As Per CBSE Syllabus 2024-25

Computer Science

Class XII

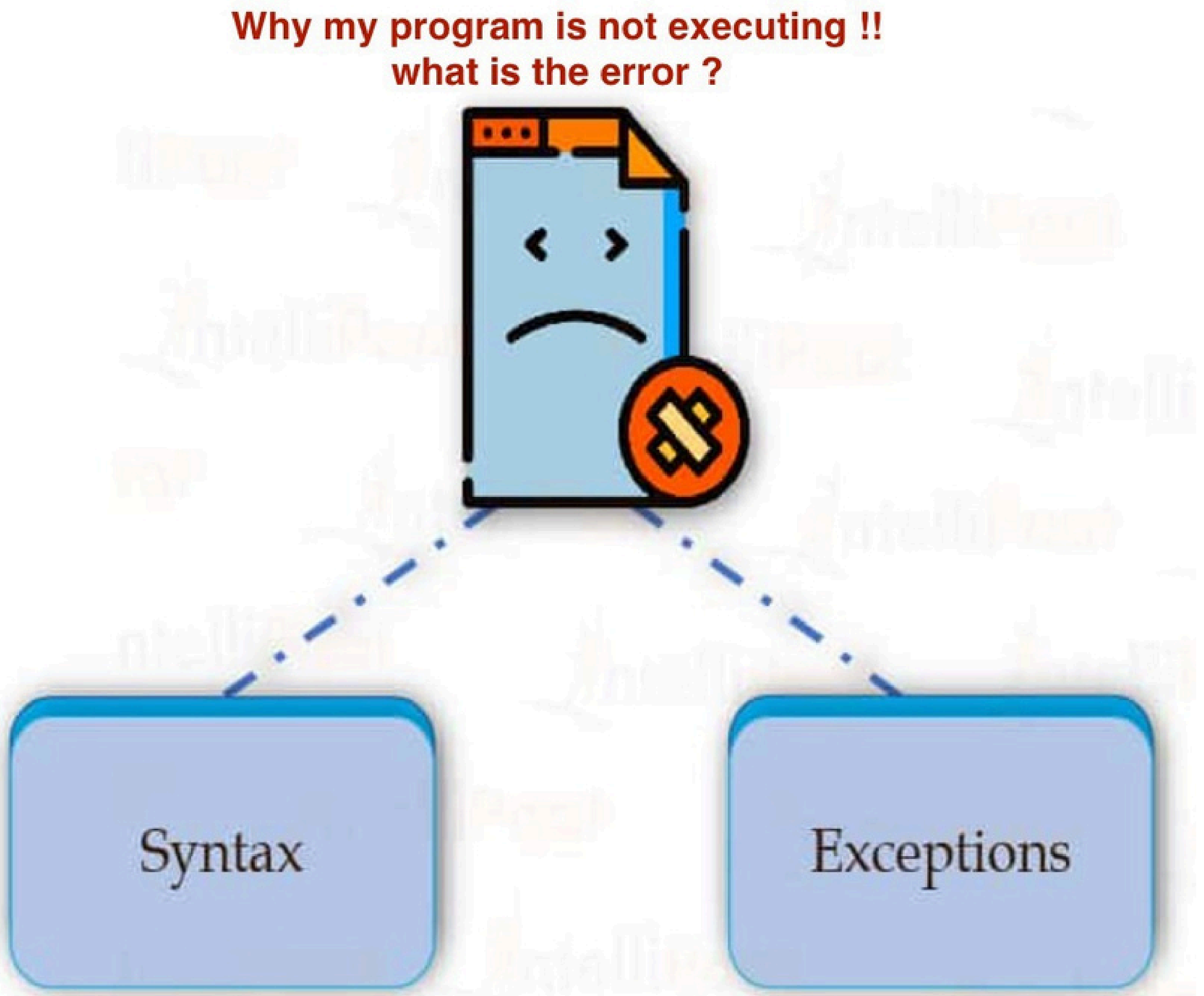
Exception Handling

Types of Errors:

Types of errors :

(i) *Compile-timeerrors.*Thesearetheerrors resulting out of violation of programming language's grammar rules. All syntax errors are reported during compilation.

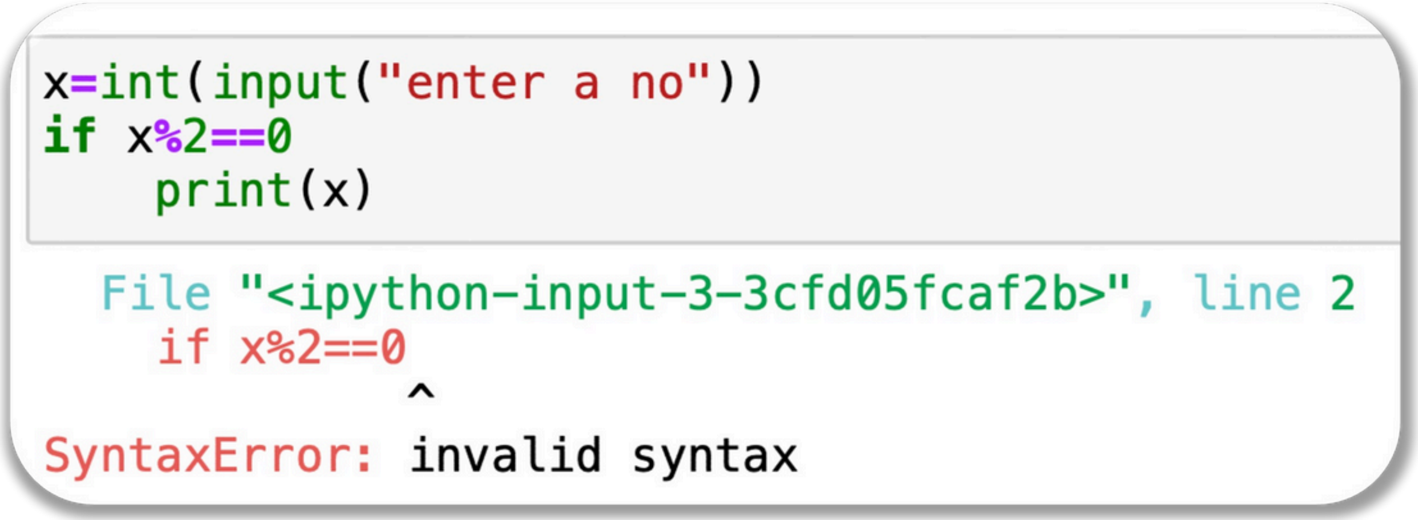
(ii) Run-time errors. The errors that occur during runtime because of unexpected situations. Such errors are handled through exception handling routines of Python.



Syntax Error Examples:

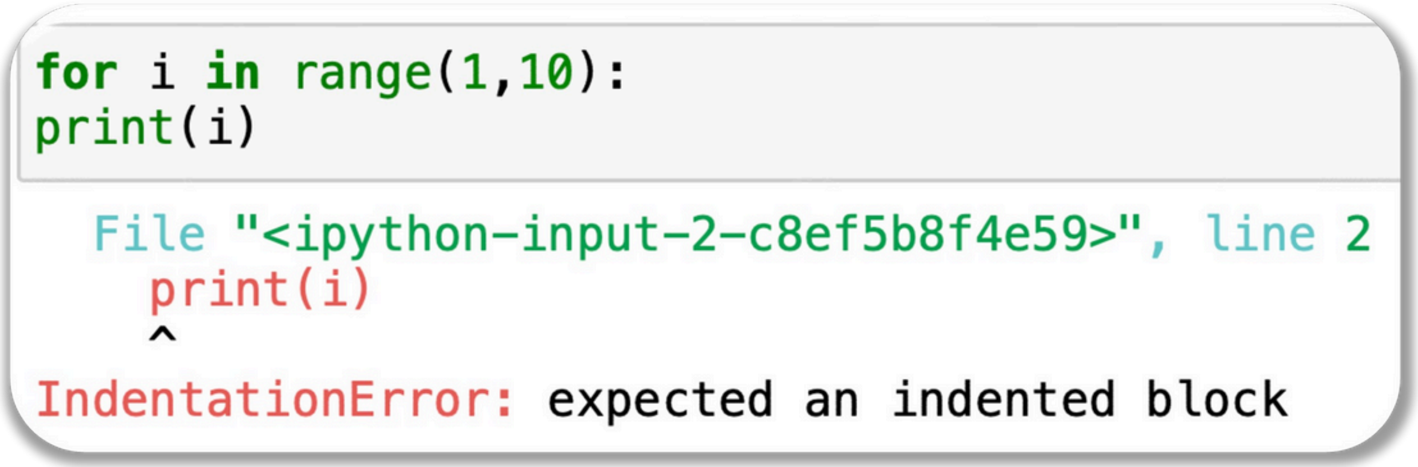
These are the errors resulting out of violation of programming language's grammar rules.

Note: All syntax errors are reported during compilation.



```
x=int(input("enter a no"))
if x%2==0
    print(x)
```

```
  File "<ipython-input-3-3cfd05fcaf2b>", line 2
    if x%2==0
             ^
SyntaxError: invalid syntax
```



```
for i in range(1,10):
print(i)
```

```
  File "<ipython-input-2-c8ef5b8f4e59>", line 2
    print(i)
        ^
IndentationError: expected an indented block
```

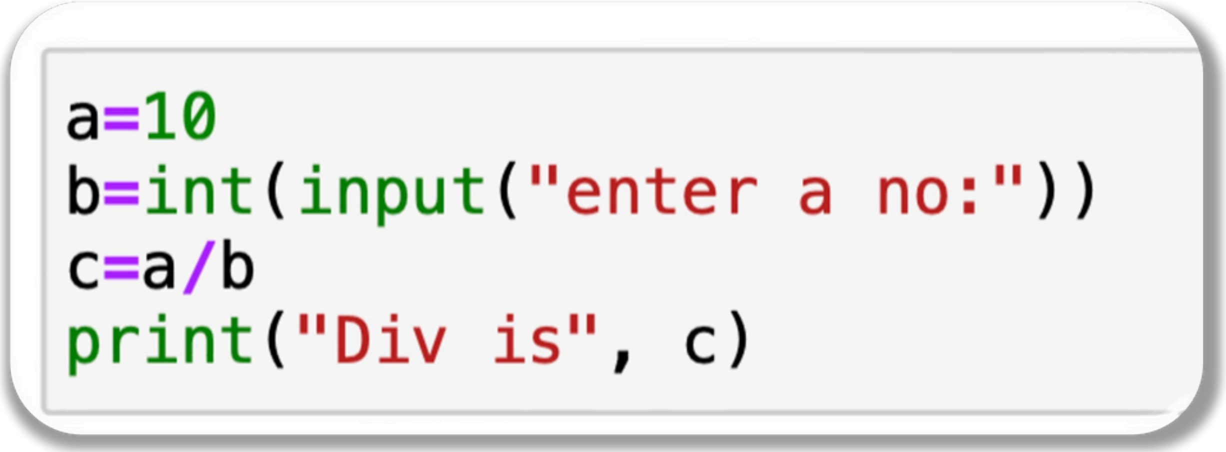
What is an Exception :

Exceptions are unexpected events or errors that occur during the execution of a program, such as a division by zero or accessing an invalid memory location. These events can lead to program termination or incorrect results.

“It is an exceptional event that occurs during runtime and causes normal program flow to be disrupted.”

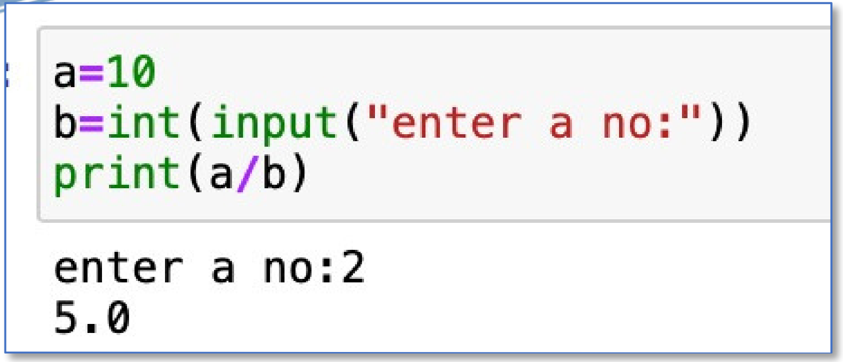
Some common examples of Exceptions are :

- Dividebyzeroerrors
- Accessingtheelementsofanarraybeyonditsrange
- InvalidinputmHarddiskcrash
- Openinganon-existentfile
- Heapmemoryexhausted

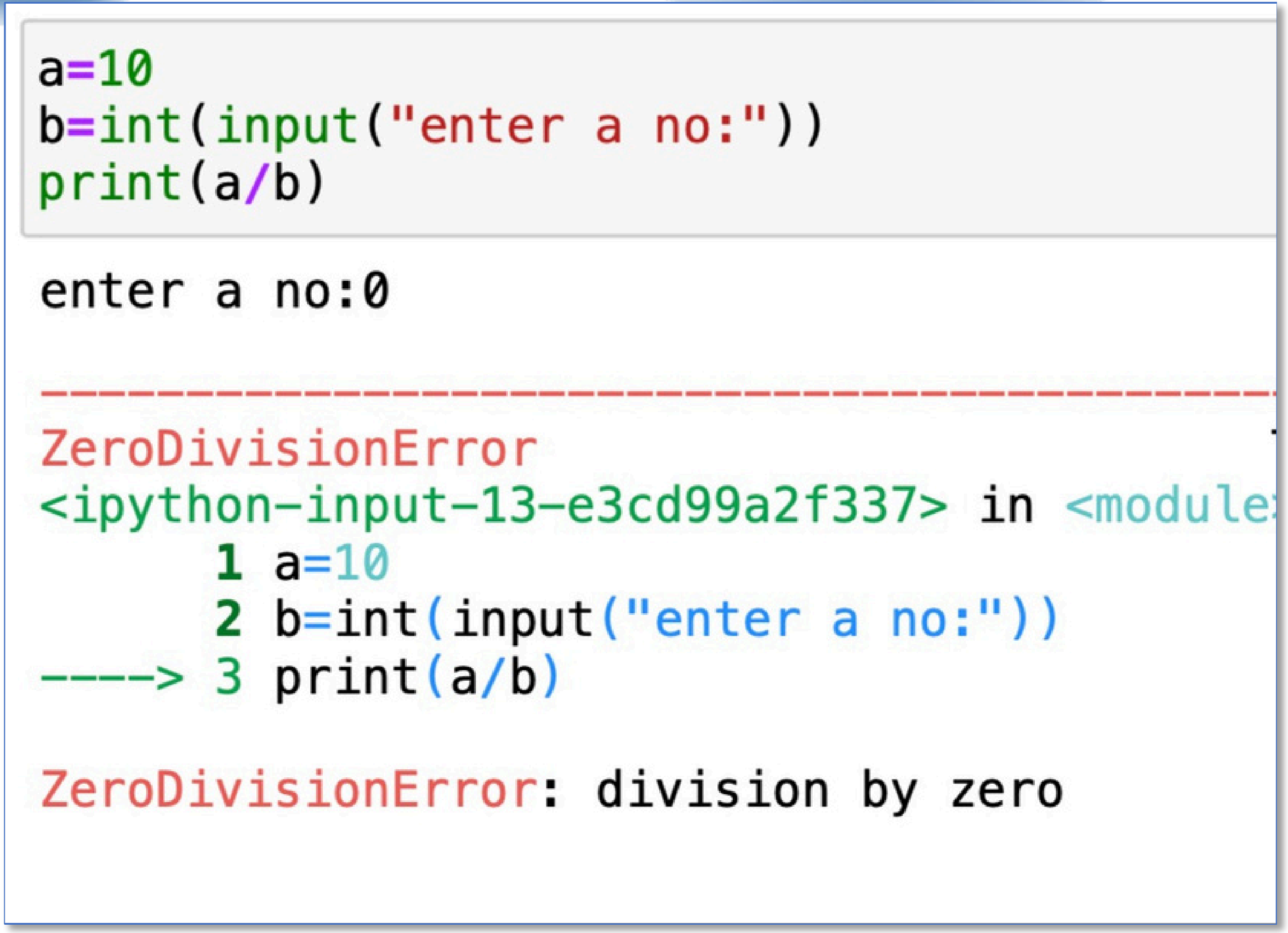


```
a=10
b=int(input("enter a no:"))
c=a/b
print("Div is", c)
```

Exception (Examples):



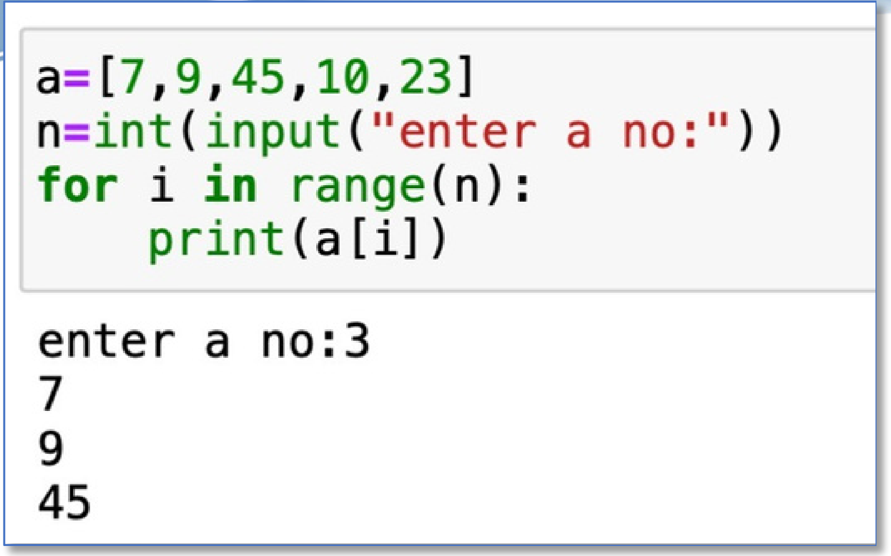
```
a=10
b=int(input("enter a no:"))
print(a/b)

enter a no:2
5.0
```



```
a=10
b=int(input("enter a no:"))
print(a/b)

enter a no:0

---------------------------------------------------------------------------
ZeroDivisionError
<ipython-input-13-e3cd99a2f337> in <module
      1 a=10
      2 b=int(input("enter a no:"))
----> 3 print(a/b)

ZeroDivisionError: division by zero
```

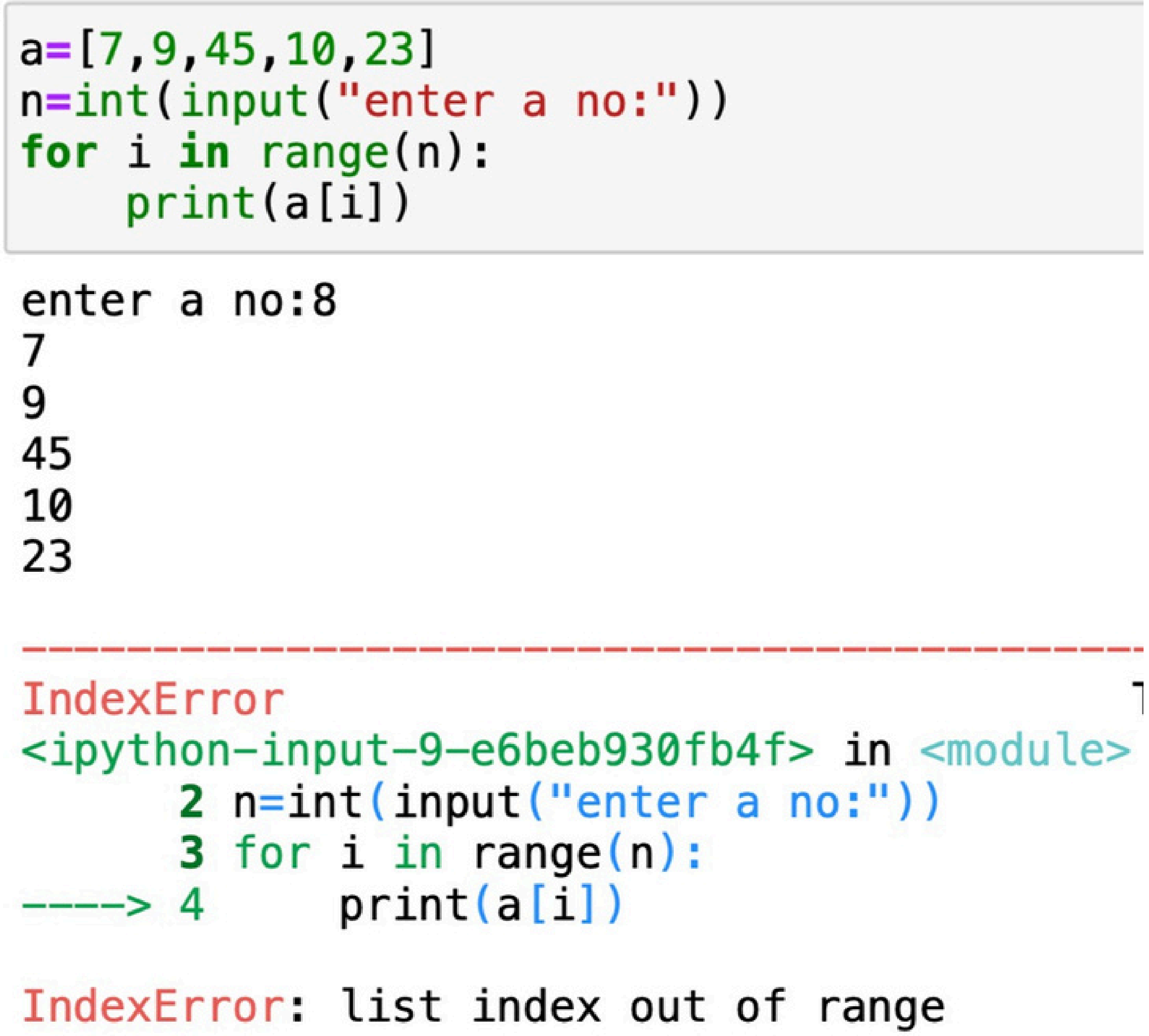
Note: Observe that there is nothing wrong with the program syntax, it is only when we try to divide an integer with zero , an exception is generated.

Exception (Examples):


```
a=[7,9,45,10,23]
n=int(input("enter a no:"))
for i in range(n):
    print(a[i])
```

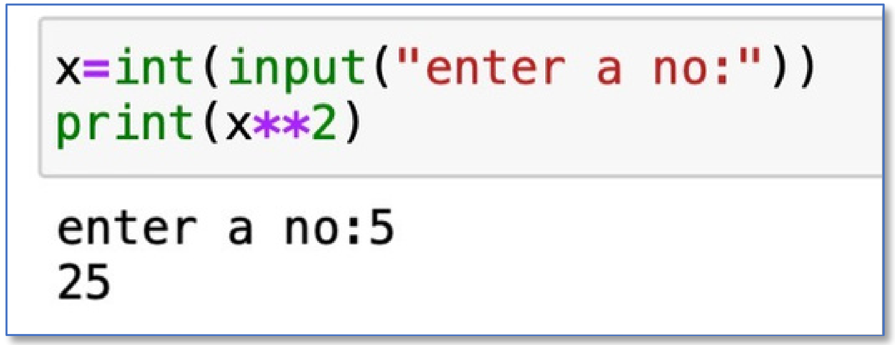
```
enter a no:3
7
9
45
```

Note: Only When we are trying to access a list element with an non existing index an exception is generated.


```
a=[7,9,45,10,23]
n=int(input("enter a no:"))
for i in range(n):
    print(a[i])
```

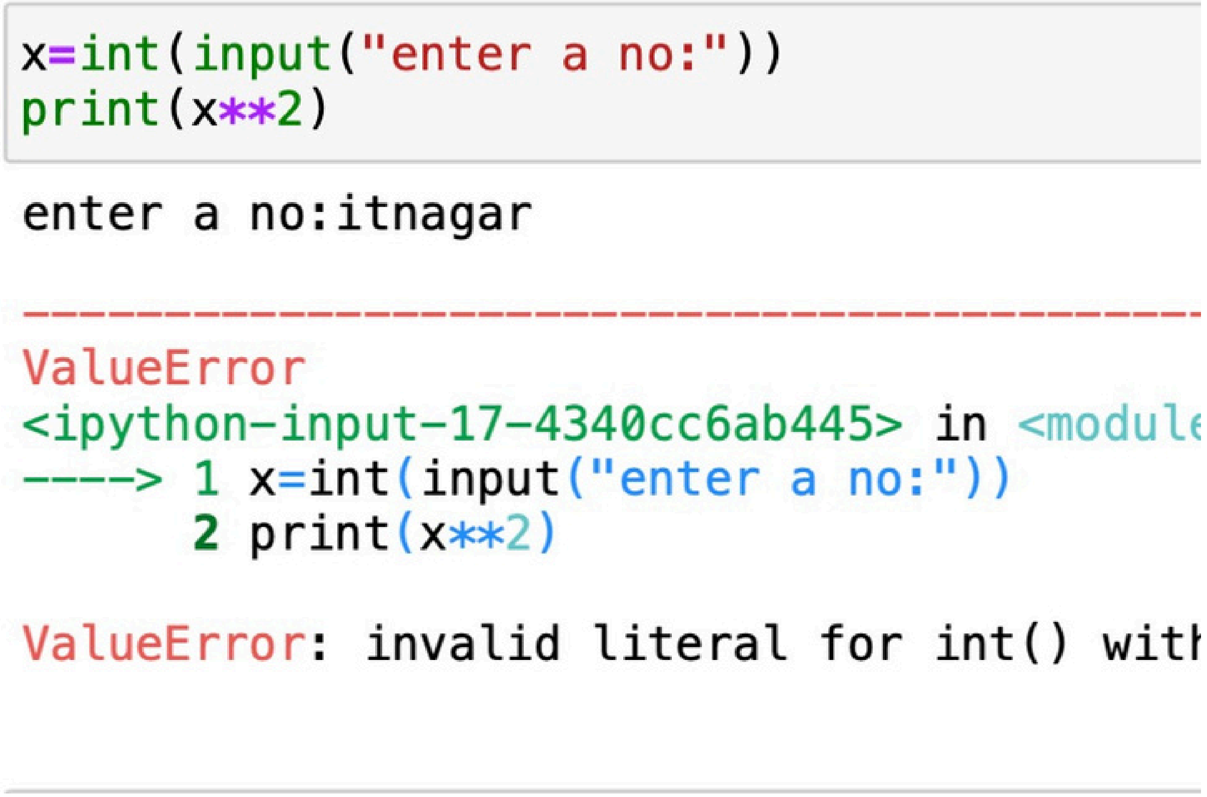
```
enter a no:8
7
9
45
10
23

-------------------------------------------
IndexError
<ipython-input-9-e6beb930fb4f> in <module>
      2 n=int(input("enter a no:"))
      3 for i in range(n):
----> 4     print(a[i])

IndexError: list index out of range
```

Exception (Examples):



```
x=int(input("enter a no:"))
print(x**2)
```

```
enter a no:5
25
```



```
x=int(input("enter a no:"))
print(x**2)
```

```
enter a no:itnagar

---------------------------------------------------------------------------
ValueError
<ipython-input-17-4340cc6ab445> in <modul
----> 1 x=int(input("enter a no:"))
      2 print(x**2)

ValueError: invalid literal for int() wit
```

Note: Only When we are trying to convert a string to integer the Exception generated.

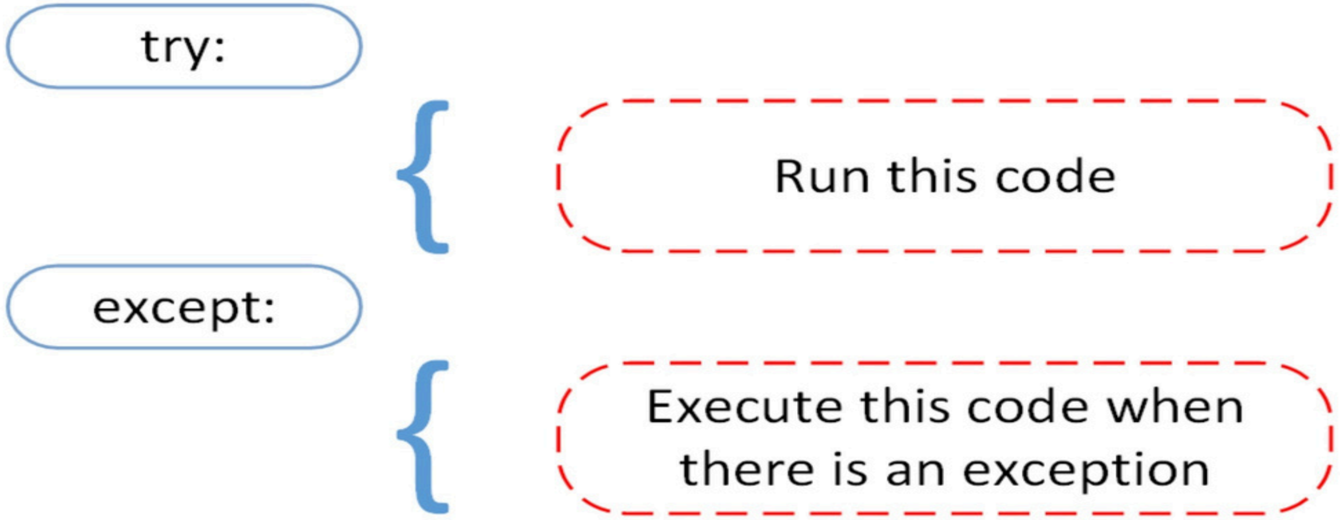
What is Exception Handling?

Exception handling in Python is a mechanism used to handle runtime errors that occur during the execution of a Python program

Exception handling allows a program to gracefully handle such exceptions and recover from errors by taking corrective actions instead of terminating abruptly. In Python, exception handling is implemented using a try-except block

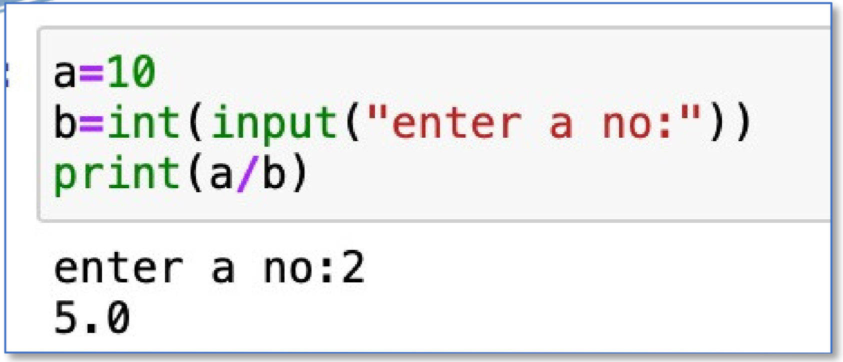
Exception Handling using. **try** and**except** Block:

The try and except block in Python is a way to handle exceptions or errors that may occur during code execution. This mechanism prevents the program from crashing by allowing it to continue running even if an error is encountered.



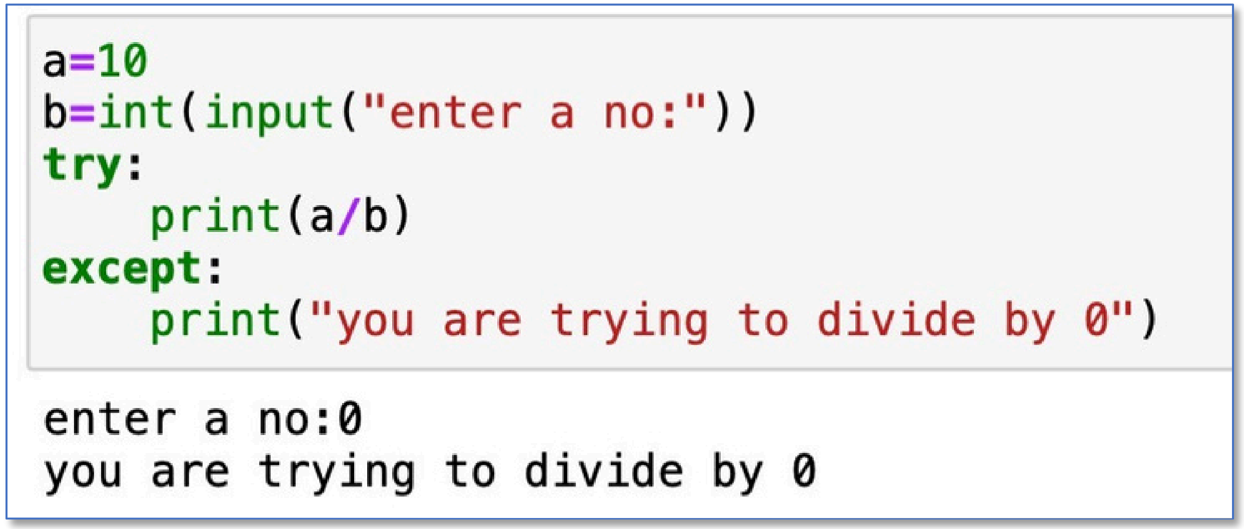
In a try block, you write the code that might raise an exception. If an exception occurs, the code execution jumps to the corresponding except block, where you can handle the error or take alternative actions.

Exception (Examples):



```
a=10
b=int(input("enter a no:"))
print(a/b)
```

```
enter a no:2
5.0
```

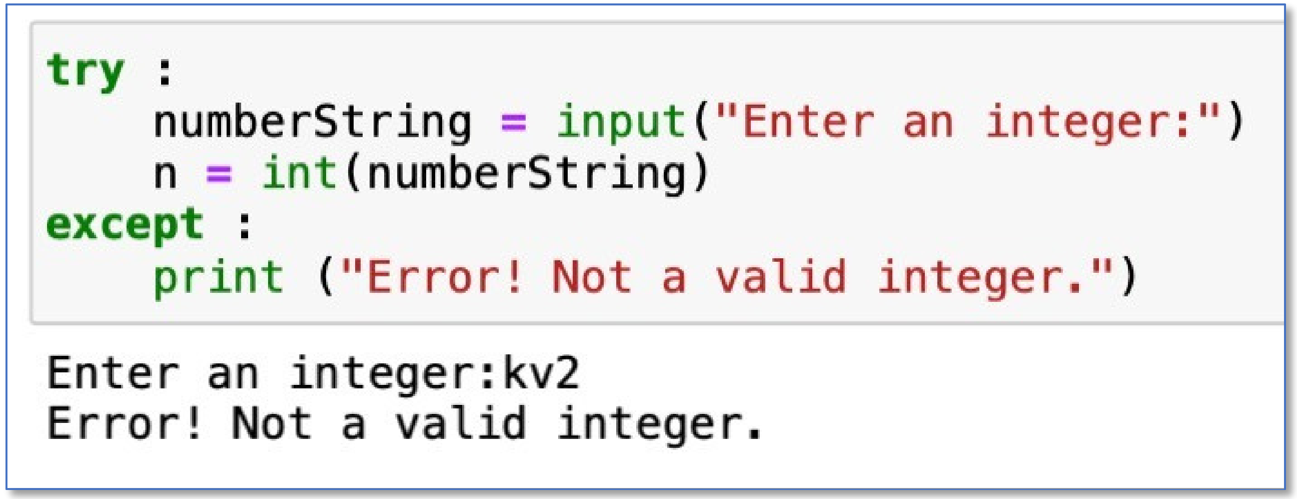


```
a=10
b=int(input("enter a no:"))
try:
    print(a/b)
except:
    print("you are trying to divide by 0")
```

```
enter a no:0
you are trying to divide by 0
```

Example:

Write a program to ensure that an integer is entered as input and in case any other value is entered, it displays a message –'Not a valid integer'



```
try :
    numberString = input("Enter an integer:")
    n = int(numberString)
except :
    print ("Error! Not a valid integer.")
```

```
Enter an integer:kv2
Error! Not a valid integer.
```

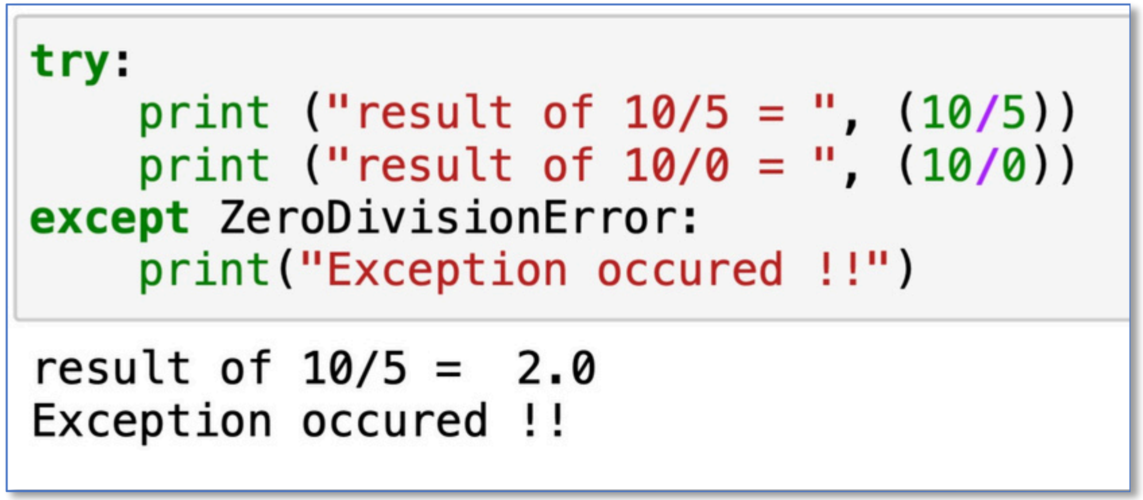
General Built-in Python Exceptions:

Exception Name	*Description*
EOFError	Raised when one of the built-in functions (input( )) hits an end-of-file condition (EOF) without reading any data. (NOTE. the file.read( ) and file.readline( ) methods return an empty string when they hit EOF.)
IO Error	Raised when an I/O operation (such as a print statement, the built-in open( ) function or a method of a file object) fails for an I/O-related reason, *e.g.*, "file not found" or "disk full".
NameError	Raised when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found.
IndexError	Raised when a sequence subscript is out of range, *e.g.*, from a list of length 4 if you try to read a value of index like 8 or E8 etc. (Slice indices are silently truncated to fall in the allowed range ; if an index is not a plain integer, *TypeError* is raised.)
ImportError	Raised when an import statement fails to find the module definition or when a from ... import fails to find a name that is tobeimported.
TypeError	Raised when an operation or function is applied to an object of inappropriate type, *e.g.*, if you try to compute a square-root of a string value. The associated value is a string giving details about the type mismatch.
ValueError	Raised when a built-in operation or function receives an argument that has the right type but an inappropriate value, and the situation is not described by a more precise exception such as *IndexError*.
ZeroDivisionError	Raised when the second argument of a division or modulo operation is zero.
OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
KeyError	Raised when a mapping (dictionary) key is not found in the set of existing keys
ImportError	Raised when the module given with import statement is not found.
KeyboardInterrupt	Raised when keys **Esc, Del** or **Ctrl+C**is pressed during program execution and normal program flow gets disturbed.

Second Argument of the Exception Block:

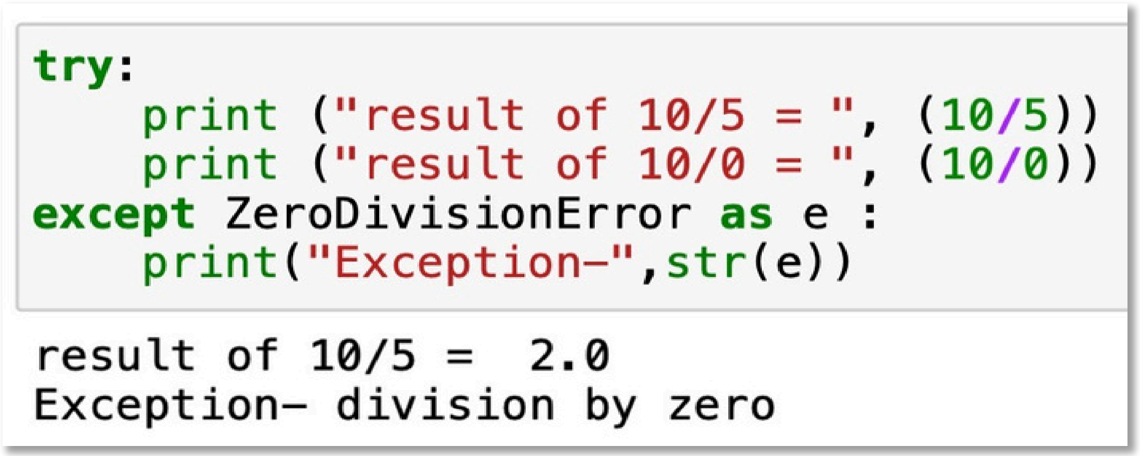
We can also provide a second argument (optional) for the except block, which gives a reference to the exception object.

```
try:
        #code
Except <Exception Name> as <Argument>:
        #code for error handling here
```



```
try:
    print ("result of 10/5 = ", (10/5))
    print ("result of 10/0 = ", (10/0))
except ZeroDivisionError:
    print("Exception occured !!")
```

```
result of 10/5 =  2.0
Exception occured !!
```



```
try:
    print ("result of 10/5 = ", (10/5))
    print ("result of 10/0 = ", (10/0))
except ZeroDivisionError as e :
    print("Exception-",str(e))
```

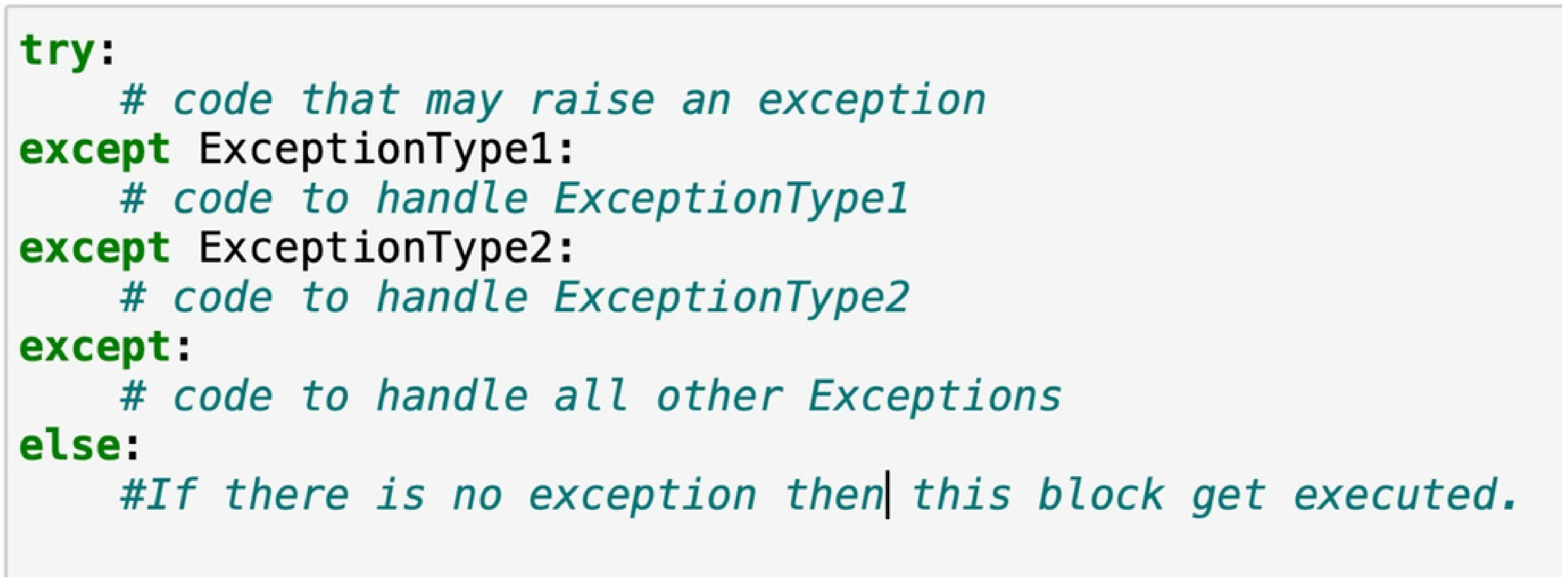
```
result of 10/5 =  2.0
Exception- division by zero
```

Handling Multiple Errors

Handling multiple exceptions in Python allows a single try-except block to handle different types of exceptions using multiple except blocks. This allows a program to handle various types of errors that may occur during runtime and take corrective measures accordingly.

In a try-except block, each except block is associated with a specific exception type, and the block containing the code to handle that exception is executed if the corresponding exception occurs in the try block. By handling multiple exceptions, programmers can write more robust and less error-prone code.

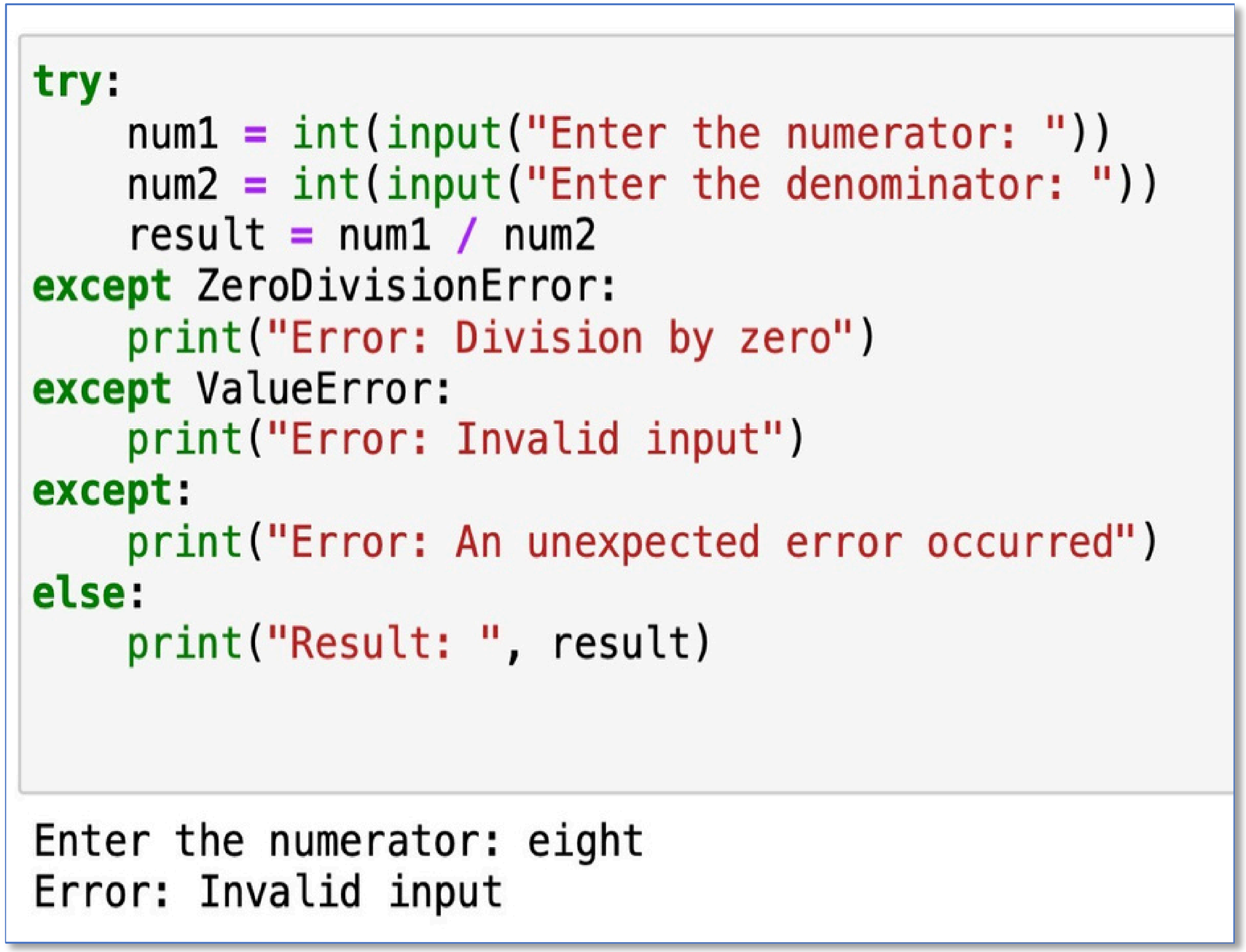
Syntax:



```
try:
    # code that may raise an exception
except ExceptionType1:
    # code to handle ExceptionType1
except ExceptionType2:
    # code to handle ExceptionType2
except:
    # code to handle all other Exceptions
else:
    #If there is no exception then this block get executed.
```

Example:

Program to handle multiple exceptions:


```
try:
    num1 = int(input("Enter the numerator: "))
    num2 = int(input("Enter the denominator: "))
    result = num1 / num2
except ZeroDivisionError:
    print("Error: Division by zero")
except ValueError:
    print("Error: Invalid input")
except:
    print("Error: An unexpected error occurred")
else:
    print("Result: ", result)
```

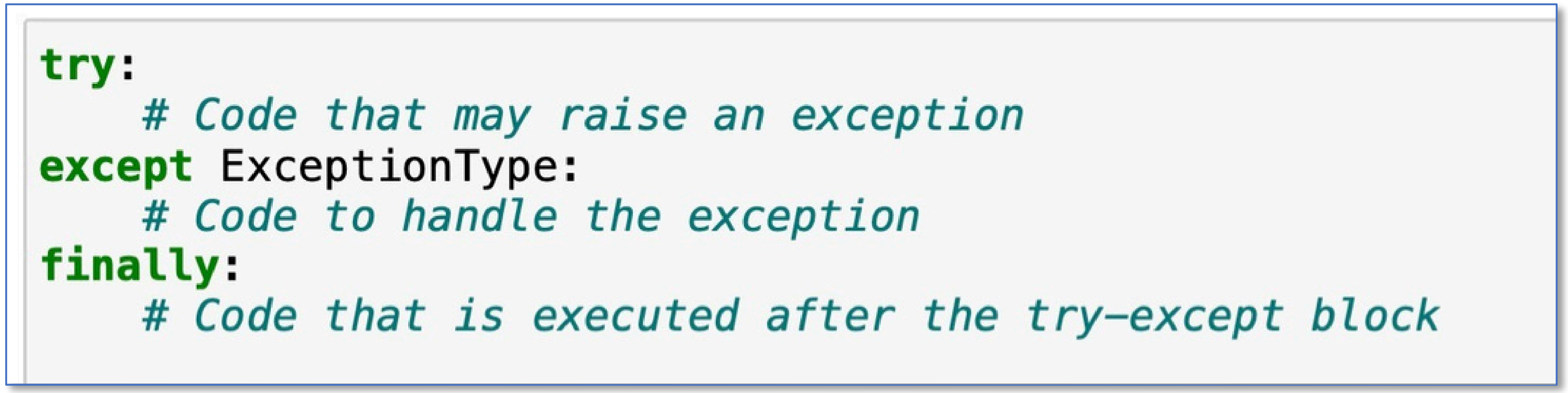
```
Enter the numerator: eight
Error: Invalid input
```

Note (Execution Order):

The <try suite> is executed first ; if, during the course of executing the <try suite>, an exception is raised that is not handled otherwise, and the <except suite> is executed, with <name> bound to the exception, if found ; if no matching except suite is found then *unnamed except suite* is executed.

finally Block :

The finally block is a part of the try-except block in Python that contains the code that is executed regardless of whether an exception is raised or not. The syntax of the try-except-finally block is as follows:

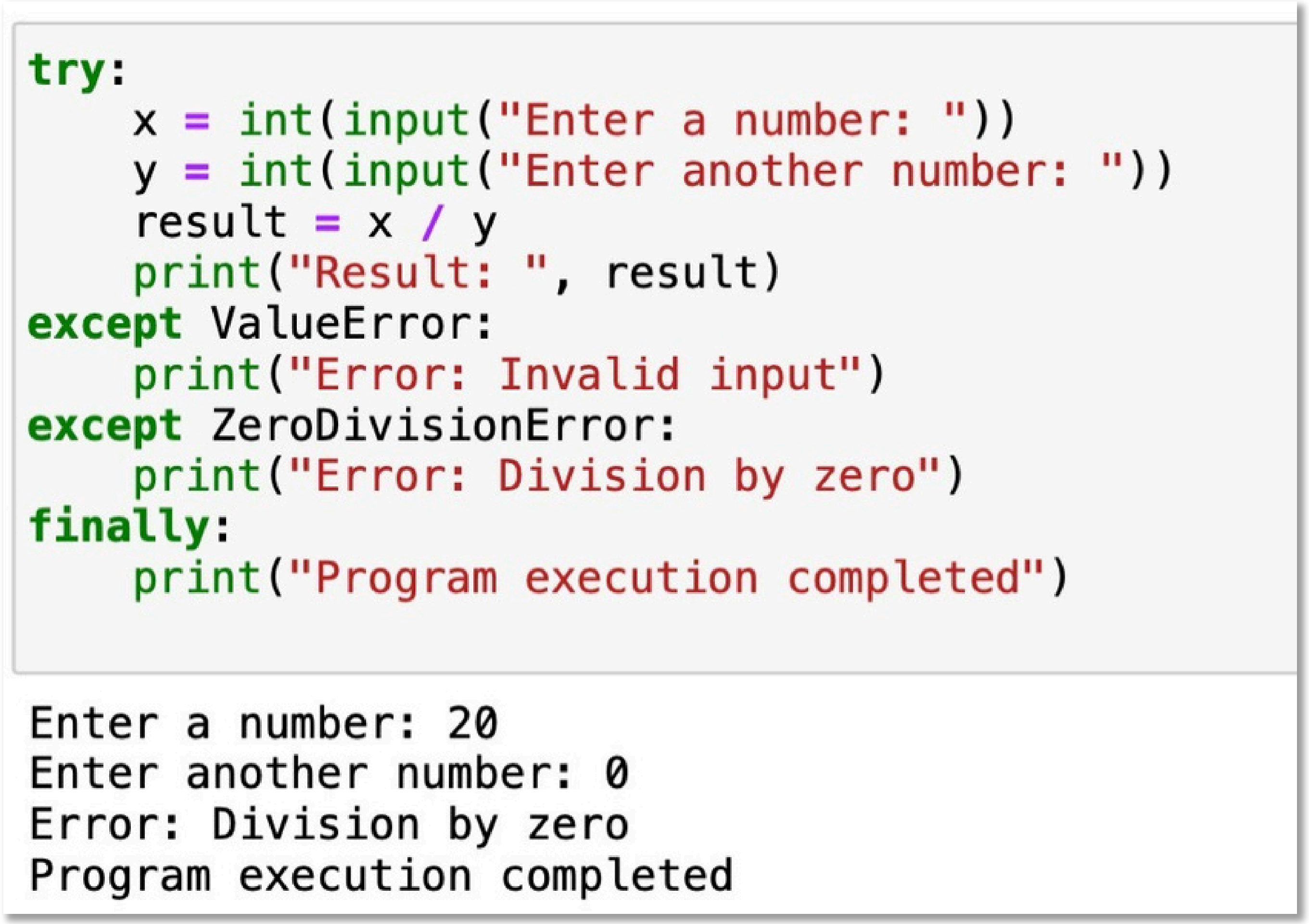


```
try:
    # Code that may raise an exception
except ExceptionType:
    # Code to handle the exception
finally:
    # Code that is executed after the try-except block
```

the try block contains the code that may raise an exception. If an exception occurs, the control is transferred to the corresponding except block, which contains the code to handle the exception. The finally block contains the code that is executed after the try-except blocks, regardless of whether an exception occurred or not.

Example :

Program using finally block



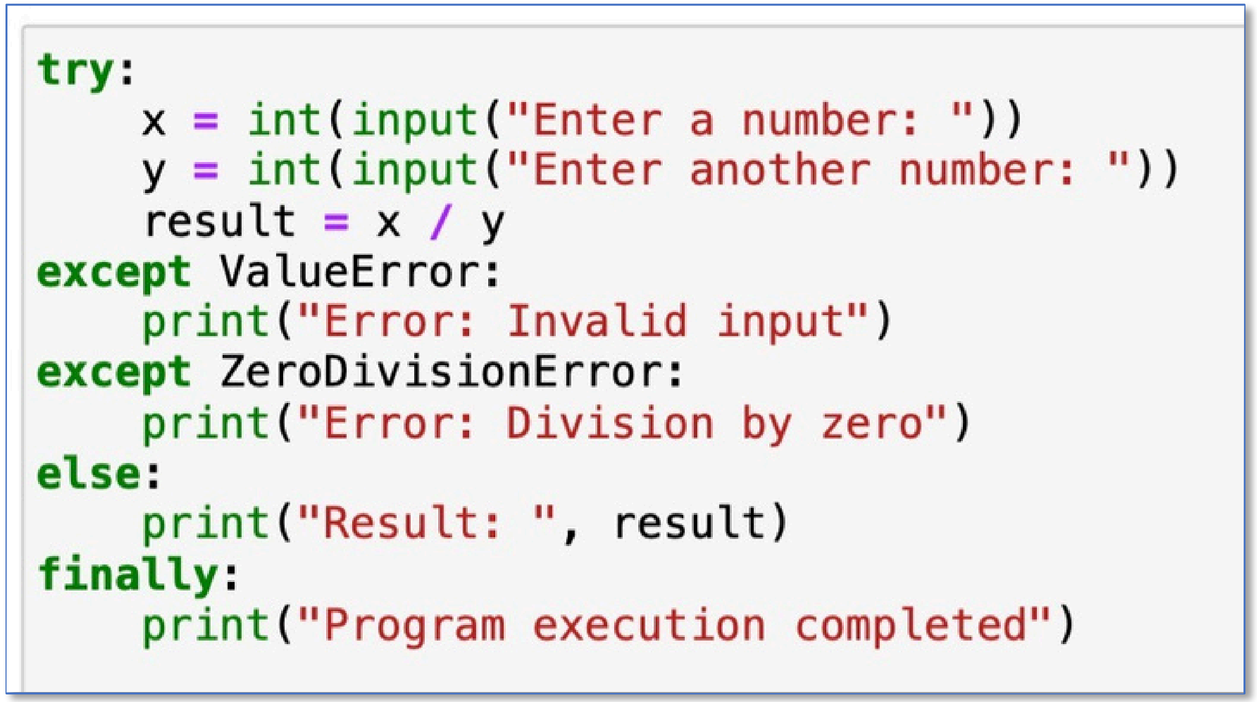
```
try:
    x = int(input("Enter a number: "))
    y = int(input("Enter another number: "))
    result = x / y
    print("Result: ", result)
except ValueError:
    print("Error: Invalid input")
except ZeroDivisionError:
    print("Error: Division by zero")
finally:
    print("Program execution completed")
```

```
Enter a number: 20
Enter another number: 0
Error: Division by zero
Program execution completed
```

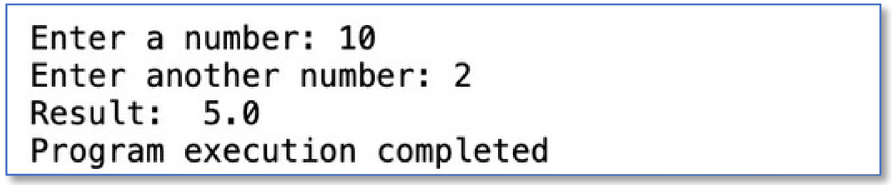
Example :

What is the order of execution?

In this example, if the user enters an invalid input or attempts to divide by zero, the corresponding except block handles the exception and prints an error message to the user. If no exception occurs, the else block is executed and prints the result. Finally, the finally block is executed and prints a message to indicate the completion of the program execution



```
try:
    x = int(input("Enter a number: "))
    y = int(input("Enter another number: "))
    result = x / y
except ValueError:
    print("Error: Invalid input")
except ZeroDivisionError:
    print("Error: Division by zero")
else:
    print("Result: ", result)
finally:
    print("Program execution completed")
```



```
Enter a number: 10
Enter another number: 2
Result:  5.0
Program execution completed
```

Practice Programs Example:

1. Write a Python program that takes two numbers as input from the user and calculates the quotient of the two numbers. Handle the exceptions that may occur during the program execution, such as invalid input or division by zero.

2. Write a Python program that reads a file and displays its contents on the screen. Handle the exceptions that may occur during the program execution, such as the file not found error or file reading error.

3. Write a Python program that takes a list of integers as input from the user and calculates the average of the numbers. Handle the exceptions that may occur during the program execution, such as invalid input or division by zero.

4. Write a Python program that reads a CSV file and displays its contents on the screen. Handle the exceptions that may occur during the program execution, such as the file not found error or file reading error.

5. Write a Python program that takes a string as input from the user and converts it to an integer. Handle the exceptions that may occur during the program execution, such as invalid input or string conversion error.

Practice Programs Example:

6. Write a Python program that takes a list of numbers as input from the user and finds the maximum and minimum numbers in the list. Handle the exceptions that may occur during the program execution, such as invalid input or empty list error.

7. Write a Python program that reads a file and writes its contents to another file. Handle the exceptions that may occur during the program execution, such as the file not found error or file reading/writing error.

8. Write a Python program that takes two strings as input from the user and concatenates them. Handle the exceptions that may occur during the program execution, such as invalid input or string concatenation error.

9. Write a Python program that reads a text file and counts the number of words in it. Handle the exceptions that may occur during the program execution, such as the file not found error or file reading error.

10. Write a Python program that takes a string as input from the user and reverses it. Handle the exceptions that may occur during the program execution, such as invalid input or string reversal error.