Numpy explained- Language of Numpy

medium.com/thrive-in-ai/numpy-explained-language-of-numpy-972cc7c99042

Rohit Raj

March 2, 2022

Numpy is the most important library for numerical processing in python. There are many great tutorials available that explain the details of functions of Numpy.

For learning Numpy basics I refer to Numpy official documentation and the following book.

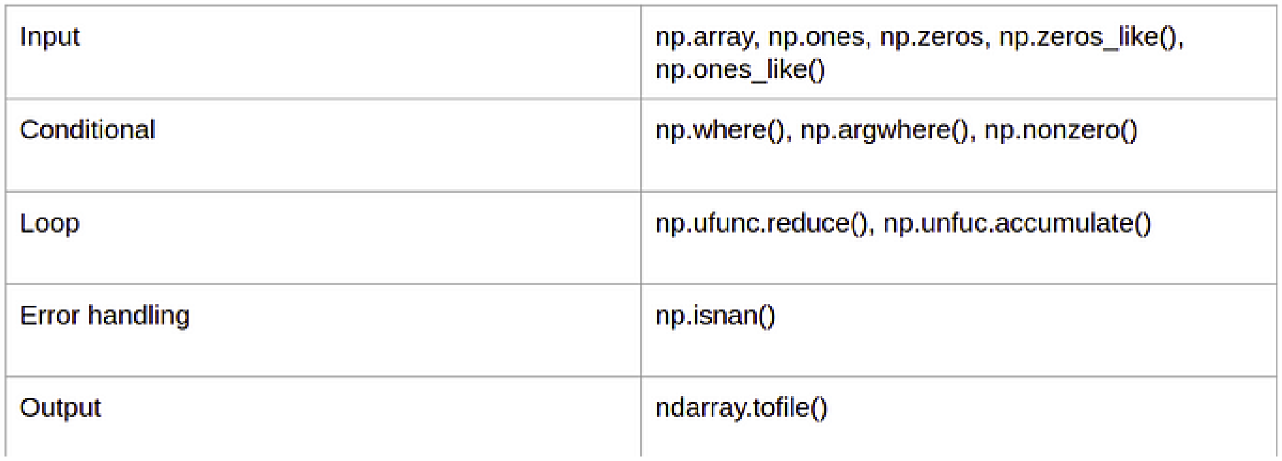
https://jakevdp.github.io/PythonDataScienceHandbook/index.html

But many times you stumble upon complicated tasks in numpy where if you look for answers in stack overflow you will find them to belong formulas spanning multiple lines.

In this article, I explain how you can make complicated expressions using Numpy.

First, the most important thing to understand is python itself is much slower than Numpy. So you have to avoid python for loops and use vectorization and Numpy universal functions as much as possible to speed up your code.

The key to cracking complicated problems in numpy is to understand numpy as a programming language itself. Programming language can be broken into key components like loops, error handling, conditional statements etc. There are numpy functions for each of the components.



Input	np.array, np.ones, np.zeros, np.zeros_like(), np.ones_like()
Conditional	np.where(), np.argwhere(), np.nonzero()
Loop	np.ufunc.reduce(), np.unfuc.accumulate()
Error handling	np.isnan()
Output	ndarray.tofile()

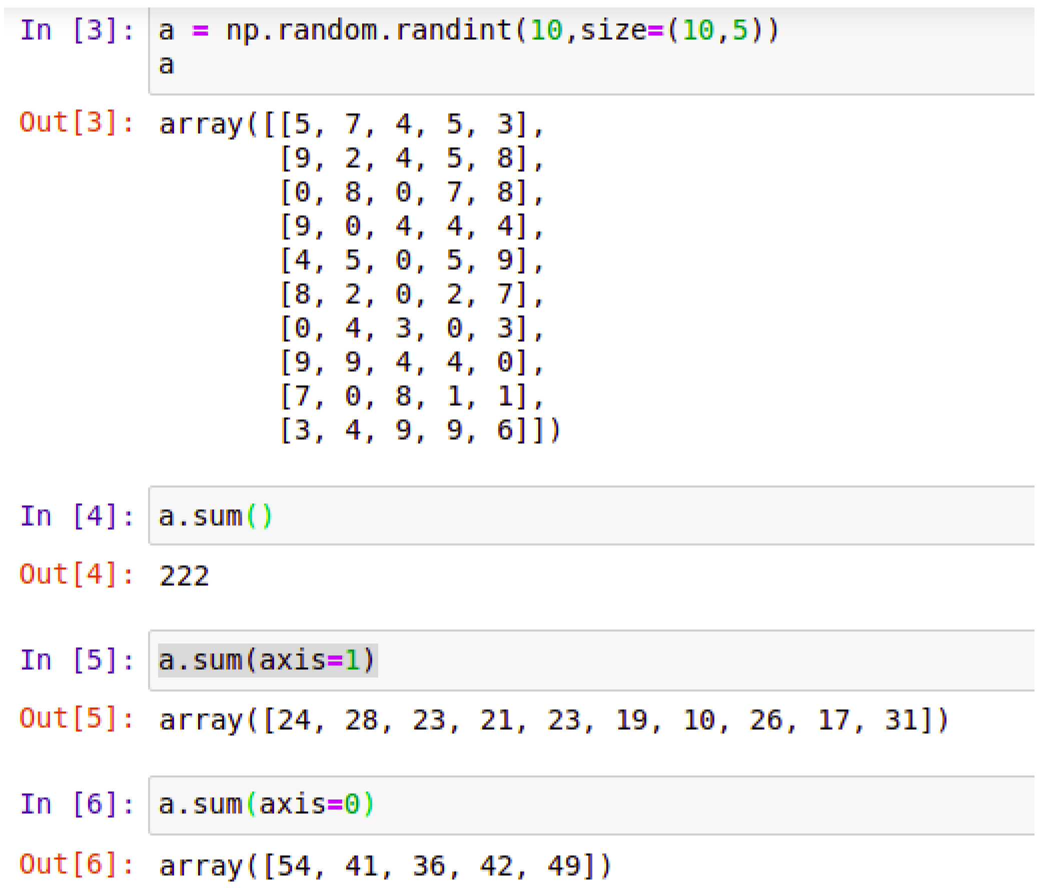
Numpy has more than 60 universal functions for arithmetic operations, logical comparisons, and other mathematical functions. For a detailed list of numpy universal functions you can refer to the link below:

Universal functions (ufunc) - NumPy v1.22 Manual

All ufuncs take optional keyword arguments. Most of these represent advanced usage and will not typically be used. The…

numpy.org

Let's start with sum universal function. We can apply sum universal function across axis in the following way:



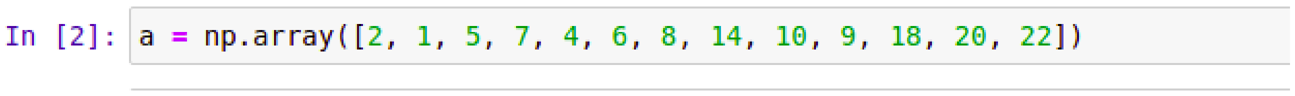
```
In [3]: a = np.random.randint(10,size=(10,5))
        a

Out[3]: array([[5, 7, 4, 5, 3],
               [9, 2, 4, 5, 8],
               [0, 8, 0, 7, 8],
               [9, 0, 4, 4, 4],
               [4, 5, 0, 5, 9],
               [8, 2, 0, 2, 7],
               [0, 4, 3, 0, 3],
               [9, 9, 4, 4, 0],
               [7, 0, 8, 1, 1],
               [3, 4, 9, 9, 6]])

In [4]: a.sum()

Out[4]: 222

In [5]: a.sum(axis=1)

Out[5]: array([24, 28, 23, 21, 23, 19, 10, 26, 17, 31])

In [6]: a.sum(axis=0)

Out[6]: array([54, 41, 36, 42, 49])
```

We can apply every universal across the axis of our choice by passing value to the axis parameter.

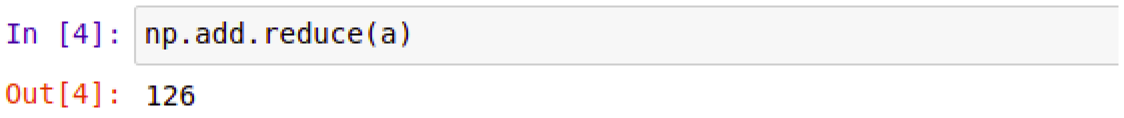
Except for the universal functions which take only one input like np.negative, every universal function supports the following five methods.

1. ufunc.reduce(): Reduces array's dimension by one, by applying ufunc along one axis.

The following function applies add function along with elements of the above array

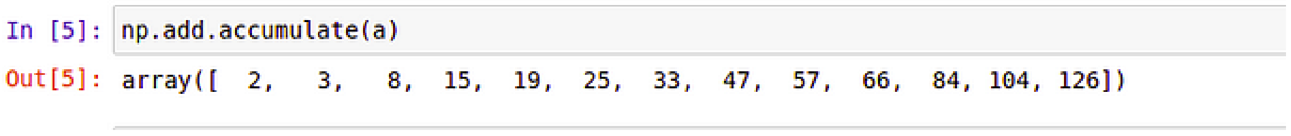


```
In [2]: a = np.array([2, 1, 5, 7, 4, 6, 8, 14, 10, 9, 18, 20, 22])
```



```
In [4]: np.add.reduce(a)

Out[4]: 126
```

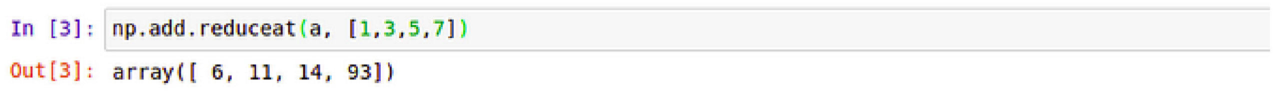
2. ufunc.accumulate(): Accumulate the result of applying the operator to all elements.



```
In [5]: np.add.accumulate(a)
Out[5]: array([  2,   3,   8,  15,  19,  25,  33,  47,  57,  66,  84, 104, 126])
```

The above function applies add function on rolling basis along elements of the array.

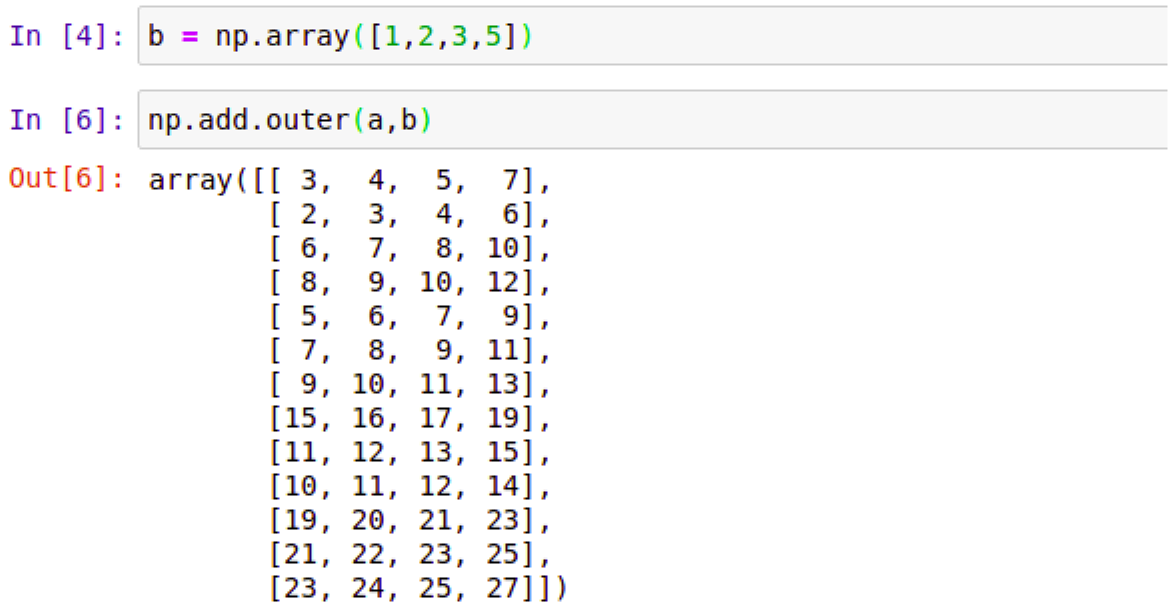
3. ufunc.reduceat(): Performs a (local) reduce with specified slices over a single axis.



```
In [3]: np.add.reduceat(a, [1,3,5,7])
Out[3]: array([ 6, 11, 14, 93])
```

Applies add function cumulatively on specified indices.

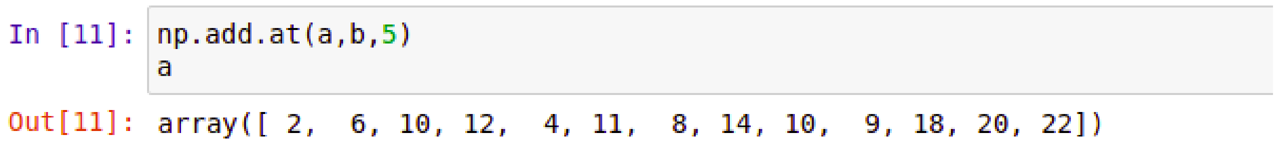
4. ufunc.outer(): Apply the ufunc op to all pairs (a, b) with a in A and b in B.



```
In [4]: b = np.array([1,2,3,5])

In [6]: np.add.outer(a,b)

Out[6]: array([[ 3,  4,  5,  7],
               [ 2,  3,  4,  6],
               [ 6,  7,  8, 10],
               [ 8,  9, 10, 12],
               [ 5,  6,  7,  9],
               [ 7,  8,  9, 11],
               [ 9, 10, 11, 13],
               [15, 16, 17, 19],
               [11, 12, 13, 15],
               [10, 11, 12, 14],
               [19, 20, 21, 23],
               [21, 22, 23, 25],
               [23, 24, 25, 27]])
```

Performs additions of elements of arrays a and b for each of their element. The shape of array a is (13,1). The shape of array b is (4,1) so the above operation returns an array of size (13,4)

5. ufunc.at(): Performs unbuffered in place operation on operand ‘a’ for elements specified by ‘indices’.



```
In [11]: np.add.at(a,b,5)
         a

Out[11]: array([ 2,  6, 10, 12,  4, 11,  8, 14, 10,  9, 18, 20, 22])
```

above command adds 5 to array ‘a’ at indices specified by array a

Next, I show we can combine conditional operators with numpy universal functions.

We can use conditional operators of python on numpy arrays

```
In [13]: a>5

Out[13]: array([False, False, False,  True, False,  True,  True,  True,  True,
                True,  True,  True,  True])
```

```
In [14]: a<10

Out[14]: array([ True,  True,  True,  True,  True,  True,  True, False, False,
                True, False, False, False])
```

We can also do element-wise comparisons using python operators or numpy universal functions

```
In [16]: b = np.random.randint(5,size=(13,))
         b

Out[16]: array([1, 4, 4, 0, 1, 3, 4, 3, 4, 2, 1, 2, 3])
```

```
In [17]: a>b

Out[17]: array([ True, False,  True,  True,  True,  True,  True,  True,  True,
                 True,  True,  True,  True])
```

```
In [18]: np.greater(a,b)

Out[18]: array([ True, False,  True,  True,  True,  True,  True,  True,  True,
                 True,  True,  True,  True])
```

We can also combine various conditions using logical and/or.

```
In [22]: np.logical_and(a>5 , a<12)

Out[22]: array([False, False,  True,  True, False,  True,  True, False,  True,
                True, False, False, False])
```

```
In [19]: np.logical_or(a>5 , np.mod(a, 2)==0)

Out[19]: array([ True, False, False,  True,  True,  True,  True,  True,  True,
                 True,  True,  True,  True])
```

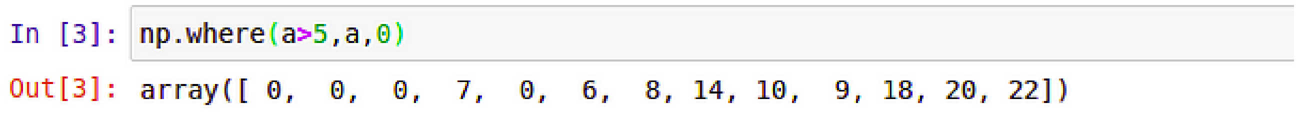
We can obtain the index of array where the condition is true by using nonzero function

```
In [21]: np.logical_or(a>5 , np.mod(a, 2)==0).nonzero()

Out[21]: (array([ 0,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12]),)
```

We can use the output of nonzero function to obtain elements of the array for which the conditions are true

```
In [22]: c= np.logical_or(a>5 , np.mod(a, 2)==0).nonzero()
         a[c]

Out[22]: array([ 2,  7,  4,  6,  8, 14, 10,  9, 18, 20, 22])
```

We can also implement if else type of functionality using np.where function. The following function replaces element of the array with 0 for which condition is not true.

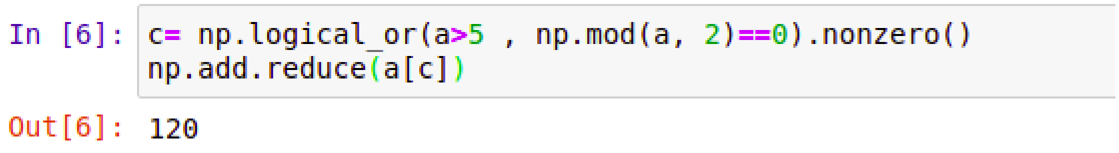


```
In [3]: np.where(a>5,a,0)
Out[3]: array([ 0,  0,  0,  7,  0,  6,  8, 14, 10,  9, 18, 20, 22])
```

2. **Combining cumulative and conditional operator**

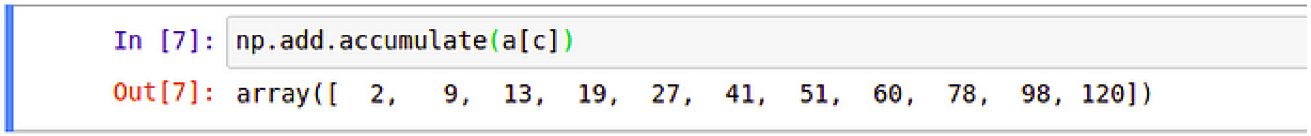
We can combine conditional function along with reduce functions of numpy universal functions.

We can add elements for which conditions are true by the following function



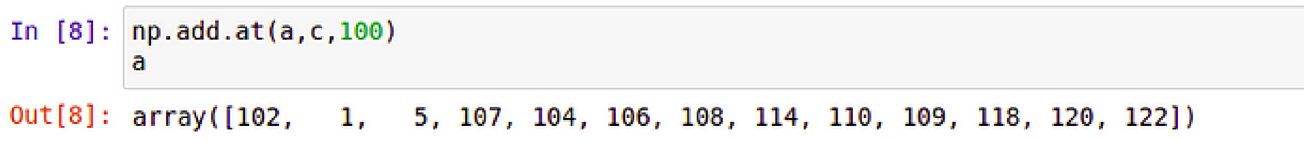
```
In [6]: c= np.logical_or(a>5 , np.mod(a, 2)==0).nonzero()
        np.add.reduce(a[c])
Out[6]: 120
```

We can obtain the cumulative sum of elements for which conditions are true



```
In [7]: np.add.accumulate(a[c])
Out[7]: array([  2,   9,  13,  19,  27,  41,  51,  60,  78,  98, 120])
```

If we want to retain the value of array where the condition is not true then we can use .at method of universal functions. The following function will add 100 to array where the condition is true



```
In [8]: np.add.at(a,c,100)
        a
Out[8]: array([102,   1,   5, 107, 104, 106, 108, 114, 110, 109, 118, 120, 122])
```

Summary

Using numpy functions as shown above you can solve any complicated problem in numpy.

If you liked the article, please like and subscribe to my newsletter.