Class 2 NumPy Functions

April 13, 2020

Discuss about other inbuild Functions in NumPy

Numpy Ref: click here

[1]:
```python
import numpy as np
```

[3]:
```python
len(dir(np))
```

[3]:
```
622
```

[2]:
```python
arr = np.arange(25)
ranarr = np.random.randint(0,50,10)
```

[7]:
```python
arr
```

[7]:
```
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])
```

[17]:
```python
arr.shape
```

[17]:
```
(25,)
```

[14]:
```python
ranarr
```

[14]:
```
array([16, 48, 29, 28, 14, 34, 14, 32, 33, 47])
```

0.1 Reshape

Returns an array containing the same data with a new shape.

[5]:
```python
arr.reshape(5,5)
```

[5]:
```
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]])
```

0.1.1 max,min,argmax,argmin

These are useful methods for finding max or min values. Or to find their index locations using argmin or argmax

```
[6]: ranarr
```

```
[6]: array([12,  9, 47, 48, 34,  2, 25, 17, 19,  4])
```

```
[7]: ranarr.max()
```

```
[7]: 48
```

```
[8]: ranarr.argmax()
```

```
[8]: 3
```

```
[9]: ranarr.min()
```

```
[9]: 2
```

```
[10]: ranarr.argmin()
```

```
[10]: 5
```

0.2 Shape

Shape is an attribute that arrays have (not a method): Vectors are a type of matrix having only one column or one row.

```
[11]: # Vector
      arr.shape
```

```
[11]: (25,)
```

```
[12]: # Notice the two sets of brackets
      arr.reshape(1,25)
```

```
[12]: 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]])
```

```
[13]: arr.reshape(1,25).shape
```

```
[13]: (1, 25)
```

```
[14]: arr.reshape(25,1)
```

```
[14]: 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]])
```

```
[15]: arr.reshape(25,1).shape
```

```
[15]: (25, 1)
```

```
[16]: arr.dtype
```

```
[16]: dtype('int32')
```

0.2.1 BroadCasting

The term broadcasting refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations. Arithmetic operations on arrays are usually done on corresponding elements. If two arrays are of exactly the same shape, then these operations are smoothly performed.

When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward. Two dimensions are compatible when

1. they are equal, or
2. one of them is 1 If these conditions are not met, a ValueError: operands could not be broadcast together exception is thrown, indicating that the arrays have incompatible shapes. The size of the resulting array is the maximum size along each dimension of the input arrays. More Details: Broadcasting

[20]:
```
a = np.array([1.0, 2.0, 3.0])
b = 2.0
c=a*b
print(c)
```

```
[2. 4. 6.]
```

[32]:
```
x = np.arange(4)
x
```

[32]:
```
array([0, 1, 2, 3])
```

[31]:
```
xx = x.reshape(4,1)
xx
```

[31]:
```
array([[0],
       [1],
       [2],
       [3]])
```

[19]:
```
y = np.ones(5)
print(y)
z = np.ones((3,4))
print(z)
```

```
[1. 1. 1. 1. 1.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

[33]:
```
print("XShape:",x.shape)
print("YShape:",y.shape)
```

```
XShape: (4,)
YShape: (5,)
```

[28]:
```
x+y
```

```
      ␣
↪---------------------------------------------------------------------------

      ValueError                                Traceback (most recent call␣
↪last)

      <ipython-input-28-259706549f3d> in <module>
   ----> 1 x+y
```

```
        ValueError: operands could not be broadcast together with shapes (4,)␣
↪(5,)
```

[39]:
```
print("The XX:")
print("")
print(xx)
print("")
print("The y:")
print("")
print(y)
```

```
The XX:

[[0]
 [1]
 [2]
 [3]]

The y:

[1. 1. 1. 1. 1.]
```

[29]:
```
(xx + y).shape
```

[29]:
```
(4, 5)
```

[30]:
```
(xx + y)
```

[30]:
```
array([[1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4.]])
```

0.3 Conditional Selection

This is a very fundamental concept that will directly translate to pandas later on, make sure you understand this part!

Let's briefly go over how to use brackets for selection based off of comparison operators.

[40]:
```
arr = np.arange(1,11)
arr
```

[40]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[41]:
```
arr > 4
```

```
[41]: array([False, False, False, False,  True,  True,  True,  True,  True,
                True])
```

```
[42]: bool_arr = arr>4
```

```
[43]: bool_arr
```

```
[43]: array([False, False, False, False,  True,  True,  True,  True,  True,
                True])
```

```
[44]: arr[bool_arr]
```

```
[44]: array([ 5,  6,  7,  8,  9, 10])
```

```
[45]: arr[arr>2]
```

```
[45]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

```
[46]: x = 2
      arr[arr>x]
```

```
[46]: array([ 3,  4,  5,  6,  7,  8,  9, 10])
```

1 Good Job