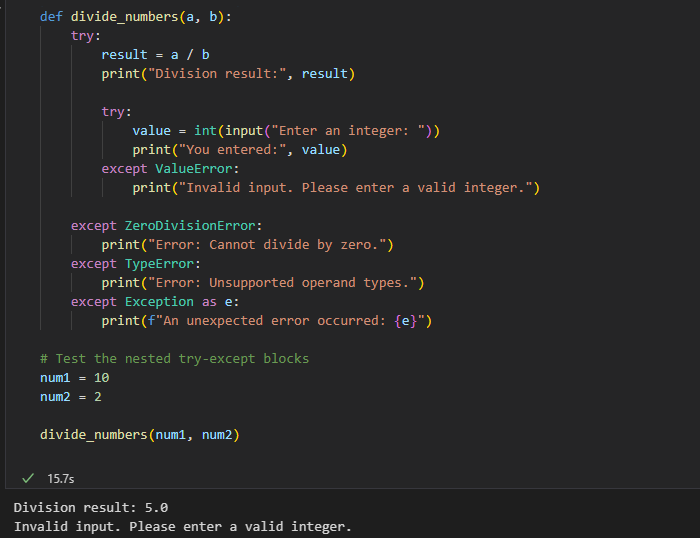
* **Specific exception type:**

Specifying a specific exception type in the except block means catching only that particular type of exception. This allows you to handle different types of errors in a more precise and meaningful way. You can choose to handle different exceptions differently, providing better feedback to users or taking appropriate corrective actions.

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

**Ans:** Yes, Python allows us to have nested try-except blocks. This means we can place one try-except block inside another try or except block. This nesting allows us to handle different exceptions at different levels of code execution and provides more fine-grained error handling.

Here's an example of nested try-except blocks:

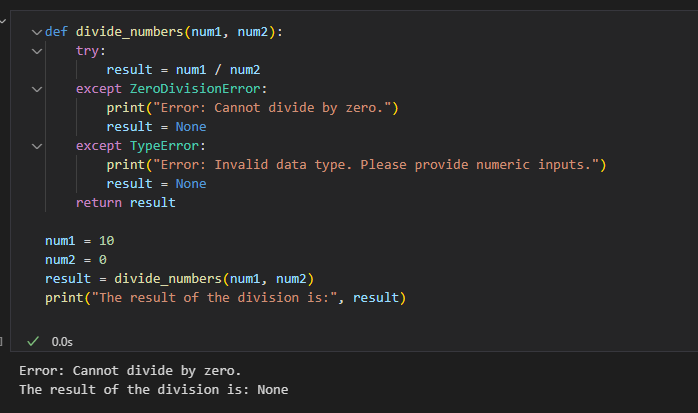


In this example, we have two functions: divide\_numbers() and process\_input(). The divide\_numbers() function takes two numbers as input and attempts to perform division. If a ZeroDivisionError occurs, it catches the exception and sets the result to None.

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

**Ans:** Yes, we can use multiple “except” blocks in Python. You can have multiple “except” blocks to handle different types of exceptions separately.

Here's an example that demonstrates the use of multiple “except” blocks

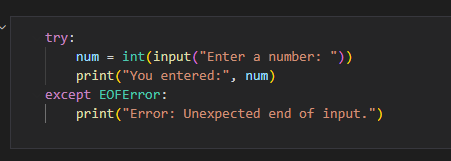


**7. Write the reason due to which following errors are raised:**

**a. EOFError**

**Ans:** The “EOFError” (End of File Error) is raised when an input function (e.g., input(), raw\_input(), readline()) tries to read from the standard input (stdin) or a file, but the end of the file is reached unexpectedly. This error typically occurs in situations where the program is expecting more input, but there is no more data to read.

Example:



**b. FloatingPointError:**

**Ans:** FloatingPointError is raised when a floating-point operation cannot be performed due to some exceptional condition or invalid operation related to floating-point numbers. These errors generally occur during arithmetic calculations involving floating-point numbers and can result from various scenarios:

* **Division by zero:** Attempting to divide a number by zero will raise a FloatingPointError. For example, 1.0 / 0.0.
* **Overflow or underflow:** Performing arithmetic operations that lead to extremely large (overflow) or extremely small (underflow) floating-point values can raise FloatingPointError. For example, 1e500 \* 1e500.
* **Invalid floating-point operation:** Some mathematical operations, such as taking the square root of a negative number, can result in an invalid operation and raise a FloatingPointError. For example, math.sqrt(-1).

**c. IndexError:**

**Ans:** The IndexError is raised when you attempt to access an index of a sequence (such as a list, tuple, or string) that is outside the valid range of indices for that sequence.

Example:

**A screenshot of a computer error

Description automatically generated**

**d. MemoryError:**

**Ans:** A MemoryError is raised in Python when the program runs out of available memory to allocate for new objects or data structures. In simpler terms, it occurs when the system does not have enough memory to meet the program's memory requirements.

The primary reasons for a MemoryError can be:

Insufficient RAM, Large Data structures, Infinite recursion, Intensive memory usage, Memory leak.

**e. OverflowError:**

**Ans: The OverflowError is raised in Python when an arithmetic operation, such as addition, subtr**action, multiplication, or exponentiation, results in a numeric value that exceeds the maximum representable value for the specific data type.

For example, if you try to perform an operation that leads to a result larger than the maximum value that can be stored in an integer variable, an OverflowError will be raised.

Example:

**f. TabError:**

**Ans:** A “TabError” is raised in Python when there is an issue with the indentation in the code. It occurs when there is a mix of tabs and spaces used for indentation, or when tabs are used inconsistently across the code.

Python relies on consistent indentation to define the block structure of the code. Mixing tabs and spaces can lead to ambiguity and errors in parsing the code, which is why Python raises a “TabError” to indicate the problem.

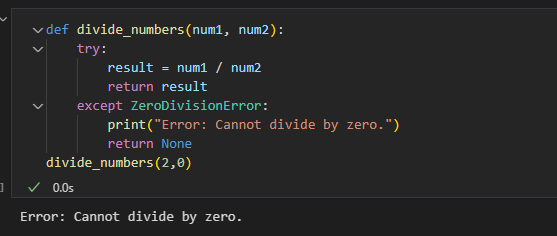
**g. ValueError:**

**Ans:** The “ValueError” is raised when an operation or function receives an argument of the correct type, but the argument's value is not suitable for the operation or function. In other words, it indicates that the data type of the argument is correct, but the value itself is not valid or acceptable for the specific operation being performed.

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

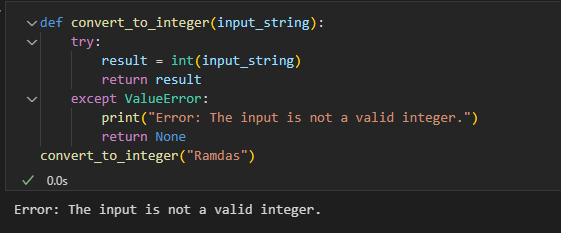
**a. Program to divide two numbers**

**Ans:**

****

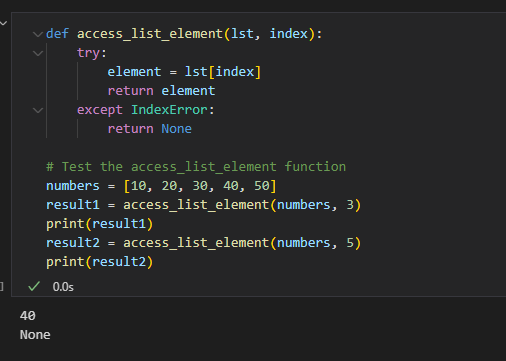
**b. Program to convert a string to an integer**

**Ans:**

****

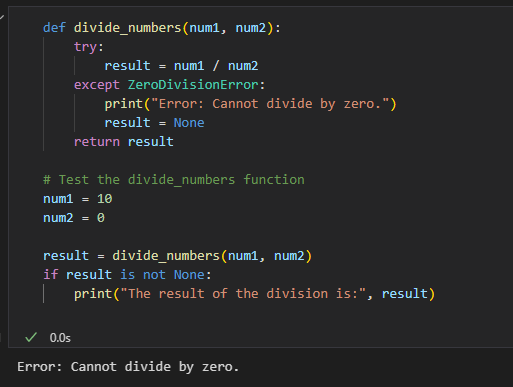
**c. Program to access an element in a list:**

**Ans:**

****

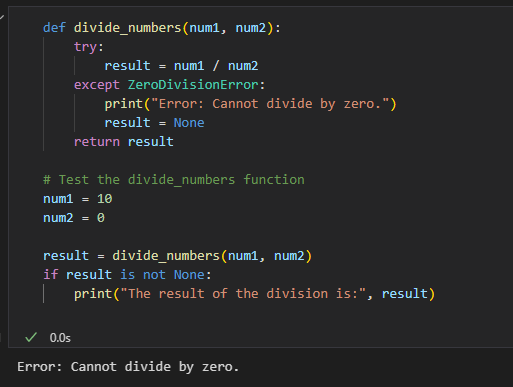
**d. Program to handle a specific exception:**

**Ans:**

****

**e. Program to handle any exception:**

**Ans:**

****