9. Library - NumPy

Table of Contents

1. Introduction

Install Numpy

- Windows

```
pip install numpy
```

- Macbook

```
pip3 install numpy
```

What is NumPy?

- NumPy stands for *Numerical Python*.
- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- It is an open source project and you can use it freely.

Why Use NumPy?

- In Python we have lists that serve the purpose of arrays, but they are slow to process.
- *NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.*
- The array object in NumPy is called `ndarray`, it provides a lot of supporting functions that make working with `ndarray` very easy.
- Arrays are very frequently used in data science, where speed and resources are very important.

2. Create a basic array

- Once NumPy is installed, import it in your applications by adding the `import` keyword:

```
import numpy as np
```

- To create a NumPy array, you can use the function `np.array()` and pass a list to it.

```
>>> import numpy as np
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

- We can create an array filled with 0's and 1's:

```
>>> import numpy as np
>>> np.zeros(2)
array([0., 0.])
>>> np.ones(2)
array([1., 1.])
```

- Sometimes, we need an empty array, which we need to fill some numbers later. The initial content in the empty array is *random* and depends on the state of our computer memory.

```
>>> import numpy as np
>>> np.empty(2)
array([2.05833592e-312, 2.33419537e-312])
```

- We can create an array with a range of elements:

```
>>> import numpy as np
>>> np.arange(4)
array([0, 1, 2, 3])
```

- Or even an array that contains of certain (e.g., evenly spaced) intervals.

```
>>> import numpy as np
>>> np.arange(2, 9, 2)
array([2, 4, 6, 8])
>>> np.arange(2, 9, 3)
array([2, 5, 8])
```

- We can also to create an array with values that are spaced linearly in a specified interval by using `np.linspace()` . Note that the boundaries would be included in the output, and the length of the output equals the parameter `num`.

```
>>> import numpy as np
>>> np.linspace(0, 10, num=6)
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

3. Indexing and slicing

- Indices in NumPy arrays are quite similar to those in lists.
- Aslo, `len` function still works here.

```
>>> import numpy as np
>>> data = np.array([1, 2, 3, 4])
>>> len(data)
4
>>> data[0]
1
>>> data[1:]
array([2, 3, 4])
>>> data[:2]
array([1, 2])
```

'Negative' index

- We can use negative index to access the element by counting inversely from the end. It also works in lists.

```
>>> import numpy as np
>>> data = np.array([1, 2, 3, 4])
>>> data[-1]
4
>>> data[-2]
3
>>> data[:-2]
array([1, 2])
>>> data[-2:]
array([3, 4])
```

4. Operations - 1: Basic

- With NumPy, we can perform easily addition, subtraction, multiplication, division, and more for a list of values.
- For example, if we would like to perform operations on the corresponding elements in two list, we need to use loop.

```
data_1 = [1, 2, 3]
data_2 = [11, 21, 31]
```

```
l = []
for i in range(len(data_1)):
    l.append(data_1[i] + data_2[i])

print(l)
```

- However, with NumPy we can easily finish it.

```
import numpy as np
data_1 = [1, 2, 3]
data_2 = [11, 21, 31]

data_1, data_2 = np.array(data_1), np.array(data_2)

print(data_1 + data_2)
print(data_1 - data_2)
print(data_1 * data_2)
```

5. Operations - 2: Broadcasting

- There are times when you might want to carry out an operation between an array and a single number.
- For example, our array (we'll call it "data") might contain information about distance in miles, but you want to convert the information to kilometers. You can perform this operation with:

```
>>> import numpy as np
>>> data = np.array([1, 2, 3])
>>> data * 1.6
array([1.6, 3.2, 4.8])
```

6. Conditionals used in NumPy

- With broadcasting, we can obtain an array with boolean values. For example, we have a data, and would like to use which values are larger than a particular value.

```
>>> import numpy as np
>>> data = np.array([23, 2, 9, 8, 54, 89, 23])
>>> data > 10
array([ True, False, False, False,  True,  True,  True])
>>> data[data>10]
array([23, 54, 89, 23])
```

- In the above program, `data>10` gives us the boolean values indicating whether an element in the array is larger than 10. Then, in `data[data>10]`, with the bracket operator we can obtain the values

in `data` that satisfy the condition `>10`. It is so convenient and efficient by avoiding loops.

- It can be applied to select certain values that fulfill any condition. For example, we discussed how to obtain even/odd number before.

```
>>> data = [23, 2, 9, 8, 54, 89, 23]
>>> odd_number = [i for i in data if i%2 != 0]
>>> odd_number
[23, 9, 89, 23]
```

- With Numpy, it is much easier to obtain the results.

```
>>> data = [23, 2, 9, 8, 54, 89, 23]
>>> data = np.array(l)
>>> odd_number = data[data%2!=0]
>>> odd_number
array([23,  9, 89, 23])
```

7. Statistical calculations

- NumPy also performs aggregation functions. In addition to `min`, `max`, and `sum`, you can easily run `mean` to get the average, `prod` to get the result of multiplying the elements together, `std` to get the standard deviation, and more.
- For example, we can use the following methods to obtain the statistics, `sum`, `min`, `max`, `mean`, `std`.

```
>>> import numpy as np
>>> data = np.array([23, 2, 9, 8, 54, 89, 23])
>>> data.sum()
208
>>> data.min()
2
>>> data.max()
89
>>> data.mean()
29.714285714285715
>>> data.std()
28.941884062051393
```

- We can use the following functions in NumPy to obtain quartiles (Q1-25th percentile, Q2 - 50th percentile (median), Q3 - 75th percentile)

```
>>> import numpy as np
>>> data = np.array([23, 2, 9, 8, 54, 89, 23])
>>> np.percentile(data, 25)
8.5
>>> np.percentile(data, 50)
```

```
23.0
>>> np.percentile(data, 75)
38.5
```

Practice questions

1. Counting. Given a list, try to get the number of positive numbers.
2. Calculate the area of a circle, given a list of values of radius.
3. Given the following to lists, one includes the student names, another one includes the ages of the students. Try to get the students whose ages are greater than 20.

```
names = ['Jack', 'Mark', 'Mary', 'Jenny', 'April', 'Jin']
ages = [23, 18, 21, 19, 22, 19]
```

4. Given the following grades about homework, project, exam_1 and exam_2, and the weights for the three assignments are 20%, 20%, 30%, and 30%, try to calculate the percentage of A (>=90), B (<90 and >=80), C (<80 and >=70), D (<70 and >=60), and F(<60).

```
homework = [81, 83, 89, 98, 70, 71, 72, 91, 80, 61, 50]
project = [90, 92, 85, 82, 84, 86, 83, 79, 70, 81, 60]
exam_1 = [83, 70, 78, 90, 82, 88, 68, 59, 59, 75, 77]
exam_2 = [82, 73, 60, 65, 95, 88, 68, 59, 62, 75, 50]
```

References

- https://numpy.org/doc/stable/user/absolute_beginners.html
- https://www.w3schools.com/python/numpy/numpy_intro.asp