

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 7)

Comparing strings

Python's strings can be compared using the same set of operators which are in use in relation to numbers.

Take a look at these operators - they can all compare strings, too:

```
==  
!=  
>  
>=  
<  
<=
```

There is one "but" - the results of such comparisons may sometimes be a bit surprising. Don't forget that Python is not aware (it cannot be in any way) of subtle linguistic issues - it just compares code point values, character by character.

The results you get from such an operation are sometimes astonishing. Let's start with the simplest cases.

Two strings are equal when they consist of the same characters in the same order. By the same fashion, two strings are not equal when they don't consist of the same characters in the same order.

Both comparisons give True as a result:

```
'alpha' == 'alpha'  
'alpha' != 'Alpha'
```

The final relation between strings is determined by comparing the first different character in both strings (keep ASCII/UNICODE code points in mind at all times.)

When you compare two strings of different lengths and the shorter one is identical to the longer one's beginning, the longer string is considered greater.

Just like here:

```
'alpha' < 'alphabet'
```

The relation is True.

String comparison is always case-sensitive (upper-case letters are taken as lesser than lower-case).

The expression is True:

```
'beta' > 'Beta'
```

Comparing strings: continued

Even if a string contains digits only, it's still not a number. It's interpreted as-is, like any other regular string, and its (potential) numerical aspect is not taken into consideration in any way.

Look at the examples:

```
'10' == '010'  
'10' > '010'  
'10' > '8'  
'20' < '8'  
'20' < '80'
```

They produce the following results:

```
False  
True  
False  
True  
True  
output
```

Comparing strings against numbers is generally a bad idea.

The only comparisons you can perform with impunity are these symbolized by the `==` and `!=` operators. The former always gives `False`, while the latter always produces `True`.

Using any of the remaining comparison operators will raise a `TypeError` exception.

Let's check it:

```
'10' == 10  
'10' != 10  
'10' == 1  
'10' != 1  
'10' > 10
```

The results in this case are:

```
False  
True  
False  
True  
TypeError exception  
output
```

Run all the examples, and carry out more experiments.

Sorting

Comparing is closely related to sorting (or rather, sorting is in fact a very sophisticated case of comparing).

This is a good opportunity to show you two possible ways to sort lists containing strings. Such an operation is very common in the real world - any time you see a list of names, goods, titles, or cities, you expect them to be sorted.

Let's assume that you want to sort the following list:

```
greek = ['omega', 'alpha', 'pi', 'gamma']
```

In general, Python offers two different ways to sort lists.

The first is implemented as a function named `sorted()`.

The function takes one argument (a list) and returns a new list, filled with the sorted argument's elements. (Note: this description is a bit simplified compared to the actual implementation - we'll discuss it later.)

The original list remains untouched.

Look at the code in the editor, and run it.

```
# Demonstrating the sorted() function:
first_greek = ['omega', 'alpha', 'pi', 'gamma']
first_greek_2 = sorted(first_greek)

print(first_greek)
print(first_greek_2)

print()

# Demonstrating the sort() method:
second_greek = ['omega', 'alpha', 'pi', 'gamma']
print(second_greek)

second_greek.sort()
print(second_greek)
```

The snippet produces the following output:

```
['omega', 'alpha', 'pi', 'gamma']
['alpha', 'gamma', 'omega', 'pi']
output
```

The second method affects the list itself - no new list is created. Ordering is performed in situ by the method named `sort()`.

The output hasn't changed:

```
['omega', 'alpha', 'pi', 'gamma']
['alpha', 'gamma', 'omega', 'pi']
output
```

If you need an order other than non-descending, you have to convince the function/method to change its default behaviors. We'll discuss it soon.

Strings vs. numbers

There are two additional issues that should be discussed here: how to convert a number (an integer or a float) into a string, and vice versa. It may be necessary to perform such a transformation. Moreover, it's a routine way to process input/output data.

The number-string conversion is simple, as it is always possible. It's done by a function named `str()`.

Just like here:

```
itg = 13
flt = 1.3
si = str(itg)
sf = str(flt)

print(si + ' ' + sf)
```

The code outputs:

```
13 1.3
output
```

The reverse transformation (string-number) is possible when and only when the string represents a valid number. If the condition is not met, expect a `ValueError` exception.

Use the `int()` function if you want to get an integer, and `float()` if you need a floating-point value.

Just like here:

```
si = '13'
sf = '1.3'
itg = int(si)
flt = float(sf)

print(itg + flt)
```

This is what you'll see in the console:

```
14.3
output
```

In the next section, we're going to show you some simple programs that process strings.

Key takeaways

1. Strings can be compared to strings using general comparison operators, but comparing them to numbers gives no reasonable result, because no string can be equal to any number. For example:

```
string == number is always False;  
string != number is always True;  
string >= number always raises an exception.
```

2. Sorting lists of strings can be done by:

a function named `sorted()`, creating a new, sorted list;
a method named `sort()`, which sorts the list in situ

3. A number can be converted to a string using the `str()` function.

4. A string can be converted to a number (although not every string) using either the `int()` or `float()` function. The conversion fails if a string doesn't contain a valid number image (an exception is raised then).

Exercise 1

Which of the following lines describe a true condition?

`'smith' > 'Smith'`

`'Smiths' < 'Smith'`

`'Smith' > '1000'`

`'11' < '8'`

Exercise 2

What is the expected output of the following code?

```
s1 = 'Where are the snows of yesteryear?'  
s2 = s1.split()  
s3 = sorted(s2)  
print(s3[1])
```

Exercise 3

What is the expected result of the following code?

```
s1 = '12.8'  
i = int(s1)  
s2 = str(i)  
f = float(s2)  
print(s1 == s2)
```

LAB

Estimated time
30 minutes

Level of difficulty
Medium

Objectives
improving the student's skills in operating with strings;
using strings to represent non-text data.

Scenario

You've surely seen a seven-segment display.

It's a device (sometimes electronic, sometimes mechanical) designed to present one decimal digit using a subset of seven segments. If you still don't know what it is, refer to the following Wikipedia article.

Your task is to write a program which is able to simulate the work of a seven-display device, although you're going to use single LEDs instead of segments.

Each digit is constructed from 13 LEDs (some lit, some dark, of course) - that's how we imagine it:

```
# ### ### # # ### ### ### ### ###  
#  #  # # # #  #  # # # # # #  
# ### ### ### ### ###  # ### ### # #  
# #      #  #  # # #  # # #  # # #  
# ### ###  # ### ###  # ### ### ###
```

Note: the number 8 shows all the LED lights on.

Your code has to display any non-negative integer number entered by the user.

Tip: using a list containing patterns of all ten digits may be very helpful.

Test data

Sample input:

123

Sample output:

```
# ### ###  
#  #  #  
# ### ###  
# #      #  
# ### ###
```

Sample input:

9081726354

Sample output:

```
#### ### # # ### ### ### ### ### # #  
# # # # # # #  #  #  # #  # #  # #  
#### # # ### #  # ### ### ### ### ###  
  # # # # # # #  #  #  # #  #  #  #  
#### ### ### #  # ### ### ### ###  #
```