Exception Handling

What is Exception Handling?

Way to manage errors that occur during the execution of a program, allowing the program to continue running or to fail gracefully

Helps prevent program crashes and provides meaningful error messages to users or developers

Common Types of Exceptions

SyntaxError

Error in the code syntax

ArithmeticError

base class for all numeric calculations errors

- ZeroDivisionError
- OverflowError
- FloatingPoint Error

AttributeError

Raised when an attribute reference or assignment fails.

EOFError

Raised when the input() function hits an end-of-file condition (EOF) without reading any data.

ImportError

Raised when an import statement fails to find the module definition/ import

- ModuleNotFoundError

Common Types of Exceptions

IndexError

Raised when a sequence subscript is out of range.

KeyError

raised when a dictionary key is not found in the set of existing keys

KeyboardInterrupt

when the user interrupts program execution, usually by pressing Ctrl+C

MemoryError

when an operation runs out of memory but the situation may still be rescued.

NameError

when a local or global name is not found.

Common Types of Exceptions

OSError

when a system-related error occurs, including I/O failures like “file not found

TypeError

when an operation or function is applied to an object of inappropriate type

ValueError

Raised when a function receives an argument of the right type but inappropriate value

UnicodeError

Raised when a Unicode-related encoding or decoding error occurs

SystemExit

Raised by the `sys.exit()` function to exit the program

TRY AND EXCEPT

```
try:
    # Code that might cause an exception
except ExceptionType:
    # Code to execute if the exception occurs
```

Task

Write a program that takes input from the user, tries to convert it to an integer, divide 999 by the number and handles possible ValueError and ZeroDivisionError.

ELSE

```
try:
    # Code that might cause an exception
except ExceptionType:
    # Code to execute if the exception occurs
else:
    # Code to execute if no exception occurs
```

Executes code if no exceptions were raised in the try block.

FINALLY

```
try:
    # Code that might cause an exception
except ExceptionType:
    # Code to execute if the exception occurs
finally:
    # Code to execute no matter what
```

Ensures that the code in the finally block runs no matter what, even if an exception is raised.

Raising Exceptions

raise ExceptionType("Error message")

Manually trigger an exception in your code

Custom Exceptions

Define your own exceptions for specific error cases in your program.

```
class MyCustomError(Exception):
    pass
```

- pass- a placeholder that is used when a statement is syntactically required but you have nothing to write in that part of the code yet.
- It allows you to create a minimal block of code that does nothing but avoids syntax errors.

Task

A basic calculator that takes user input, performs arithmetic operations, and handles different exceptions like division by zero, invalid input, and custom exceptions.

Thank you very much!