Python Essentials 1:
Module 3
Boolean Values, Conditional Execution, Loops, Lists and List Processing, Logical and Bitwise Operations

In this module, you will cover the following topics:

the Boolean data type;
relational operators;
making decisions in Python (if, if-else, if-elif,else)
how to repeat code execution using loops (while, for)
how to perform logic and bitwise operations in Python;
lists in Python (constructing, indexing, and slicing; content manipulation)
how to sort a list using bubble-sort algorithms;
multidimensional lists and their applications.

(part 6)

The inner life of lists

Now we want to show you one important, and very surprising, feature of lists, which strongly distinguishes them from ordinary variables.

We want you to memorize it - it may affect your future programs, and cause severe problems if forgotten or overlooked.

Take a look at the snippet in the editor.

```
list_1 = [1]
list_2 = list_1
list_1[0] = 2
print(list_2)
```

The program:

creates a one-element list named list_1;
assigns it to a new list named list_2;
changes the only element of list_1;
prints out list_2.

The surprising part is the fact that the program will output: [2], not [1], which seems to be the obvious solution.

Lists (and many other complex Python entities) are stored in different ways than ordinary (scalar) variables.

You could say that:

the name of an ordinary variable is the name of its content;
the name of a list is the name of a memory location where the list is stored.

Read these two lines once more - the difference is essential for understanding what we are going to talk about next.

The assignment: list_2 = list_1 copies the name of the array, not its contents. In effect, the two names (list_1 and list_2) identify the same location in the computer memory. Modifying one of them affects the other, and vice versa.

How do you cope with that?

Powerful slices

Fortunately, the solution is at your fingertips - its name is the slice.

A slice is an element of Python syntax that allows you to make a brand new copy of a list, or parts of a list.

It actually copies the list's contents, not the list's name.

This is exactly what you need. Take a look at the snippet below:

```
list_1 = [1]
list_2 = list_1[:]
list_1[0] = 2
print(list_2)
```

Its output is [1].

This inconspicuous part of the code described as [:] is able to produce a brand new list.

One of the most general forms of the slice looks as follows:

```
my_list[start:end]
```

As you can see, it resembles indexing, but the colon inside makes a big difference.

A slice of this form makes a new (target) list, taking elements from the source list - the elements of the indices from start to end - 1.

Note: not to end but to end - 1. An element with an index equal to end is the first element which does not take part in the slicing.

Using negative values for both start and end is possible (just like in indexing).

Take a look at the snippet:

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[1:3]
print(new_list)
```

The new_list list will have end - start (3 - 1 = 2) elements - the ones with indices equal to 1 and 2 (but not 3).

The snippet's output is: [8, 6]

Slices - negative indices

Look at the snippet below:

```
my_list[start:end]
```

To repeat:

start is the index of the first element included in the slice;
end is the index of the first element not included in the slice.

This is how negative indices work with the slice:

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[1:-1]
print(new_list)
```

The snippet's output is:

[8, 6, 4]
output

If the start specifies an element lying further than the one described by the end (from the list's beginning point of view), the slice will be empty:

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[-1:1]
print(new_list)
```

The snippet's output is:

[]

Slices: continued

If you omit the start in your slice, it is assumed that you want to get a slice beginning at the element with index 0.

In other words, the slice of this form:

```
my_list[:end]
```

is a more compact equivalent of:

```
my_list[0:end]
```

Look at the snippet below:

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[:3]
print(new_list)
```

This is why its output is: [10, 8, 6].

Similarly, if you omit the end in your slice, it is assumed that you want the slice to end at the element with the index len(my_list).

In other words, the slice of this form:

```
my_list[start:]
```

is a more compact equivalent of:

```
my_list[start:len(my_list)]
```

Look at the following snippet:

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[3:]
print(new_list)
```

Its output is therefore: [4, 2].

Slices: continued

As we've said before, omitting both start and end makes a copy of the whole list:

```
my_list = [10, 8, 6, 4, 2]
new_list = my_list[:]
print(new_list)
```

The snippet's output is: [10, 8, 6, 4, 2].

The previously described del instruction is able to delete more than just a list's element at once - it can delete slices too:

```
my_list = [10, 8, 6, 4, 2]
del my_list[1:3]
print(my_list)
```

Note: in this case, the slice doesn't produce any new list!

The snippet's output is: [10, 4, 2].

Deleting all the elements at once is possible too:

```
my_list = [10, 8, 6, 4, 2]
del my_list[:]
print(my_list)
```

The list becomes empty, and the output is: [].

Removing the slice from the code changes its meaning dramatically.

Take a look:

```
my_list = [10, 8, 6, 4, 2]
del my_list
print(my_list)
```

The del instruction will delete the list itself, not its content.

The print() function invocation from the last line of the code will then cause a runtime error.

The in and not in operators

Python offers two very powerful operators, able to look through the list in order to check whether a specific value is stored inside the list or not.

These operators are:

```
elem in my_list
elem not in my_list
```

The first of them (in) checks if a given element (its left argument) is currently stored somewhere inside the list (the right argument) - the operator returns True in this case.

The second (not in) checks if a given element (its left argument) is absent in a list - the operator returns True in this case.

Look at the code in the editor. The snippet shows both operators in action. Can you guess its output? Run the program to check if you were right.

```
my_list = [0, 3, 12, 8, 2]

print(5 in my_list)
print(5 not in my_list)
print(12 in my_list)
```

Lists - some simple programs

Now we want to show you some simple programs utilizing lists.

The first of them tries to find the greater value in the list. Look at the code in the editor.

The concept is rather simple - we temporarily assume that the first element is the largest one, and check the hypothesis against all the remaining elements in the list.

The code outputs 17 (as expected).

The code may be rewritten to make use of the newly introduced form of the for loop:

```
my_list = [17, 3, 11, 5, 1, 9, 7, 15, 13]
largest = my_list[0]

for i in my_list:
    if i > largest:
        largest = i

print(largest)
```

The program above performs one unnecessary comparison, when the first element is compared with itself, but this isn't a problem at all.

The code outputs 17, too (nothing unusual).

If you need to save computer power, you can use a slice:

```
my_list = [17, 3, 11, 5, 1, 9, 7, 15, 13]
largest = my_list[0]

for i in my_list[1:]:
    if i > largest:
        largest = i

print(largest)
```

The question is: which of these two actions consumes more computer resources - just one comparison, or slicing almost all of a list's elements?

Lists - some simple programs

Now let's find the location of a given element inside a list:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
to_find = 5
found = False

for i in range(len(my_list)):
    found = my_list[i] == to_find
    if found:
        break

if found:
    print("Element found at index", i)
else:
    print("absent")
```

Note:

- the target value is stored in the to_find variable;
- the current status of the search is stored in the found variable (True/False)
- when found becomes True, the for loop is exited.

Let's assume that you've chosen the following numbers in the lottery: 3, 7, 11, 42, 34, 49.

The numbers that have been drawn are: 5, 11, 9, 42, 3, 49.

The question is: how many numbers have you hit?

The program will give you the answer:

```
drawn = [5, 11, 9, 42, 3, 49]
bets = [3, 7, 11, 42, 34, 49]
hits = 0

for number in bets:
    if number in drawn:
        hits += 1

print(hits)
```

Note:

- the drawn list stores all the drawn numbers;
- the bets list stores your bets;
- the hits variable counts your hits.

The program output is: 4.

LAB

Estimated time
10-15 minutes

Level of difficulty
Easy

Objectives
Familiarize the student with:

list indexing;
utilizing the in and not in operators.
Scenario
Imagine a list - not very long, not very complicated, just a simple list containing some integer numbers. Some of these numbers may be repeated, and this is the clue. We don't want any repetitions. We want them to be removed.

Your task is to write a program which removes all the number repetitions from the list. The goal is to have a list in which all the numbers appear not more than once.

Note: assume that the source list is hard-coded inside the code - you don't have to enter it from the keyboard. Of course, you can improve the code and add a part that can carry out a conversation with the user and obtain all the data from her/him.

Hint: we encourage you to create a new list as a temporary work area - you don't need to update the list in situ.

We've provided no test data, as that would be too easy. You can use our skeleton instead.

```
my_list = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]
#
# Write your code here.
#
print("The list with unique elements only:")
print(my_list)
```

Key takeaways

1. If you have a list l1, then the following assignment: l2 = l1 does not make a copy of the l1 list, but makes the variables l1 and l2 point to one and the same list in memory. For example:

```
vehicles_one = ['car', 'bicycle', 'motor']
print(vehicles_one) # outputs: ['car', 'bicycle', 'motor']

vehicles_two = vehicles_one
del vehicles_one[0] # deletes 'car'
print(vehicles_two) # outputs: ['bicycle', 'motor']
```

2. If you want to copy a list or part of the list, you can do it by performing slicing:

```
colors = ['red', 'green', 'orange']

copy_whole_colors = colors[:]  # copy the entire list
copy_part_colors = colors[0:2]  # copy part of the list
```

3. You can use negative indices to perform slices, too. For example:

```
sample_list = ["A", "B", "C", "D", "E"]
new_list = sample_list[2:-1]
print(new_list)  # outputs: ['C', 'D']
```

4. The start and end parameters are optional when performing a slice: list[start:end], e.g.:

```
my_list = [1, 2, 3, 4, 5]
slice_one = my_list[2: ]
slice_two = my_list[ :2]
slice_three = my_list[-2: ]

print(slice_one)  # outputs: [3, 4, 5]
print(slice_two)  # outputs: [1, 2]
print(slice_three)  # outputs: [4, 5]
```

5. You can delete slices using the del instruction:

```
my_list = [1, 2, 3, 4, 5]
del my_list[0:2]
print(my_list)  # outputs: [3, 4, 5]

del my_list[:]
print(my_list)  # deletes the list content, outputs: []
```

6. You can test if some items exist in a list or not using the keywords in and not in, e.g.:

```
my_list = ["A", "B", 1, 2]

print("A" in my_list)  # outputs: True
print("C" not in my_list)  # outputs: True
print(2 not in my_list)  # outputs: False
```

Exercise 1

What is the output of the following snippet?

```
list_1 = ["A", "B", "C"]
list_2 = list_1
list_3 = list_2

del list_1[0]
del list_2[0]

print(list_3)
```

Exercise 2

What is the output of the following snippet?

```
list_1 = ["A", "B", "C"]
list_2 = list_1
list_3 = list_2

del list_1[0]
del list_2

print(list_3)
```

Exercise 3

What is the output of the following snippet?

```
list_1 = ["A", "B", "C"]
list_2 = list_1
list_3 = list_2

del list_1[0]
del list_2[:]

print(list_3)
```

Exercise 4

What is the output of the following snippet?

```
list_1 = ["A", "B", "C"]
list_2 = list_1[:]
list_3 = list_2[:]

del list_1[0]
del list_2[0]

print(list_3)
```

Exercise 5

Insert in or not in instead of ??? so that the code outputs the expected result.

```
my_list = [1, 2, "in", True, "ABC"]

print(1 ??? my_list)  # outputs True
print("A" ??? my_list)  # outputs True
print(3 ??? my_list)  # outputs True
print(False ??? my_list)  # outputs False
```