NumPy Basics: Arrays and Vectorized Computation

Part 3

The NumPy ndarray: A Multidimensional Array Object

Part 3

Indexing with slices

- Like one-dimensional objects such as Python lists, ndarrays can be sliced with the familiar syntax:

```
In [61]: arr
Out[61]: array([ 0,  1,  2,  3,  4, 64, 64, 64,  8,  9])

In [62]: arr[1:6]
Out[62]: array([ 1,  2,  3,  4, 64])
```

- Consider the two-dimensional array from before, `arr2d`.
- Slicing this array is a bit different:

```
In [63]: arr2d
Out[63]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [64]: arr2d[:2]
Out[64]: array([[1, 2, 3],
                [4, 5, 6]])
```

- As you can see, it has sliced along axis 0, the first axis.
- A slice, therefore, selects a range of elements along an axis.
- It can be helpful to read the expression `arr2d[:2]` as “select the first two rows of `arr2d`.”

- You can pass multiple slices just like you can pass multiple indexes:

```
In [65]: arr2d
Out[65]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [66]: arr2d[:2, 1:]
Out[66]: array([[2, 3],
                [5, 6]])
```

- When slicing like this, you always obtain array views of the same number of dimensions.

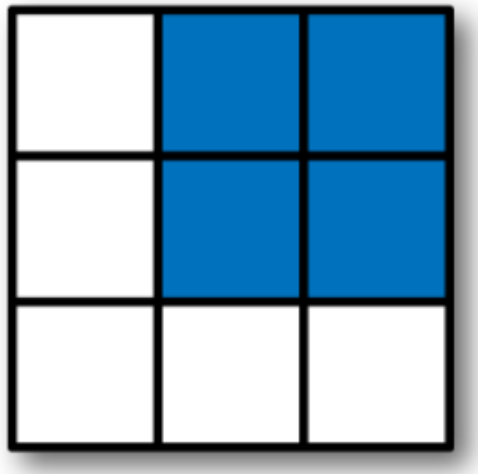
- By mixing integer indexes and slices, you get lower dimensional slices.
- For example, I can select the second row but only the first two columns like so:

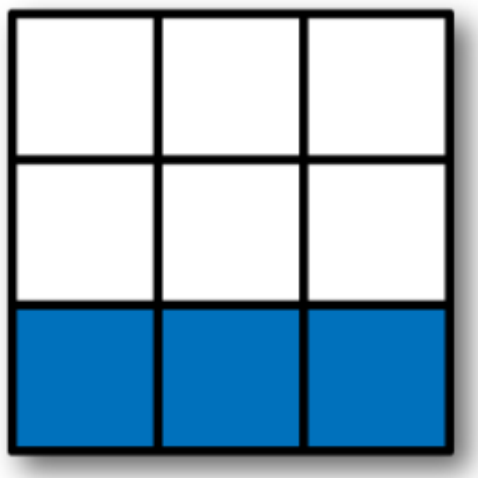
```
In [67]: arr2d
Out[67]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [68]: arr2d[1, :2]
Out[68]: array([4, 5])
```

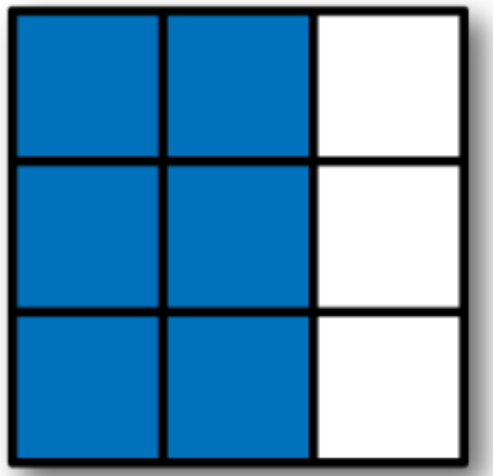
- Similarly, I can select the third column but only the first two rows like so:

```
In [69]: arr2d[:2, 2]
Out[69]: array([3, 6])
```

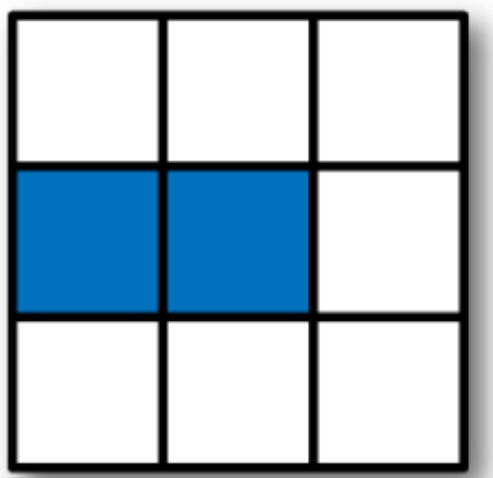




Expression	Shape
`arr[:2, 1:]`	`(2, 2)`
`arr[2]`	`(3,)`
`arr[2, :]`	`(3,)`
`arr[2:, :]`	`(1, 3)`



```
arr[:, :2]          (3, 2)
```



```
arr[1, :2]             (2,)
arr[1:2, :2]        (1, 2)
```

- Note that a colon by itself means to take the entire axis, so you can slice only higher dimensional axes by doing:

```
In [70]: arr2d
Out[70]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [71]: arr2d[:, :1]
Out[71]: array([[1],
                [4],
                [7]])
```

- Of course, assigning to a slice expression assigns to the whole selection:

```
In [72]: arr2d
Out[72]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [73]: arr2d[:2, 1:] = 0

In [74]: arr2d
Out[74]: array([[1, 0, 0],
                [4, 0, 0],
                [7, 8, 9]])
```

Boolean Indexing

- Let’s consider an example where we have some data in an array and an array of names with duplicates.

```
In [75]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
         names

Out[75]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [76]: data = np.random.randn(7, 4)
         data

Out[76]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])
```

- Suppose each name corresponds to a row in the `data` array and we wanted to select all the rows with corresponding name `'Bob'`.
- Like arithmetic operations, comparisons (such as ==) with arrays are also vectorized.
- Thus, comparing `names` with the string `'Bob'` yields a boolean array:

```
In [77]: names
Out[77]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [78]: names == 'Bob'
Out[78]: array([ True, False, False,  True, False, False, False])
```

- This boolean array can be passed when indexing the array:

```
In [79]: data
Out[79]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])

In [80]: names == 'Bob'
Out[80]: array([ True, False, False,  True, False, False, False])

In [81]: data[names == 'Bob']
Out[81]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823]])
```

- In these examples, I select from the rows where `names == 'Bob'` and index the columns, too:

```
In [82]: data
Out[82]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])

In [83]: names == 'Bob'
Out[83]: array([ True, False, False,  True, False, False, False])

In [84]: data[names == 'Bob', 2:]
Out[84]: array([[-1.40681423,  0.02569201],
                [-0.1173356 ,  0.56321823]])

In [85]: data[names == 'Bob', 3]
Out[85]: array([0.02569201, 0.56321823])
```

- To select everything but `'Bob'`, you can either use `!=` or negate the condition using `~`:

```
In [86]: names != 'Bob'
Out[86]: array([False,  True,  True, False,  True,  True,  True])

In [87]: data[~(names == 'Bob')]
Out[87]: array([[ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])
```

- The ~ operator can be useful when you want to invert a general condition:

```
In [88]: cond = names == 'Bob'
         data[~cond]

Out[88]: array([[ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])
```

- Selecting two of the three names to combine multiple boolean conditions, use boolean arithmetic operators like `&` (and) and `|` (or):

```
In [89]: names
Out[89]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [90]: data
Out[90]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])

In [91]: mask = (names == 'Bob') | (names == 'Will')
         mask
Out[91]: array([ True, False,  True,  True,  True, False, False])

In [92]: data[mask]
Out[92]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215]])
```

- Selecting data from an array by boolean indexing *always* creates a copy of the data, even if the returned array is unchanged.

- The Python keywords `and` and `or` do not work with boolean arrays.
- Use `&` (and) and `|` (or) instead.

- Setting values with boolean arrays works in a common-sense way.
- To set all of the negative values in `data` to `0` we need only do:

```
In [93]: data
Out[93]: array([[ 0.60213884,  1.16152378, -1.40681423,  0.02569201],
                [ 1.96391964,  0.77421578, -0.46176619, -0.82894886],
                [-0.14635458, -0.25735535, -1.43502398, -1.67862161],
                [ 1.66486039,  0.95193674, -0.1173356 ,  0.56321823],
                [ 0.22405003, -1.66374869, -0.4432533 , -1.46083215],
                [-0.63752033, -0.85573213, -0.89384526, -0.29027859],
                [-0.23961896,  2.61369281,  0.0260379 , -0.16887009]])

In [94]: data[data < 0] = 0
         data
Out[94]: array([[0.60213884, 1.16152378, 0.        , 0.02569201],
                [1.96391964, 0.77421578, 0.        , 0.        ],
                [0.        , 0.        , 0.        , 0.        ],
                [1.66486039, 0.95193674, 0.        , 0.56321823],
                [0.22405003, 0.        , 0.        , 0.        ],
                [0.        , 0.        , 0.        , 0.        ],
                [0.        , 2.61369281, 0.0260379 , 0.        ]])
```

- Setting whole rows or columns using a one-dimensional boolean array is also easy:

```
In [95]: names
Out[95]: array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [96]: data
Out[96]: array([[0.60213884, 1.16152378, 0.        , 0.02569201],
                [1.96391964, 0.77421578, 0.        , 0.        ],
                [0.        , 0.        , 0.        , 0.        ],
                [1.66486039, 0.95193674, 0.        , 0.56321823],
                [0.22405003, 0.        , 0.        , 0.        ],
                [0.        , 0.        , 0.        , 0.        ],
                [0.        , 2.61369281, 0.0260379 , 0.        ]])

In [97]: data[names != 'Joe'] = 7
         data
Out[97]: array([[7.        , 7.        , 7.        , 7.        ],
                [1.96391964, 0.77421578, 0.        , 0.        ],
                [7.        , 7.        , 7.        , 7.        ],
                [7.        , 7.        , 7.        , 7.        ],
                [7.        , 7.        , 7.        , 7.        ],
                [0.        , 0.        , 0.        , 0.        ],
                [0.        , 2.61369281, 0.0260379 , 0.        ]])
```