CHAPTER 16

ERRORS AND EXCEPTIONS

Learning Objectives

After completing this chapter, the reader will be familiar with the following concepts:

- Understand errors and exceptions.
- Understand the standard exceptions
- Understand how exception-handling aids to handle run-time exceptions.

- Learn to extract information from exceptions, raise an exception finally statement, and catch exceptions in programs.
- Learn to handle exceptions such as using except clauses with no exceptions, except clauses to specify exception handlers, and using except clause with multiple exceptions.
- To try-finally clause—learn to use the try statement to delimit the code in which exceptions may occur and use the final clause to release resources.

- Understand the argument of an exception and raise an exception.
- Understand the user-defined exceptions.
- To create program defined exceptions.

16.1 Introduction to Errors and Exceptions

1. Anomalies

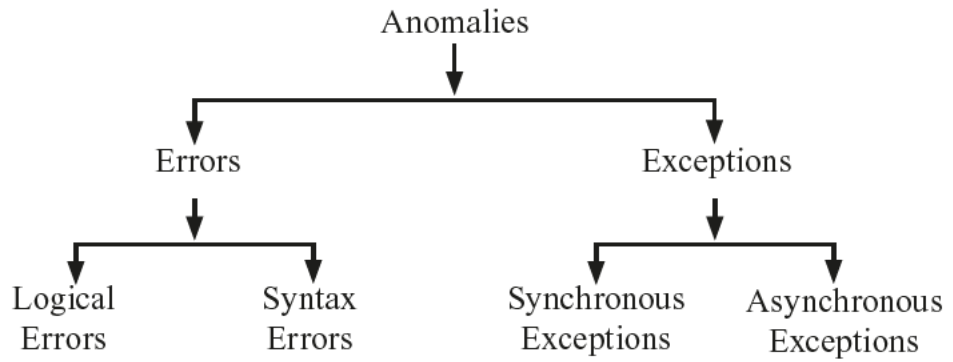


Fig. 16.1: Classification of Anomalies

2. Errors

Types of Errors

Fig. 16.2: Types of Errors

- **Semantic errors:** the semantic error occurs when the program's semantics (meaning) is erroneous, or the statements have no meaning

- **Logical errors:** Logical errors are difficult to trace since they are caused by faulty logical implementation and might happen at any time.

Table 16.1: Correct Logic Versus Logical Errors

Correct Logic	Logical Errors
Go to kindergarten school at the age of 4. Pass schooling up to 12th standard. Enter Engineering degree.	Get admitted into Engineering college at the age of 4. Go to kindergarten school at the age of 8.

Runtime errors: "Runtime errors" occur during an application's execution.

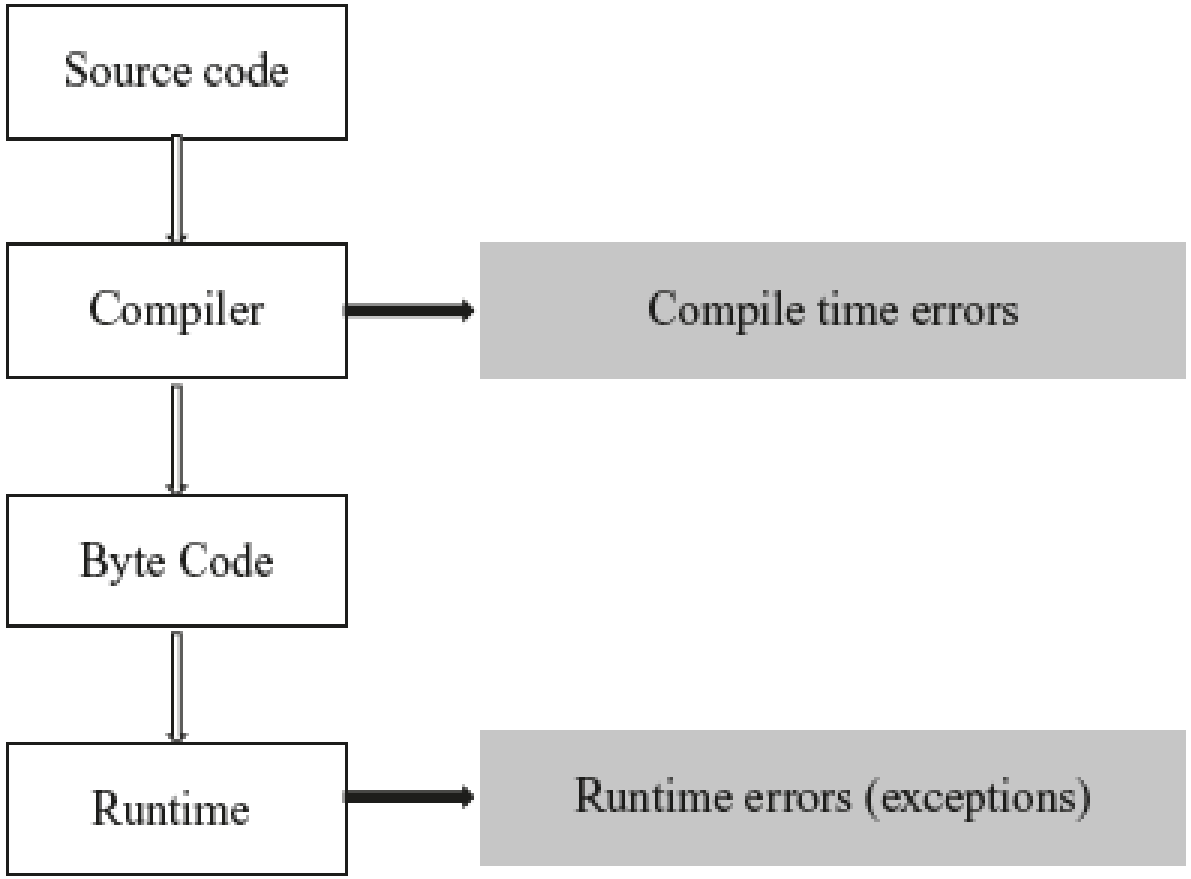


Fig. 16.3: Compile Time Errors and Runtime Errors

Other Types of Errors

- Type errors
- ValueError
- ZeroDivisionError
- FileNotFoundError
- NotImplementedError

Table 16.2: Difference between Syntax Errors and Logic Errors

Syntax Errors	Logic Errors
Syntax errors occur when the programming language rules are violated, such as the word being misspelt.	The Logic errors take work to identify. Although the anticipated conclusion will not be achieved by the code in any way resembling the way it has been written to operate, the code will function correctly.

3. Exception

- **Exceptional conditions:** When a system goes through something that is not typical of the system's behaviour, extenuating factors can be accused of the event.
- ***Exception failures:*** two-thirds of all system breakdowns and 60 per cent of all security holes in computer systems are attributed to exception failures.

Fig. 16.4: Python Exception Handling

Table 16.3: The Difference between an Error and an Exception

Error	Exception
It occurs whenever something in the program fails to work as intended, such as a syntactical error.	Exceptions are the errors that happen when the program is being executed. An occurrence that occurs during the execution of a program and impedes the usual flow of the instructions contained inside that program is referred to as an exception.
It takes place during the compilation process.	When a Python script runs into an error condition that cannot be handled, it will raise an exception and construct an exception object. The script reacts quickly and takes care of the exception. If the software cannot control the situation, it will exit and publish a trace back to the issue, including its location and origin.

16.2 Handling Exceptions

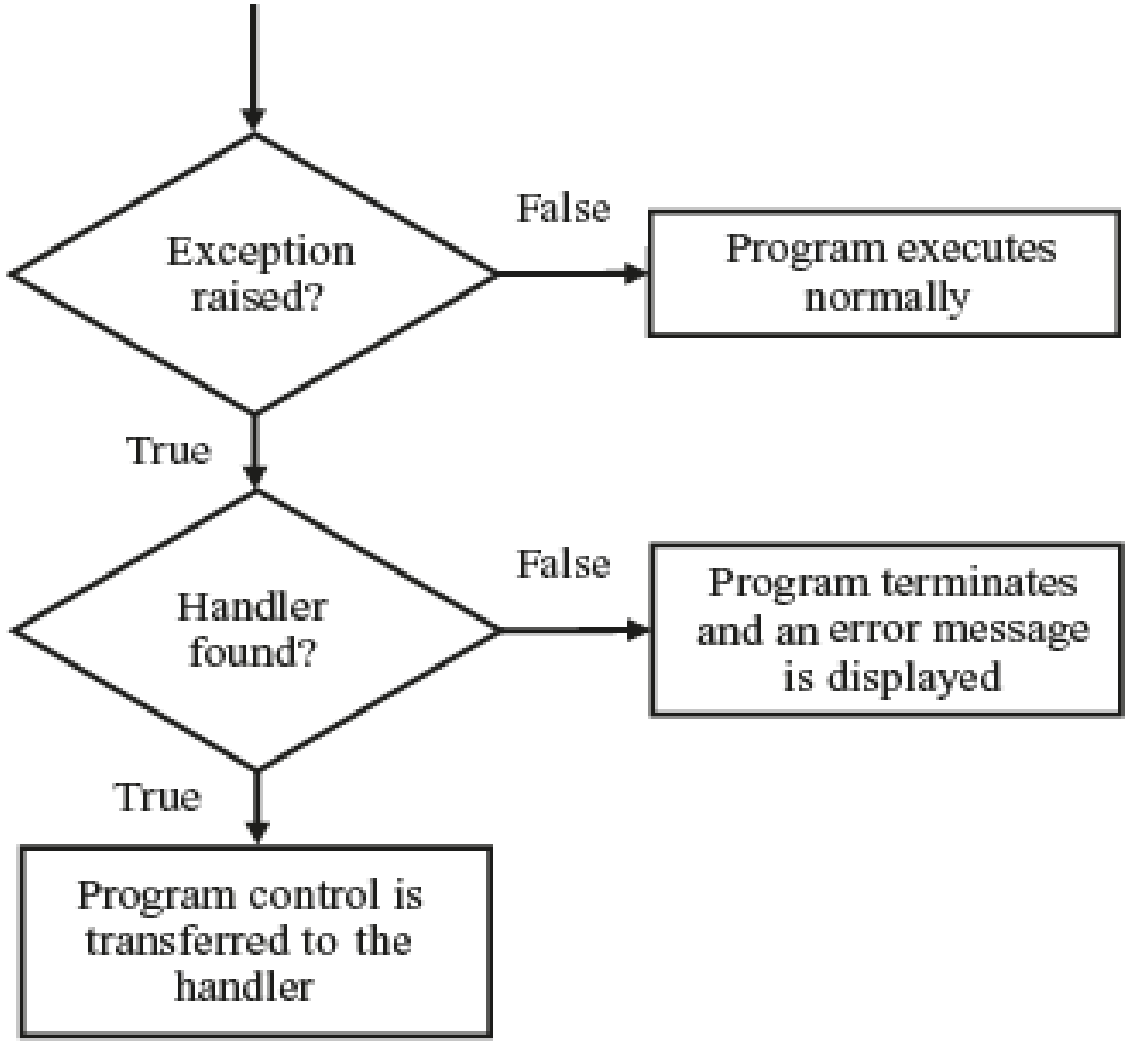


Fig. 16.5: Handling Exceptions

Rule of Exceptions

- If an exception arises but is not always catastrophic, it can be handled.
- The exception clause in a try statement should include a pointer to a specific class.
- A specific object type must exist for an exception to apply.

Listed below are a few examples of an exception.

- An exception based on arithmetic.
- Attempting to access a file that cannot possibly exist.
- Exception to the standard number format.
- If the datatype server is inaccessible, the table can be deleted.
- The combination of two distinct yet incompatible kinds.
- The index of the array is outside the bounds.
- Zero cannot be divided by a number.

Advantages of Handling Exceptions

- The advantages of Handling Exceptions are as follows.
- Decoupling the "Regular" code from the error-handling code.
- Aids in grouping and differentiating error types.
- Sending errors up the call stack when they occur.
- To correct errors that occur during the runtime.
- Use events to communicate the right circumstances without transferring results all over a program by using the notify function.

Benefits of Throwing Exceptions

Throwing and handling the exceptions benefits us from the following.

- Better user experience.
- More helpful error messages for everyone: from users to fellow developers.
- Reliability: to prevent the application from crashing suddenly by avoiding potential problems in advance and taking preventative measures.

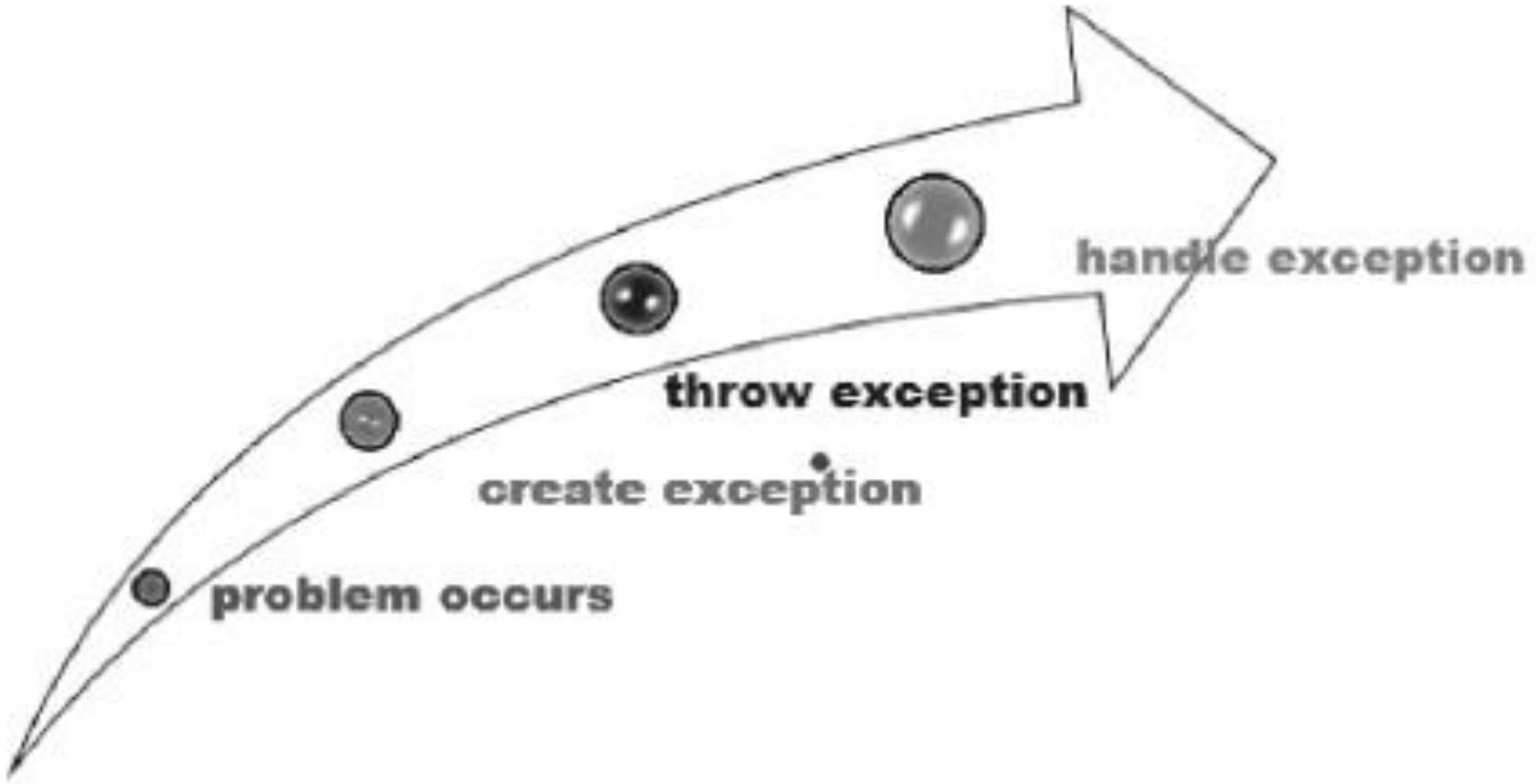


Fig. 16.6: The Sequence of Events in Handling Exceptions

Fig. 16.7: Exception Handling in Python

try - except - else

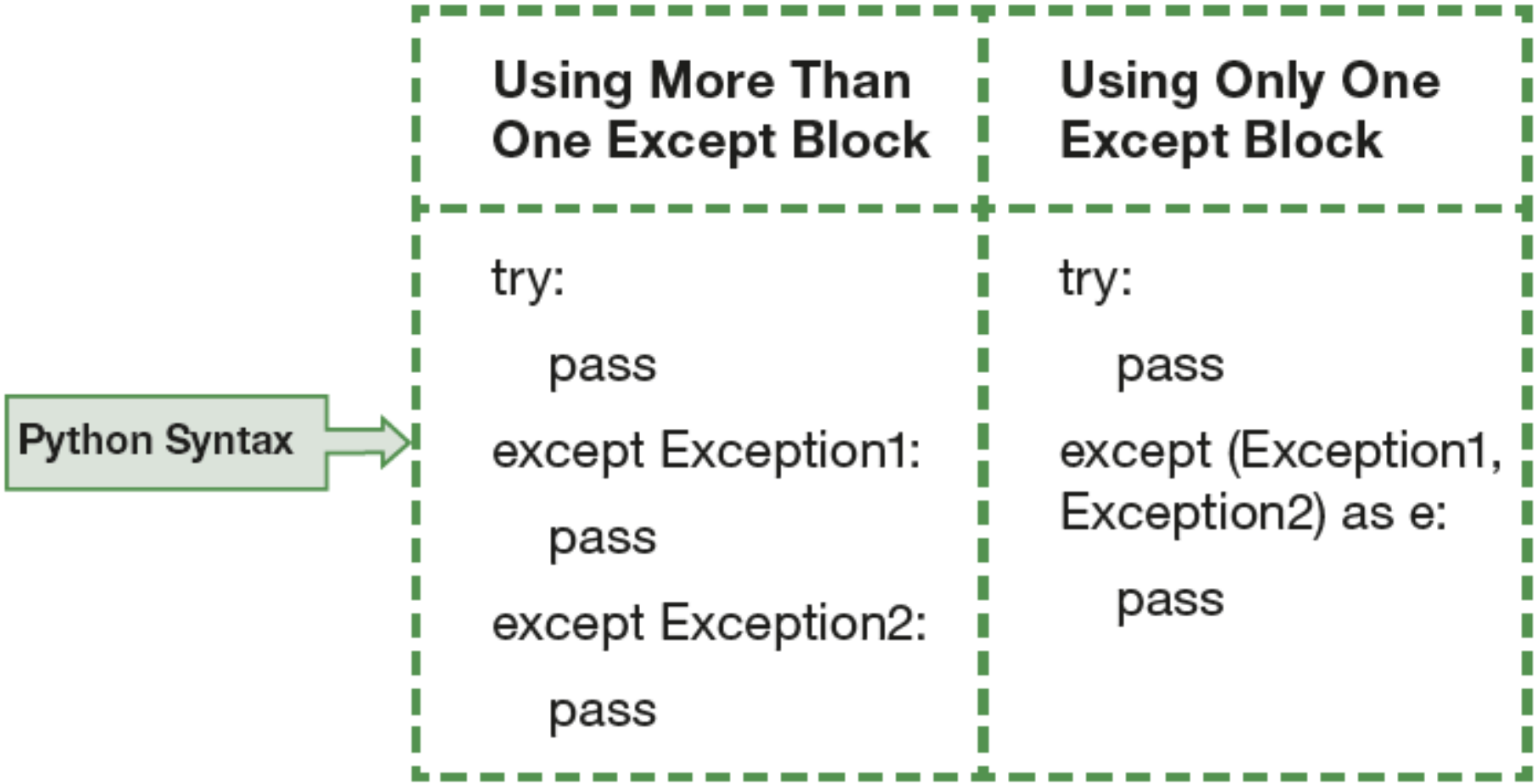
Python Syntax

```
try:
    code
except:
    Code to be executed in case an exception occurs.
else:
    Code to be executed in case an exception does not occur.
```

1. **Using more than one except block**: when many exceptions occur, write exception blocks for each one.
2. **Using only one except block**: a single except clause can be used to include numerous exceptions. Multiple exceptions can be caught in a single except block only if they all have the same behaviour. The program's time can be saved, if the output of several exceptions is the same, which avoids unnecessary code duplication. " Python interpreter will execute code defined within the except clause, if it discovers a matching exception. The following is a generalized syntax for multiple exceptions:

16.3 Multiple Exceptions

Exceptions can be handled in two ways with Python.



16.4 Raising Exceptions

Python's "raise" statement is used to throw an exception explicitly. Using the "raise" statement, an exception object is created, and the expected program execution path is immediately abandoned in search of a matching except clause in the surrounding try statements. If no matching except clause could be located to handle the exception, the raise statement aborts execution and puts an end to the program. Raise will cause an exception to be thrown, so the program's execution will be halted, if it is not handled. The syntax for the raise statement is given by

Where,

- The exception is the name of the exception.
- The "args" is optional and represents the value of the exception argument.
- The final argument, traceback, is optional and, if present, is the traceback object used for the exception.

The message string attaches a meaningful error message to the exception object. Additional attributes are connected to the exception owing to the string. Two types of raise statements exist.

1. raise exceptionClass <, value >

2. raise < exception >

16.5 Exception Chaining

Like Nested if statements, nested loops and nested class, the concept of nested is also extended to try - except as well. Reader could identify that the try block raises the respective exception either by interpreter or user-defined exception raise, when an error is occurring and the except block handles the raised exception. What happens when another exception is raised in the except block? There comes Exception chaining into the picture.

The process of exception chaining involves introducing a new try – except block into a except block or, a new exception is raised inside the except block.

16.6 Built-In Exceptions

Types of Exceptions

Python supports two diverse kinds of exceptions (Fig. 18.11).

- Built-in exceptions
- User-defined exceptions

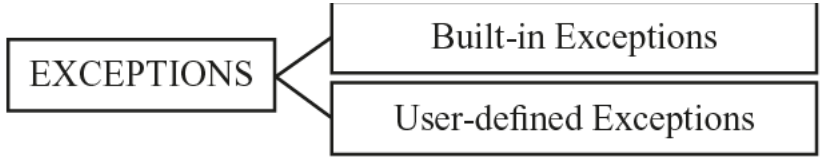


Fig. 16.7: Types of Exceptions

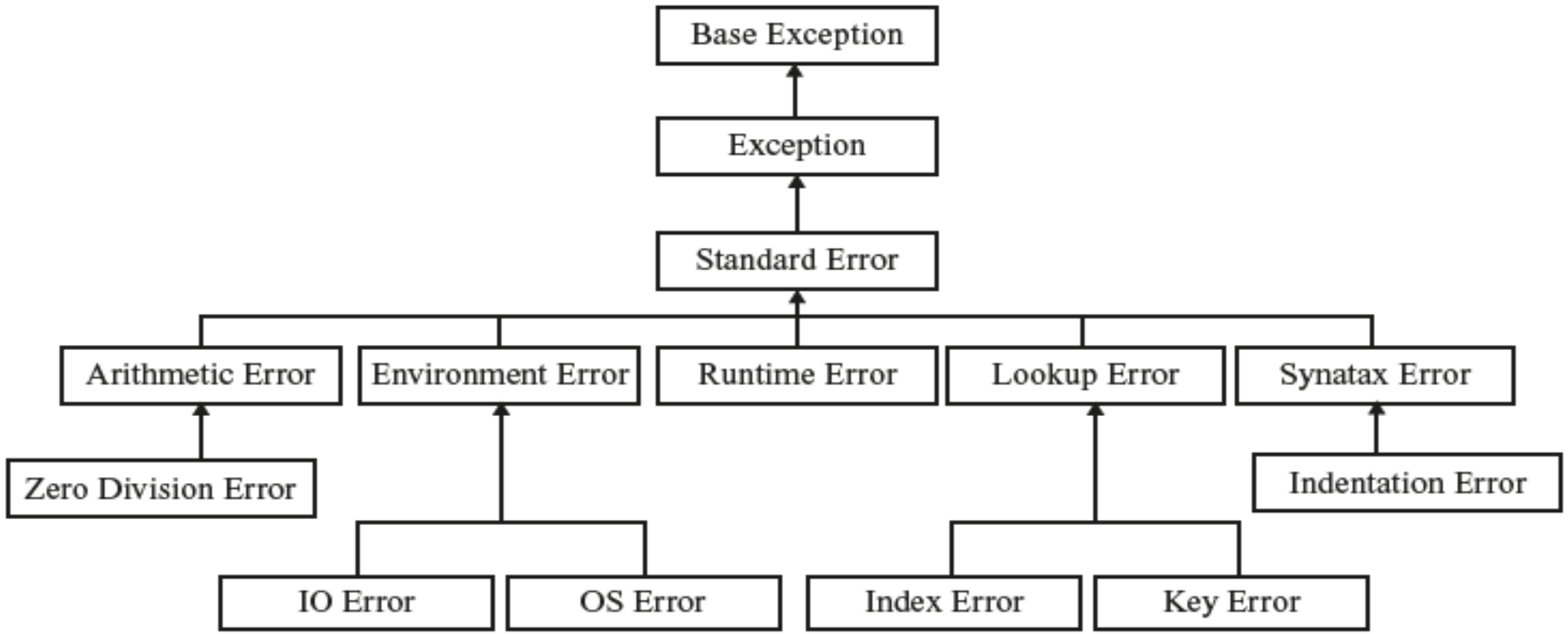


Fig. 16.8: Most Relevant Exception Classes

Table 16.4: Base Class Exception Name

Base Class Exception Name	Description
ArithmeticError	Base class for all errors that occur for numeric calculation. *Arithmetic error exceptions category:* the superclass of Over flowError, ZeroDivisionError, and FloatingPointError, and a subclass of exception.
EnvironmentError	Base class for all exceptions that occur outside the Python environment.
Exception	Base class for all exceptions.
LookupError	Base class for all lookup errors.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.

Table 16.5: Predefined Exception Classes Name

Predefined Exception Classes Name	Description
AssertionError	Raised in case of failure of the assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of the file is reached.
FloatingPointError	Raised when a floating-point calculation fails.
ImportError	Raised when an import statement fails.
IndentationError	Raised when indentation is not specified correctly.
IndexError	Raised when an index is not found in a sequence.
IOError	Raised when an input or output operation fails, such as the print statement or the open () function, when trying to open a file that does not exist. Raised for operating system related errors.

KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
KeyError	It is raised when the specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not implemented.
OverflowError	Raised when a calculation exceeds the maximum limit for a numeric type.
RuntimeError	Raised when a generated error does not fall into any category.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.

Predefined Exception Classes Name	Description
StopIteration	Raised when the following () method of an iterator does not point to any object.
SyntaxError	Raised when there is an error in Python syntax.
SystemError	Raised when the interpreter finds an internal problem, the Python interpreter does not exist when this error is encountered.
SystemExit	Raised when the Python interpreter is quit by using the sys.exit() function. If not handled in the code, it causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted invalid for the specified data type.
UnboundLocalError	Raised when trying to access a local variable in a function or method, but no value has been assigned to it.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.

Table 16.6: Python Built-In Exceptions and Their Classes

Python Built-In Exceptions	Classes
GeneratorExit	Raised if a generator's close() method gets called.
MemoryError	Raised if an operation runs out of memory.
OSError	Raised when system operation causes a system related error.
ReferenceError	Raised when a weak reference proxy accesses a garbage collected reference.
TabError	Raised by inconsistent tabs and spaces.
UnicodeError	Raised when a Unicode related encoding or decoding error occurs.
UnicodeEncodeError	Raised when a Unicode related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode related error occurs during translating.

16.8 Clean-Up Actions

- Code can have three distinct scenarios as follows.
- Code normally runs:
- Error handling is done in the except clause:
- Error without any except clause:

Try-Finally Clause

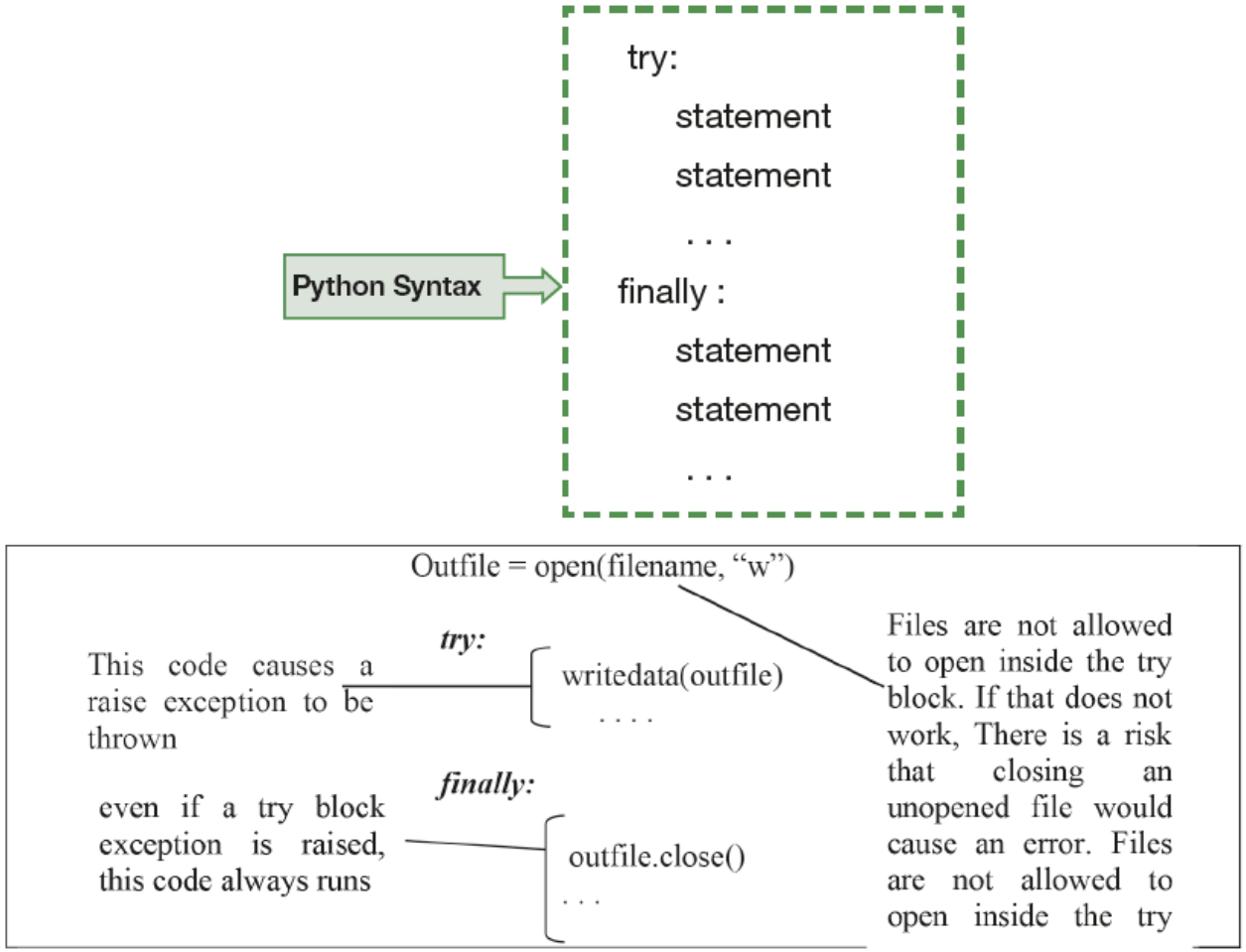


Fig. 16.9: Try Finally Clause