Lesson 10. Tuple objects: sequences of any kind of object

After reading lesson 10, you'll be able to

- Create a sequence of any kind of object by using a tuple
- Do a few operations on tuple objects
- Swap variable values by using tuples

Suppose I give you the simple task of keeping track of your favorite super-hero characters. Let's say you have three: Spiderman, Batman, and Superman.

Using what you know so far, you could try to create a string containing every one of these names, separated by a space, like so: `"Spiderman Batman Superman"`. Using the commands you learned in lessons 7 and 8, you'd be able, with a little effort and care, to keep track of indices in the string and extract each name as needed.

But what if you kept full names in the string, like so: `"Peter Parker Bruce Wayne Clark Kent"`. It now becomes considerably harder to extract each person's name because the first and last names are also separated by spaces. You could use other special characters, such as a comma, to separate full names, but this doesn't solve the most annoying problem with using strings to store this data: it's tedious to extract items of interest because you have to keep track of starting and ending indices.

Consider this

Look in your fridge. Write down all the objects you can see in there, separated by commas. Now look in your clothes hamper. Write down all the objects you can see in there, separated by commas.

In each set of items:

- How many items did you put down?
- What is the first item? What is the middle item (if you have an even number of items, round down)?

Answer:

Fridge: Milk, cheese, cauliflower, carrots, eggs

- Five items
- First: milk; middle: cauliflower

Hamper: T-shirt, socks

- Two items
- First: T-shirt; middle: T-shirt

10.1. TUPLES AS SEQUENCES OF DATA

Strings store sequences of characters. It'd be a lot more convenient if there were a way to store individual objects in a sequence, as opposed to only string characters. As you write more-complicated code, it becomes useful to be able to represent sequences of any kind of objects.

10.1.1. Creating tuple objects

Python has a data type that represents a sequence of any objects, not just single-character strings. This data type is a *tuple*. In the same way that a string is represented within quotation marks, a tuple is represented in parentheses, `()`. Individual objects within the tuple are separated by a comma. An example of a tuple is `(1, "a", 9.9)`. Other examples of tuples are as follows:

- `(1, 2, 3)`—A tuple containing three integer objects.
- `("a", "b", "cde", "fg", "h")`—A tuple containing five string objects.
- `(1, "2", False)`—A tuple containing an integer, a string, and a Boolean object.
- `(5, (6, 7))`—A tuple containing an integer and another tuple made up of two integers.
- `(5,)`—A tuple containing a single object. Notice the extra comma, which tells Python that the parentheses are used to hold a singleton tuple and not to denote precedence in a mathematical operation.

Quick check 10.1

Are each of the following valid tuple objects?

1

```
("carnival",)
```

2

```
("ferris wheel", "rollercoaster")
```

3

```
("tickets")
```

4

```
((), ())
```

10.2. UNDERSTANDING OPERATIONS ON TUPLES

Tuples are a more general version of strings, because every item in the tuple is a separate object. Many operations on tuples are the same as on strings.

10.2.1. Getting the tuple length with len()

Recall that the command `len()` can be used on other objects, not just on strings. When you use `len()` on a tuple, you get a value that represents the number of objects inside the tuple. For example, the expression `len((3, 5, "7", "9"))` means that you're finding the length (the number of objects) of the tuple `(3, 5, "7", "9")`. The expression evaluates to `4` because this tuple has four elements.

Quick check 10.2

Evaluate the following expressions. Then try them in Spyder to check yourself:

1

```
len(("hi", "hello", "hey", "hi"))
```

2

```
len(("abc", (1, 2, 3)))
```

3

```
len(((1, 2),))
```

4

```
len(())
```

10.2.2. Indexing into and slicing a tuple with []

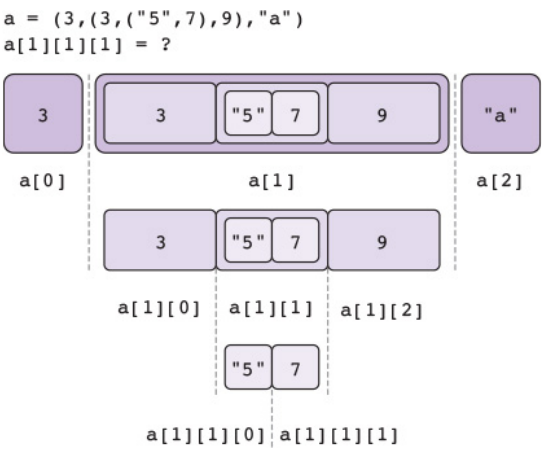
Because tuples are a sequence of objects, indexing into a tuple is the same as indexing into a string. You use the `[]` operator, and the first object is at index 0, the second object is at index 1, and so on. For example,

- `(3, 5, "7", "9")[1]` evaluates to `5`.
- `(3, (3, 5), "7", "9")[1]` evaluates to `(3, 5)`.

One difference from strings is in the special case when one of the objects in the tuple is another tuple. For example, `(3, (3, ("5", 7), 9), "a")`

You can access an element deep down in a sequence of nested tuples by doing a series of indexing operations. For example, `(3, (3, ("5", 7), 9), "a")[1][1][1]` evaluates to 7. This is a bit tricky because you can have tuples inside tuples inside tuples. Figure 10.1 shows how you can visualize the expression.

Figure 10.1. The structure of the tuple `(3, (3, ("5", 7), 9), "a")`. The dashed lines indicate separate objects in the tuple.



Going step-by-step, you evaluate the tuple as follows:

- `(3, (3, ("5", 7), 9), "a")[1]` evaluates to the tuple `(3, ("5", 7), 9)`.
- `(3, ("5", 7), 9)[1]` evaluates to the tuple `("5", 7)`.
- `("5", 7)[1]` evaluates to `7`.

Slicing a tuple is the same as slicing a string, with the same rules. But you have to be careful to recognize that you might have other tuples as elements at a certain position.

Quick check 10.3

Evaluate the following expressions. Then try them in Spyder to check yourself:

1

```
("abc", (1, 2, 3))[1]
```

2

```
("abc", (1, 2, "3"))[1][2]
```

3

```
("abc", (1, 2), "3", 4, ("5", "6"))[1:3]
```

4

```
a = 0
t = (True, "True")
t[a]
```

10.2.3. Performing mathematical operations

The same operations you're allowed to do on strings, you're allowed to do on tuples: addition and multiplication.

You can add two tuples to *concatenate* them. For example, `(1, 2) + (-1, -2)` evaluates to `(1, 2, -1, -2)`.

You can multiply a tuple by an integer to get a tuple that contains the original tuple repeated that many times. For example, `(1, 2) * 3` evaluates to `(1, 2, 1, 2, 1, 2)`.

Quick check 10.4

Evaluate the following expressions. Then try them in Spyder to check yourself:

1

2

```
2 * ("no", "no", "no")
```

3

```
(0, 0, 0) + (1,)
```

4

```
(1, 1) + (1, 1)
```

10.2.4. Swapping objects inside tuples

In this section, you'll see one more interesting way to use tuples. You can use tuples to swap the object values associated with variable names, if the variables are elements of the tuple. For example, say you start with these two variables:

```
long = "hello"
short = "hi"
```

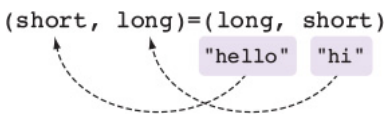
You want to write a line that yields the equivalent of the following swap:

```
long = "hi"
short = "hello"
```

Figure 10.2 shows the visualization you should have in mind for the following code, which accomplishes the swap:

```
long = "hello"
short = "hi"
(short, long) = (long, short)
```

Figure 10.2. Using tuples to swap the values of the two objects between the variable names



You start with `"hello"` bound to the variable `long`, and `"hi"` bound to the variable `short`. After the line `(short, long) = (long, short)` is executed, the value of `short` is `"hello"`, and the value of `long` is `"hi"`.

You might think that having two variables on the left side of the equal sign isn't allowed. But recall that the variables are in the context of a tuple object, and surrounded by parentheses, so the item to the left of the equal sign is only one object, a tuple. This tuple has variables that are bound to other objects at each of its indices. You'll see why using tuples in this way is useful in lesson 19.

Quick check 10.5

Write a line to swap the values of the following:

1

```
s = "strong"
w = "weak"
```

2

```
yes = "affirmative"
no = "negative"
```

SUMMARY

In this lesson, my objective was to teach you about tuples and how they behave in a similar way to strings. You learned how to get elements at each position by indexing into a tuple and how to slice a tuple to get at elements inside it; these elements can be primitive objects or might even be tuples themselves. Unlike strings, an object inside a tuple can be another tuple,

- Tuples are sequences of any kind of object, even other tuples.
- You can index into multiple levels of tuples.
- You can use tuples to swap the values of variables.

Let's see if you got this...

Q10.1

Write a program that initializes the string `word = "echo"`, the empty tuple `t = ()`, and the integer `count = 3`. Then, write a sequence of commands by using the commands you learned in this lesson to make `t = ("echo", "echo", "echo", "cho", "cho", "cho", "ho", "ho", "ho", "o", "o", "o")` and print it. The original word is added to the end of the tuple, and then the original word without the first letter is added to the tuple, and so on. Each substring of the original word gets repeated `count` number of times.

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

PREV
Lesson 9. Simple error messages

NEXT
Lesson 11. Interacting with the user