NP02_The_Basics_of_NumPy_Arrays

December 19, 2022

1 The Basics of NumPy Arrays

[1]:
```
import numpy as np
np.__version__
```

[1]:
```
'1.23.5'
```

1.1 NumPy Array Attributes

[2]:
```
np.random.seed(0) # seed for reproducibility
```

[3]:
```
x1 = np.random.randint(10,size=6) # one-dimensional array
x2 = np.random.randint(10,size=(3,4)) # two-dimensional array
x3 = np.random.randint(10,size=(3,4,5)) # three-dimensional array
```

[4]:
```
print("x3 ndim :",x3.ndim)
print("x3 shape :",x3.shape)
print("x3 size :",x3.size)
```

```
x3 ndim : 3
x3 shape : (3, 4, 5)
x3 size : 60
```

[5]:
```
print("dtype :",x3.dtype)
```

```
dtype : int32
```

[6]:
```
print("itemsize :",x3.itemsize,"bytes")
print("nbytes :",x3.nbytes,"bytes")
```

```
itemsize : 4 bytes
nbytes : 240 bytes
```

1.2 Array Indexing: Accessing Single Elements

[7]:
```
x1
```

[7]:
```
array([5, 0, 3, 3, 7, 9])
```

[8]:
```
x1[0]
```

[8]:
```
5
```

[9]:
```
x1[4]
```

[9]:
```
7
```

[10]:
```
x1[-1]
```

[10]:
```
9
```

[11]:
```
x2
```

[11]:
```
array([[3, 5, 2, 4],
       [7, 6, 8, 8],
       [1, 6, 7, 7]])
```

[12]:
```
x2[0,0]
```

[12]:
```
3
```

[13]:
```
x2[2,0]
```

[13]:
```
1
```

[14]:
```
x2[2,-1]
```

[14]:
```
7
```

[15]:
```
x2[0,0] = 2
x2
```

[15]:
```
array([[2, 5, 2, 4],
       [7, 6, 8, 8],
       [1, 6, 7, 7]])
```

[16]:
```
x1[0] = 3.14159 # this will be truncated
x1
```

[16]:
```
array([3, 0, 3, 3, 7, 9])
```

1.3 Array Slicing: Accessing Subarrays

1.3.1 One-dimensional Subarrays

[17]:
```
x = np.arange(10)
x
```

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

```
[18]: x[:5] # first 5 elements
```

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

```
[19]: x[5:] # elements after index 5
```

```
[19]: array([5, 6, 7, 8, 9])
```

```
[20]: x[4:7] # middle subarray
```

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

```
[21]: x[::2] # every other element
```

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

```
[22]: x[1::2] # every other element starting at index 1
```

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

```
[23]: x[::-1] # all elements reversed
```

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

```
[24]: x[5::-2] # reversed every other from index 5
```

```
[24]: array([5, 3, 1])
```

1.3.2 Multidimensional Subarrays

```
[25]: x2
```

```
[25]: array([[2, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])
```

```
[26]: x2[:2,:3] # two rows and three columns
```

```
[26]: array([[2, 5, 2],
             [7, 6, 8]])
```

```
[27]: x2[:3,::2] # three rows and every other column
```

```
[27]: array([[2, 2],
             [7, 8],
             [1, 7]])
```

```
[28]: x2[::1,::-1] # subarrays reversed
```

```
[28]: array([[4, 2, 5, 2],
             [8, 8, 6, 7],
             [7, 7, 6, 1]])
```

```
[29]: x2[::1,::-1] # arrays and subarray reversed together
```

```
[29]: array([[4, 2, 5, 2],
             [8, 8, 6, 7],
             [7, 7, 6, 1]])
```

1.3.3 Accessing Array Rows and Columns

```
[30]: x2
```

```
[30]: array([[2, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])
```

```
[31]: x2[:,0] # first column of x2
```

```
[31]: array([2, 7, 1])
```

```
[32]: x2[0,:] # first row of x2
```

```
[32]: array([2, 5, 2, 4])
```

```
[33]: x2[0] # equivalent to x2[0,:]
```

```
[33]: array([2, 5, 2, 4])
```

Subarrays as No-Copy Views

```
[34]: x2
```

```
[34]: array([[2, 5, 2, 4],
             [7, 6, 8, 8],
             [1, 6, 7, 7]])
```

```
[35]: x2_sub = x2[:2,:2] # array slicing returns views rather than copies of the data
      x2_sub
```

```
[35]: array([[2, 5],
             [7, 6]])
```

```
[36]: x2_sub[0,0] = 99
      x2_sub
```

```
[36]: array([[99,  5],
              [ 7,  6]])
```

```
[37]: x2 # original array is also changed
```

```
[37]: array([[99,  5,  2,  4],
              [ 7,  6,  8,  8],
              [ 1,  6,  7,  7]])
```

1.3.4 Creating Copies of Arrays

```
[38]: x2
```

```
[38]: array([[99,  5,  2,  4],
              [ 7,  6,  8,  8],
              [ 1,  6,  7,  7]])
```

```
[39]: x2_sub = x2[:2,:2].copy() # copy() method is used to explicitly copy data␣
        ↪within an array
      x2_sub
```

```
[39]: array([[99,  5],
              [ 7,  6]])
```

```
[40]: x2_sub[0,0] = 42
      x2_sub
```

```
[40]: array([[42,  5],
              [ 7,  6]])
```

```
[41]: x2 # original array is not changed
```

```
[41]: array([[99,  5,  2,  4],
              [ 7,  6,  8,  8],
              [ 1,  6,  7,  7]])
```

1.4 Reshaping Arrays

```
[42]: grid = np.arange(1,10)
      grid
```

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

```
[43]: grid = grid.reshape((3,3)) # reshape() is used for reshaping arrays
      grid
```

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

```
[44]: x = np.array([1,2,3])
      x
```

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

```
[45]: x.reshape((1,3)) # row vector via reshape
```

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

```
[46]: x.reshape((3,1)) # column vector via reshape
```

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

```
[47]: x[np.newaxis,:] # row vector via newaxis
```

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

```
[48]: x[:,np.newaxis] # column vector via newaxis
```

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

1.5 Array Concatenation and Splitting

1.5.1 Concatenation of Arrays

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

      np.concatenate([x,y]) # np.concatenate() takes a tuple or list of arrays as its␣
       ↪first argument
```

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

```
[50]: z = np.array([99,99,99])
      np.concatenate([x,y,z])
```

```
[50]: array([ 1,  2,  3,  3,  2,  1, 99, 99, 99])
```

```
[51]: grid = np.array([[1,2,3],[4,5,6]])
      grid
```

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

```
[52]: np.concatenate([grid,grid]) # concatenate along the first axis
```

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

```
[53]: np.concatenate([grid,grid],axis=1) # concatenate along the second axis␣
       ↪(zero-indexed)
```

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

```
[54]: np.vstack([x,grid]) # vertically stack the arrays (along the first axis)
```

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

```
[55]: w = np.array([[99],[98]])

      np.hstack([grid,w]) # horizontally stack the arrays (along the second axis)
```

```
[55]: array([[ 1,  2,  3, 99],
             [ 4,  5,  6, 98]])
```

```
[56]: np.dstack([grid,grid]) # diagonally stack the arrays (along the third axis)
```

```
[56]: array([[[1, 1],
              [2, 2],
              [3, 3]],

             [[4, 4],
              [5, 5],
              [6, 6]]])
```

1.5.2 Splitting of Arrays

```
[57]: x = [1,2,3,99,99,3,2,1]

      x1,x2,x3 = np.split(x,[3,5]) # split(x,[3,5]) splits x at position 3 and 5
      print(x1,x2,x3)
```

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

```
[58]: grid = np.arange(16).reshape([4,4])
      grid
```

[58]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

[59]:
```
upper,lower = np.vsplit(grid,[2]) # vsplit(grid,[2]) splits grid vertically at␣
 ↪position 2
print(upper)
print(lower)
```

```
[[0 1 2 3]
 [4 5 6 7]]
[[ 8  9 10 11]
 [12 13 14 15]]
```

[60]:
```
left,right = np.hsplit(grid,[2]) # hsplit(grid,[2]) splits grid horizontally at␣
 ↪position 2
print(left)
print(right)
```

```
[[ 0  1]
 [ 4  5]
 [ 8  9]
 [12 13]]
[[ 2  3]
 [ 6  7]
 [10 11]
 [14 15]]
```

[61]:
```
y = np.arange(16).reshape([2,2,4])
y
```

[61]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],

       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]]])
```

[62]:
```
d1,d2 = np.dsplit(y,[2]) # dsplit(y,[2]) splits y along third axis at position 2
print(d1)
print(d2)
```

```
[[[ 0  1]
  [ 4  5]]

 [[ 8  9]
  [12 13]]]
[[[ 2  3]
  [ 6  7]]
 [[10 11]
  [14 15]]]
```