# Theory: Slicing

18 minutes 16 / 16 problems solved

Start practicing

4169 users solved this topic. Latest completion was about 3 hours ago.

Python provides the ability to access individual elements of lists, strings, and tuples by using indexes. It is possible because these types are considered as ordered **sequences**.

Take a look at a list containing Fibonacci numbers. We can print out any number from the sequence. Don't forget, indexes start from zero, not one:

```
fib_nums = [0, 1, 1, 2, 3, 5, 8, 13, 21]

print(fib_nums[0])  # 0, the first element
print(fib_nums[8])  # 21, the last element
```

## §1. Getting sections of sequences

Another thing you may want to do with any sequence is retrieving its **part**. Usually, it means getting elements from a particular section by their indexes. This is called **slicing**, and there is a special notation for it. It looks like accessing an element by its index, but in an improved manner:

```
print(fib_nums[2:5])  # [1, 2, 3]
```

Looks great, doesn't it? Just like with the indexing, we use **square brackets**, but here we add a **colon** to indicate that we're slicing. Actually, slicing is one of the most famous and widely used features of Python. It allows developers to do a lot of cool things.

**Pay attention** to the end index: it is not the index of the last element of the slice, but rather an index of the first element that is NOT in the slice (i.e. excluded index)! So the last element **is not included**.

A string can also be sliced:

```
text = 'Python is not only a snake!'
print(text[10:18])  # 'not only'
```

In the same way, slicing can be applied to tuples. We hope you can try it on your own.

There's another interesting thing to note: if you set the end index higher than the last element of the sequence, no Error would be raised. Instead, all elements up to the end will be taken:

```
text = 'Python is not only a snake!'
print(text[10:9999])  # 'not only a snake!'

words = ['Python', 'is', 'not', 'only', 'a', 'snake', '!']
print(words[2:9999])  # ['not', 'only', 'a', 'snake', '!']
```

Now, we will explore slicing in more detail with the help of lists.

## §2. Forms of slicing

We've demonstrated the use of slicing with starting and ending indexes. But this is not the only possible form.

The full syntax for slicing looks like this:

```
sequence[start:stop:step]  # from start to stop-1, by step
```

This statement produces a slice of the sequence where `start` is an index of the first needed element (the element is included in the slice) and `stop` is an index of the last element (the element is **not** included in the slice), `step` is an

Current topic:

✓ Slicing

Topic depends on:

✓ Indexes 7★ Stage 3

Topic is required for:

- Recursion in Python
- Testing user input
- Aggregations and Ordering
- Array creation and indexing
- Text corpora in NLTK

Table of contents:

interval between elements to be chosen.

Let's slice a list of planets picking every second planet. We start from the third (with the index 2) planet and stop at the seventh (with the index 6) one. The eighth planet (with the index 7) is not included in the slice.

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
print(planets[2:7:2])  # ['Earth', 'Jupiter', 'Uranus']
```

Each part of the slice has a default value, so it can be omitted. If we don't specify the `start` index, it is considered to be **0**; if we don't specify the `stop` index, it is equal **to the length of the sequence**. The default `step` is **1**, i.e. every element between the beginning and the end is put in the slice.

Here's what happens if we slice without specifying some indexes:

```
sequence[:end]    # element from the 1st element to end-1
sequence[start:]  # elements from start to the last element
sequence[:]       # the full copy of the sequence
sequence[::step]  # every element with a given step
```

Let's take a look at some examples to make understanding more practical.

```
snakes = ['python', 'cobra', 'viper']
print(snakes[:2])     # ['python', 'cobra']
print(snakes[0][:2])  # py

degrees_of_two = [1, 2, 4, 8, 16, 32, 64, 128]
print(degrees_of_two[4:])  # [16, 32, 64, 128]

colors = ['red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet']
print(colors[::3])  # ['red', 'green', 'violet']
```

An interesting way to use slicing is to create **a copy** of the sequence using `[:]` notation.

```
sheep = ['Dolly', 'Polly', 'Molly']
cloned_sheep = sheep[:]  # ['Dolly', 'Polly', 'Molly']
```

Indexes can also be **negative.** We saw this before when we accessed a single element: it means counting from right to left and starting at -1. When the `step` value is negative, the elements are returned in reverse order.

```
pets = ['dog', 'cat', 'parrot', 'gecko']

print(pets[-2:])   # ['parrot', 'gecko']
print(pets[:-2])   # ['dog', 'cat']
print(pets[::-1])  # ['gecko', 'parrot', 'cat', 'dog']
print(pets[::-2])  # ['gecko', 'cat']
```

If you're using negative `step` with the `start` and `end` indexes, those should be chosen accordingly, that is, the `start` index should be greater than the `end` index!

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(numbers[7:2:-1])  # [8, 7, 6, 5, 4]
print(numbers[2:7:-1])  # []
```

We hope you get the general idea of slicing now.

## §3. Conclusion

Slicing allows you to get sections of sequences such as lists, strings, and tuples specifying `start`, `end` and `step`. Remember that all the indexes are optional in the slice syntax because there is a default value for all of them.

In some sense, slicing is just an extension of the standard indexing with similar rules: the first index of a sequence is zero, and negative indexes start from the end. We believe, that after a bit of practice, you will be good at it.

Report a typo

**353** users liked this theory. **4** didn't like it. **What about you?**

Start practicing

Comments (8) Hints (5) Useful links (1) Show discussion