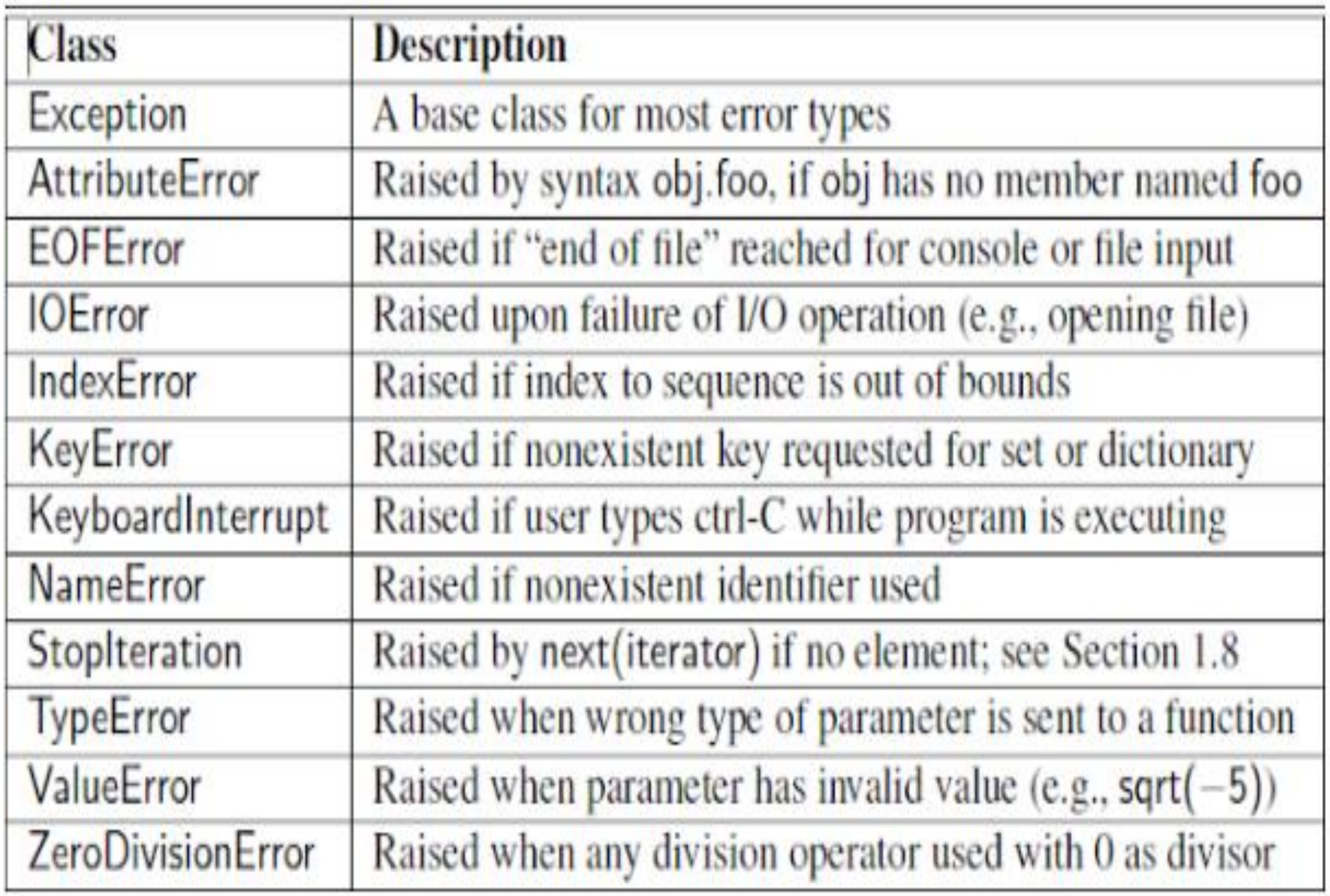
Exception Handling

- **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.

- **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.

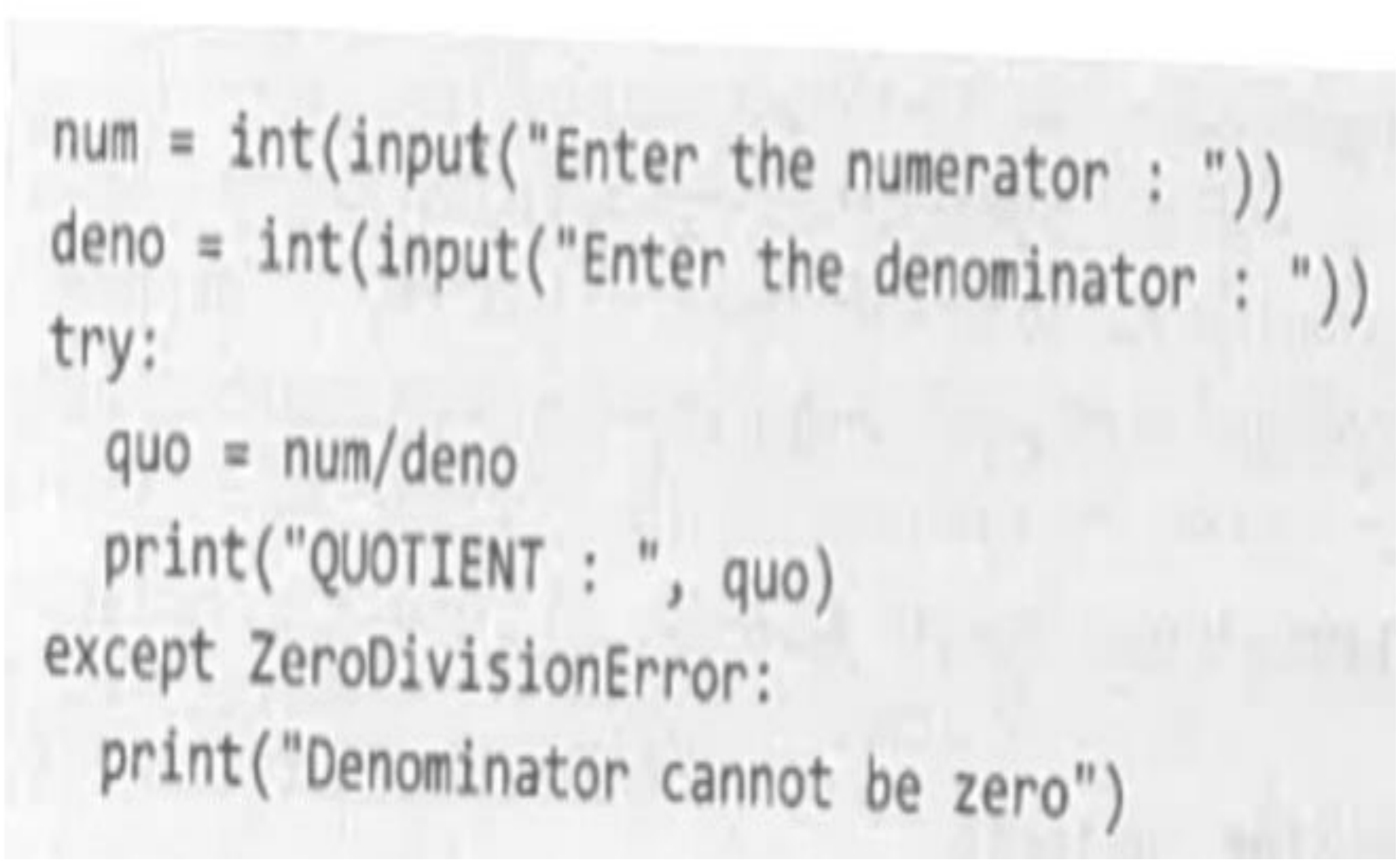
- **Different types of exceptions in python:**



Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax obj.foo, if obj has no member named foo
EOFError	Raised if "end of file" reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by next(iterator) if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., sqrt(−5))
ZeroDivisionError	Raised when any division operator used with 0 as divisor

try and except Statement - Catching Exceptions

- In Python, we catch exceptions and handle them using try and except code blocks. The try clause contains the code that can raise an exception, while the except clause contains the code lines that handle the exception.



```
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")
```

The except Clause with No Exceptions:

- This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.
- **Example:**

```
try:
    x = 10 / 0  # Division by zero
except:
    print("An exception occurred")

print("Program continues...")
```

Multiple except block

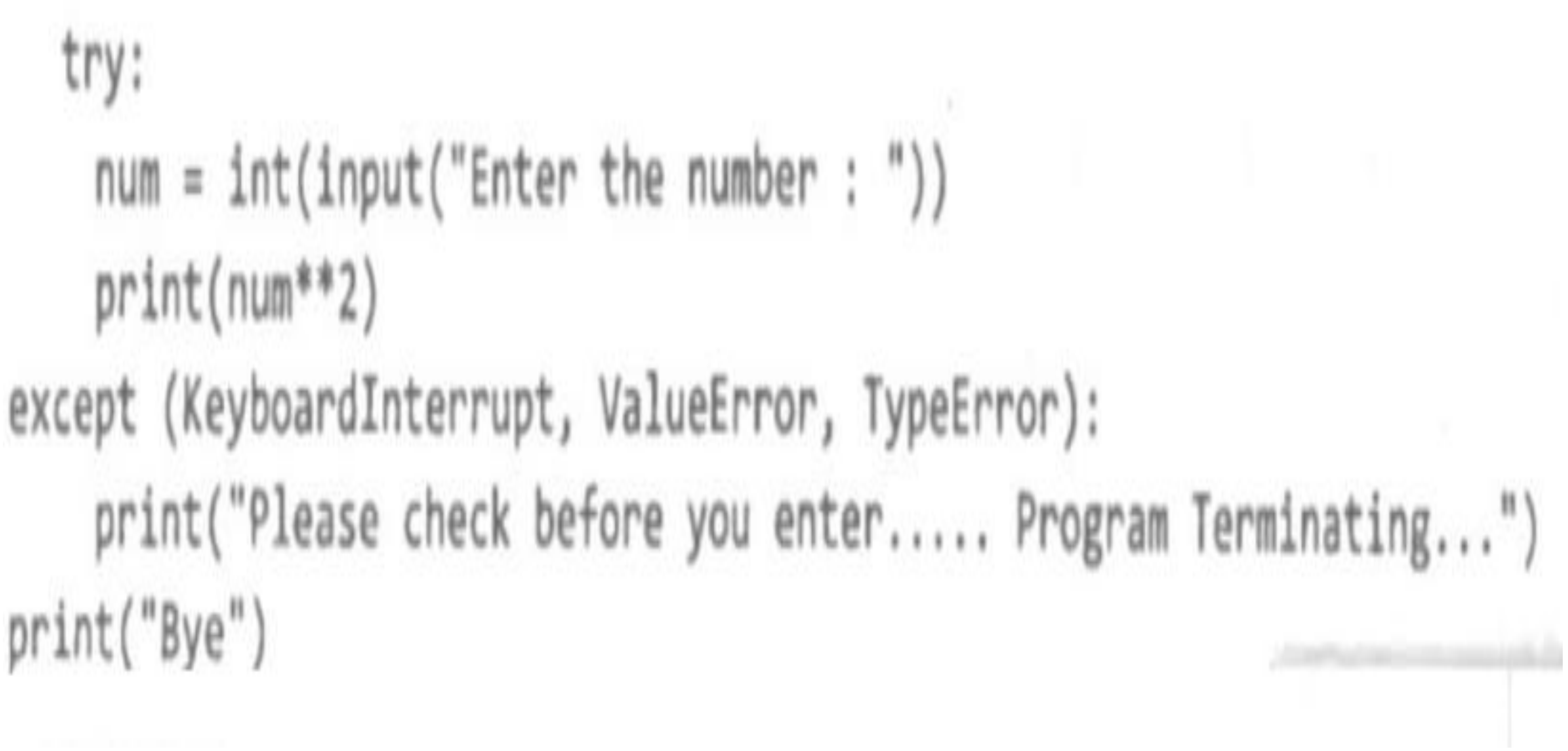
- In Python, you can use multiple except blocks to handle different types of exceptions individually. This allows you to provide specific error handling for each exception type. Here's an example:
- **Syntax:**

```
try:
 ……………….
except Exception 1:
…………….
except Exception 2:
………….
except:
……………………
```

Example:

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
    print("Result:", result)
except ValueError:
    print("Invalid input. Please enter a valid integer.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
except:
   print("An error occurred:")
```

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")
```

Try with Else Clause

- Python also supports the else clause, which should come after every except clause, in the try, and except blocks. Only when the try clause fails to throw an exception the Python interpreter goes on to the else block.
- **Example:**

```
# Defining a function which returns reciprocal of a number
def reciprocal( num1 ):
    try:
        reci = 1 / num1
    except ZeroDivisionError:
       print( "We cannot divide by zero" )
    else:
        print ( reci )
# Calling the function and passing values
reciprocal( 4 )
reciprocal( 0 )
```

Try with Finally Block

- The finally keyword is available in Python, and it is always used after the try-except block. The finally code block is always executed after the try block has terminated normally or after the try block has terminated for some other reason.
- **Here is an example of finally keyword with try-except clauses**:

```
# Raising an exception in try block
try:
    div = 4 // 0
    print( div )
# this block will handle the exception raised
except ZeroDivisionError:
    print( "Atepting to divide by zero" )
# this will always be executed no matter exception is raised or not
finally:
    print( 'This is code of finally clause' )
```

How to 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.
- If we wish to use raise to generate an exception when a given condition happens, we may do so as follows:
- #Python code to show how to raise an exception in Pytho n
- num = [3, 4, 5, 7]
- **if** len(num) > 3:
-     **raise** Exception(f"Length of the given list must be less than or equal to 3 but is {len(num)}")