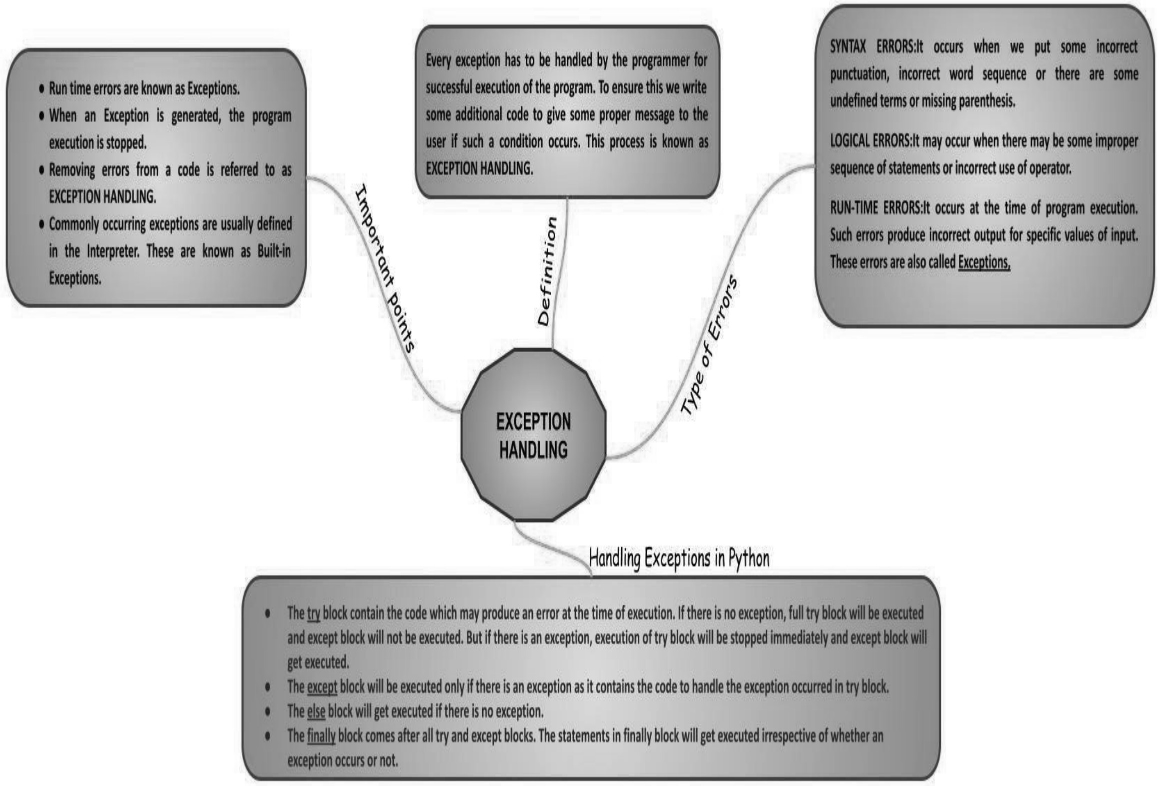
Exception Handling

<u>*Topics to be covered*</u>

- *Introduction*
- *Types of errors*
- *Handling Exceptions Using Try–Except–Finally Blocks*



- Run time errors are known as Exceptions.
- When an Exception is generated, the program execution is stopped.
- Removing errors from a code is referred to as EXCEPTION HANDLING.
- Commonly occurring exceptions are usually defined in the Interpreter. These are known as Built-in Exceptions.

Every exception has to be handled by the programmer for successful execution of the program. To ensure this we write some additional code to give some proper message to the user if such a condition occurs. This process is known as EXCEPTION HANDLING.

SYNTAX ERRORS: It occurs when we put some incorrect punctuation, incorrect word sequence or there are some undefined terms or missing parenthesis.

LOGICAL ERRORS: It may occur when there may be some improper sequence of statements or incorrect use of operator.

RUN-TIME ERRORS: It occurs at the time of program execution. Such errors produce incorrect output for specific values of input. These errors are also called Exceptions,

- The try block contain the code which may produce an error at the time of execution. If there is no exception, full try block will be executed and except block will not be executed. But if there is an exception, execution of try block will be stopped immediately and except block will get executed.
- The except block will be executed only if there is an exception as it contains the code to handle the exception occurred in try block.
- The else block will get executed if there is no exception.
- The finally block comes after all try and except blocks. The statements in finally block will get executed irrespective of whether an exception occurs or not.

EXCEPTION HANDLING

While executing a Python program, it may happen that the program does not execute at all or it can generate unexpected output. This happens when there are syntax errors, run time errors, logical errors or any semantic errors in the code.

SYNTAX ERROR	LOGICAL ERROR	RUN TIME ERROR

It occurs when we put some incorrect punctuation, incorrect word sequence or there are some undefined terms or missing parenthesis. Syntax errors are also known as **Parsing errors.**
For example:
>>> p=2(num1+num2)
 	
This statement is mathematically correct but the Python interpreter will raise a SYNTAX error as there is no sign present between 2 and parenthesis. The correct statement will be:
>>> p=2*(num1+num2)
	It may occur when there may be some improper sequence of statements or incorrect use of operators. It will not stop a program from executing but will produce incorrect output. It will generate incorrect output for every value of input.
	For example:
	If we want to find the sum of two numbers, write the following code:
	A, B = 10, 15
	C = A * B
	print (“Sum is: “, C)
	Here, the code will generate A * B but we wanted to find Sum. Hence it is a logical error.
		It occurs at the time of program execution. Such errors produce incorrect output for specific values of input. These errors are also called **Exceptions,** which occur when something unexpected happens leading to stopping the program execution.
		For example:
		1. Division by zero.
		2. Finding the square root of a negative number.
		3. Insufficient memory is available on the computer.
		4. Trying to open a file that does not exist.

EXCEPTIONS:

- Run time errors are known as Exceptions.
- When an Exception is generated, the program execution is stopped.
- Removing errors from a code is referred to as EXCEPTION HANDLING.
- Commonly occurring exceptions are usually defined in the Interpreter. These are known as Built-in Exceptions.

Some Built-in Exceptions are listed below:

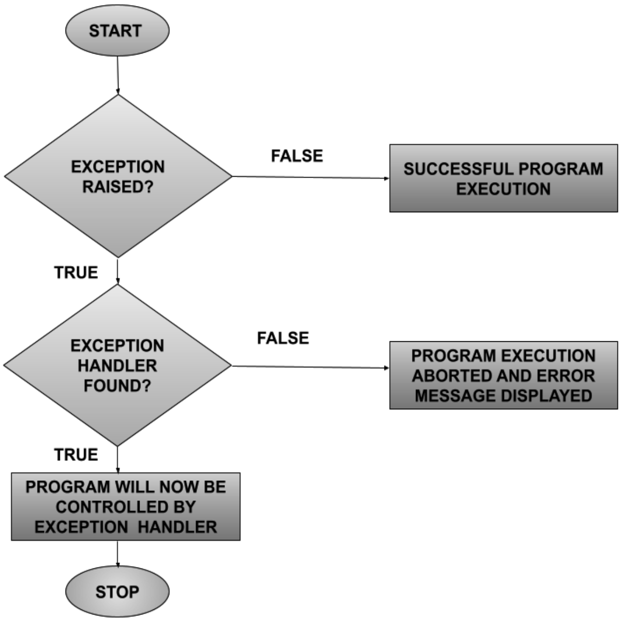
EXCEPTION	**DESCRIPTION**
ZeroDivisionError	It is raised when an expression or a value is getting divided by zero (0). For example: If c = 0, then p = b/c will result in 'ZeroDivisionError'.
NameError	It is raised when an identifier is not assigned any value earlier and is being used in some expression. For example: if p = a*b/c then it will result in 'NameError' when one or more variables are not assigned values.
TypeError	It is raised when variables used in any expression have values of different data types. For example: if p=(a+b)/c then it will result in 'TypeError' when the variables a, b and c are of different data types.
ValueError	It is raised when the given value of a variable is of the right data type but not appropriate according to the expression.
IOError	It is raised when the file specified in a program statement cannot be opened.
IndexError	It is raised when the index of a sequence is out of the range.
KeyError	It is raised when a key doesn't exist or is not found in a dictionary.

EXCEPTION HANDLING:

Every exception has to be handled by the programmer for the successful execution of the program. To ensure this we write some additional code to give some proper message to the user if such a condition occurs. This process is known as **EXCEPTION HANDLING.**

Exception handlers separate the main logic of the program from the error detection and correction code. The segment of code where there is any possibility of error or exception is placed inside one block. The code to be executed in case the exception has occurred is placed inside another block. These statements for detection and reporting the execution do not affect the main logic of the program.

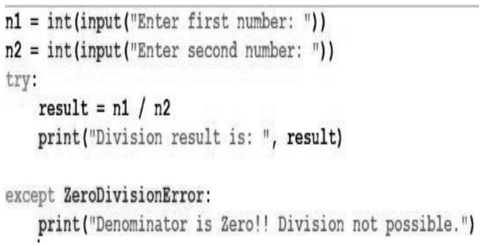
STEPS FOR EXCEPTION HANDLING:


START
EXCEPTION RAISED?
FALSE
SUCCESSFUL PROGRAM EXECUTION
TRUE
EXCEPTION HANDLER FOUND?
FALSE
PROGRAM EXECUTION ABORTED AND ERROR MESSAGE DISPLAYED
TRUE
PROGRAM WILL NOW BE CONTROLLED BY EXCEPTION HANDLER
STOP

An exception is said to be caught when a code designed for handling that particular exception is executed. In Python, exceptions, if any, are handled by using the try-except-finally block. While writing code, a programmer might doubt a particular part of code to raise an exception. Such suspicious lines of code are written inside a **try** block which will be followed by an **except** block. The code to handle every possible exception, that may arise in the try block, will be written inside the **except** block.

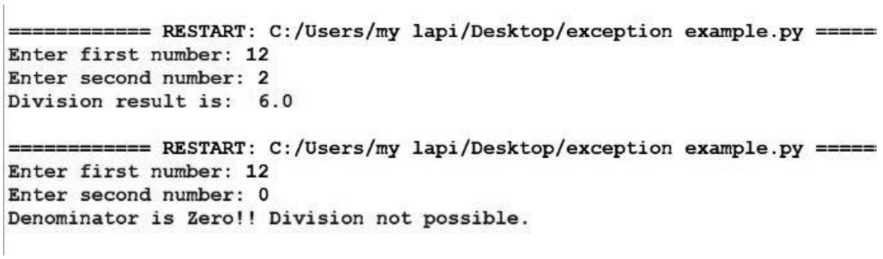
If no exception occurs during the execution of the program, the program produces the desired output successfully. But if an exception is encountered, further execution of the code inside the try block will be stopped and the control flow will be transferred to the except block.

Example 1:

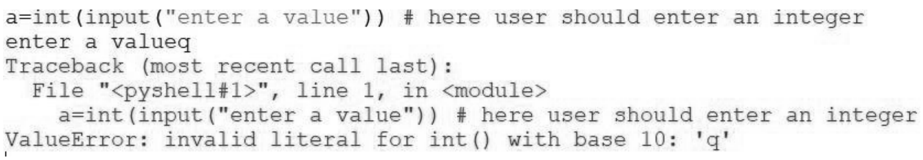


```
n1 = int(input("Enter first number: "))
n2 = int(input("Enter second number: "))
try:
    result = n1 / n2
    print("Division result is: ", result)

except ZeroDivisionError:
    print("Denominator is Zero!! Division not possible.")
```

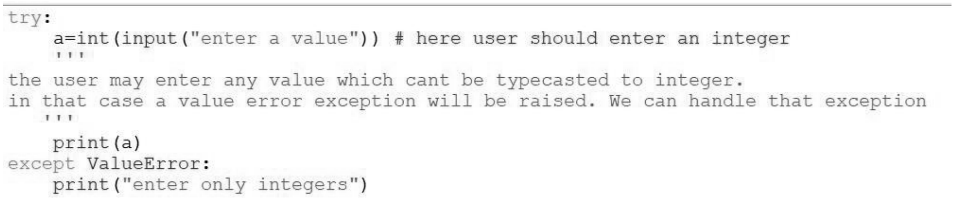
The output will be:


```
============ RESTART: C:/Users/my lapi/Desktop/exception example.py =====
Enter first number: 12
Enter second number: 2
Division result is:  6.0

============ RESTART: C:/Users/my lapi/Desktop/exception example.py =====
Enter first number: 12
Enter second number: 0
Denominator is Zero!! Division not possible.
```

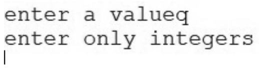
Example 2:


```
a=int(input("enter a value")) # here user should enter an integer
enter a valueq
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    a=int(input("enter a value")) # here user should enter an integer
ValueError: invalid literal for int() with base 10: 'q'
```

In the above example, the user entered a wrong value that raised ValueError. We can handle this exception by using the ValueError exception.


```
try:
    a=int(input("enter a value")) # here user should enter an integer
    '''
the user may enter any value which cant be typecasted to integer.
in that case a value error exception will be raised. We can handle that exception
   '''
    print(a)
except ValueError:
    print("enter only integers")
```

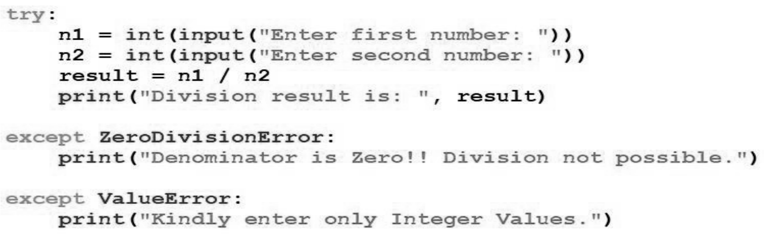
Result:


```
enter a valueq
enter only integers
```

Use of multiple “except” blocks:

Sometimes, a single piece of code in a program may have more than one type of error. If such an event happens, we can use multiple **except** blocks for a single **try** block.

Example 1:


```
try:
    n1 = int(input("Enter first number: "))
    n2 = int(input("Enter second number: "))
    result = n1 / n2
    print("Division result is: ", result)

except ZeroDivisionError:
    print("Denominator is Zero!! Division not possible.")

except ValueError:
    print("Kindly enter only Integer Values.")
```

The output will be:

```
Enter first number: 25
Enter second number: a
Kindly enter only Integer Values.
```

Example 2:

```
try:
    file = open("nonexistent_file.txt", "r")
    data = file.read()
    file.close()
    num = int(data)
    result = 10 / num
    print("Result:", result)
except FileNotFoundError:
    print("Error: File not found.")
except ZeroDivisionError:
    print("Error: Cannot divide by zero.")
except ValueError:
    print("Error: Invalid data. Please make sure the file contains a valid number.")
```

We can also handle exceptions without naming them.

```
#handling exceptions without naming them.
try:
    n1 = int(input("Enter first number: "))
    n2 = int(input("Enter second number: "))
    result = n1 / n2
    print("Division result is: ", result)

except ValueError:
    print("Kindly enter only Integer Values.")

except:
    print("oops!! There are some Exceptions.")
```

The output will be:

```
Enter first number: 56
Enter second number: 0
oops!! There are some Exceptions.
```

Default exception messages can also be displayed when we are not handling exceptions by name.

```
try:
    n1 = int(input("Enter first number: "))
    n2 = int(input("Enter second number: "))
    result = n1 / n2
    print("Division operation performed")
except ValueError:
    print("Kindly enter only Integer Values.")
except Exception as e:
    print("There is an Exception, ", str(e))
else:
    print("Division result is: ", result)
finally:
    print("Have a good day!!")
```

The output will be:

```
Enter first number: 12
Enter second number: 0
There is an Exception,  division by zero
Have a good day!!
```

<u>try…except…else Clause:</u> Just like Conditional and Iterative statements we can use an optional **else** clause along with the **try…except** clause. An except block will be executed only when some exceptions are raised in the try block. But if there is no error then except blocks will not be executed. **In this case, the <u>else</u> clause will be executed.**

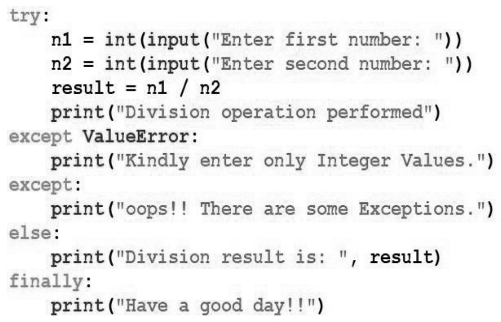
```
try:
    n1 = int(input("Enter first number: "))
    n2 = int(input("Enter second number: "))
    result = n1 / n2
    print("Division operation performed")
except ValueError:
    print("Kindly enter only Integer Values.")
except:
    print("oops!! There are some Exceptions.")
else:
    print("Division result is: ", result)
```

The output will be:

```
Enter first number: 12
Enter second number: 2
Division operation performed
Division result is:  6.0
```

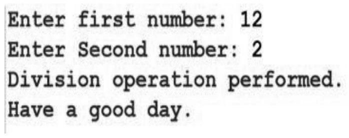
<u>finally CLAUSE:</u>

The **try…except…else** block in Python has an optional **finally** clause. The statements inside the **finally** block are always executed whether an exception has occurred in the try block or not. If we want to use the **finally** block, it should always be placed at the end of the clause i.e. after all except blocks and the else block.



```
try:
    n1 = int(input("Enter first number: "))
    n2 = int(input("Enter second number: "))
    result = n1 / n2
    print("Division operation performed")
except ValueError:
    print("Kindly enter only Integer Values.")
except:
    print("oops!! There are some Exceptions.")
else:
    print("Division result is: ", result)
finally:
    print("Have a good day!!")
```

The output will be:



```
Enter first number: 12
Enter Second number: 2
Division operation performed.
Have a good day.
```