An Introduction to Slicing Syntax in Python

David W. Agler

2023-11-17

Slicing in Python

Slicing allows for accessing (or extracting) part of sequence, e.g., a string, list, or tuple. In this article, I'll focus on slicing strings. Slicing is achieved by using a syntax to the right of a sequence that contains indices to select a range of items from the sequence.

Slicing Syntax: Start and Stop

Slicing is achieved using the following syntax:

`sequence[start:stop]`

That is, there is the sequence to be sliced (e.g., a string, list, or tuple) followed by integers and a colon in square brackets. The first integer is the start of the slice (including it in the slice) and the second integer is the stop of the slice (excluding it from the slice).

The `start` argument is the index of the first item we want to access. So, if we had the string `hello`, we might specify the start of the slice with its index:

```
sequence = "hello"
new_seq = sequence[0:stop]
```

The index specifying the start of the slice is *inclusive* so "0" includes the first letter in the string. The `stop` argument is the *index of the first value that we do not want to include in the slice.* So, if we specify an index of "0" for the start and "2" for the stop, the slice will give us the items at "0, 1" but not 2. This is because "2" is the index of the first item that we don't want to include in the slice. To illustrate, if we had the string `hello`, if the `start` is `0` and the `stop` is 2, then printing the slice will return `he` (the items at index 0 and 1). If the `stop` is 3, then printing the slice will return `hel` (the items at index 0, 1, 2).

What if we want to slice the entire string? To do this, we cannot use the index of the last item in the sequence as the stop. Instead, we need to use the index of the item after the last item in the sequence. Since the index of the last character in `hello` is "4", we would specify the stop of the slice as "5":

```
sequence = "hello"
new_seq = sequence[0:5] # start is 0, stop is 5
print(new_seq) # prints "hello"
```

Step Argument

We can add a third `step` argument to our slice syntax. This is the number of items we step up or down through the sequence from the start to the stop.

```
sequence[start:stop:step]
```

The default value for the step is "1" so `"hello"[0:5:1]` is equivalent to `"hello"[0:5]`. But we can also specify a different value for the step. For example, if we wanted to start at index "0" and step up by "2", we could specify the step as "2":

```
sequence = "hello"
new_seq = sequence[0:5:1]
print(new_seq) # prints "hello"
new_seq2 = sequence[0:5:2]
print(new_seq2) # prints "hlo"
```

Negative Start, Stop, and Step Value

We can also use negative numbers for the start, stop, and step arguments. When using positive indices, the first item is 0. When using negative integers, the last item in the sequence is -1.

H	e	l	l	o
0	1	2	3	4
-5	-4	-3	-2	-1

So, if we wanted to start at the last item in the sequence and step down by "1" to "-4", we could specify the start as "-1", the step as "-4", and the step as "-1":

```
sequence = "hello"
new_seq = sequence[-1:-4:-1]
print(new_seq) # prints "oll"
```

The above starts at the last item in the sequence and steps down by 1 until it reaches the item at index 0. It does not include the item at index -4 since the stop is *the first value that we do not want to include in the slice.*

What if we wanted to start at the last item in the sequence and step down by "1" to the first item in the sequence? One thought would be to set the start to "-1" the end of the sequence, the step to "-1" to step down through the sequence, and the index to "0" which is the index of the first item in the sequence. But

this will not work. The reason is, again, that the stop is *the first value that we do not want to include in the slice.* So, if we set the stop to "0", then the slice will not include the item at index "0". To illustrate, consider the following:

```
sequence = "hello"
new_seq = sequence[-1:0:-1]
print(new_seq) # prints "olle"
```

One way we could slice the sequence in reverse would be to count past the last item in the sequence. As the last item in the sequence is at index "-5", we could set the stop to "-6" which is the index of the item after the last item in the sequence.

```
sequence = "hello"
new_seq = sequence[-1:-6:-1]
print(new_seq) # prints "olleh"
```

However, as we will see in the next section, there is a much better way to slice a sequence in reverse.

Default Values for Start and Stop

Finally, we mentioned that the step defaults to "1". What about the *start* and *stop*? I find explanations of the default values for the start and stop arguments to be a bit confusing. Here is how I understand it. The default value of both the stop and start is `None`. The values are determined by the sign of the step size.

1. If the step is positive (e.g., 1), then the start is 0 and the stop is the length of the sequence (`len(sequence)`).
2. If the step is negative (e.g., -1), then the start is length of the sequence (`len(sequence)`) and the stop is `-(len(sequence) + 1)`

Let's consider the first case where we compare the default values of the start and stop when the step is positive. We'll use the string `hello world` as our sequence. The length of the sequence is 11. The step is 1.

```
sequence = "hello world"
seq_len = len(sequence) # 11

seq1 = sequence[::1]
seq2 = sequence[0:seq_len:1]

print(seq1) # prints "hello world"
print(seq2) # prints "hello world"
```

In the first case, we use the default values for the start and stop. In the second case, we specify the start and stop explicitly. In both cases, the slice returns the entire sequence.

In the second case, we'll use the string `testing` as our `sequence`, but this time

we'll use a negative step. In this case, we'll compare what is returned using several equivalent slices. The length of the sequence is 7. The step is -1.

```
sequence = "testing"
seq_len = len(sequence)
seq3 = sequence[None:None:-1]
seq4 = sequence[::-1]
seq5 = sequence[seq_len:-(seq_len + 1):-1]
seq6 = sequence[-1::-1]
print(seq3, seq4, seq5, seq6)
# prints "gnitset gnitset gnitset gnitset"
```

As we can see, all of the slices return the sequence in reverse. The first slice uses `None` and `None`. The second slice uses the default value for the start and the explicit value. The third slice uses the length of the sequence as the start and `-(seq_len + 1)`. The fourth slice uses the index of the last item in the sequence as the start and the default value for the stop.

Each reverses the sequence. But the slice `sequence[::-1]` is the most concise.