Python Essentials 2:
Module 5
Modules, packages string and list methods, and exceptions

In this module, you will learn about:

Python modules: their rationale, function, how to import them in different ways, and present the content of some standard modules provided by Python;
the way in which modules are coupled together to make packages.
the concept of an exception and Python's implementation of it, including the try-except instruction, with its applications, and the raise instruction.
strings and their specific methods, together with their similarities and differences compared to lists.

(part 11)

Built-in exceptions

We're going to show you a short list of the most useful exceptions. While it may sound strange to call "useful" a thing or a phenomenon which is a visible sign of failure or setback, as you know, to err is human and if anything can go wrong, it will go wrong.

Exceptions are as routine and normal as any other aspect of a programmer's life.

For each exception, we'll show you:

- its name;
- its location in the exception tree;
- a short description;
- a concise snippet of code showing the circumstances in which the exception may be raised.

There are lots of other exceptions to explore - we simply don't have the space to go through them all here.

ArithmeticError
Location: BaseException ← Exception ← ArithmeticError

Description: an abstract exception including all exceptions caused by arithmetic operations like zero division or an argument's invalid domain

AssertionError
Location: BaseException ← Exception ← AssertionError

Description: a concrete exception raised by the assert instruction when its argument evaluates to False, None, 0, or an empty string

Code:

```
from math import tan, radians
angle = int(input('Enter integral angle in degrees: '))

# We must be sure that angle != 90 + k * 180
assert angle % 180 != 90
print(tan(radians(angle)))
```

BaseException
Location: BaseException

Description: the most general (abstract) of all Python exceptions - all other exceptions are included in this one; it can be said that the following two except branches are equivalent: except: and except BaseException:.

IndexError
Location: BaseException ← Exception ← LookupError ← IndexError

Description: a concrete exception raised when you try to access a non-existent sequence's element (e.g., a list's element)

Code:

```
# The code shows an extravagant way
# of leaving the loop.

the_list = [1, 2, 3, 4, 5]
ix = 0
do_it = True

while do_it:
    try:
        print(the_list[ix])
        ix += 1
    except IndexError:
        do_it = False

print('Done')
```

KeyboardInterrupt

Location: BaseException ← KeyboardInterrupt

Description: a concrete exception raised when the user uses a keyboard shortcut designed to terminate a program's execution (Ctrl-C in most OSs); if handling this exception doesn't lead to program termination, the program continues its execution.

Note: this exception is not derived from the Exception class. Run the program in IDLE.

Code:

```
# This code cannot be terminated
# by pressing Ctrl-C.

from time import sleep

seconds = 0

while True:
    try:
        print(seconds)
        seconds += 1
        sleep(1)
    except KeyboardInterrupt:
        print("Don't do that!")
```

LookupError

Location: BaseException ← Exception ← LookupError

Description: an abstract exception including all exceptions caused by errors resulting from invalid references to different collections (lists, dictionaries, tuples, etc.)

MemoryError

Location: BaseException ← Exception ← MemoryError

Description: a concrete exception raised when an operation cannot be completed due to a lack of free memory.

Code:

```
# This code causes the MemoryError exception.
# Warning: executing this code may affect your OS.
# Don't run it in production environments!

string = 'x'
try:
    while True:
        string = string + string
        print(len(string))
except MemoryError:
    print('This is not funny!')
```

OverflowError

Location: BaseException ← Exception ← ArithmeticError ← OverflowError

Description: a concrete exception raised when an operation produces a number too big to be successfully stored

Code:

```
# The code prints subsequent
# values of exp(k), k = 1, 2, 4, 8, 16, ...

from math import exp

ex = 1

try:
    while True:
        print(exp(ex))
        ex *= 2
except OverflowError:
    print('The number is too big.')
```

ImportError

Location: BaseException ← Exception ← StandardError ← ImportError

Description: a concrete exception raised when an import operation fails

Code:

```
# One of these imports will fail - which one?

try:
    import math
    import time
    import abracadabra

except:
    print('One of your imports has failed.')
```

KeyError

Location: BaseException ← Exception ← LookupError ← KeyError

Description: a concrete exception raised when you try to access a collection's non-existent element (e.g., a dictionary's element)

Code:

```
# How to abuse the dictionary
# and how to deal with it?

dictionary = { 'a': 'b', 'b': 'c', 'c': 'd' }
ch = 'a'

try:
    while True:
        ch = dictionary[ch]
        print(ch)
except KeyError:
    print('No such key:', ch)
```

We are done with exceptions for now, but they'll return when we discuss object-oriented programming in Python. You can use them to protect your code from bad accidents, but you also have to learn how to dive into them, exploring the information they carry.

Exceptions are in fact objects - however, we can tell you nothing about this aspect until we present you with classes, objects, and the like.

For the time being, if you'd like to learn more about exceptions on your own, you look into Standard Python Library at https://docs.python.org/3.6/library/exceptions.html.

LAB

Estimated time
15-25 minutes

Level of difficulty
Medium

Objectives
improving the student's skills in defining functions;
using exceptions in order to provide a safe input environment.
Scenario
Your task is to write a function able to input integer values and to check if they are within a specified range.

The function should:

- accept three arguments: a prompt, a low acceptable limit, and a high acceptable limit;
- if the user enters a string that is not an integer value, the function should emit the message Error: wrong input, and ask the user to input the value again;
- if the user enters a number which falls outside the specified range, the function should emit the message Error: the value is not within permitted range (min..max) and ask the user to input the value again;
- if the input value is valid, return it as a result.

Test data
Test your code carefully.

```
def read_int(prompt, min, max):
    #
    # Write your code here.
    #


v = read_int("Enter a number from -10 to 10: ", -10, 10)

print("The number is:", v)
```

This is how the function should react to the user's input:

```
Enter a number from -10 to 10: 100
Error: the value is not within permitted range (-10..10)
Enter a number from -10 to 10: asd
Error: wrong input
Enter number from -10 to 10: 1
The number is: 1
```

Key takeaways

1. Some abstract built-in Python exceptions are:

    ArithmeticError,
    BaseException,
    LookupError.

2. Some concrete built-in Python exceptions are:

    AssertionError,
    ImportError,
    IndexError,
    KeyboardInterrupt,
    KeyError,
    MemoryError,
    OverflowError.

Exercise 1

Which of the exceptions will you use to protect your code from being interrupted through the use of the keyboard?

Exercise 2

What is the name of the most general of all Python exceptions?

Exercise 3

Which of the exceptions will be raised through the following unsuccessful evaluation?

```
huge_value = 1E250 ** 2
```

Congratulations! You have completed PE2: Module 5.

Well done! You've reached the end of Module 5 and completed a major milestone in your Python programming education. Here's a short summary of the objectives you've covered and got familiar with in Module 5:

working with Python modules; importing, creating, and using modules;
using selected Python STL modules (math, random, and platform)
constructing and using packages in Python;
characters, strings, and coding standards;
the nature of strings in Python; strings vs. lists - similarities and differences;
list and string methods;
handling errors in Python;
controlling the flow of errors using try and except;
the hierarchy of exceptions; review of the most useful exceptions.

You are now ready to take the module quiz and attempt the final challenge: Module 5 Test, which will help you gauge what you've learned so far.