Lesson 7. Introducing string objects: sequences of characters

After reading lesson 7, you'll be able to

- Understand what string objects are
- See what values a string object can have
- Do some basic operations using string objects

Working with sequences of characters is common. These sequences are known as *strings*, and you can store any sequence of characters inside a string object: your name, your phone number, your address including the new lines, and so on. Storing information in a string format is useful. You can do many operations after you have data represented in a string. For example, if you and a friend are both doing research for a project, you can take notes on separate concepts and then combine your findings. If you're writing an essay and discover that you overused a word, you can remove all instances of that word or replace some instances with another word. If you discover that you accidentally had Caps Lock on, you can convert the entire text to lowercase instead of rewriting.

Consider this

Look at your keyboard. Pick out 10 characters and write them down. String them together in any order. Now try to string them in some combination to form words.

Answer:

- `hjklasdfqw`
- `shawl` or `hi` or `flaw`

7.1. STRINGS AS SEQUENCES OF CHARACTERS

In lesson 4, you learned that a string is a sequence of characters and that all characters in that string are denoted inside quotation marks. You can use either `"` or `'` as long as you're consistent for one string. The type of a string in Python is `str`. The following are examples of string objects:

- `"simple"`
- `'also a string'`
- `"a long string with Spaces and special sym&@L5_!"`
- `"525600"`
- `""` (Nothing between double quotes is an empty string.)
- `''` (Nothing between single quotes is an empty string.)

The sequence of characters can contain numbers, uppercase and lowercase letters, spaces, special characters representing a newline, and symbols in any order. You know that an object is a string because it starts with a quotation mark and ends with a quotation mark. Again, the kind of quotation mark used to end a string must be the same as the kind you used to start it.

Quick check 7.1

Are each of the following valid string objects?

1

```
"444"
```

2

3

```
'combo"
```

4

```
checkered_flag
```

5

```
"99 bbaalloonnss"
```

7.2. BASIC OPERATIONS ON STRINGS

Before working with strings, you must create a string object and add content to it. Then you can start using the string by performing operations on it.

7.2.1. Creating a string object

You can create a string object by initializing a variable to be bound to the object. For example:

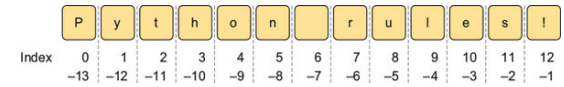
- In the statement `num_one = "one"`, the variable `num_one` is bound to an object of type `str` whose value is `"one"`.
- In the statement `num_two = "2"`, the variable `num_two` is bound to an object of type `str` whose value is `"2"`. It's important to understand that `"2"` is a string and not the integer `2`.

7.2.2. Understanding indexing into a string

Because strings are made up of a sequence of characters, you can determine the value of a character at a certain position in the string. Called *indexing* into a string, this is the most basic operation you can do with a string object.

In computer science, you start counting from 0. This is used when you manipulate string objects. Take a look at figure 7.1, which shows a string object whose value is `"Python rules!"` Each character is located at an index. The first character in a string is always at index 0. For the string `"Python rules!"`, the last character is at index 12.

Figure 7.1. The string "Python rules!" and the index of each character. The first row shows indexing with positive integers, and the second row shows indexing with negative integers.



You can also count backward. The last character in any string is always at index -1 when you're counting backward. For the string `"Python rules!"`, the first character, `P`, is at index -13. Notice that the space is also a character.

Quick check 7.2

For the string, `"fall 4 leaves"`, what's the index number of the following characters? Give the forward and backward index values:

1

```
4
```

2

```
f
```

3

```
s
```

There's a special way to index into a string to give you the value of the character at a particular index. You use square brackets, `[]`, and inside the square brackets you put any index value that you want, provided that it's an integer value. Here are two examples using the string `"Python rules!"`:

- `"Python rules!"[0]` evaluates to `'P'`.
- `"Python rules!"[7]` evaluates to `'r'`.

The number for the index is allowed to be any integer. What happens if it's a negative number? The last character in the string is considered to be at index -1. You're essentially counting through the strings backward.

If you assign the string to a variable, you can index into it as well, in a more concise way. For example, if `cheer = "Python rules!"`, then `cheer[2]` gives you the value `'t'`.

Quick check 7.3

To what do the following expressions evaluate? Try them in Spyder to check!

1

```
"hey there"[1]
```

2

```
"TV guide"[2]
```

3

```
code = "L33t hax0r5"
code[0]
code[-4]
```

7.2.3. Understanding slicing a string

So far, you know how to get the character at one index in the string. But sometimes you may want to know the value of a group of characters, starting from one index and ending with another. Let's say you're a teacher and have information on all students in your class in the form "##### FirstName LastName". You're only interested in the names and notice that the first six characters are always the same: five digits and then a space. You can extract the data you want by looking at the part of the string that starts from the seventh character until the end of the string.

Extracting data in this way is called getting a *substring* of the string. For example, the characters `snap` in the string `s = "snap crackle pop"` are a substring of `s`.

The square brackets can be used in a more sophisticated way. You can use them to *slice* the string between two indices and get a substring, according to certain rules. To slice a string, you can put up to three integers, separated by colons, in square brackets:

```
[start_index:stop_index:step]
```

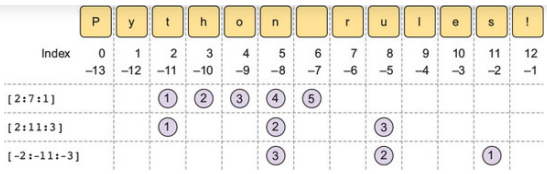
where

- `start_index` represents the index of the first character to take.
- `stop_index` represents the index up to which you take the characters, but not including the one at `stop_index`.
- `step` represents how many characters to skip (for example, take every second character or every fourth character). A positive step means that you're going left to right through the string, and vice versa for a negative step. It's not necessary to explicitly give the step value. If omitted, the step is 1, meaning you take every character (you don't skip any characters).

The following examples are depicted in figure 7.2, showing the order in which the characters are selected and put together to give a final value. If `cheer = "Python rules!"`, then

- `cheer[2:7:1]` evaluates to `'thon '` because you're stepping left to right, taking every character in order, starting with the one at index 2 and not including the one at index 7.
- `cheer[2:11:3]` evaluates to `'tnu'` because you're stepping left to right, taking every third character, starting with the one at index 2 and not including the one at index 11.
- `cheer[-2:-11:-3]` evaluates to `'sun'` because you're stepping right to left, taking every third character, starting with the one at index -2 and not including the one at index -11.

Figure 7.2. Three examples of slicing into the string "Python rules!". The numbered circles on each row indicate the order in which Python retrieves the characters from the string to form a new substring from the slice.



Quick check 7.4

To what do the following expressions evaluate? Try them in Spyder to check yourself!

1

```
"it's not impossible"[1:2:1]
```

2

```
"Keeping Up With Python"[-1:-20:-2]
```

3

```
secret = "mai p455w_zero_rD"
secret[-1:-8]
```

7.3. OTHER OPERATIONS ON STRING OBJECTS

A string is an interesting object type because you can do quite a few complex operations with strings.

7.3.1. Getting the number of characters in a string with len()

Suppose you're reading student essays and you've imposed a 2,000-character limit. How can you determine the number of characters a student used? You can set up the entire essay inside a string and then use command, `len()`, to get the number of characters in the string. This includes all characters between the quotation marks, including spaces and symbols. The empty string has a length of 0. For example,

- `len("")` evaluates to `0`.
- `len("Boston 4 ever")` evaluates to `13`.
- If `a = "eh?"`, then `len(a)` evaluates to `3`.

The command `len()` is special in that you can use it on other types of objects, not just on a string object.

The next few operations that you can do on strings will take on a different look. You'll use *dot notation* to make commands to the string objects. You have to use dot notation when the command you want was created to work on only a specific type of object. For example, a command to convert all letters in a string to uppercase was created to work with only a string object. It doesn't make sense to use this command on a number, so this command uses dot notation on a string object.

The dot notation commands look a little different from `len()`. Instead of putting the string object name in the parentheses, you put the name before the command and place a dot between them; for example, `a.lower()` instead of `lower(a)`. You can think of the dot as indicating a command that will work with only a given object—in this case, a string. This touches upon a much deeper idea called *object-oriented programming*, something that you'll see in greater detail in lesson 30.

7.3.2. Converting between letter cases with upper() and lower()

Suppose you're reading student essays, and one student wrote everything in capital letters. You can set up the essay inside a string and then change the case of letters in the string.

A few commands are available to manipulate the case of a string. These commands affect only the letter characters in the string. Numbers and special characters aren't affected:

- `lower()` converts all letters in the string to lowercase. For example, `"Ups AND Downs".lower()` evaluates to `'ups and downs'`.
- `upper()` converts all the letters in the string to uppercase. For example, `"Ups AND Downs".upper()` evaluates to `'UPS AND DOWNS'`.
- `swapcase()` converts lowercase letters in the string to uppercase, and vice versa. For example, `"Ups AND Downs".swapcase()` evaluates to `'uPS and dOWNS'`.
- `capitalize()` converts the first character in the string to a capital

`long Time Ago...".capitalize()` evaluates to `'A long time ago... '`.

Quick check 7.5

You're given `a = "python 4 ever&EVER"`. Evaluate the following expressions. Then try them in Spyder to check yourself:

1

```
a.capitalize()
```

2

```
a.swapcase()
```

3

```
a.upper()
```

4

```
a.lower()
```

SUMMARY

In this lesson, my objective was to teach you about string objects. You saw how to get elements at each position by indexing a string and how to slice a string to get substrings. You saw how to get the length of a string, and how to convert all letters to lowercase or uppercase. Here are the major takeaways:

- Strings are sequences of single-character strings.
- String objects are denoted by quotation marks.
- You can do many operations on strings to manipulate them.

Let's see if you got this...

Q7.1

Write one or more commands that uses the string `"Guten Morgen"` to get `TEN`. There is more than one way to do this.

Q7.2

Write one or more commands that uses the string `"RaceTrack"` to get `Ace`.

Recommended / Playlists / History / Topics / Settings / Get the App / Sign Out

PREV
Unit 2. Strings, tuples, and interacting with the user

NEXT
Lesson 8. Advanced string operations