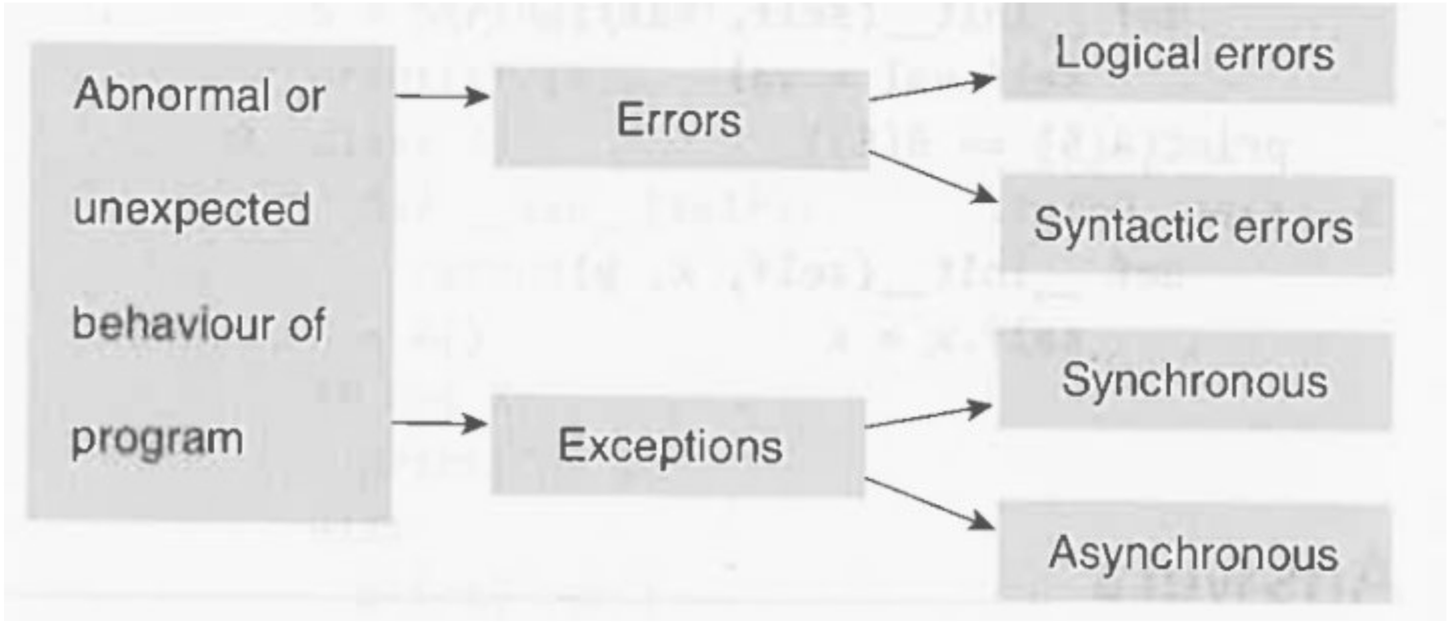
Module 4.4 :

Exceptions Handling

Topic

- Errors in python program
- Exceptions
- Exception Handling
- Types of Exceptions,
- The Except Block
- The assert statement

Errors in python program



Errors in python program

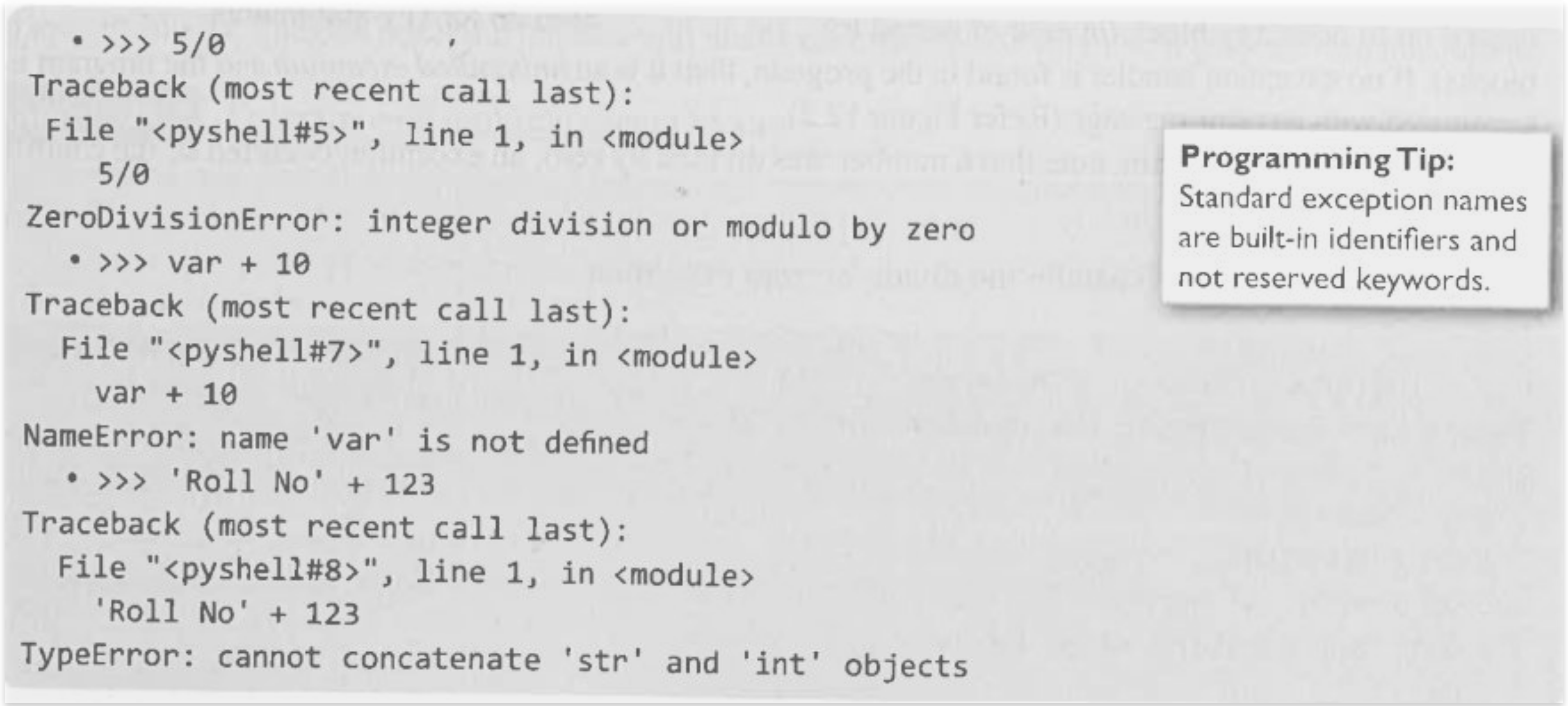
Logical Error : Due to poor understanding of problems and its solution

Syntax Error: arises due to poor understanding of the language

Exceptions are run-time anomalies or unusual conditions (such as divide by zero, accessing arrays out of its bounds, running out of memory or disk space, overflow, and underflow) that a program may encounter during Execution.

Synchronous exceptions (like divide by zero, array index out of bound, etc.) can be controlled by the program,
Asynchronous exceptions (like an interrupt from the keyboard, hardware malfunction, or disk failure), on the other hand, arc caused by events that are beyond the control of the program.

Exceptions



```
  • >>> 5/0
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    5/0
ZeroDivisionError: integer division or modulo by zero
  • >>> var + 10
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    var + 10
NameError: name 'var' is not defined
  • >>> 'Roll No' + 123
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    'Roll No' + 123
TypeError: cannot concatenate 'str' and 'int' objects
```

Programming Tip: Standard exception names are built-in identifiers and not reserved keywords.

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Exceptions

- Even if a statement is syntactically correct, it may still cause an error when executed. Such errors that occur at run-time (or during execution) are known as exceptions.
- An exception is an event, which occurs during the execution of a program and disrupts the normal flow of the program's instructions. When a program encounters a situation which it cannot deal with, it raises an exception. Therefore, we can say that an exception is a Python object that represents an error.
- When a program raises an exception, it must handle the exception or the program will be immediately terminated. You can handle exceptions in your programs to end it gracefully, otherwise, if exceptions are not handled by programs, then error messages are generated. Let us see some examples in which exceptions occurs

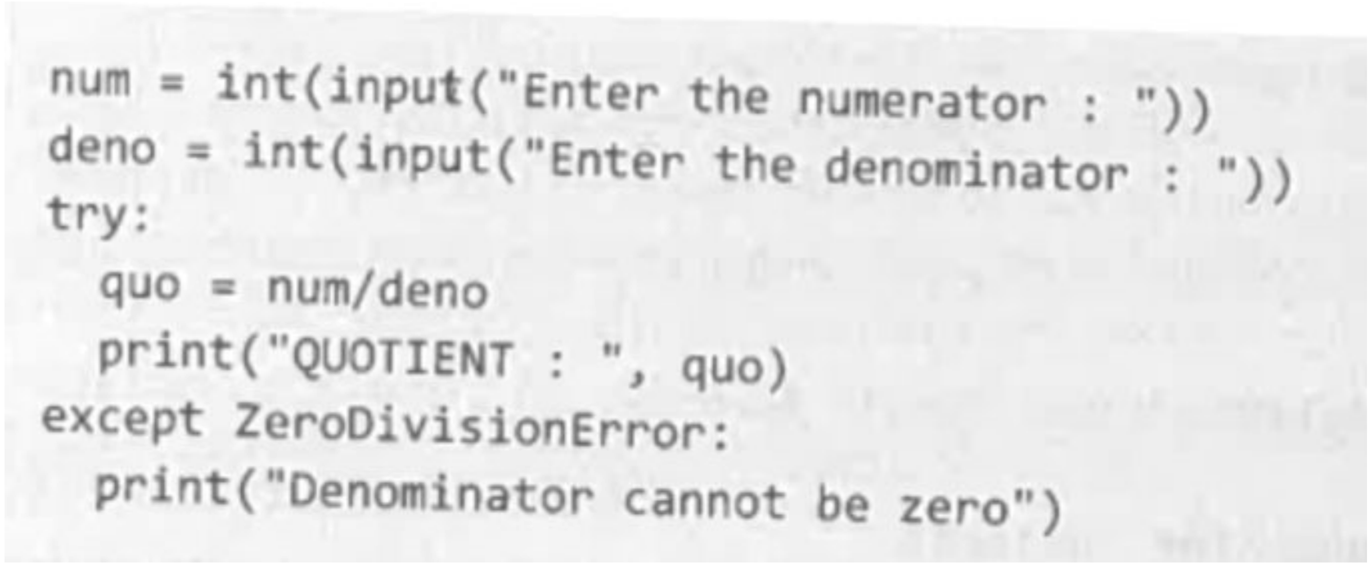
Handling Exceptions

- Even if a statement is syntactically correct, it may still cause an error when executed. Such errors that occur at run-time (or during execution) are known as exceptions.
- An exception is an event, which occurs during the execution of a program and disrupts the normal flow of the program's instructions. When a program encounters a situation which it cannot deal with, it raises an exception. Therefore, we can say that an exception is a Python object that represents an error.
- When a program raises an exception, it must handle the exception or the program will be immediately terminated. You can handle exceptions in your programs to end it gracefully, otherwise, if exceptions are not handled by programs, then error messages are generated. Let us see some examples in which exceptions occurs

Handling Exceptions

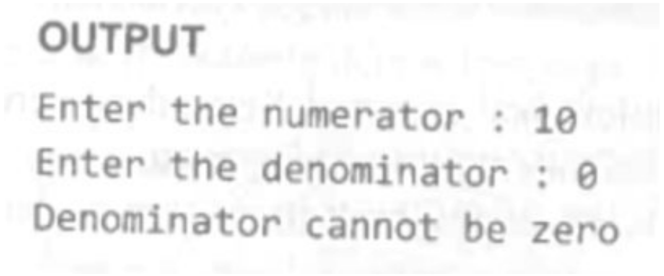
- The try block lets you test a block of code for errors.
- The except block lets you handle the error.
- The else block lets you execute code when there is no error.
- The finally block lets you execute code, regardless of the result of the try- and except blocks.

try --- except



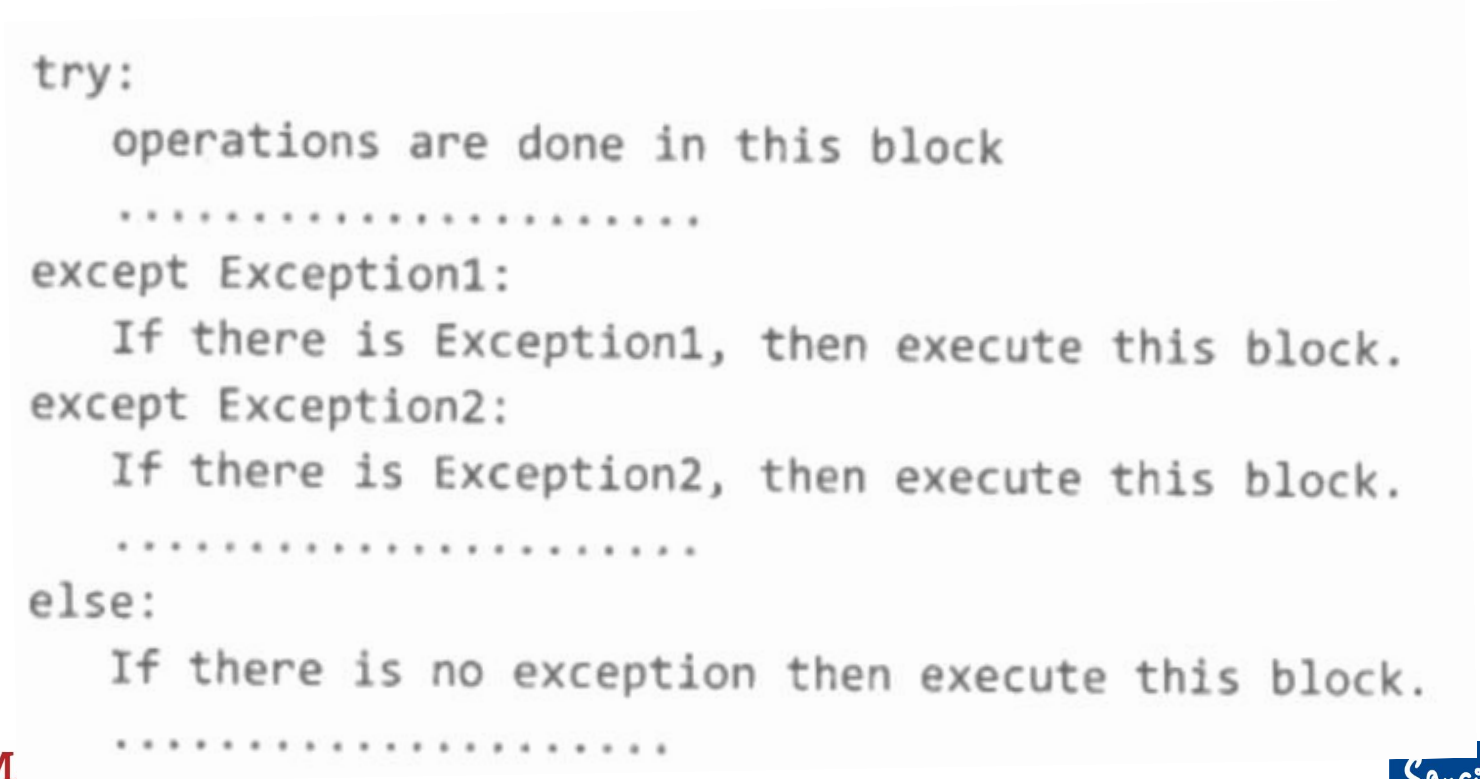
```
num = int(input("Enter the numerator : "))
deno = int(input("Enter the denominator : "))
try:
  quo = num/deno
  print("QUOTIENT : ", quo)
except ZeroDivisionError:
  print("Denominator cannot be zero")
```

try --- except


OUTPUT

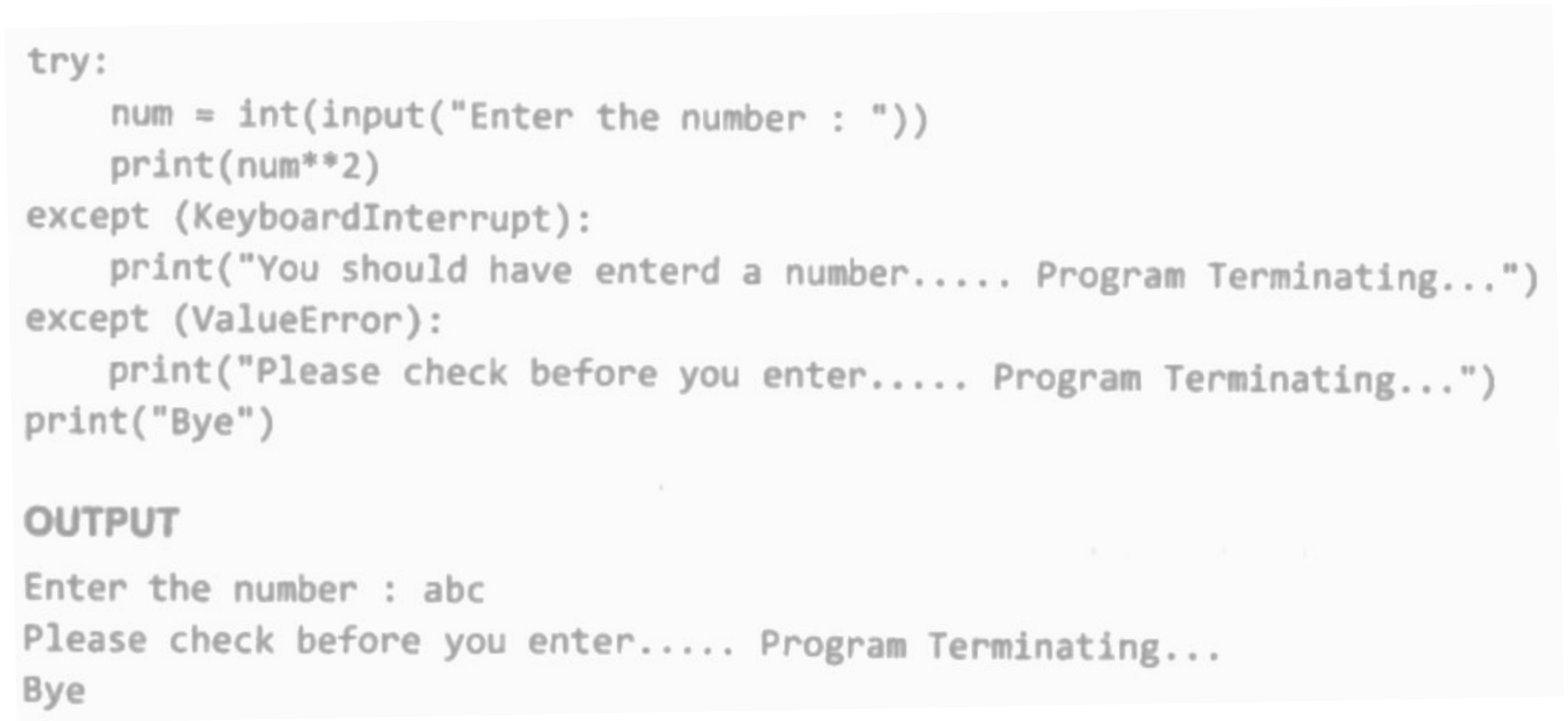
```
Enter the numerator : 10
Enter the denominator : 0
Denominator cannot be zero
```

Multiple except block



```
try:
    operations are done in this block
    .......................
except Exception1:
    If there is Exception1, then execute this block.
except Exception2:
    If there is Exception2, then execute this block.
    .......................
else:
    If there is no exception then execute this block.
    .......................
```

Multiple except block



```
try:
    num = int(input("Enter the number : "))
    print(num**2)
except (KeyboardInterrupt):
    print("You should have enterd a number..... Program Terminating...")
except (ValueError):
    print("Please check before you enter..... Program Terminating...")
print("Bye")
```

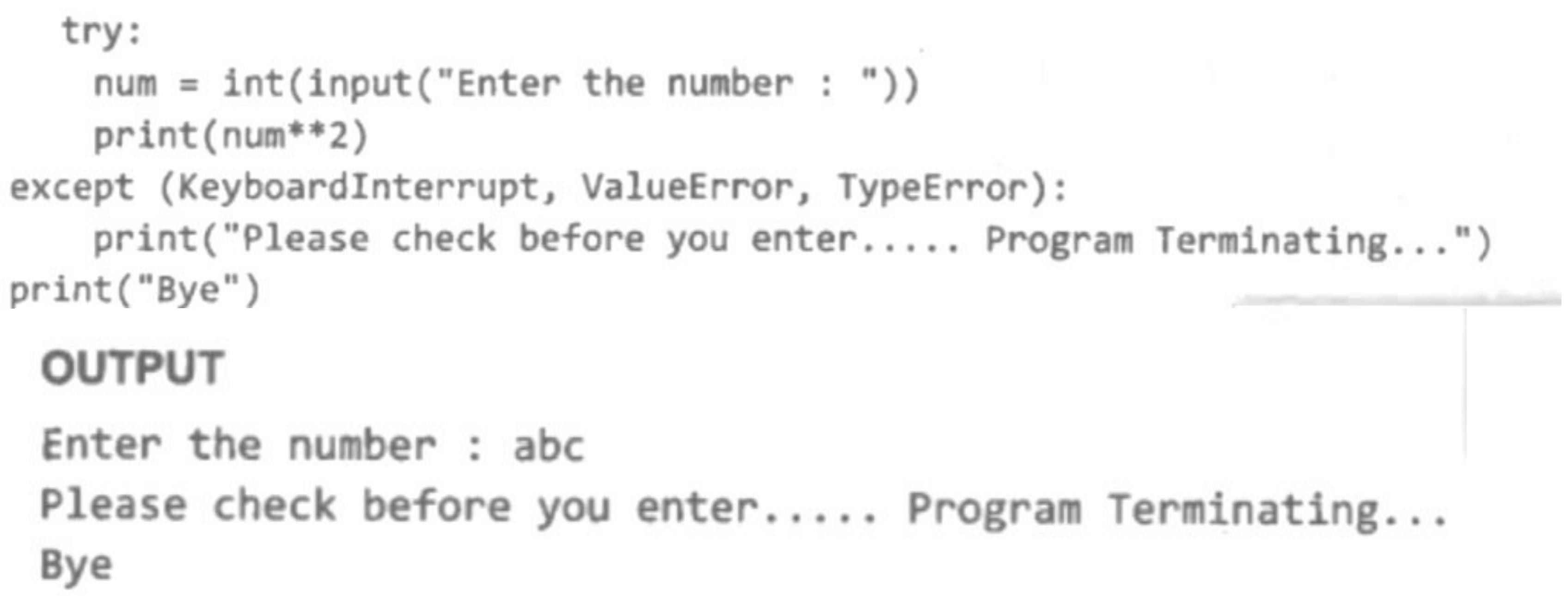
OUTPUT

```
Enter the number : abc
Please check before you enter..... Program Terminating...
Bye
```

SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya College of Engineering

Somaiya TRUST

Multiple exception in single block

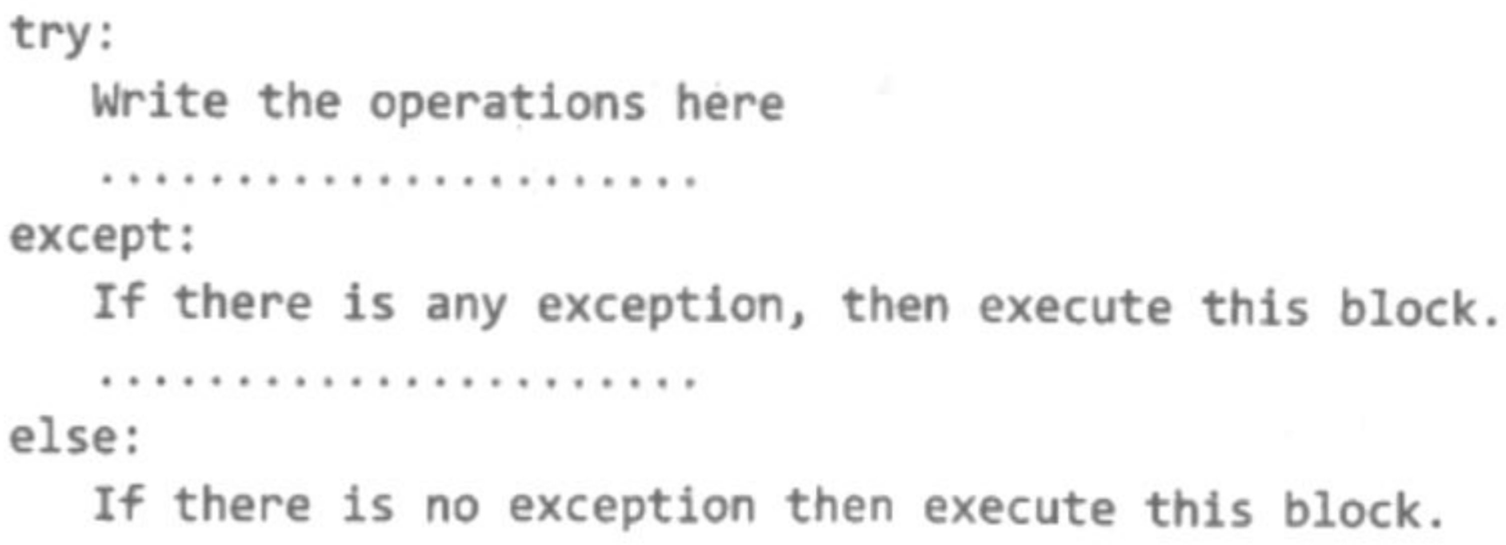


```
  try:
    num = int(input("Enter the number : "))
    print(num**2)
except (KeyboardInterrupt, ValueError, TypeError):
    print("Please check before you enter..... Program Terminating...")
print("Bye")
```

OUTPUT

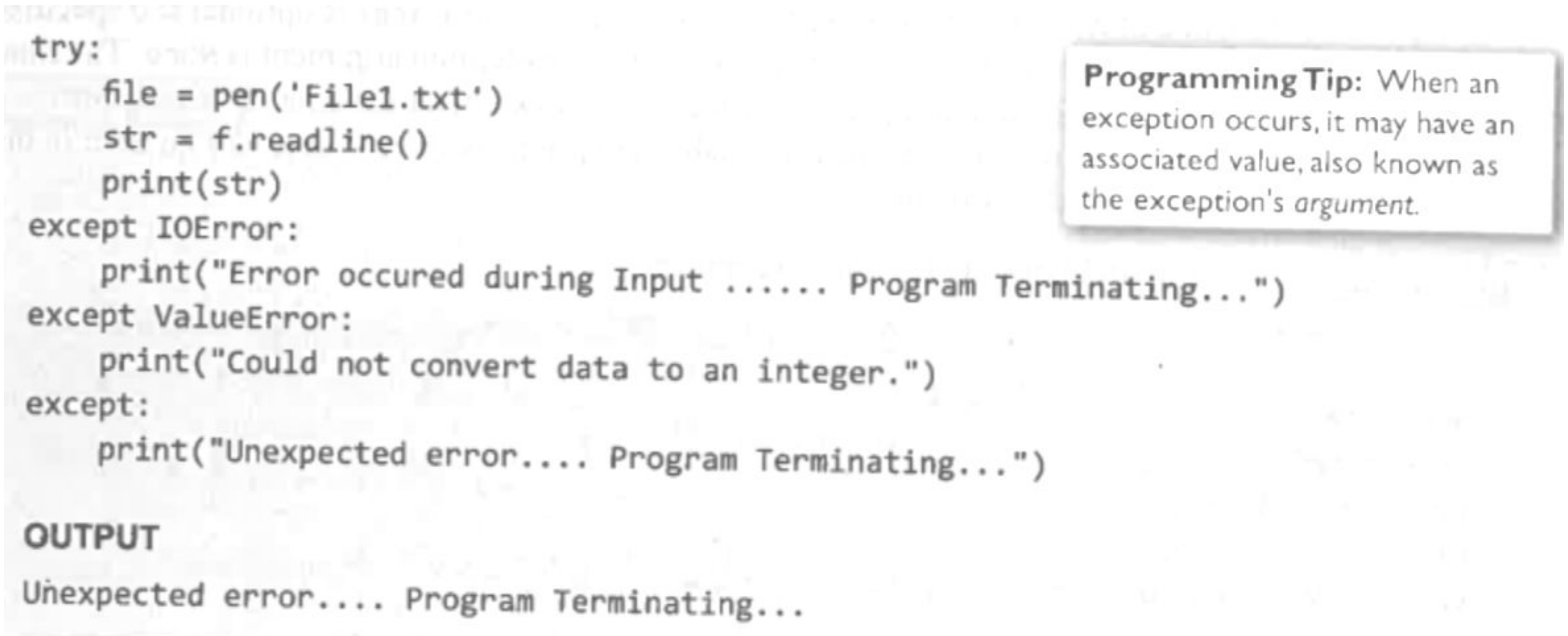
```
Enter the number : abc
Please check before you enter..... Program Terminating...
Bye
```

except block without exception



```
try:
    Write the operations here
    ......................
except:
    If there is any exception, then execute this block.
    ......................
else:
    If there is no exception then execute this block.
```

except block without exception



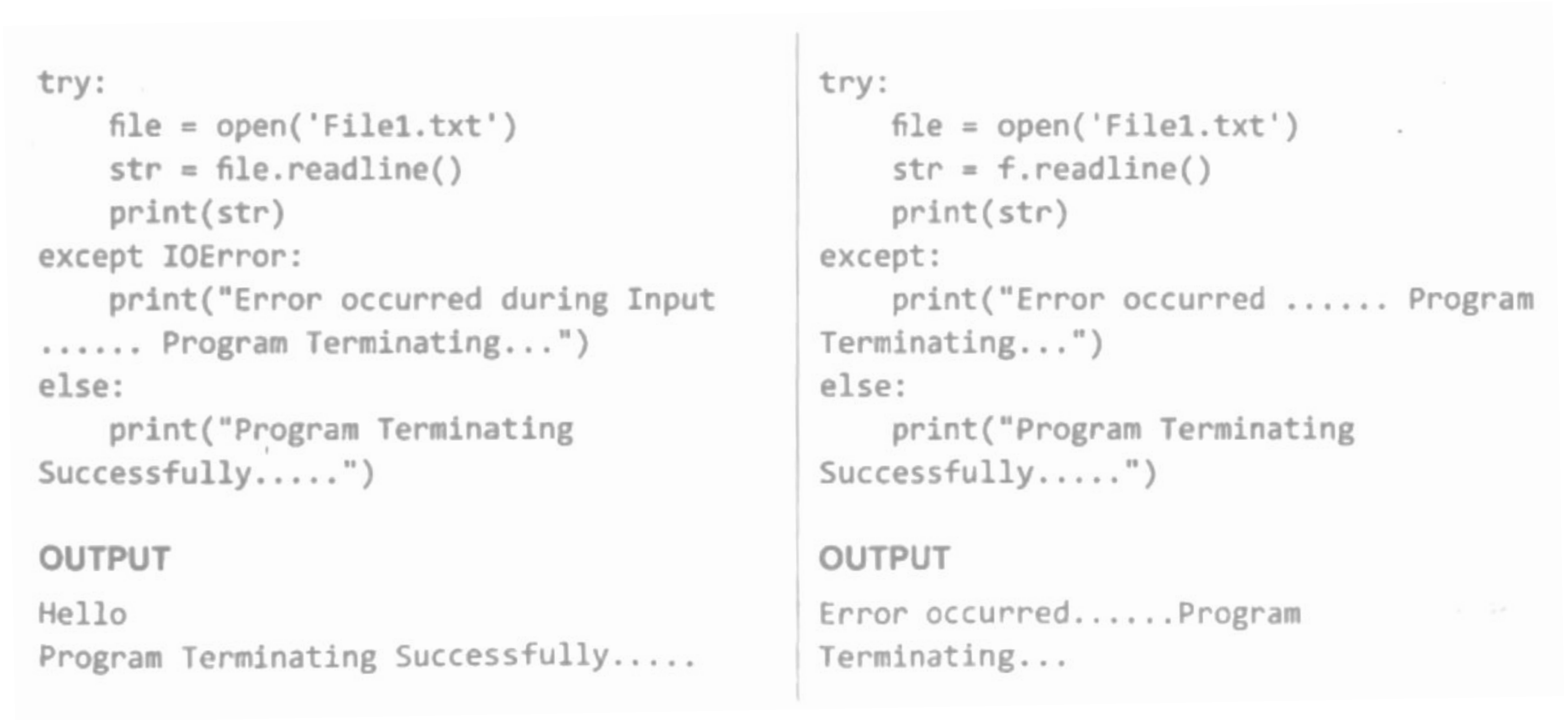
```
try:
    file = pen('File1.txt')
    str = f.readline()
    print(str)
except IOError:
    print("Error occured during Input ...... Program Terminating...")
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error.... Program Terminating...")
```

Programming Tip: When an exception occurs, it may have an associated value, also known as the exception's *argument*.

OUTPUT

```
Unexpected error.... Program Terminating...
```

THE else CLAUSE



```
try:
    file = open('File1.txt')
    str = file.readline()
    print(str)
except IOError:
    print("Error occurred during Input
...... Program Terminating...")
else:
    print("Program Terminating
Successfully.....")
```

OUTPUT

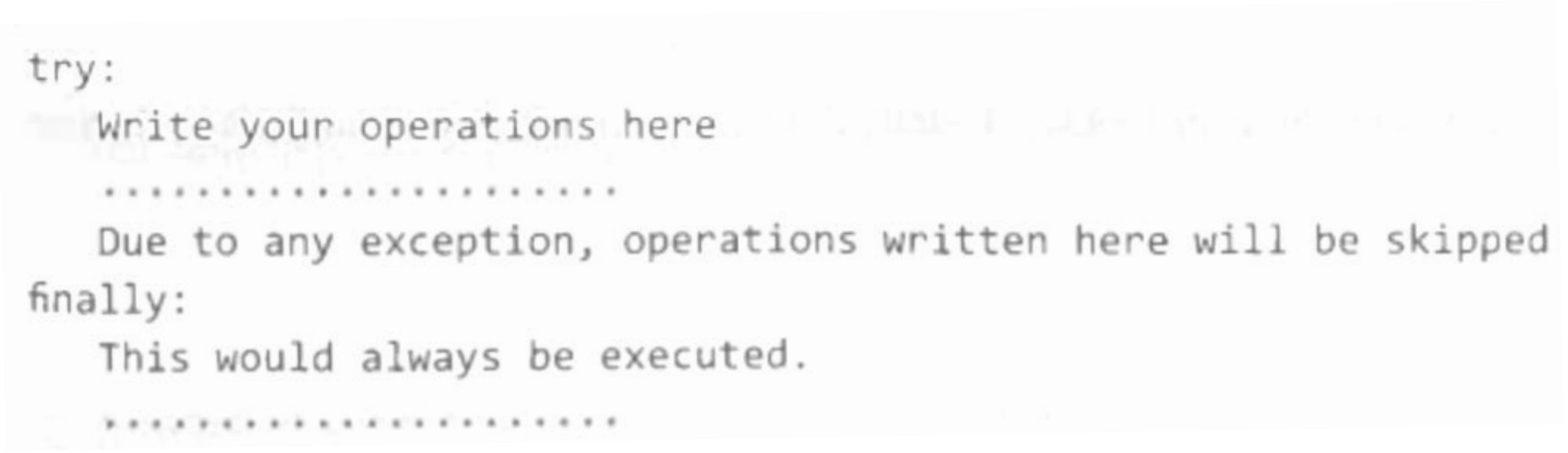
```
Hello
Program Terminating Successfully.....
```

```
try:
    file = open('File1.txt')
    str = f.readline()
    print(str)
except:
    print("Error occurred ...... Program
Terminating...")
else:
    print("Program Terminating
Successfully.....")
```

OUTPUT

```
Error occurred......Program
Terminating...
```

THE finally BLOCK



```
try:
    Write your operations here
    .......................
    Due to any exception, operations written here will be skipped
finally:
    This would always be executed.
    .......................
```

THE finally BLOCK

```
try:
        print(“Raising Exception..... i)
        raise ValueError
finally:
        print("Performing clean up in Finally..... a)
```

OUTPUT

```
Raising Exception.....
Performing clean up in Finally......
Traceback (most recent call last):
        File “C:\Python34\Try.py"”, line 4, in <module>
        raise ValueError
ValueError
```

THE finally BLOCK

```
try:
        print(“Raising Exception..... ")
        raise ValueError
except:
        print(“Exception caught..... ”)
finally:
        print("Performing clean up in Finally...... ”)
```

OUTPUT

```
Raising Exception.....
Exception caught.....
Performing clean up in Finally......
```

ASSERTIONS IN PYTHON

An assertion is a basic check that can be tuned on or off when the program is being tested. You can think of assert as a raise-if statement (or raise-if-not statement). Using assert statement, an expression is tested and if the result of the expression is False then an exception is raised. The assert statement is intended for debugging statements. It can be seen as an abbreviated notation for a conditional raise statement.

In Python, assertions are implemented using **assert** keyword. Assertions are usually placed at the start of a function to check for valid input, and after a function call to check for valid output.

When Python encounters an assert statement, the expression associated with it is calculated and if the expression is False, an **AssertionError** is raised. The syntax for assert statement is:

assert expression[, arguments]

ASSERTIONS IN PYTHON

```
c = int(input(“Enter the temperature in Celsius: "))
f= (c* 9/5) + 32
assert(f<=32), "Its freezing"
print("Temperature in Fahrenheit = ”,f)
```

OUTPUT:

```
Enter the temperature in Celsius: 100
Traceback (most recent call last):
    File "C:\Python34\Try.py", line 3, in <module>
        assert(f<=32), "Its freezing"
AssertionError: Its freezing
```

Key points to remember:

1. Do not catch exceptions that you cannot handle.

2 User defined exceptions can be very useful if some complex or specific information has to be stored in exception instances.

3. Do not create new exception classes when the built-in exceptions already have all the functionality you need.

Thank You