**Experiment Number – 16**

**Title – Write a program to demonstrate exception handling using raise, try, except and finally statements.**

**Theory-**

Error in Python can be of two types i.e. [Syntax errors and Exceptions](https://www.geeksforgeeks.org/errors-and-exceptions-in-python/). Errors are problems in a program due to which the program will stop the execution. On the other hand, exceptions are raised when some internal events occurs which change the normal flow of the program.

An exception in Python is an incident that happens while executing a program that causes the regular course of the program's commands to be disrupted. When a Python code comes across a condition it can't handle, it raises an exception. An object in Python that describes an error is called an exception.

When a Python code throws an exception, it has two options: handle the exception immediately or stop and quit.Python uses try and except keywords to handle exceptions. Both keywords are followed by indented blocks.

* The try block lets you test a block of code for errors.
* The except block lets you handle the error.
* The else block lets you execute code when there is no error.
* The finally block lets you execute code, regardless of the result of the try- and except blocks.

The try: block contains one or more statements which are likely to encounter an exception. If the statements in this block are executed without an exception, the subsequent except: block is skipped.

If the exception does occur, the program flow is transferred to the except: block. The statements in the except:block are meant to handle the cause of the exception appropriately. For example, returning an appropriate error message.

You can specify the type of exception after the except keyword. The subsequent block will be executed only if the specified exception occurs. There may be multiple except clauses with different exception types in a single try block. If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

The rest of the statements after the except block will continue to be executed, regardless if the exception is encountered or not.

**Syntax:**

try:

# code that may cause exception

except:

# code to run when exception occurs

Here, we have placed the code that might generate an exception inside the try block. Every try block is followed by an except block.

When an exception occurs, it is caught by the except block. The except block cannot be used without the try block.

In Python, keywords else and finally can also be used along with the try and except clauses. While the except block is executed if the exception occurs inside the try block, the else block gets processed if the try block is found to be exception free.

**Syntax:**

try:

#statements in try block

except:

#executed when error in try block

else:

#executed if try block is error-free

finally:

#executed irrespective of exception occured or not

The finally block consists of statements which should be processed regardless of an exception occurring in the try block or not. As a consequence, the error-free try block skips the except clause and enters the finally block before going on to execute the rest of the code. If, however, there's an exception in the try block, the appropriate except block will be processed, and the statements in the finally block will be processed before proceeding to the rest of the code.

## Catching Specific Exception

A try statement can have more than one except clause, to specify handlers for different exceptions. Please note that at most one handler will be executed. For example, we can add IndexError in the above code. The general syntax for adding specific exceptions are –

try:

# statement(s)

exceptIndexError:

# statement(s)

exceptValueError:

# statement(s)

### Different types of exceptions in python:

In Python, there are several built-in exceptions that can be raised when an error occurs during the execution of a program. Here are some of the most common types of exceptions in Python:

* **SyntaxError:** This exception is raised when the interpreter encounters a syntax error in the code, such as a misspelled keyword, a missing colon, or an unbalanced parenthesis.
* **TypeError**: This exception is raised when an operation or function is applied to an object of the wrong type, such as adding a string to an integer.
* **NameError**: This exception is raised when a variable or function name is not found in the current scope.
* **IndexError**: This exception is raised when an index is out of range for a list, tuple, or other sequence types.
* **KeyError**: This exception is raised when a key is not found in a dictionary.
* **ValueError**: This exception is raised when a function or method is called with an invalid argument or input, such as trying to convert a string to an integer when the string does not represent a valid integer.
* **AttributeError**: This exception is raised when an attribute or method is not found on an object, such as trying to access a non-existent attribute of a class instance.
* **IOError**: This exception is raised when an I/O operation, such as reading or writing a file, fails due to an input/output error.
* **ZeroDivisionError**: This exception is raised when an attempt is made to divide a number by zero.
* **ImportError**: This exception is raised when an import statement fails to find or load a module.

These are just a few examples of the many types of exceptions that can occur in Python. It’s important to handle exceptions properly in your code using try-except blocks or other error-handling techniques, in order to gracefully handle errors and prevent the program from crashing.

**Raise an Exception**

If a condition does not meet our criteria but is correct according to the Python interpreter, we can intentionally raise an exception using the raise keyword. We can use a customized exception in conjunction with the statement.

**Example**

try:

    age=int(input("Please enter your age"))

    ifage<5:

        raiseValueError("Not allowed! Your age is less than 5")

    ifage>20:

        raiseValueError("Not allowed! Your age is greater than 20")

exceptValueErrorasv:

    print("ValueError Exception thrown")

    print(v)

### Advantages of Exception Handling:

* **Improved program reliability**: By handling exceptions properly, you can prevent your program from crashing or producing incorrect results due to unexpected errors or input.
* **Simplified error handling**: Exception handling allows you to separate error handling code from the main program logic, making it easier to read and maintain your code.
* **Cleaner code:** With exception handling, you can avoid using complex conditional statements to check for errors, leading to cleaner and more readable code.
* **Easier debugging**: When an exception is raised, the Python interpreter prints a traceback that shows the exact location where the exception occurred, making it easier to debug your code.

### Disadvantages of Exception Handling:

* **Performance overhead:** Exception handling can be slower than using conditional statements to check for errors, as the interpreter has to perform additional work to catch and handle the exception.
* **Increased code complexity**: Exception handling can make your code more complex, especially if you have to handle multiple types of exceptions or implement complex error handling logic.
* **Possible security risks:** Improperly handled exceptions can potentially reveal sensitive information or create security vulnerabilities in your code, so it’s important to handle exceptions carefully and avoid exposing too much information about your program.

## Assertions in Python

An assertion is a consistency test that we can switch on or off.The simplest way to understand an assertion is to compare it with an if-then condition. An exception is thrown if the outcome is false when an expression is evaluated.Assertions are made via the assert statement.Assertions are commonly used at the beginning of a function to inspect for valid input and at the end of calling the function to inspect for valid output.

### The assert Statement

Python examines the adjacent expression, preferably true when it finds an assert statement. Python throws an AssertionError exception if the result of the expression is false.

**The syntax for the assert clause is −**

assert Expressions[, Argument]

Python uses ArgumentException, if the assertion fails, as the argument for the AssertionError. We can use the try-except clause to catch and handle AssertionError exceptions, but if they aren't, the program will stop, and the Python interpreter will generate a traceback.

**Exmaple**

try:

n = int(input("Enter a number: "))

d = int(input("Enter a number: "))

assert d!=0,"Denominator can not be zero"

except:

print("Not an even number!")

else:

r=n//d

print(r)