# Indexing for Sequence and Collection in Python 3

K. S. Ooi
*Foundation in Science*
*Faculty of Health and Life Sciences*
*INTI International University*
*Persiaran Perdana BBN, Putra Nilai,*
*71800 Nilai, Negeri Sembilan, Malaysia*
E-mail: kuansan.ooi@newinti.edu.my
dr.k.s.ooi@gmail.com

## Abstract

String slicing technique can be applied to only one kind of Python 3 iterables. In Python 3, there are two kinds of iterables, the sequence and the collection. The string slicing technique can only be applied to sequence, and can not be applied to collection. For example, keys of dictionary are keys that uniquely identifies their respective values. Any values other than the keys cannot be used as identifiers. The implementation of Python in this respect is therefore clean.

**Keywords**: Python 3, iterables, slicing, sequence, collection
**Date**: Dec 25, 2020

## 1. Slicing the Sequence

In a previous article [1], I discuss string slicing, using three indices: *start*, *end*, and *step*. When it comes to Python *iterables*, how much of that knowledge can be applied here? Let us use a list with eleven elements, a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. The following table the results of applying the previous knowledge in string [KSOoi2020] into list.

**Table 1**: Slicing the list a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] and *n* = len(a) = 11.

| Print Slice | Result | Comment |
|---|---|---|
| print(a) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | All the print statements output the whole list |
| print(a[:]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[::]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[0:]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[0:n]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[:n]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[-n:n]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[-n:]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |
| print(a[0:n:1]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11] | |

| | | |
|---|---|---|
| print(a[:-1]) | [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] | Output the list, but exclude the last element. |
| print(a[1:6]) | [2, 3, 4, 5, 6] | Output a list with 5 elements, starting from the second element. |
| print(a[:6]) | [1, 2, 3, 4, 5, 6] | Output a list with the first 6 elements. |
| print(a[-6:]) | [6, 7, 8, 9, 10, 11] | Output a list with the last 6 elements. |
| print(a[2:-3]) | [3, 4, 5, 6, 7, 8] | Output the list, excluding the first 2 and the last 3. |
| print(a[-6:-2]) | [6, 7, 8, 9] | Output the list with the last 6 elements, but excluding the last 2. |
| print(a[-6:7]) | [6, 7] | Use the formula *i* – *n* = -6 to get a positive value. Since *n* is 11, we have *i* = 5. So, output the 5th and 6th elements. |
| print(a[::-1]) | [11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1] | Output the reversed list. |
| print(a[1:8:2])<br>print(a[1:-3:2])<br>print(a[-10:8:2])<br>print(a[-10:-3:2]) | [2, 4, 6, 8]<br>[2, 4, 6, 8]<br>[2, 4, 6, 8]<br>[2, 4, 6, 8] | Output the by selecting 7 elements starting from the 2nd element, but skip an element in between. All negative should be converted to positive numbers by the formula *i* – *n* = -*m*, where -*m* is the negative number. |
| print(a[9:1:-2])<br>print(a[9:-10:-2])<br>print(a[-2:1:-2])<br>print(a[-2:-10:-2]) | [10, 8, 6, 4]<br>[10, 8, 6, 4]<br>[10, 8, 6, 4]<br>[10, 8, 6, 4] | Output the list reversed, starting from the 10th element, till the 3rd element, and skip one element in between. Again, you had better convert the negative numbers into positive ones. Of course, do not convert the negative numbers assigned to *step*! |
| print(a[::-2])<br>print(a[-1::-2])<br>print(a[10::-2])<br>print(a[10:-12:-2]) | [11, 9, 7, 5, 3, 1]<br>[11, 9, 7, 5, 3, 1]<br>[11, 9, 7, 5, 3, 1]<br>[11, 9, 7, 5, 3, 1] | Output the list reversed, skip an element in between. Let me repeat: you had better convert the negative numbers into positive ones. Of course, do not convert the negative numbers assigned to *step*! |

So, we can conclude that slicing a string is the same as slicing an iterable.

## 2. Slicing of Collection?

Let us use a dictionary as an example of collection.

```
a = {1:"One", 2:"Two", 3:"Three", 4:"Four", 5:"Five", 6:"Six", 7:"Seven", 8:"Eight",
     9:"Nine", 10:"Ten", 11:"Eleven"}
```

Indexing the *key* is straightforward. However, if the values of the dictionary are iterables, slicing as we did with string and iterables is valid. So, Program 1 output all the values of dictionary a, from “One” to “Eleven”.

**Program 1**

```
a = {1:"One",
   2:"Two",
   3:"Three",
   4:"Four",
   5:"Five",
   6:"Six",
   7:"Seven",
   8:"Eight",
   9:"Nine",
   10:"Ten",
   11:"Eleven",
  }

for i in range(1,12):
  print(a[i])
```

In Program 2, the output is the value of item with the key 4, but since the value is an iterable, we can print it reversed.

**Program 2**

```
a = {1:[0,1],
   2:[0,1,2],
   3:[0,1,2,3],
   4:[0,1,2,3,4],
   5:[0,1,2,3,4,5],
   6:[0,1,2,3,4,5,6],
   7:[0,1,2,3,4,5,6,7],
   8:[0,1,2,3,4,5,6,7,8],
   9:[0,1,2,3,4,5,6,7,8,9],
   10:[0,1,2,3,4,5,6,7,8,9,10],
   11:[0,1,2,3,4,5,6,7,8,9,11],
  }

print(a[4][::-1])
```

## 3. Conclusion

In this experimental study of Python, the result points to consistency of implementation. Sequences, do not have unique identifiers, and therefore we must be able to slice them like strings, as string is a sequence of characters. This is not true for collection. For example, the collection dictionary has keys. Keys are unique identifiers, and therefore their values cannot be sliced like string.

## References

1. K. S. Ooi, *Python String Slicing*, ePrint: https://vixra.org/abs/2012.0177 (2020)