

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 6)

The capitalize() method

Let's go through some standard Python string methods. We're going to go through them in alphabetical order - to be honest, any order has as many disadvantages as advantages, so the choice may as well be random.

The capitalize() method does exactly what it says - it creates a new string filled with characters taken from the source string, but it tries to modify them in the following way:

- if the first character inside the string is a letter (note: the first character is an element with an index equal to 0, not just the first visible character), it will be converted to upper-case;

- all remaining letters from the string will be converted to lower-case.

Don't forget that:

- the original string (from which the method is invoked) is not changed in any way (a string's immutability must be obeyed without reservation)

- the modified (capitalized in this case) string is returned as a result - if you don't use it in any way (assign it to a variable, or pass it to a function/method) it will disappear without a trace.

Note: methods don't have to be invoked from within variables only. They can be invoked directly from within string literals. We're going to use that convention regularly - it will simplify the examples, as the most important aspects will not disappear among unnecessary assignments.

Take a look at the example in the editor. Run it.

```
# Demonstrating the capitalize() method:  
print('aBcD'.capitalize())
```

This is what it prints:

```
Abcd  
output
```

Try some more advanced examples and test their output:

```
print("Alpha".capitalize())  
print('ALPHA'.capitalize())  
print(' Alpha'.capitalize())  
print('123'.capitalize())  
print("αβγδ".capitalize())
```

The center() method

The one-parameter variant of the center() method makes a copy of the original string, trying to center it inside a field of a specified width.

The centering is actually done by adding some spaces before and after the string.

Don't expect this method to demonstrate any sophisticated skills. It's rather simple.

The example in the editor uses brackets to clearly show you where the centered string actually begins and terminates.

Its output looks as follows:

```
[  alpha  ]  
output
```

If the target field's length is too small to fit the string, the original string is returned.

You can see the center() method in more examples here:

```
print([' + 'Beta'.center(2) + ''])  
print([' + 'Beta'.center(4) + ''])  
print([' + 'Beta'.center(6) + ''])
```

Run the snippets above and check what output they produce.

The two-parameter variant of center() makes use of the character from the second argument, instead of a space. Analyze the example below:

```
print([' + 'gamma'.center(20, '*') + ''])
```

This is why the output now looks like this:

```
[*****gamma*****]  
output
```

Carry out more experiments.

The endswith() method

The endswith() method checks if the given string ends with the specified argument and returns True or False, depending on the check result.

Note: the substring must adhere to the string's last character - it cannot just be located somewhere near the end of the string.

```
# Demonstrating the endswith() method:
if "epsilon".endswith("on"):
    print("yes")
else:
    print("no")
```

Look at our example in the editor, analyze it, and run it. It outputs:

```
yes
output
```

You should now be able to predict the output of the snippet below:

```
t = "zeta"
print(t.endswith("a"))
print(t.endswith("A"))
print(t.endswith("et"))
print(t.endswith("eta"))
```

Run the code to check your predictions.

The find() method

The find() method is similar to index(), which you already know - it looks for a substring and returns the index of first occurrence of this substring, but:

- it's safer - it doesn't generate an error for an argument containing a non-existent substring (it returns -1 then)
- it works with strings only - don't try to apply it to any other sequence.

Look at the code in the editor. This is how you can use it.

```
# Demonstrating the find() method:
print("Eta".find("ta"))
print("Eta".find("mma"))
```

The example prints:

```
1
-1
output
```

Note: don't use find() if you only want to check if a single character occurs within a string - the in operator will be significantly faster.

Here is another example:

```
t = 'theta'
print(t.find('eta'))
print(t.find('et'))
print(t.find('the'))
print(t.find('ha'))
```

Can you predict the output? Run it and check your predictions.

If you want to perform the find, not from the string's beginning, but from any position, you can use a two-parameter variant of the find() method. Look at the example:

```
print('kappa'.find('a', 2))
```

The second argument specifies the index at which the search will be started (it doesn't have to fit inside the string).

Among the two a letters, only the second will be found. Run the snippet and check.

You can use the find() method to search for all the substring's occurrences, like here:

```
the_text = """A variation of the ordinary lorem ipsum
text has been used in typesetting since the 1960s
or earlier, when it was popularized by advertisements
for Letraset transfer sheets. It was introduced to
the Information Age in the mid-1980s by the Aldus Corporation,
which employed it in graphics and word-processing templates
for its desktop publishing program PageMaker (from Wikipedia)"""
```

```
fnd = the_text.find('the')
while fnd != -1:
    print(fnd)
    fnd = the_text.find('the', fnd + 1)
```

The code prints the indices of all occurrences of the article the, and its output looks like this:

```
15
80
198
221
238
output
```

There is also a three-parameter mutation of the find() method - the third argument points to the first index which won't be taken into consideration during the search (it's actually the upper limit of the search).

Look at our example below:

```
print('kappa'.find('a', 1, 4))
print('kappa'.find('a', 2, 4))
```

The second argument specifies the index at which the search will be started (it doesn't have to fit inside the string).

Therefore, the modified example outputs:

```
1
-1
output
```

(a cannot be found within the given search boundaries in the second print()).

The isalnum() method

The parameterless method named `isalnum()` checks if the string contains only digits or alphabetical characters (letters), and returns `True` or `False` according to the result.

Look at the example in the editor and run it.

```
# Demonstrating the isalnum() method:
print('lambda30'.isalnum())
print('lambda'.isalnum())
print('30'.isalnum())
print('@'.isalnum())
print('lambda_30'.isalnum())
print('').isalnum()
```

Note: any string element that is not a digit or a letter causes the method to return `False`. An empty string does, too.

The example output is:

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

Three more intriguing examples are here:

```
t = 'Six lambdas'
print(t.isalnum())      False

t = 'AßΓδ'
print(t.isalnum())      True

t = '20E1'
print(t.isalnum())      True
```

Run them and check their output.

Hint: the cause of the first result is a space - it's neither a digit nor a letter.

The isalpha() method

The isalpha() method is more specialized - it's interested in letters only.

```
# Example 1: Demonstrating the isalpha() method:  
print("Moooo".isalpha())  
print('Mu40'.isalpha())
```

```
# Example 2: Demonstrating the isdigit() method:  
print('2018'.isdigit())  
print("Year2019".isdigit())
```

Look at Example 1 - its output is:

```
True  
False  
output
```

The isdigit() method

In turn, the isdigit() method looks at digits only - anything else produces False as the result.

Look at Example 2 - its output is:

```
True  
False  
output
```

Carry out more experiments.

The islower() method

The islower() method is a fussy variant of isalpha() - it accepts lower-case letters only.

```
# Example 1: Demonstrating the islower() method:
print("Moooo".islower())
print('moooo'.islower())
```

```
# Example 2: Demonstrating the isspace() method:
print(' \n '.isspace())
print(" ".isspace())
print("moos moos moos".isspace())
```

```
# Example 3: Demonstrating the isupper() method:
print("Moooo".isupper())
print('moooo'.isupper())
print('MOOOO'.isupper())
```

Look at Example 1 in the editor - it outputs:

```
False
True
output
```

The isspace() method

The isspace() method identifies whitespaces only - it disregards any other character (the result is False then).

Look at Example 2 in the editor - the output is:

```
True
True
False
output
```

The isupper() method

The isupper() method is the upper-case version of islower() - it concentrates on upper-case letters only.

Again, Look at the code in the editor - Example 3 produces the following output:

```
False
False
True
```

The join() method

The join() method is rather complicated, so let us guide you step by step thorough it:

- as its name suggests, the method performs a join - it expects one argument as a list; it must be assured that all the list's elements are strings - the method will raise a `TypeError` exception otherwise;
- all the list's elements will be joined into one string but...
- ...the string from which the method has been invoked is used as a separator, put among the strings;
- the newly created string is returned as a result.

```
# Demonstrating the join() method:  
print(",".join(["omicron", "pi", "rho"]))
```

Take a look at the example in the editor. Let's analyze it:

- the join() method is invoked from within a string containing a comma (the string can be arbitrarily long, or it can be empty)
- the join's argument is a list containing three strings;
- the method returns a new string.

Here it is:

omicron,pi,rh

The lower() method

The lower() method makes a copy of a source string, replaces all upper-case letters with their lower-case counterparts, and returns the string as the result. Again, the source string remains untouched.

If the string doesn't contain any upper-case characters, the method returns the original string.

Note: The lower() method doesn't take any parameters.

```
# Demonstrating the lower() method:  
print("SiGmA=60".lower())
```

The example in the editor outputs:

```
sigma=60  
output
```

As usual, carry out your own experiments.

The lstrip() method

The parameterless lstrip() method returns a newly created string formed from the original one by removing all leading whitespaces.

Analyze the example code in the editor.

```
# Demonstrating the lstrip() method:  
print "[" + " tau ".lstrip() + "]"
```

The brackets are not a part of the result - they only show the result's boundaries.

The example outputs:

```
[tau ]  
output
```

The one-parameter lstrip() method does the same as its parameterless version, but removes all characters enlisted in its argument (a string), not just whitespaces:

```
print("www.cisco.com".lstrip("w."))
```

Brackets aren't needed here, as the output looks as follows:

```
cisco.com  
output
```

Can you guess the output of the snippet below? Think carefully. Run the code and check your predictions.

```
print("pythoninstitute.org".lstrip(".org"))      pythoninstitute.org
```

Surprised? Leading characters, leading whitespaces. Again, experiment with your own examples.

The replace() method

The two-parameter `replace()` method returns a copy of the original string in which all occurrences of the first argument have been replaced by the second argument.

Look at the example code in the editor. Run it.

```
# Demonstrating the replace() method:
print("www.netacad.com".replace("netacad.com", "pythoninstitute.org"))
print("This is it!".replace("is", "are"))
print("Apple juice".replace("juice", ""))
```

The example outputs:

```
www.pythoninstitute.org
Thare are it!
Apple
output
```

If the second argument is an empty string, replacing is actually removing the first argument's string. What kind of magic happens if the first argument is an empty string?

The three-parameter `replace()` variant uses the third argument (a number) to limit the number of replacements.

Look at the modified example code below:

```
print("This is it!".replace("is", "are", 1))
print("This is it!".replace("is", "are", 2))
```

Can you guess its output? Run the code and check your guesses.

The rfind() method

The one-, two-, and three-parameter methods named `rfind()` do nearly the same things as their counterparts (the ones devoid of the `r` prefix), but start their searches from the end of the string, not the beginning (hence the prefix `r`, for right).

Take a look at the example code in the editor and try to predict its output. Run the code to check if you were right.

```
# Demonstrating the rfind() method:
print("tau tau tau".rfind("ta"))           8
print("tau tau tau".rfind("ta", 9))        -1
print("tau tau tau".rfind("ta", 3, 9))      4
```

The rstrip() method

Two variants of the rstrip() method do nearly the same as lstrip, but affect the opposite side of the string.

Look at the code example in the editor. Can you guess its output? Run the code to check your guesses.

```
# Demonstrating the rstrip() method:
print "[" + "  epsilon ".rstrip() + "]"
print "cisco.com".rstrip(".com")
```

As usual, we encourage you to experiment with your own examples.

```
outputs:
[  epsilon]
cis
```

The split() method

The split() method does what it says - it splits the string and builds a list of all detected substrings.

The method assumes that the substrings are delimited by whitespaces - the spaces don't take part in the operation, and aren't copied into the resulting list.

If the string is empty, the resulting list is empty too.

Look at the code in the editor.

```
# Demonstrating the split() method:
print "phi      chi\npsi".split()
```

The example produces the following output:

```
['phi', 'chi', 'psi']
output
```

Note: the reverse operation can be performed by the join() method.

The startswith() method

The startswith() method is a mirror reflection of endswith() - it checks if a given string starts with the specified substring.

Look at the example in the editor.

```
# Demonstrating the startswith() method:
print("omega".startswith("meg"))
print("omega".startswith("om"))

print()

# Demonstrating the strip() method:
print "[" + "   aleph   ".strip() + "]"
```

This is the result from it:

```
False
True
output
```

The strip() method

The strip() method combines the effects caused by rstrip() and lstrip() - it makes a new string lacking all the leading and trailing whitespaces.

Look at the second example in the editor. This is the result it returns:

```
[aleph]
output
```

Now carry out your own experiments with the two methods.

The swapcase() method

The swapcase() method makes a new string by swapping the case of all letters within the source string: lower-case characters become upper-case, and vice versa.

All other characters remain untouched.

Look at the first example in the editor.

```
# Demonstrating the swapcase() method:
print("I know that I know nothing.".swapcase())

print()

# Demonstrating the title() method:
print("I know that I know nothing. Part 1.".title())

print()

# Demonstrating the upper() method:
print("I know that I know nothing. Part 2.".upper())
```

Can you guess the output? It won't look good, but you must see it:

```
i KNOW THAT i KNOW NOTHING.
output
```

The title() method

The title() method performs a somewhat similar function - it changes every word's first letter to upper-case, turning all other ones to lower-case.

Look at the second example in the editor. Can you guess its output? This is the result:

```
I Know That I Know Nothing. Part 1.
output
```

The upper() method

Last but not least, the upper() method makes a copy of the source string, replaces all lower-case letters with their upper-case counterparts, and returns the string as the result.

Look at the third example in the editor. It outputs:

```
I KNOW THAT I KNOW NOTHING. PART 2.
output
```

Hooray! We've made it to the end of this section. Are you surprised with any of the string methods we've discussed so far? Take a couple of minutes to review them, and let's move on to the next part of the course where we'll show you what great things we can do with strings.

Key takeaways

1. Some of the methods offered by strings are:

<code>capitalize()</code>	- changes all string letters to capitals;
<code>center()</code>	- centers the string inside the field of a known length;
<code>count()</code>	- counts the occurrences of a given character;
<code>join()</code>	- joins all items of a tuple/list into one string;
<code>lower()</code>	- converts all the string's letters into lower-case letters;
<code>lstrip()</code>	- removes the white characters from the beginning of the string;
<code>replace()</code>	- replaces a given substring with another;
<code>rfind()</code>	- finds a substring starting from the end of the string;
<code>rstrip()</code>	- removes the trailing white spaces from the end of the string;
<code>split()</code>	- splits the string into a substring using a given delimiter;
<code>strip()</code>	- removes the leading and trailing white spaces;
<code>swapcase()</code>	- swaps the letters' cases (lower to upper and vice versa)
<code>title()</code>	- makes the first letter in each word upper-case;
<code>upper()</code>	- converts all the string's letter into upper-case letters.

2. String content can be determined using the following methods (all of them return Boolean values):

<code>endswith()</code>	- does the string end with a given substring?
<code>isalnum()</code>	- does the string consist only of letters and digits?
<code>isalpha()</code>	- does the string consist only of letters?
<code>islower()</code>	- does the string consists only of lower-case letters?
<code>isspace()</code>	- does the string consists only of white spaces?
<code>isupper()</code>	- does the string consists only of upper-case letters?
<code>startswith()</code>	- does the string begin with a given substring?

Exercise 1

What is the expected output of the following code?

```
for ch in "abc123XYX":
    if ch.isupper():
        print(ch.lower(), end='')
    elif ch.islower():
        print(ch.upper(), end='')
    else:
        print(ch, end='')
```

Exercise 2

What is the expected output of the following code?

```
s1 = 'Where are the snows of yesteryear?'
s2 = s1.split()
print(s2[-2])
```


Exercise 3

What is the expected output of the following code?

```
the_list = ['Where', 'are', 'the', 'snows?']  
s = '*'.join(the_list)  
print(s)
```

Exercise 4

What is the expected output of the following code?

```
s = 'It is either easy or impossible'  
s = s.replace('easy', 'hard').replace('im', '')  
print(s)
```

LAB

Estimated time
20-25 minutes

Level of difficulty
Medium

Objectives

improving the student's skills in operating with strings;
using built-in Python string methods.

Scenario

You already know how `split()` works. Now we want you to prove it.

Your task is to write your own function, which behaves almost exactly like the original `split()` method, i.e.:

it should accept exactly one argument - a string;

it should return a list of words created from the string, divided in the places where the string contains whitespaces;

if the string is empty, the function should return an empty list;

its name should be `mysplit()`

Use the template in the editor. Test your code carefully.

```
def mysplit(strng):  
    #  
    # put your code here  
    #  
  
print(mysplit("To be or not to be, that is the question"))  
print(mysplit("To be or not to be,that is the question"))  
print(mysplit("  "))  
print(mysplit(" abc "))  
print(mysplit(""))
```

Expected output

```
['To', 'be', 'or', 'not', 'to', 'be,', 'that', 'is', 'the', 'question']  
['To', 'be', 'or', 'not', 'to', 'be,that', 'is', 'the', 'question']  
[]  
['abc']  
[]
```