- 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 restriction, we will only cover the basic Python concepts which are needed for the rest of session. However, if you want to learn more on Python programming, please refer to the Carpenteries 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 occasions, defining a variable for every single value is tedious. Instead, we 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 common features amongst all arrays in Python; however, each type of array enjoys its own range of unique features that facilitate specific operations.

`list` is the most frequently used type of arrays in Python. It is therefore important to understand how it works.

The easiest way to imagine 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.

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

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

```
my_list
```

```
[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 ne "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 incliding, right excludsing 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
```

Get the number of items in a list with the function `len`

Arguments of functions are provided in round parenthesis.

```
no_items = len(my_list)

```

```
no_items
```

```
3
```

Note

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

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

```
my_list.append(4)

my_list
```

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

```
my_list.append('Something')

my_list
```

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

```
my_list[-1]
```

```
'Something'
```

Similary, items can also be deleted by using `remove` function.

```
my_list.remove('Something')

my_list
```

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

Numpy Arrays

Problem with List

Check the results of executing the following code:

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

```

```
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
```

Here, One might expect 10 to be added to each item in the list. Instead, an error is thrown. This, and other features of `list` are not intuitive.

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

Convert List to 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]
```

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`.

Read here about `Numpy` arrays:

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

Iterations

Iteration with a For Loop

Let us create a `list` first and then iterate through it.

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

my_list
```

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

Perform a set of operations on each item of the list. As an example, we want the item printed on the 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. It provides first the index, then the value of an 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 position 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 one of the type of arrays (`list`, `tuple` and `set`).
- `NumPy` is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices.
- Repetitive execution of the same block of code over and over is referred to as `iteration`.