python+numpy

July 7, 2021

1 Python and Numpy

TechX 2021

1.1 Introduction

Numpy is a basic package for scientific computing. It is a **Python** language implementation which includes:

- The powerful N-dimensional array structure
- Sophisticated functions
- Tools that can be integrated into C/C++ and Fortran code
- Linear algebra, Fourier transform and random number features

In addition to being used for scientific computing, NumPy also can be used as an efficient multi-dimensional container for general data. Because it can work with any type of data, NumPy can be integrated into multiple types of databases seamlessly and efficiently.

This notebook mainly introduce the basic usage of Numpy in Python and how it can help us analyze data efficiently.

1.2 Basic Operation of Numpy

We begin by importing numpy from the python library. Make sure that you've installed numpy or anaconda to use the package.

[2]:
```
# !pip install numpy
import numpy as np
```

Different ways to create arrays in numpy:

[3]:
```
np.zeros(10, dtype="int") #The code segement create a length-10 integer array␣
↪filled with zeros
```

[3]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[5]:
```
np.zeros((5,6),dtype="float") #5*6 zero matrix with float data type
```

[5]:
```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
```

```
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

[6]:
```
np.full((3,5),3.14,dtype="float") #creating 3*5 array filled with 3.14
```

[6]:
```
array([[3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14]])
```

[7]:
```
# Create an array filled with a linear sequence (similiar to range() function)

np.arange(0,20,2)
```

[7]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

[8]:
```
# Create a 3*3 array with random value

np.random.random((3,3))
```

[8]:
```
array([[0.06102396, 0.73204281, 0.45035684],
       [0.10359489, 0.01084566, 0.35060221],
       [0.39855614, 0.80070236, 0.46779828]])
```

[9]:
```
# Create a 3*3 array of normally distributed random values with mean 0 and std 1

np.random.normal(0,1,(3,3))
```

[9]:
```
array([[ 1.35478344, -0.9435556 , -0.14172825],
       [ 1.58091535, -0.31263571, -0.59702689],
       [ 1.84729697, -1.61422102, -1.08839475]])
```

[10]:
```
# Create a 3*3 integer arrays in interval [0,10)

np.random.randint(0,10,(3,3))
```

[10]:
```
array([[3, 4, 3],
       [7, 6, 9],
       [2, 2, 1]])
```

[11]:
```
# Create a 3*3 identity matrix

np.eye(3)
```

[11]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Basic array manipulations: * Attributes of arrays * Determining the size, shape, memory consumption, and data types of arrays * Indexing of arrays * Getting and setting the value of

individual array elements * Slicing of arrays * Getting and setting smaller subarrays within a larger array * Reshaping of arrays * Changing the shape of a given array * Joining and splitting of arrays * Combining multiple arrays into one, and splitting one array into many

[15]:
```
np.random.seed(0) #seed for reproducibility
x1 = np.random.randint(10,size = 6)
print("x1 ndim:",x1.ndim)
print("x1 shape:", x1.shape)
print("x1 size:",x1.size)
print("x1 dtype:", x1.dtype)
print("x1 itemsize:",x1.itemsize,"bytes")  #Size of each element
print("x1 nbytes:",x1.nbytes,"bytes") #Total size
```

```
x1 ndim: 1
x1 shape: (6,)
x1 size: 6
x1 dtype: int64
x1 itemsize: 8 bytes
x1 nbytes: 48 bytes
```

[16]:
```
# Array indexing

print(x1)
print(x1[0])
print(x1[-1]) #Indexing from the end of the array
```

```
[5 0 3 3 7 9]
5
9
```

[23]:
```
x2 = np.random.randint(20,size=(3,4))
print(x2)
print(x2[2,1])
print(x2[-2][-3])
```

```
[[ 7  0  1  9]
 [ 0 10  3 11]
 [18  2  0  0]]
2
10
```

[24]:
```
print(x2)
x2[0][0] = 100  #Modify value using index notations
print(x2)
```

```
[[ 7  0  1  9]
 [ 0 10  3 11]
 [18  2  0  0]]
[[100   0   1   9]
```

```
 [  0  10   3  11]
 [ 18   2   0   0]]
```

[25]:
```
print(x1)
x1[0] = 3.1415926  #This will be truncated
print(x1)
```

```
[5 0 3 3 7 9]
[3 0 3 3 7 9]
```

[26]:
```
# Array Slicing
x = np.arange(10)
x
```

[26]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[27]:
```
x[:5] #first five element
```

[27]:
```
array([0, 1, 2, 3, 4])
```

[28]:
```
x[4:7] #sub array
```

[28]:
```
array([4, 5, 6])
```

[29]:
```
x[::2] #slicing every 2 element
```

[29]:
```
array([0, 2, 4, 6, 8])
```

[30]:
```
x[1::2] #same as above, starting from 1
```

[30]:
```
array([1, 3, 5, 7, 9])
```

[31]:
```
x[::-1] #reversing array
```

[31]:
```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

[32]:
```
## for multidimensional arrays
x2
```

[32]:
```
array([[100,   0,   1,   9],
       [  0,  10,   3,  11],
       [ 18,   2,   0,   0]])
```

[33]:
```
x2[:3,::2] #all rows, first and third column
```

[33]:
```
array([[100,   1],
       [  0,   3],
       [ 18,   0]])
```

[34]:
```
x2[::-1,::-1] #all reversed
```

[34]:
```
array([[  0,   0,   2,  18],
       [ 11,   3,  10,   0],
       [  9,   1,   0, 100]])
```

[35]:
```
# reshaping arrays
x2
```

[35]:
```
array([[100,   0,   1,   9],
       [  0,  10,   3,  11],
       [ 18,   2,   0,   0]])
```

[36]:
```
x2.reshape(4,3)
```

[36]:
```
array([[100,   0,   1],
       [  9,   0,  10],
       [  3,  11,  18],
       [  2,   0,   0]])
```

[37]:
```
# Array concatenation and splitting

x = np.array([1,2,3])
y = np.array([3,2,1])
np.concatenate((x,y))
```

[37]:
```
array([1, 2, 3, 3, 2, 1])
```

[38]:
```
a = np.array([[1,2,3],
              [4,5,6]])
np.concatenate((a,a))
```

[38]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

[39]:
```
np.concatenate((a,a),axis=1) #concatenate along the second axis
```

[39]:
```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

[40]:
```
# Splitting of arrays

x = [1,2,3,99,99,3,2,1]

x1,x2,x3 = np.split(x,[3,5])
print(x1,x2,x3)
```

```
[1 2 3] [99 99] [3 2 1]
```

[41]:
```
a = np.arange(36).reshape((6,6))
a
```

[41]:
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

[45]:
```
upper, middle, lower = np.vsplit(a, [2,3])
print("upper:",upper)
print("middle:",middle)
print("lower:",lower)
```

```
upper: [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
middle: [[12 13 14 15 16 17]]
lower: [[18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
```

[46]:
```
#Array arithmetic
# Very similiar to native arithmetic operators

x = np.arange(4)
print("x+5", x+5)
print("x // 2", x // 2)
```

```
x+5 [5 6 7 8]
x // 2 [0 0 1 1]
```

[47]:
```
print(np.add(3,2))

print(np.add(x,2)) #Addition +
print(np.subtract(x,5)) #Subtraction -
print(np.negative(x)) #Unary negation -
print(np.multiply(x,3)) #Multiplication *
print(np.divide(x,2)) #Division /
print(np.floor_divide(x,2)) #Floor division //
print(np.power(x,2)) #Exponentiation **
print(np.mod(x,2)) #Modulus/remainder **

print(np.multiply(x, x))
```

```
5
[2 3 4 5]
```

```
[-5 -4 -3 -2]
[ 0 -1 -2 -3]
[0 3 6 9]
[0.  0.5 1.  1.5]
[0 0 1 1]
[0 1 4 9]
[0 1 0 1]
[0 1 4 9]
```

[49]:
```
# summing values in array
print(np.sum(x))
print(sum(x))
```

[49]:
```
6
```

[50]:
```
#The only difference between two functions is the speed
big_array = np.random.rand(100000)
%timeit sum(big_array)
%timeit np.sum(big_array)
```

```
13.2 ms ± 210 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
22.1 µs ± 57.3 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

[51]:
```
np.min(big_array), np.max(big_array)
```

[51]:
```
(4.011685566074341e-06, 0.9999993688244624)
```