Python Slicing in Depth

Summary: in this tutorial, you'll learn about Python slicing and how to use it to extract data from and assign data to a sequence.

Python slicing review

So far you've learned about slicing such as list slicing (https://www.pythontutorial.net/python-basics/python-list-slice/) .

Technically, slicing relies on indexing. Therefore, slicing only works with sequence types (https://www.pythontutorial.net/advanced-python/python-sequences/) .

For mutable sequence types such as lists (https://www.pythontutorial.net/python-basics/python-list/) , you can use slicing to extract and assign data. For example:

```
colors = ['red', 'green', 'blue', 'orange']

# extract data
print(colors[1:3])

# assign data
```

```
colors[1:3] = ['pink', 'black']
print(colors)
```

Output:

```
['green', 'blue']
['red', 'pink', 'black', 'orange']
```

However, you can use slicing to extract data from immutable sequences. For example:

```
topic = 'Python Slicing'

# Extract data
print(topic[0:6])
```

Output:

```
Python
```

If you attempt to use slicing to assign data to an immutable sequence, you'll get an error. For example:

```
topic[0:6] = 'Java'
```

Error:

```
TypeError: 'str' object does not support item assignment
```

The slicing `seq[start:stop]` returns the elements starting at the index `start` up to the index `stop - 1`.

Therefore, it's easier to visualize that the indexes are between the elements when you slice the sequence:

Python slice type

Everything in Python is an object including the slice. A slice is actually an object of the `slice` type.

When you use the slicing notation:

```
seq[start:stop]
```

The `start:stop` is a `slice` object.

```
slice(start, stop)
```

For example:

```
s = slice(1, 3)

print(type(s))
print(s.start, s.stop)
```

Output:

```
<class 'slice'>
1 3
```

So instead of using the slicing notation:

```
colors[1:3]
```

... you can use the slice object instead:

```
colors = ['red', 'green', 'blue', 'orange']
s = slice(1, 3)

print(colors[s])
```

Output:

```
['green', 'blue']
```

Python Slicing: start and stop bounds

The slice `seq[start:stop]` selects elements start at the index `start` and stop at the index `stop` (excluding the element at the index `stop`).

In other words, it returns all elements of the sequence at the index n where n satisfies the following expression:

```
start <= n < stop
```

When `start` or `stop` is greater than the length of the sequence:

```
len(seq)
```

... Python uses `len(seq)` for the `start` or `stop` .

Both `start` and `stop` are optional. The `start` defaults to 0 and `stop` defaults to `len(seq)` when you don't specify it.

The following example returns the entire list:

```
colors = ['red', 'green', 'blue', 'orange']
print(colors[0:100])
```

Output:

```
['red', 'green', 'blue', 'orange']
```

Since the stop bound is 100, Python uses the `len(colors)` for the stop bound.

The following example returns an empty list:

```
colors = ['red', 'green', 'blue', 'orange']
print(colors[10:])
```

Because the start bound is 10, Python assigns the `len(colors)` to it.

Negative start and stop bounds

The slice object also accepts negative `start` and `stop` bounds.

The following example uses the negative start and stop bounds to slice a list:

```
colors = ['red', 'green', 'blue', 'orange']

print(colors[-4:-2])
```

Output:

```
['red', 'green']
```

To get the `'blue'` and `'orange'` elements from the `colors` list, you can combine the negative and positive bounds:

```
colors = ['red', 'green', 'blue', 'orange']
print(colors[-2:4])
```

Output:

```
['blue', 'orange']
```

The step value

Slices support the third argument, which is the step value. The step value defaults to 1 if you don’t specify it:

```
seq[star:stop:step]
```

It’s equivalent to:

```
s = slice(start, stop, step)
seq[s]
```

See the following example:

```
colors = ['red', 'green', 'blue', 'orange']
print(colors[0:4:2])
```

Output:

```
['red', 'blue']
```

The indices method

A slice object essentially defines a sequence of indices for selecting elements of a sequence.

To make it more convenient, the `slice` type has the `indices` method that returns the equivalent range `(start, stop, step)` for any slice of a sequence with a specified length:

```
slice(start, stop, step).indices(length)
```

It return a new tuple:

```
(i, j, k)
```

And you can use the values of this tuple to generate a list of indices using the `range` function. For example:

```
colors = ['red', 'green', 'blue', 'orange']

s = slice(0, 4, 2)
t = s.indices(len(colors))

for index in range(*t):
    print(colors[index])
```

Output:

```
red
blue
```

How it works.

- First, create a slice object whose start is 0, stop is 4, and step is 2.
- Second, return a tuple of indices of the slice of the sequence whose length is the length of the colors list.
- Third, pass the result tuple to the range function to select elements from the colors list.

Summary

- Slicing only works for sequence types including mutable and immutable sequences.
- A slice is an object the `slice` type.