PY0101EN-2-1-Tuples

January 17, 2019

```
<a href="https://cocl.us/PY0101EN_edx_add_top">
     <img src="https://s3-api.us-geo.objectstorage.softlayer.net/cf-courses-data/CognitiveClass/
</a>
```

Tuples in Python

Welcome! This notebook will teach you about the tuples in the Python Programming Language. By the end of this lab, you'll know the basics tuple operations in Python, including indexing, slicing and sorting.

Table of Contents

```
<ul>
    <li>
        <a href="#dataset">About the Dataset</a>
    </li>
    <li>
        <a href="#tuple">Tuples</a>
        <ul>
            <li><a href="index">Indexing</a></li>
            <li><a href="slice">Slicing</a></li>
            <li><a href="sort">Sorting</a></li>
        </ul>
    </li>
    <li>
        <a href="#escape">Quiz on Tuples</a>
    </li>
</ul>
<p>
    Estimated time needed: <strong>15 min</strong>
</p>
```

About the Dataset

Imagine you received album recommendations from your friends and compiled all of the recommandations into a table, with specific information about each album.

The table has one row for each movie and several columns:

- **artist** - Name of the artist
- **album** - Name of the album
- **released_year** - Year the album was released

- **length_min_sec** - Length of the album (hours,minutes,seconds)
- **genre** - Genre of the album
- **music_recording_sales_millions** - Music recording sales (millions in USD) on SONG://DATABASE
- **claimed_sales_millions** - Album's claimed sales (millions in USD) on SONG://DATABASE
- **date_released** - Date on which the album was released
- **soundtrack** - Indicates if the album is the movie soundtrack (Y) or (N)
- **rating_of_friends** - Indicates the rating from your friends from 1 to 10

The dataset can be seen below:

```
<th>Artist</th>
<th>Album</th>
<th>Released</th>
<th>Length</th>
<th>Genre</th>
<th>Music recording sales (millions)</th>
<th>Claimed sales (millions)</th>
<th>Released</th>
<th>Soundtrack</th>
<th>Rating (friends)</th>

<td>Michael Jackson</td>
<td>Thriller</td>
<td>1982</td>
<td>00:42:19</td>
<td>Pop, rock, R&B</td>
<td>46</td>
<td>65</td>
<td>30-Nov-82</td>
<td></td>
<td>10.0</td>

<td>AC/DC</td>
<td>Back in Black</td>
<td>1980</td>
<td>00:42:11</td>
<td>Hard rock</td>
<td>26.1</td>
<td>50</td>
<td>25-Jul-80</td>
<td></td>
<td>8.5</td>

<tr>
<td>Pink Floyd</td>
<td>The Dark Side of the Moon</td>
<td>1973</td>
<td>00:42:49</td>
```

```
<td>Progressive rock</td>
<td>24.2</td>
<td>45</td>
<td>01-Mar-73</td>
<td></td>
<td>9.5</td>

<tr>
<td>Whitney Houston</td>
<td>The Bodyguard</td>
<td>1992</td>
<td>00:57:44</td>
<td>Soundtrack/R&B, soul, pop</td>
<td>26.1</td>
<td>50</td>
<td>25-Jul-80</td>
<td>Y</td>
<td>7.0</td>

<tr>
<td>Meat Loaf</td>
<td>Bat Out of Hell</td>
<td>1977</td>
<td>00:46:33</td>
<td>Hard rock, progressive rock</td>
<td>20.6</td>
<td>43</td>
<td>21-Oct-77</td>
<td></td>
<td>7.0</td>

<tr>
<td>Eagles</td>
<td>Their Greatest Hits (1971-1975)</td>
<td>1976</td>
<td>00:43:08</td>
<td>Rock, soft rock, folk rock</td>
<td>32.2</td>
<td>42</td>
<td>17-Feb-76</td>
<td></td>
<td>9.5</td>

<tr>
<td>Bee Gees</td>
<td>Saturday Night Fever</td>
<td>1977</td>
<td>1:15:54</td>
<td>Disco</td>
```

```
<td>20.6</td>
<td>40</td>
<td>15-Nov-77</td>
<td>Y</td>
<td>9.0</td>

<tr>
<td>Fleetwood Mac</td>
<td>Rumours</td>
<td>1977</td>
<td>00:40:01</td>
<td>Soft rock</td>
<td>27.9</td>
<td>40</td>
<td>04-Feb-77</td>
<td></td>
<td>9.5</td>
```

Tuples

In Python, there are different data types: string, integer and float. These data types can all be contained in a tuple as follows:

Now, let us create your first tuple with string, integer and float.

```
In [1]: # Create your first tuple

        tuple1 = ("disco",10,1.2 )
        tuple1
```

```
Out[1]: ('disco', 10, 1.2)
```

The type of variable is a **tuple**.

```
In [2]: # Print the type of the tuple you created

        type(tuple1)
```

```
Out[2]: tuple
```

Indexing

Each element of a tuple can be accessed via an index. The following table represents the relationship between the index and the items in the tuple. Each element can be obtained by the name of the tuple followed by a square bracket with the index number:

We can print out each value in the tuple:

```
In [3]: # Print the variable on each index

        print(tuple1[0])
        print(tuple1[1])
        print(tuple1[2])
```

```
disco
10
1.2
```

We can print out the **type** of each value in the tuple:

```
In [4]: # Print the type of value on each index

        print(type(tuple1[0]))
        print(type(tuple1[1]))
        print(type(tuple1[2]))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

We can also use negative indexing. We use the same table above with corresponding negative values:

We can obtain the last element as follows (this time we will not use the print statement to display the values):

```
In [5]: # Use negative index to get the value of the last element

        tuple1[-1]
```

```
Out[5]: 1.2
```

We can display the next two elements as follows:

```
In [6]: # Use negative index to get the value of the second last element

        tuple1[-2]
```

```
Out[6]: 10
```

```
In [7]: # Use negative index to get the value of the third last element

        tuple1[-3]
```

```
Out[7]: 'disco'
```

Concatenate Tuples

We can concatenate or combine tuples by using the **+** sign:

```
In [8]: # Concatenate two tuples

        tuple2 = tuple1 + ("hard rock", 10)
        tuple2
```

```
Out[8]: ('disco', 10, 1.2, 'hard rock', 10)
```

We can slice tuples obtaining multiple values as demonstrated by the figure below:
Slicing
We can slice tuples, obtaining new tuples with the corresponding elements:

```
In [9]: # Slice from index 0 to index 2

        tuple2[0:3]

Out[9]: ('disco', 10, 1.2)
```

We can obtain the last two elements of the tuple:

```
In [10]: # Slice from index 3 to index 4

         tuple2[3:5]

Out[10]: ('hard rock', 10)
```

We can obtain the length of a tuple using the length command:

```
In [11]: # Get the length of tuple

         len(tuple2)

Out[11]: 5
```

This figure shows the number of elements:
Sorting
Consider the following tuple:

```
In [13]: # A sample tuple

         Ratings = (0, 9, 6, 5, 10, 8, 9, 6, 2)
```

We can sort the values in a tuple and save it to a new tuple:

```
In [14]: # Sort the tuple

         RatingsSorted = sorted(Ratings)
         RatingsSorted

Out[14]: [0, 2, 5, 6, 6, 8, 9, 9, 10]
```

Nested Tuple
A tuple can contain another tuple as well as other more complex data types. This process is called 'nesting'. Consider the following tuple with several elements:

```
In [15]: # Create a nest tuple

         NestedT =(1, 2, ("pop", "rock") ,(3,4),("disco",(1,2)))
```

Each element in the tuple including other tuples can be obtained via an index as shown in the figure:

```
In [16]: # Print element on each index

         print("Element 0 of Tuple: ", NestedT[0])
         print("Element 1 of Tuple: ", NestedT[1])
         print("Element 2 of Tuple: ", NestedT[2])
         print("Element 3 of Tuple: ", NestedT[3])
         print("Element 4 of Tuple: ", NestedT[4])
```

```
Element 0 of Tuple:  1
Element 1 of Tuple:  2
Element 2 of Tuple:  ('pop', 'rock')
Element 3 of Tuple:  (3, 4)
Element 4 of Tuple:  ('disco', (1, 2))
```

We can use the second index to access other tuples as demonstrated in the figure:
We can access the nested tuples :

```
In [17]: # Print element on each index, including nest indexes

         print("Element 2, 0 of Tuple: ",   NestedT[2][0])
         print("Element 2, 1 of Tuple: ",   NestedT[2][1])
         print("Element 3, 0 of Tuple: ",   NestedT[3][0])
         print("Element 3, 1 of Tuple: ",   NestedT[3][1])
         print("Element 4, 0 of Tuple: ",   NestedT[4][0])
         print("Element 4, 1 of Tuple: ",   NestedT[4][1])
```

```
Element 2, 0 of Tuple:  pop
Element 2, 1 of Tuple:  rock
Element 3, 0 of Tuple:  3
Element 3, 1 of Tuple:  4
Element 4, 0 of Tuple:  disco
Element 4, 1 of Tuple:  (1, 2)
```

We can access strings in the second nested tuples using a third index:

```
In [18]: # Print the first element in the second nested tuples

         NestedT[2][1][0]
```

```
Out[18]: 'r'
```

```
In [19]: # Print the second element in the second nested tuples

         NestedT[2][1][1]
```

```
Out[19]: 'o'
```

We can use a tree to visualise the process. Each new index corresponds to a deeper level in the tree:

Similarly, we can access elements nested deeper in the tree with a fourth index:

```
In [20]: # Print the first element in the second nested tuples

         NestedT[4][1][0]

Out[20]: 1

In [21]: # Print the second element in the second nested tuples

         NestedT[4][1][1]

Out[21]: 2
```

The following figure shows the relationship of the tree and the element NestedT[4][1][1]:

Quiz on Tuples

Consider the following tuple:

```
In [22]: # sample tuple

         genres_tuple = ("pop", "rock", "soul", "hard rock", "soft rock", \
                         "R&B", "progressive rock", "disco")
         genres_tuple

Out[22]: ('pop',
          'rock',
          'soul',
          'hard rock',
          'soft rock',
          'R&B',
          'progressive rock',
          'disco')
```

Find the length of the tuple, genres_tuple:

```
In [23]: len(genres_tuple)

Out[23]: 8
```

Double-click **here** for the solution.

Access the element, with respect to index 3:

```
In [24]: genres_tuple[3]

Out[24]: 'hard rock'
```