# Section 8

More About Strings

Mohammed Asir Shahid

2021-08-03

## Contents

## 1 Advanced String Syntax

Text is one of the most common forms of data that we will work with.

How do we add quotes and apostrophes in Python Strings?

### 1.1 Escape Characters

```
print("Hello")
print('That is Alice\'s cat')
print('Say hi to Bob\'s mother')
```

```
Hello
That is Alice's cat
Say hi to Bob's mother
```

We can have several different types of escape characters.

1. \' - Prints as Single quote
2. \“ - Prints as Double quote
3. -⁀Prints as tab
4. - Prints as newline
5. \ - Prints as backslash.

```
print("Hello there!\nHow are you?\nI\'m fine")
```

```
Hello there!
How are you?
I'm fine
```

### 1.2 Raw Strings

We can also have raw strings which should be used when we have text that includes many backslashes that we don't want to use for escape characters. There are similar to regular strings, but have “r” right at the beginning.

```
print(r"Hello")
print(r"That is Carol\'s cat.")
```

```
Hello
That is Carol\'s cat.
```

### 1.3 Multiline Strings with Triple Quotes

Instead of using in order to put a newline in the string, it can be easier to use multiline strings.

```
print("""Dear Alice,
Eve’s cat has been arrested for catnapping, cat burgularly, and extortion.
Sincerly,
Bob""")
```

```
Dear Alice,
Eve’s cat has been arrested for catnapping, cat burgularly, and extortion.
Sincerly,
Bob
```

Above, Python automatically formats this using and în the background.

### 1.4 Similarities Between Strings and Lists

Strings can use indices and slices similar to lists. The string “Hello” can be seen as a list with 5 items, each of which are individual strings.

```
spam="Hello world!"

print(spam[0])
print(spam[1:5])
print(spam[-1])

print("Hello" in spam)
print("x" in spam)
print("HELLO" in spam)
```

```
H
ello
!
True
False
False
```

## 2 String Methods

Unlike list methods, string methods return a new string value rather than modifying the existing string in place. This is due to the fact that strings are immutable.

### 2.1 upper(), lower()

```
spam="Hello world!"

spam.upper()

print(spam)

spam=spam.upper()

print(spam)

spam=spam.lower()

print(spam)
```

```
Hello world!
HELLO WORLD!
hello world!
```

This can be helpful when we allow the user to input an answer and we do not care about uppercase or lowercase. There is also a corresponding title() method.

### 2.2 isupper(), islower()

These methods return a boolean value depending on whether the strings are uppercase or lowercase. If we have a blank string or a string without any uppercase or lowercase characters, both isupper() and islower() will be false.

```
spam="Hello world"
```

```
print(spam.islower())

spam="hello world"

print(spam.islower())

spam="HELLO WORLD"

print(spam.isupper())
```

```
False
True
True
```

We can also call both upper() and isupper() on the same string.

```
print("Hello".upper().isupper())
```

```
True
```

## 2.3 The isX() methods

There are several other string methods that begin with is and return a boolean.

1. isalpha() Returns True if the string consists only of letters and isn't blank
2. isalnum() Returns True if the string consists only of letters and numbers and is not blank
3. isdecimal() Returns True if the string consists only of numeric characters and is not blank
4. isspace() Returns True if the string consists only of spaces, tabs, and newlines and is not blank
5. istitle() Returns True if the string consists only of words that begin with an uppercase letter followed by only lowercase letters

```
print("hello".isalpha())
print("hello123".isalnum())
print("123".isdecimal())
print("    ".isspace())
print("Hello world!".isspace())
print("Hello world!"[5].isspace())
print("This Is Title Case!".istitle())
```

```
True
True
True
True
False
True
True
```

## 2.4 startswith(), endswith()

These return true if the string value that they are called on begins or ends, respectively, with the string that is passed to the method.

```
print("Hello world!".startswith("Hello"))
print("Hello world!".startswith("h"))
print("Hello world!".startswith("ello"))

print("Hello world!".endswith("world!"))
print("Hello world!".endswith("world"))
print("Hello world!".endswith("!"))
```

```
True
False
False
True
False
True
```

## 2.5 join() and split()

The join() method is useful when we have a list of strings that need to be joined together into a single string value. It gets passed a list of strings and returns a string.

```
print(",".join(["cats","rats","bats"]))

print("".join(["cats","rats","bats"]))

print(" ".join(["cats","rats","bats"]))

print("\n\n".join(["cats","rats","bats"]))
```

```
cats,rats,bats
catsratsbats
cats rats bats
cats

rats

bats
```

The split method does the opposite. It is called on a string value and returns a list of strings.

```
print("My name is Simon".split())

print("My name is Simon".split("m"))

print("My name is Simon".split("!"))
```

```
['My', 'name', 'is', 'Simon']
['My na', 'e is Si', 'on']
['My name is Simon']
```

## 2.6 rjust(), ljsut(), center()

rjust() and ljust() return a padded version of the string they are called upon in order to either right or left justify the string. They take an argument that dictates the character length of the justification. We can also give an argument to change the character that is used for the justification.

The center method works similarly, but it centers the text instead of justifying to the right or left.

```
print("Hello".rjust(10))
print("Hello".rjust(30))
print("Hello".ljust(10))

print("Hello".rjust(20, "*"))
print("Hello".ljust(10, "-"))

print("Hello".center(20))
print("Hello".center(20,"*"))
print("Hello".center(20,"="))
```

```
     Hello
                         Hello
Hello
***************Hello
Hello-----
       Hello
*******Hello********
=======Hello========
```

## 2.7 strip(), rstrip(), lstrip()

These methods can be used when we want to strip off whitespace characters like spaces, tabs, or new lines from the right side, left side, or both sides. We can also pass a string that we want to remove instead of just the whitespace.

```
spam="Hello".rjust(10)

print(spam)
```

```
spam=spam.strip()

print(spam)

print("   x    ".strip())
print("   x    ".rstrip())
print("   x    ".lstrip())


spam="SpamSpamBaconSpamEggsSpamSpam"

print(spam.strip("amSp"))
```

```
     Hello
Hello
x
   x
x
BaconSpamEggs
```

## 2.8 replace()

This method looks for two arguments, a string to look for and a string to replace it with.

```
spam="Hello there!"

print(spam)

spam.replace("e","XYZ")

print(spam)
```

```
Hello there!
Hello there!
```

### 2.9 The pyperclip Module

We installed this in a previous section. It comes with copy and paste functions that lets it read and write to your computer's clipboard.

```
pip install pyperclip
```

```
import pyperclip

#pyperclip.copy("Hello!!!!!!")
#pyperclip.paste("Hello!!!!!!")
```

## 3 String Formatting

Often times when we want to combine strings to form a single string, we use the plus operator. However, this can get complicated when we have long strings with a lot of variables. Python has string formatting techniques that make this easier.

```
name = "Alice"
place = "Main Street"
time = "6 pm"
food = "turnips"

print("Hello " + name + ", you are invited to a party at " + place + " at " + time + "

print("Hello %s, you are invited to a party at %s at %s. Please bring %s." % (name, pla
```

```
Hello Alice, you are invited to a party at Main Street at 6 pm. Please bring turnips.
Hello Alice, you are invited to a party at Main Street at 6 pm. Please bring turnips.
```