Lists in Python

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Perform list operations in Python, including indexing, list manipulation and copy/clone list.

Table of Contents

- About the Dataset
- Lists
    - Indexing
    - List Content
    - List Operations
    - Copy and Clone List
- Quiz on Lists

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

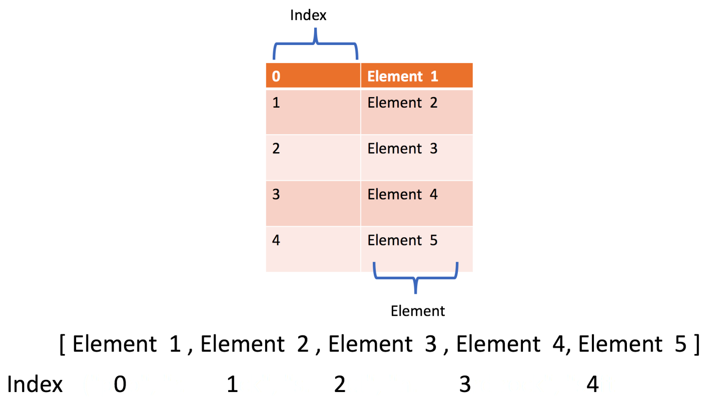
The dataset can be seen below:

<table font-size:xx-small> Artist Album Released Length Genre Music recording sales (millions) Claimed sales (millions) Released Soundtrack Rating (friends) Michael Jackson Thriller 1982 00:42:19 Pop, rock, R&B 46 65 30-Nov-82 10.0 AC/DC Back in Black 1980 00:42:11 Hard rock 26.1 50 25-Jul-80 8.5 Pink Floyd The Dark Side of the Moon 1973 00:42:49 Progressive rock 24.2 45 01-Mar-73 9.5 Whitney Houston The Bodyguard 1992 00:57:44 Soundtrack/R&B, soul, pop 26.1 50 25-Jul-80 Y 7.0 Meat Loaf Bat Out of Hell 1977 00:46:33 Hard rock, progressive rock 20.6 43 21-Oct-77 7.0 Eagles Their Greatest Hits (1971-1975) 1976 00:43:08 Rock, soft rock, folk rock 32.2 42 17-Feb-76 9.5 Bee Gees Saturday Night Fever 1977 1:15:54 Disco 20.6 40 15-Nov-77 Y 9.0 Fleetwood Mac Rumours 1977 00:40:01 Soft rock 27.9 40 04-Feb-77 9.5 </table>

Lists

Indexing

We are going to take a look at lists in Python. A list is a sequenced collection of different objects such as integers, strings, and other lists as well. The address of each element within a list is called an **index**. An index is used to access and refer to items within a list.



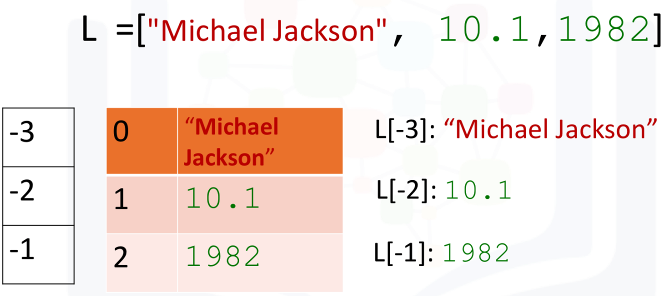
To create a list, type the list within square brackets **[ ]**, with your content inside the parenthesis and separated by commas. Let's try it!

```
In [1]: # Create a list

        L = ["Michael Jackson", 10.1, 1982]
        L
```

```
Out[1]: ['Michael Jackson', 10.1, 1982]
```

We can use negative and regular indexing with a list :



```
In [2]: # Print the elements on each index

        print('the same element using negative and positive indexing:\n Postive:',L[0],
        '\n Negative:' , L[-3]  )
        print('the same element using negative and positive indexing:\n Postive:',L[1],
        '\n Negative:' , L[-2]  )
        print('the same element using negative and positive indexing:\n Postive:',L[2],
        '\n Negative:' , L[-1]  )
```

```
the same element using negative and positive indexing:
 Postive: Michael Jackson
 Negative: Michael Jackson
the same element using negative and positive indexing:
 Postive: 10.1
 Negative: 10.1
the same element using negative and positive indexing:
 Postive: 1982
 Negative: 1982
```

List Content

Lists can contain strings, floats, and integers. We can nest other lists, and we can also nest tuples and other data structures. The same indexing conventions apply for nesting:

```
In [3]: # Sample List

        ["Michael Jackson", 10.1, 1982, [1, 2], ("A", 1)]
```

```
Out[3]: ['Michael Jackson', 10.1, 1982, [1, 2], ('A', 1)]
```

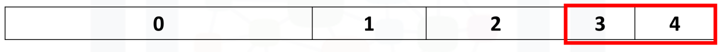
List Operations

We can also perform slicing in lists. For example, if we want the last two elements, we use the following command:

```
In [4]: # Sample List

        L = ["Michael Jackson", 10.1,1982,"MJ",1]
        L
```

```
Out[4]: ['Michael Jackson', 10.1, 1982, 'MJ', 1]
```



```
In [5]: # List slicing

        L[3:5]
```

```
Out[5]: ['MJ', 1]
```

We can use the method `extend` to add new elements to the list:

```
In [6]: # Use extend to add elements to list

        L = [ "Michael Jackson", 10.2]
        L.extend(['pop', 10])
        L
```

```
Out[6]: ['Michael Jackson', 10.2, 'pop', 10]
```

Another similar method is `append`. If we apply `append` instead of `extend`, we add one element to the list:

```
In [7]: # Use append to add elements to list

        L = [ "Michael Jackson", 10.2]
        L.append(['pop', 10])
        L
```

```
Out[7]: ['Michael Jackson', 10.2, ['pop', 10]]
```

Each time we apply a method, the list changes. If we apply `extend` we add two new elements to the list. The list `L` is then modified by adding two new elements:

In [8]:
```
# Use extend to add elements to list

L = [ "Michael Jackson", 10.2]
L.extend(['pop', 10])
L
```

Out[8]:
```
['Michael Jackson', 10.2, 'pop', 10]
```

If we append the list `['a','b']` we have one new element consisting of a nested list:

In [9]:
```
# Use append to add elements to list

L.append(['a','b'])
L
```

Out[9]:
```
['Michael Jackson', 10.2, 'pop', 10, ['a', 'b']]
```

As lists are mutable, we can change them. For example, we can change the first element as follows:

In [10]:
```
# Change the element based on the index

A = ["disco", 10, 1.2]
print('Before change:', A)
A[0] = 'hard rock'
print('After change:', A)
```

```
Before change: ['disco', 10, 1.2]
After change: ['hard rock', 10, 1.2]
```

We can also delete an element of a list using the `del` command:

In [11]:
```
# Delete the element based on the index

print('Before change:', A)
del(A[0])
print('After change:', A)
```

```
Before change: ['hard rock', 10, 1.2]
After change: [10, 1.2]
```

We can convert a string to a list using `split`. For example, the method `split` translates every group of characters separated by a space into an element in a list:

In [12]:
```
# Split the string, default is by space

'hard rock'.split()
```

Out[12]:
```
['hard', 'rock']
```

We can use the split function to separate strings on a specific character. We pass the character we would like to split on into the argument, which in this case is a comma. The result is a list, and each element corresponds to a set of characters that have been separated by a comma:

In [13]:
```
# Split the string by comma

'A,B,C,D'.split(',')
```

Out[13]:
```
['A', 'B', 'C', 'D']
```

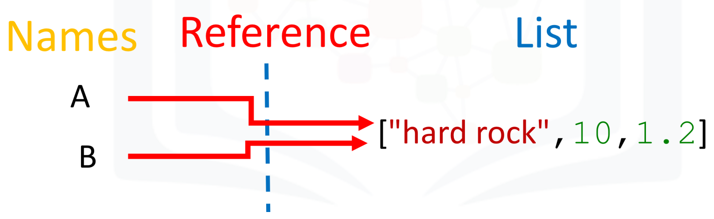
Copy and Clone List

When we set one variable **B** equal to **A**; both **A** and **B** are referencing the same list in memory:

In [14]:
```
# Copy (copy by reference) the list A

A = ["hard rock", 10, 1.2]
B = A
print('A:', A)
print('B:', B)
```

```
A: ['hard rock', 10, 1.2]
B: ['hard rock', 10, 1.2]
```



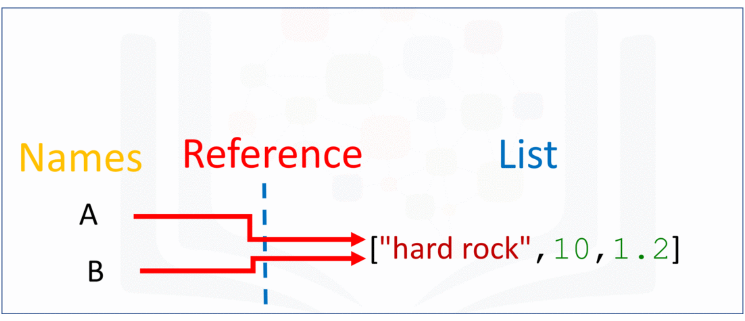
Initially, the value of the first element in **B** is set as hard rock. If we change the first element in **A** to **banana**, we get an unexpected side effect. As **A** and **B** are referencing the same list, if we change list **A**, then list **B** also changes. If we check the first element of **B** we get banana instead of hard rock:

In [15]:
```
# Examine the copy by reference

print('B[0]:', B[0])
A[0] = "banana"
print('B[0]:', B[0])
```

```
B[0]: hard rock
B[0]: banana
```

This is demonstrated in the following figure:



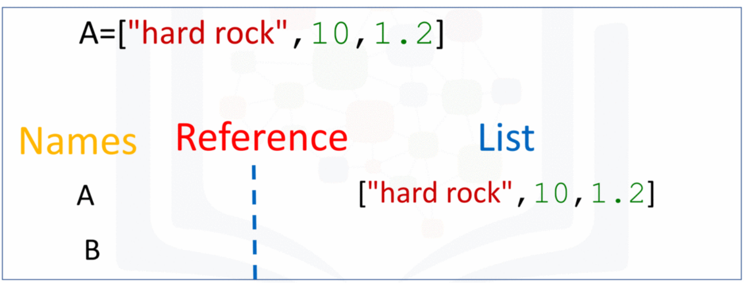
You can clone list **A** by using the following syntax:

In [16]:
```
# Clone (clone by value) the list A

B = A[:]
B
```

Out[16]:
```
['banana', 10, 1.2]
```

Variable **B** references a new copy or clone of the original list; this is demonstrated in the following figure:



Now if you change **A**, **B** will not change:

In [17]:
```
print('B[0]:', B[0])
A[0] = "hard rock"
print('B[0]:', B[0])
```

```
B[0]: banana
B[0]: banana
```

Quiz on List

Create a list `a_list`, with the following elements `1`, `hello`, `[1,2,3]` and `True`.

In [18]:
```
# Write your code below and press Shift+Enter to execute
a_list = [1, 'hello', [1, 2, 3], True]
```

▶ Click here for the solution

Find the value stored at index 1 of `a_list`.

In [19]:
```
# Write your code below and press Shift+Enter to execute
a_list[1]
```

Out[19]:
```
'hello'
```

▶ Click here for the solution

Retrieve the elements stored at index 1, 2 and 3 of `a_list`.

In [20]:
```
# Write your code below and press Shift+Enter to execute
a_list[1:4]
```

Out[20]:
```
['hello', [1, 2, 3], True]
```

▶ Click here for the solution

Concatenate the following lists `A = [1, 'a']` and `B = [2, 1, 'd']`:

In [21]:
```
# Write your code below and press Shift+Enter to execute
A = [1, 'a']
B = [2, 1, 'd']
A + B
```

Out[21]:
```
[1, 'a', 2, 1, 'd']
```

▶ Click here for the solution

The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow this article to learn how to share your work.

Author

Joseph Santarcangelo

Other contributors

Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab

© IBM Corporation 2020. All rights reserved.