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.

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:

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

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 even 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!

```
# Create a list

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

We can use negative and regular indexing with a list:

```
# 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]  )
```

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:

```
# Sample List

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

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

```
# Sample List

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

```
# List slicing

L[3:5]
```

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

```
# Use extend to add elements to list

L = [ "Michael Jackson", 10.2]
```

```
L.extend(['pop', 10])
L
```

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

```
# Use append to add elements to list

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

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:

```
# Use extend to add elements to list

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

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

```
# Use append to add elements to list

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

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

```
# Change the element based on the index

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

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

```
# Delete the element based on the index

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

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:

```
# Split the string, default is by space

'hard rock'.split()
```

We can use the split function to separate strings on a specific character which we call a **delimiter**. 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:

```
# Split the string by comma

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

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

```
# Copy (copy by reference) the list A

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

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":

```
# Examine the copy by reference

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

This is demonstrated in the following figure:

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

```
# Clone (clone by value) the list A

B = A[:]
B
```

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:

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

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

```
# Write your code below and press Shift+Enter to execute
```

Find the value stored at index 1 of a_list.

```
# Write your code below and press Shift+Enter to execute
```

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

```
# Write your code below and press Shift+Enter to execute
```

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

```
# Write your code below and press Shift+Enter to execute
```

Congratulations, you have completed your first lesson and hands-on lab in Python.

Author

Joseph Santarcangelo

Other contributors

Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-01-10	2.1	Malika	Removed the readme for GitShare
2020-08-26	2.0	Lavanya	Moved lab to course repo in GitLab