Download Chapter notebook (ipynb)

- How are arrays useful in Python?
- Why do we need Numpy arrays?
- How can array data be iterated?

- Understanding arrays in Python
- Explaining the use of Numpy arrays
- Iterations on arrays

Note

Due to time restrictions, we will only cover the basic Python concepts required for the rest of this session. However, if you want to learn more about Python programming, please refer to the Software Carpentries full day course on Programming in Python.

Arrays

List

We can use variables to store individual data in Python. However, in data science we often need to access multiple values to perform operations. In such cases, defining a variable for every single value becomes tedious. So instead, we can use arrays.

Arrays are variables that hold any number of values. Python provides 3 types of built-in arrays: `list`, `tuple`, and `set`. There are a several shared features among all arrays in Python; however, each type of array enjoys its own range of unique features that each facilitate specific operations.

`list` is the most frequently used type of arrays in Python. The easiest way to explain how a `list` works is to think of it as a table that can have any number of rows. This is akin to a spreadsheet with one column.

Here is how to create a list using square brackets:

```
1 my_list = [1, 2, 3]
```

Type the name and execute a cell to see its content.

```
1 my_list
```

```
1 [1, 2, 3]
```

To access an item in a list, we use the name of the list, followed by square brackets. Inside the brackets we provide the index of the item requested. In Python, indices start from 0.

```
my_list[2]
```

```
3
```

The last item in an array is referred to with index −1

```
my_list[-1]
```

```
3
```

If the index refers to a non-existing item, an error will be “thrown” (in Python speak).

```
my_list[3]
```

```
Error: IndexError: list index out of range
```

All items will be shown with the colon `:`

```
my_list[:]
```

```
[1, 2, 3]
```

Get segments of slices using index values to the left or right of the colon. Left is inclusive of, right excludes the item with the index specified.

```
my_list[:2]
```

```
[1, 2]
```

```
my_list[1:]
```

```
[2, 3]
```

If you want to find index of a particular number:

```
my_list.index(3)
```

```
2
```

You can obtain the number of items in a list using the function `len`

Arguments of functions are provided in round parentheses.

```
no_items = len(my_list)

```

```
3 no_items
```

```
1 3
```

Note

A `list` is a so-called mutable type of array. This means that it is possible to change a list's content. In contrast, if you create an array with round parentheses, it is immutable, and is referred to as a `tuple`. This cannot be changed in its current location in memory.

More items can also be added in a list using the `append` function.

```
1 my_list.append(4)
2
3 my_list
```

```
1 [1, 2, 3, 4]
```

```
1 my_list.append('Something')
2
3 my_list
```

```
1 [1, 2, 3, 4, 'Something']
```

```
1 my_list[-1]
```

```
1 'Something'
```

Similarly, items can also be deleted from a list using the `remove` function.

```
1 my_list.remove('Something')
2
3 my_list
```

```
1 [1, 2, 3, 4]
```

Numpy Arrays

Problems with Lists

Observe the results of executing the following code:

```
1 my_list = [1, 2, 3]
2
```

```
print(my_list)
```

```
[1, 2, 3]
```

```
print(my_list*2)
```

```
[1, 2, 3, 1, 2, 3]
```

One might expect each item in the list to be squared. Instead, the number of items is doubled.

Or:

```
print(my_list)
```

```
[1, 2, 3]
```

```
print(my_list + 10)
```

```
Error: TypeError: can only concatenate list (not "int") to list
```

Looking at this, it might be expected that 10 would be added to each item in the list. Instead, an error is thrown. This, and other features of `list` are not that intuitive.

In data science and machine learning, we therefore predominantly use `Numpy` arrays.

Converting a List to a Numpy Array

First we import a function from the Numpy library:

```
from numpy import array
```

```
my_np_array = array(my_list)

my_np_array
```

```
array([1, 2, 3])
```

This is more intuitive:

```
print(my_np_array)
```

```
[1 2 3]
```

```
print(my_np_array*2)
```

```
[2 4 6]
```

Similarly:

```
print(my_np_array)
```

```
[1 2 3]
```

```
print(my_np_array + 10)
```

```
[11 12 13]
```

In order to get the maximum and the index of the maximum in the array:

```
my_np_array.max()
```

```
3
```

```
my_np_array.argmax()
```

```
2
```

Indexing is with square brackets, as for `list` and `tuple`.

Click here to read more about `Numpy` arrays:

https://www.nature.com/articles/s41586-020-2649-2

Iterations

Iteration with a For Loop

Let us firstly create a `list` first and then iterate through it using a **`for`** loop.

```
my_list = [1, 22, 333, 4444, 55555]

my_list
```

```
[1, 22, 333, 4444, 55555]
```

Perform a set of operations on each item in the list. Display the item on your screen, using `print`.

```
for number in my_list:

    print(number)
```

```
1
22
333
```

```
4444
55555
```

To directly get the indices in a **for** Loop, use `enumerate`. This firstly provides the index, followed by the value of the item.

```
for index, item in enumerate(my_list):

    print('Index:', index, '    Item:', item)
```

```
Index: 0     Item: 1
Index: 1     Item: 22
Index: 2     Item: 333
Index: 3     Item: 4444
Index: 4     Item: 55555
```

If you want to print the positions of items in the list, remember to add `1` in the index and array start from index `0`.

```
for index, item in enumerate(my_list):

    print('Position:', index+1, '    Item:', item)
```

```
Position: 1     Item: 1
Position: 2     Item: 22
Position: 3     Item: 333
Position: 4     Item: 4444
Position: 5     Item: 55555
```

Keypoints

- `list` is a mutable types of arrays.
- `NumPy` is a Python library optimised to deal for large, multi-dimensional arrays and matrices.
- Repetitive execution of the same block of code over and over is referred to as `iteration`.