

TITLE: Python Programming: Problem Solving, Packages and Libraries

Edition

Lecture PPT Chapter 16: Python
Exceptions

LEARNING OBJECTIVES

After studying this chapter, you will be able to:

- LO 1** Understand the difference between “syntax error” and “exceptions”.
- LO 2** Use the try-except-else block of code in Python
- LO 3** Understand that in Python 2.x it was possible to derive exceptions from two different classes, i.e., (1) exception class and (2) string class. But in Python 3.x, it is possible to derive exceptions only from the exception class.
- LO 4** Understand the various built-in exceptions in Python such as IOError, EOFError, ZeroDivisionError, ImportError, IndexError, NameError, IndentationError, TypeError and ValueError.

Error vs Exceptions -- 1

There are two distinguishable types of errors:

1. Syntax errors
2. Exceptions

Some noteworthy aspects about errors are as follows:

- Syntax errors are also known as parsing errors.
- They are generated if the rules for syntax are not followed while writing a script in Python.
- Syntax errors are the most basic type of errors.
- They arise when the Python parser is unable to understand a line of code.
- Syntax errors are almost always fatal, i.e., there is almost no way to successfully execute a piece of code containing syntax errors.
- Most syntax errors are typos (i.e., typing errors), incorrect indentation or incorrect arguments.

[.... Continued on next slide]

Error vs Exceptions -- 2

[.... Continued from previous slide]

On the other hand, an exception is

- **An error which happens during the execution of a program or a script.**
- **An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's instructions.**
- **In general, when a Python script encounters a situation that it can't cope with, it raises an exception.**

The raise statement

- In general it is the Python interpreter which “raises” an exception.
- However a programmer can himself raise an exception using the “raise” statement.
- Further an exception can have an argument, which is a value that gives additional information about the problem. The content of the argument vary with exception.
- Syntax for exception statement is as follows (Note that the argument is optional):

```
1 raise [exceptionName[, argument]]
```

The following example script raises an error named `nameError` and gives it an argument -> *'I raised this error'*

```
1 >>>raise NameError('I raised this error')
2 Traceback (most recent call last):
3   File "<pyshell#18>", line 1, in<module>
4 raise NameError('I raised this error')
5 NameError: I raised this error
6 >>>
```

The try-except-else block of code in Python -- 1

- If there is a piece of code which might throw an exception, then the good way to deal with this code would be to enclose it in a try block of code. The try block gives the programmer an opportunity to deal with the error.
- So if you have a try block and you get an error, the interpreter will give you an opportunity to deal with it in the except block. However if the script in the except block is unable to deal with the error, then the Python interpreter will stop the execution of the script and raise an in-built error.
- When you enclose a suspicious code in a try block, there are following two possibilities:
 - Possibility 1 is that it will throw an exception.
 - Possibility 2 is that it will not throw an exception.

[.... Continued on next slide]

The try-except-else block of code in Python -- 1

The following pseudo-code shows how the try -- except -- else blocks of code work.

- The suspicious code (With possible bugs) is put in the “try” block.
- If an exception is thrown in execution of the code in the “try” block, then the code in the “except” block is executed.
- If however no exception is thrown by the code in the “try” block, then the code in the “else” block is executed

```
1  # Pseudo code
2  try:
3      # Code with bugs... could throw exception
4  except(Exception1[, Exception2 [, ... ExceptionN]]):
5      # If any of above exceptions ie Exception1 to
6      # ... ExceptionN occur, execute this block
7  else:
8      # If there is no exception in the list of exceptions,
9  then execute this code
```

The try-except-else block of code in Python -- 2

The following points about the above pseudo-code relating to a try-except-else block are noteworthy:

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception (This means that if you have an except block without giving any exception type, then it will catch all exceptions).
- After the except clause(s), you can include an else-clause (The else block is optional). The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.
- However, if there is no exception, then the else block will be executed.

The try-except-else block of code in Python -- 3

The following figure shows how the try – except – lese blocks of code work

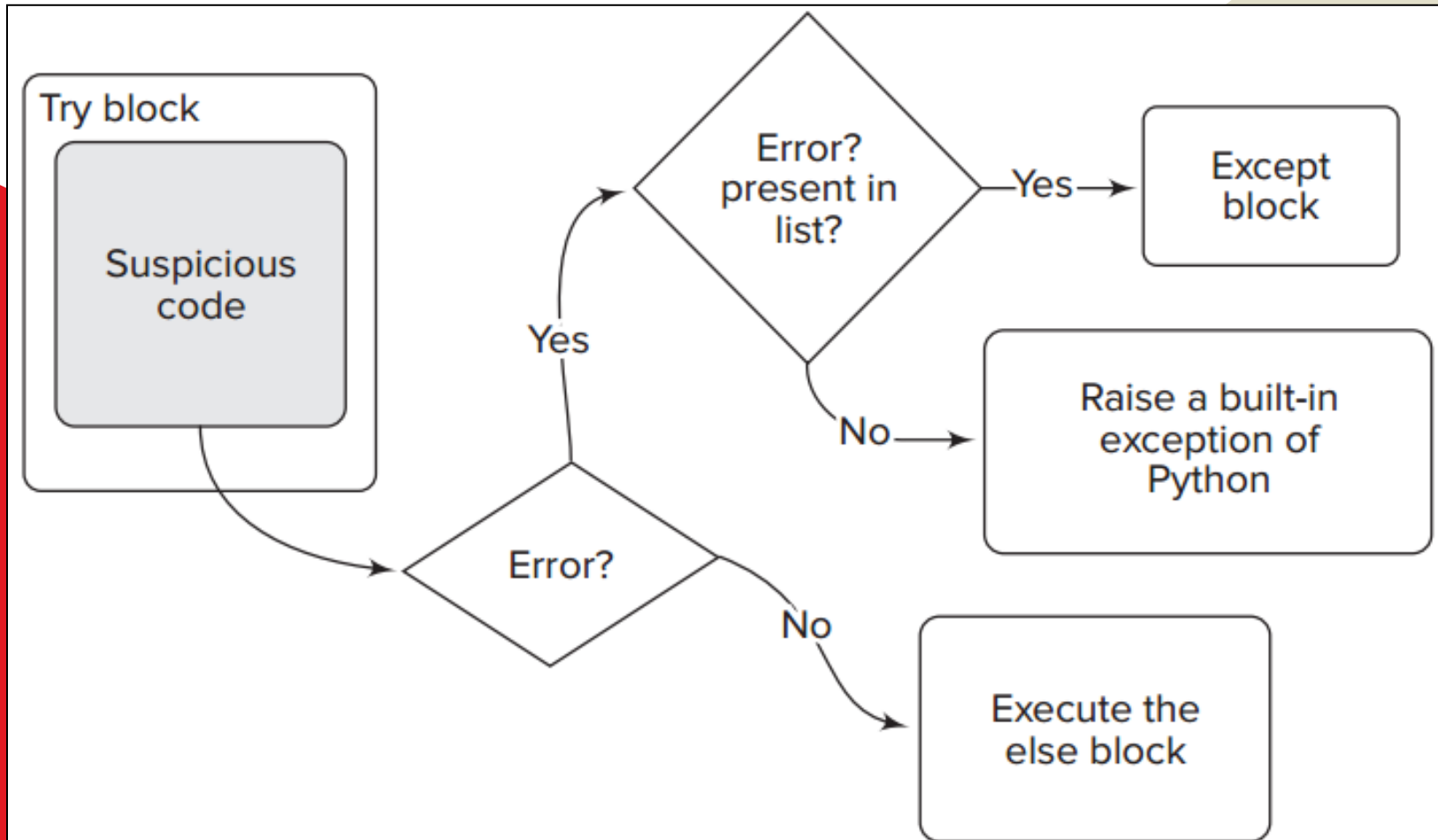


FIGURE 16.1 The “flow” of execution in a “try-except-else” block

The try-except-else-finally block with multiple except -- 1

- Note that when you are writing the except block you have the following three options:
 - Option 1 you write one type of exception.
 - Option 2 you write more than one types of exception.
 - Option 3 you do not write any type of exception.
- So for except block with one type of exception, the block is executed only if that type of exception is raised.
- For except block with multiple exceptions, the except block is executed if any of the exceptions in the exceptions given is raised. So if an exception which is not given in the list of exceptions, then it will not be raised.
- For except block with no exception, the block is executed if there is any type of exception.

The try-except-else-finally block with multiple except -- 2

- The following example code from book has a function which takes 4 parameters; a list -> myL, an -> index idx, a dividend -> dividend and a divisor -> divisor.
- The function has a try block where 2 types of errors namely IndexError and ZeroDivisionError may be raised.
- It has an “except” block which is executed only if no errors are raised.
- The function also has a “finally” block which is executed irrespective of whether an exception is raised or not.
- You can try out this script by giving such values to a function call so that errors are raised and check which of the blocks are executed. You can know the block executed by the print output.

```
1 def myFunc(myL, idx, dividend, divisor):
2     try:
3         print('If this is printed, index OK->',myL[idx])
4         print('If this printed, divisor not 0->',dividend/ divisor)
5     except IndexError:
6         print('Index is out of range')
7     except ZeroDivisionError:
8         print(' Cant divide by 0')
9     else:
10        print('No exception raised')
11    finally:
12        print('Exception or not, this will be printed')
```

The try-except-else-finally block with multiple except -- 3

The following figure shows the flow of execution in a “try – except -- else -- finally” block

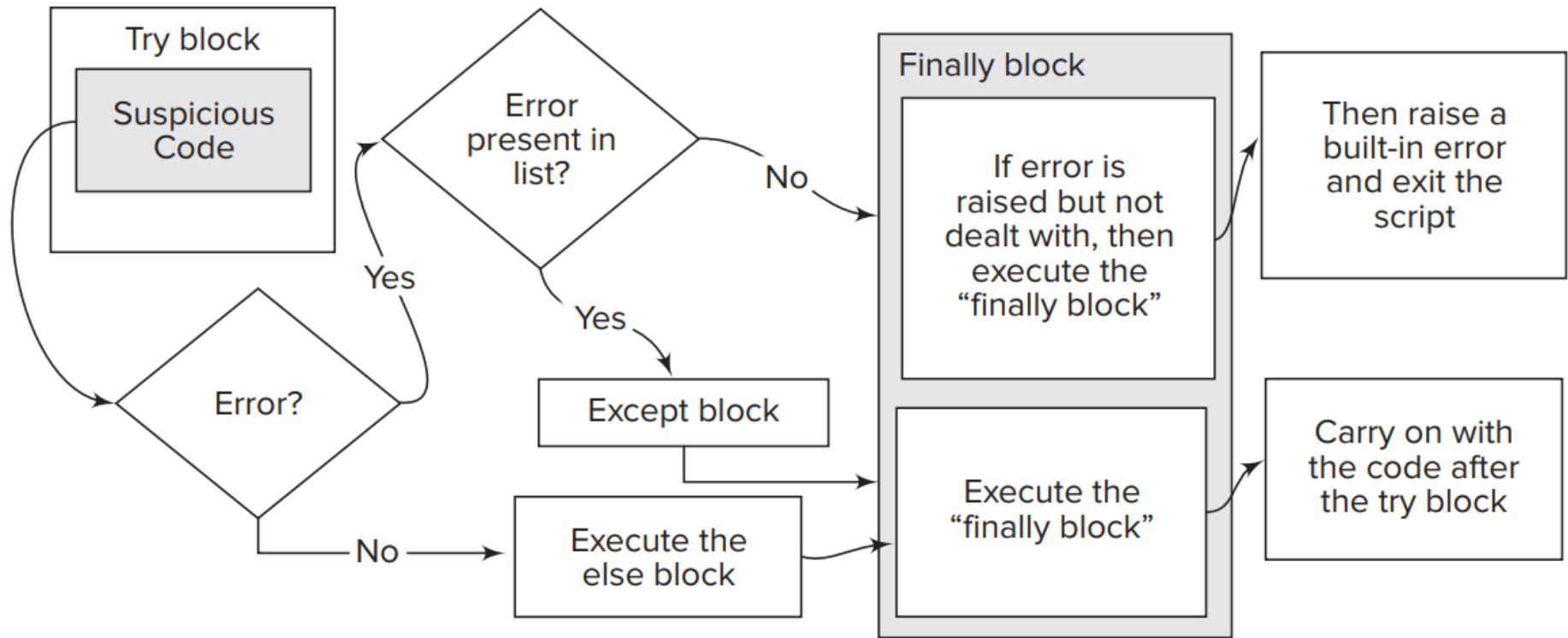


FIGURE 16.2 “Flow of execution” in a “try-except-else-finally” block

User-defined exceptions

- Sometimes a user needs to create his own exceptions. Python allows the user to derive his own exceptions.
- In older Python versions, there were following two ways in which exceptions could be derived.
 1. One version of exceptions was derived from exception class.
 2. other from the string class.
- But from Python 2.6 onwards, it is possible to derive exceptions only from exception class and not from string class.
- What does deriving exception from class or exception class or string class mean?
 - Note that Python being an object-oriented programming language, everything in it is a class or an object.
 - This is true for exceptions also. So exceptions when defined are classes and when used are objects.
 - Further exceptions are inherited from some base class. In earlier versions of Python, the exception class (and therefore objects) could be inherited both from the exception base class or the string class.
 - However, version 2.6 onwards, it is possible to derive exceptions only from the exception base class.

Built-in exceptions -- 1

Some built-in exceptions in Python are shown in Table

Sl. No	Error type	Error description
1	IOError	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”.
2	EOFError	Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data. (Note that the file.read() and file.readline() methods return an empty string when they hit EOF.)
3	ZeroDivisionError	Raised when the second argument of a division or modulo operation is zero.
4	ImportError	Raised when an import statement fails to find the module definition or when a from ... import fails to find a name that is to be imported.
5	IndexError	Raised when a sequence subscript is out of range. (Slice indices are silently truncated to fall in the allowed range; if an index is not a plain integer, TypeError is raised.)

[..... Continued on next slide]

Built-in exceptions -- 2

[Continued from previous slide.....]

Sl. No	Error type	Error description
6	NameError	Raised when a local or global name is not found.
7	IndentationError	Syntax errors related to incorrect indentation.
8	TypeError	Raised when an operation or function is applied to an object of inappropriate type
9	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.

An except clause may name multiple exceptions as a parenthesized tuple

From Python 3.x onwards, you may raise a number of exceptions with the same except statement. But the exceptions must be in the form of a “parenthesized tuple”.

Suppose you have some errors, say `RuntimeError`, `TypeError` and `NameError`, then you could raise these three errors in a single except as a “tuple of errors” as follows:

```
1 except (RuntimeError, TypeError, NameError):  
2     pass
```


The topics of “Beyond Text” and “Assignment” sections

The “Beyond Text” section has 2 very interesting and important topics:

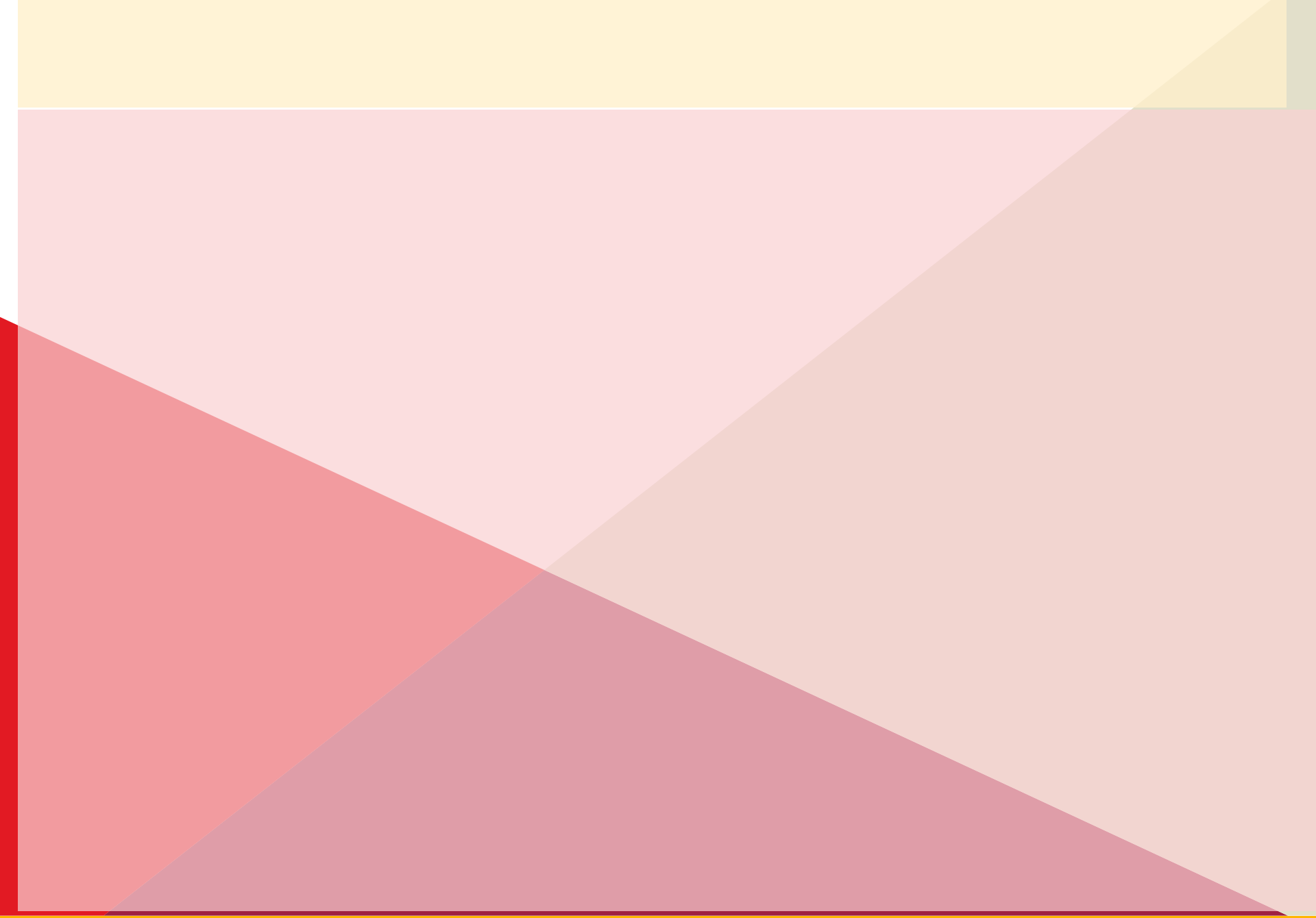
The exception hierarchy

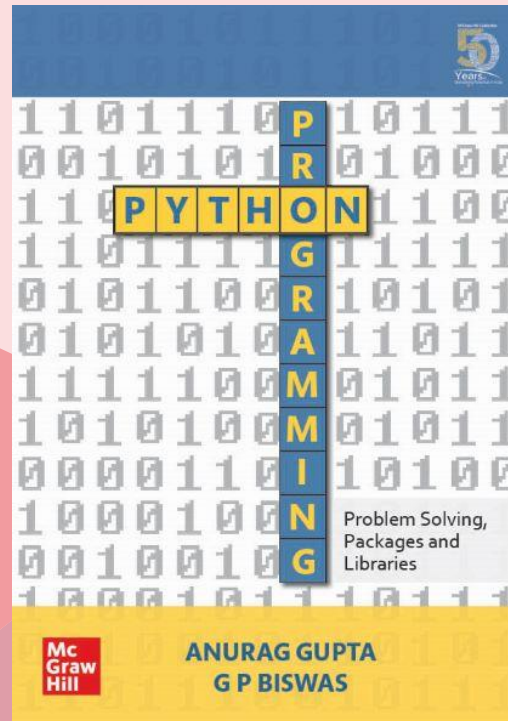
The exception object

These 2 topics are worth studying

The “Assignment” section also has a very interesting topic:

- studying the traceback module








Because learning changes everything.®

Thank You!

For any queries or feedback contact us at:

 support.india@mheducation.com

 1800-103-5875

 www.mheducation.co.in

in

