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 cannot 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 \(n = \text{len}(a) = 11\).

<table>
<thead>
<tr>
<th>Print Slice</th>
<th>Result</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>print(a)</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td>All the print statements output the whole list</td>
</tr>
<tr>
<td>print(a[:])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>print(a::&lt;])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>print(a[0:1])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>print(a[0:n])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>print(a[-n:n])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>print(a[-n:])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>print(a[0:n+1])</td>
<td>[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]</td>
<td></td>
</tr>
<tr>
<td>Print Expression</td>
<td>Output</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>print(a[:-1])</code></td>
<td><code>[1, 2, 3, 4, 5, 6, 7, 8, 9]</code></td>
<td>Output the list, but exclude the last element.</td>
</tr>
<tr>
<td><code>print(a[1:6])</code></td>
<td><code>[2, 3, 4, 5, 6]</code></td>
<td>Output a list with 5 elements, starting from the second element.</td>
</tr>
<tr>
<td><code>print(a[:6])</code></td>
<td><code>[1, 2, 3, 4, 5, 6]</code></td>
<td>Output a list with the first 6 elements.</td>
</tr>
<tr>
<td><code>print(a[-6:])</code></td>
<td><code>[6, 7, 8, 9, 10, 11]</code></td>
<td>Output a list with the last 6 elements.</td>
</tr>
<tr>
<td><code>print(a[2:3])</code></td>
<td><code>[3, 4, 5, 6, 7, 8]</code></td>
<td>Output the list, excluding the first 2 and the last 3.</td>
</tr>
<tr>
<td><code>print(a[-6:-2])</code></td>
<td><code>[6, 7, 8, 9]</code></td>
<td>Output the list with the last 6 elements, but excluding the last 2.</td>
</tr>
<tr>
<td><code>print(a[-6:7])</code></td>
<td><code>[6, 7]</code></td>
<td>Use the formula <code>i – n = -6</code> to get a positive value. Since <code>n</code> is 11, we have <code>i = 5</code>. So, output the 5th and 6th elements.</td>
</tr>
<tr>
<td><code>print(a[:1])</code></td>
<td><code>[1, 2, 3, 4, 5, 6, 7, 8, 9]</code></td>
<td>Output the reversed list.</td>
</tr>
<tr>
<td><code>print(a[1:10:2])</code></td>
<td><code>[2, 4, 6, 8]</code></td>
<td>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 <code>i – n = -m</code>, where <code>-m</code> is the negative number.</td>
</tr>
<tr>
<td><code>print(a[10::2])</code></td>
<td><code>[10, 8, 6, 4]</code></td>
<td>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!</td>
</tr>
<tr>
<td><code>print(a[1:2])</code></td>
<td><code>[11, 9, 7, 5, 3, 1]</code></td>
<td>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!</td>
</tr>
</tbody>
</table>

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.

```python
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