

# Lists

February 13, 2023



## 1 Lists in Python

- Slides by [Ahmed Moustafa](#)
- Content modified from [Pierian Data](#)

## 2 Git & GitHub Quick Review

- What is Git? What is it used for?
- What are the different components of the Git system?
- What are the function of the following commands, `commit`, `pull`, and `push`?
- On GitHub, what does **fork** do?
- On GitHub, what does **pull request** do?

## 3 Lists

- Lists can be thought of the most general version of a sequence in Python.
- Unlike strings, they are **mutable**, i.e. elements inside a list can be changed.
- Lists are constructed with brackets `[ ]` and commas `,` separating every element in the list.

## 4 Creating a List

```
[2]: weights = [65.0, 70.5, 72.3, 68.0, 77.2] # list of numbers
weights
```

```
[2]: [65.0, 70.5, 72.3, 68.0, 77.2]
```

```
[3]: cities = ["London", "Paris", "New York", "Tokyo", "Berlin"] # list of strings
cities
```

```
[3]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin']
```

```
[4]: types = [1, 2.5, "hello", "world", 42, "python"] # list of different data types
types
```

```
[4]: [1, 2.5, 'hello', 'world', 42, 'python']
```

## 5 List of Lists

We can also create a list of lists. For example, combining the two list we just created, `cities` and `weights` into a new list `my_list`:

```
[5]: my_list = [cities, weights]
my_list
```

```
[5]: [['London', 'Paris', 'New York', 'Tokyo', 'Berlin'],
      [65.0, 70.5, 72.3, 68.0, 77.2]]
```

```
[6]: len(my_list)
```

```
[6]: 2
```

## 6 Indexing and Slicing 1/3

Indexing and slicing work just like in strings:

```
[7]: cities[0]
```

```
[7]: 'London'
```

```
[8]: cities[1:]
```

```
[8]: ['Paris', 'New York', 'Tokyo', 'Berlin']
```

```
[9]: cities[::-1]
```

```
[9]: ['Berlin', 'Tokyo', 'New York', 'Paris', 'London']
```

```
[10]: cities + ["Cairo", "Alexandria"]
```

```
[10]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

## 7 Indexing and Slicing 2/3

```
[11]: cities
```

```
[11]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin']
```

```
[12]: cities += ["Cairo", "Alexandria"]
      cities
```

```
[12]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[13]: cities * 2
```

```
[13]: ['London',
      'Paris',
      'New York',
      'Tokyo',
      'Berlin',
      'Cairo',
      'Alexandria',
      'London',
      'Paris',
      'New York',
      'Tokyo',
      'Berlin',
      'Cairo',
      'Alexandria']
```

## 8 Indexing and Slicing 3/3

```
[14]: my_list
```

```
[14]: [['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria'],
      [65.0, 70.5, 72.3, 68.0, 77.2]]
```

```
[15]: len(my_list)
```

```
[15]: 2
```

```
[16]: my_list[0]
```

```
[16]: ['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[17]: my_list[1][2]
```

```
[17]: 72.3
```

## 9 List Methods: append

The `append()` method adds an item to the end of the list

```
[18]: print(cities)
      len(cities)
```

```
['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[18]: 7
```

```
[19]: cities.append("Aswan")  
print(cities)  
len(cities)
```

```
['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria',  
'Aswan']
```

```
[19]: 8
```

## 10 List Methods: pop

The `pop()` method removes the item at the given index from the list and returns the removed item

```
[20]: cities.pop() # pop (remove) the last element
```

```
[20]: 'Aswan'
```

```
[21]: print(cities)  
len(cities)
```

```
['London', 'Paris', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[21]: 7
```

```
[22]: cities.pop(1) # pop (remove) at the given index
```

```
[22]: 'Paris'
```

```
[23]: print(cities)  
len(cities)
```

```
['London', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[23]: 6
```

## 11 List Methods: reverse

The `reverse()` method reverses the elements of the list

```
[24]: print("Before: ", cities)
```

```
Before:  ['London', 'New York', 'Tokyo', 'Berlin', 'Cairo', 'Alexandria']
```

```
[25]: cities.reverse()
```

```
[26]: print("After: ", cities)
```

After: ['Alexandria', 'Cairo', 'Berlin', 'Tokyo', 'New York', 'London']

## 12 List Methods: sort

The `sort()` method sorts the items of a list in ascending or descending order

```
[27]: cities.sort()  
cities
```

```
[27]: ['Alexandria', 'Berlin', 'Cairo', 'London', 'New York', 'Tokyo']
```

## 13 List Methods: index

The `index()` method returns the index of the specified element in the list

```
[28]: cities.index("Tokyo")
```

```
[28]: 5
```

```
[29]: # cities.index("Dubai")
```

## 14 Lists Exercise

GitHub Classroom Assignment [https://classroom.github.com/a/a24f\\_RDP](https://classroom.github.com/a/a24f_RDP)



## 15 Dictionaries

Dictionaries in Python is a form of mapping, between keys and their corresponding value

## 16 Constructing a Dictionary

```
[31]: my_dict = {'key1': 'value1', 'key2': 'value2'} # Make a dictionary with {} and :  
      ↪ to signify a key and a value  
      my_dict
```

```
[31]: {'key1': 'value1', 'key2': 'value2'}
```

```
[32]: my_dict['key2'] # Call values by their key
```

```
[32]: 'value2'
```

```
[33]: my_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}  
      ↪#dictionaries are very flexible  
      my_dict['key3']
```

```
[33]: ['item0', 'item1', 'item2']
```

## 17 Dictionary Methods

```
[34]: d = {'key1':1,'key2':2,'key3':3} # Create a typical dictionary  
      d
```

```
[34]: {'key1': 1, 'key2': 2, 'key3': 3}
```

```
[35]: d.keys() # Method to return a list of all keys
```

```
[35]: dict_keys(['key1', 'key2', 'key3'])
```

```
[36]: d.values() # Method to grab all values
```

```
[36]: dict_values([1, 2, 3])
```

```
[37]: d.items() # Method to return tuples of all items (we'll learn about tuples  
      ↪soon)
```

```
[37]: dict_items([('key1', 1), ('key2', 2), ('key3', 3)])
```