Special Topics - NumPy

Claire Konz for EBEC

Table of Contents

Installing NumPy

Installing NumPy

- Use PIP, just like for matplotlib

Terminal

```
$ python -m pip install numpy
```

NumPy Background

What is NumPy?

- “Fundamental package for scientific computing in Python”
- A library (meaning you must import it to use it)
- Makes use of an “array” object
- Numpy is used in…
  - Data science
  - Image processing
  - Machine learning
  - Simulation
  - Other matrix applications

Operations in NumPy vs regular Python

- Numpy is MUCH faster
- Code is usually more concise
- Example: add 2 to every data point

Script - for_loops.py

```
data = list(range(1000000))
for i in range(len(data)):
    data[i] +=2
```

Script - nparrays.py

```
data = np.arange(1000000)
data += 2
```

Terminal

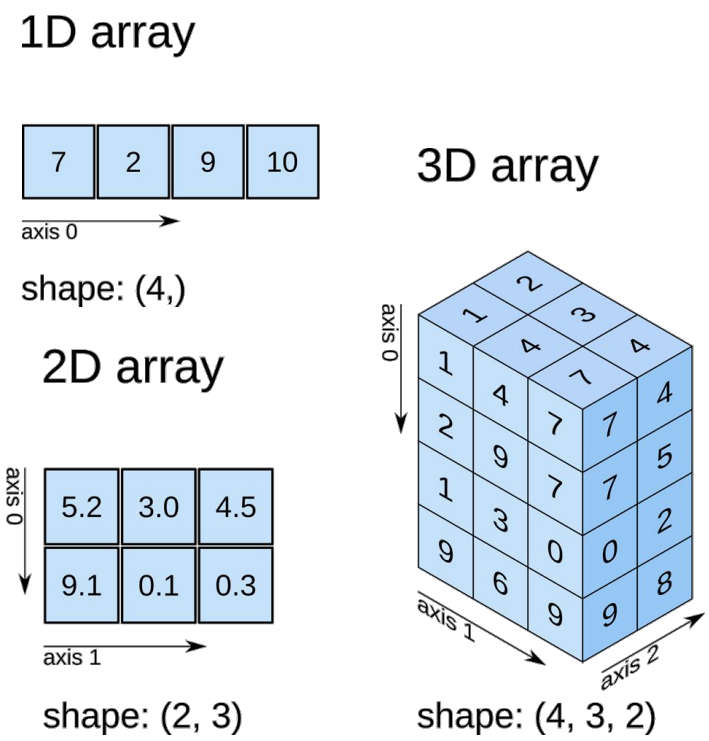
```
Average for loop time: 0.2881
Average NumPy array time: 0.0015

On average, NumPy was 189.0621
times faster
```

Timed each process 30 times

len(data) = 1,000,000

What are arrays?

- Central data structure for NumPy
- Grid of values
  - 1D array = vector
  - 2D array = matrix
  - 3D array = stack of matrices
  - N-dimensional array
- Array attributes:
  - Rank - number of dimensions the array has
  - Size - number of elements in the array
  - Shape - length of each dimension
  - *Note*: array dimensions are called “axes”



Creating Arrays

Array from a list

`np.array(list)`

- Creates a numpy array
- Accepts a list as inputs (including lists of list to make N-D arrays)

Terminal

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

Array of zeros

`np.zeros(s)`

- Creates an array of zeros with shape s
- s must be an integer (for a 1D array) or tuple (for arrays with more than one dimension)

Terminal

```
>>> np.zeros(5)
array([0., 0., 0., 0., 0.])
>>> np.zeros((2,3))
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Array of ones

`np.ones(s)`

- Creates an array of ones with shape s
- s must be an integer (for a 1D array) or tuple (for arrays with more than one dimension)

Terminal

```
>>> np.ones(3)
array([1., 1., 1.])
>>> np.ones((2,3))
array([[1., 1., 1.],
       [1., 1., 1.]])
```

Array over a range

```
np.arange(end)
np.arange(start, end)
np.arange(start, end, step size)
```

- Creates an array with specified start, end, and step size
- Can be helpful in creating axes for plots

Terminal

```
>>> np.arange(5)
array([0, 1, 2, 3, 4])
>>> np.arange(2,7)
array([2, 3, 4, 5, 6])
```

Terminal

```
>>> np.arange(2,9,2)
array([2, 4, 6, 8])
>>> np.arange(2,4,0.5)
array([2. , 2.5, 3. , 3.5])
```

Array with equally spaced values

```
np.linspace(start, end, num = 50)
```

- Creates an array of numbers with equally spaced values between the start and end
- Includes start and end values
- “Num” is an optional third argument that specifies how many values between the start and end
  - Default value = 50

Terminal

```
>>> np.linspace(2,5)
array([2., 2.06122449,
2.12244898,...,4.87755102,
4.93877551, 5.]
```

Terminal

```
>>> np.linspace(2,5,6)
array([2. , 2.6, 3.2, 3.8,
4.4, 5. ])
```

Creating an identity matrix

`np.identity(n)`

- Creates an n x n identity matrix
- Can be a useful operation in matrix math

Terminal

```
>>> np.identity(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Stacking arrays together

```
np.vstack((arr1, arr2))
np.hstack((arr1, arr2))
```

- Stack arrays together vertically or horizontally
- Accepts tuple of arrays as input

Terminal

```
>>> a = np.array([[1,2],[3,4]])
>>> b = np.array([[5,6],[7,8]])
>>> np.hstack((a,b))
array([[1, 2, 5, 6],
       [3, 4, 7, 8]])
```

Terminal

```
>>> np.vstack((a,b))
array([[1, 2],
       [3, 4],
       [5, 6],
       [7, 8]])
```

Getting Array Attributes

Getting array attributes - ndim

`arr.ndim`

- returns rank (number of axes) of the array

Terminal

```
>>> a = np.array([1,2,3])
>>> a.ndim
1
>>> b = np.array([[1,2,3],[4,5,6]])
>>> b.ndim
2
```

Getting array attributes - size

`arr.size`

- returns total number of elements in an array
- *Note*: this is the same as the product of the length of all axes

Terminal

```
>>> a = np.array([1,2,3])
>>> a.size
3
>>> b = np.array([[1,2,3],[4,5,6]])
>>> b.size
6
```

Getting array attributes - shape

`arr.shape`

- returns a tuple of the length of each array axis

Terminal

```
>>> a = np.array([1,2,3])
>>> a.shape
(3,)
>>> b = np.array([[1,2,3],[4,5,6]])
>>> b.shape
(2,3)
```

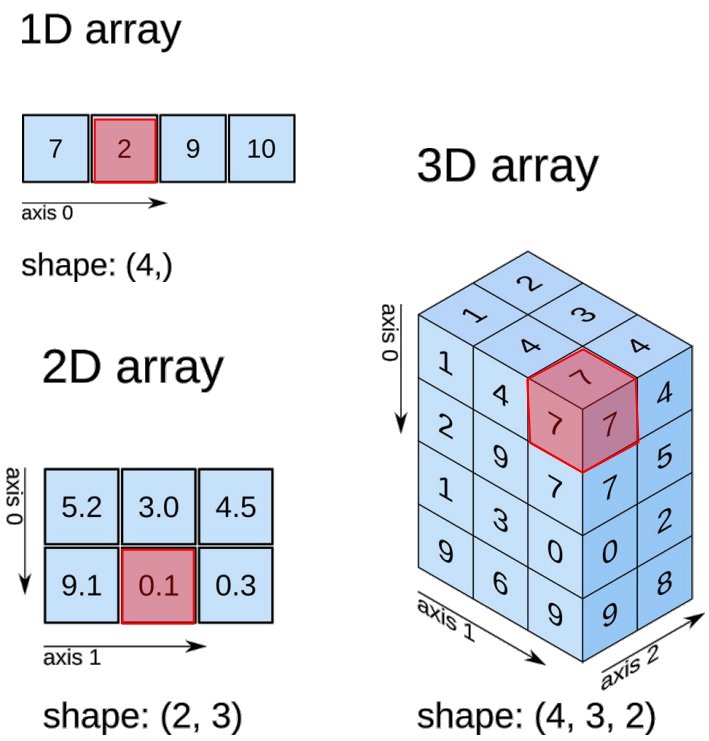
Indexing and Slicing

Array indexing

- Works like list indexing
- To index a single element, provide the same number of coordinates as there are axes in the array
- The order in which axes are indexed matters

Terminal

```
>>> one_d[1]
2
>>> two_d[1, 1]
0.1
>>> three_d[0, 2, 0]
7
```



Array Slicing

- Just like lists, you can take “slices” of an array

Terminal

```
>>> stack = np.arange(27).reshape((3, 3,
3))
>>> stack
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]]])
```

Terminal

```
>>> stack[2, 2, :]
array([24, 25, 26])
>>> stack[2, 1:, 2]
array([23, 26])
>>> stack[1:, 1:, 2]
array([[14, 17],
       [23, 26]])
```

Filtering

Filtering

- Formatted as if indexing a list
- Used to get elements in a list that meet a specific criteria

Terminal

```
>>> a = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[a >= 5]
array([5, 6, 7, 8, 9])
>>> a[a % 2 == 0]
array([0, 2, 4, 6, 8])
>>> a[(a < 3) | (a > 7)]
array([0, 1, 2, 8, 9])
```

Filtering with boolean values

- Use logical operators to return an array of boolean values that indicate whether or not a given criteria is met
- Indexing an array with a boolean array returns values corresponding to True

Terminal

```
>>> b = np.array([7,0,2,5,9,3,4,8,1,6])
>>> t = (b < 4) & (b % 2 == 0)
>>> t
array([False,  True,  True, False, False, False,
False, False, False,False])
>>> b[t]
array([0, 2])
```

Where - find indices where a condition is true

`np.where(condition)`

- Returns a tuple of lists of indices where a given criteria is true
- Returns one list of indices for every dimension
- Can use results to index arrays and get specific elements

Terminal

```
>>> a = np.array([8,4,5,3,0,7,2,6,1,9])
>>> np.where(a <= 4)
(array([1, 3, 4, 6, 8], dtype=int32),)
>>> b = np.array([[2, 7, 4],[3, 9, 4]])
>>> ys, xs = np.where(b == 4)
>>> ys
array([0, 1], dtype=int32)
>>> xs
array([2, 2], dtype=int32)
```

Where - replace values if a condition is true

```
np.where(condition, x, y)
```

- x and y are optional arguments
- Returns x where the condition is true and y otherwise
- x and/or y can be operations, but operations must be broadcastable

Terminal

```
>>> a = np.array([8,4,5,3,0,7,2,6,1,9])
>>> np.where(a <= 4, 1, 0)
array([0, 1, 0, 1, 1, 0, 1, 0, 1, 0])
>>> b = np.array([3,1,6,5,7,2,9,8,1,4])
>>> np.where(a<b, b-a, a-b)
array([5, 3, 1, 2, 7, 5, 7, 2, 0, 5])
```

Basic Operations

Addition, subtraction, multiplication, division

- All operations are performed element-wise between two or more arrays

Terminal

```
>>> a = np.array([1,2,3,4])
>>> b = np.ones(4)
>>> a + b
array([2., 3., 4., 5.])
>>> a - b
array([0., 1., 2., 3.])
>>> a * a
array([ 1,  4,  9, 16])
>>> a / a
array([1., 1., 1., 1.])
```

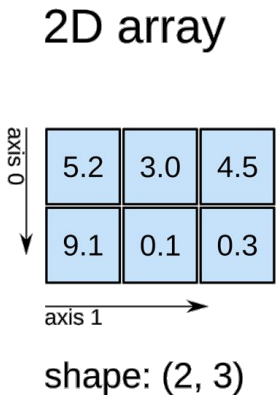
Adding all elements in an array

`np.sum(axis)`

- Used to add together the value of all elements in an array
- If the axis argument is specified, elements on that axis are summed and returned as an array

Terminal

```
>>> a = np.array([[5.2,3.0,4.5],[9.1,0.1,0.3]])
>>> a.sum() # sum of all elements
22.2
>>> a.sum(axis=0) # sum of columns
array([14.3,  3.1,  4.8])
>>> a.sum(axis=1) # sum of rows
array([12.7,  9.5])
```



Finding the product

`arr.prod()`

- Finds the product of all elements in an array
- Like sum, this can also be performed over a specific axis

Terminal

```
>>> b = np.array([[1,2,3],[4,5,6]])
>>> b.prod()
720
>>> b.prod(axis=0)
array([ 4, 10, 18])
```

Descriptive statistics

- The NumPy library has built in functions for most common descriptive statistic calculations used in data analysis
- Includes: min, max, mean, standard deviation, variance
- Like sum, these can also be performed over a specific axis

Terminal

```
>>> data =
np.array([[5.2,3.0,4.5],
          [9.1,0.1,0.3]])
>>> data.min()
0.1
>>> data.max()
9.1
```

Terminal

```
>>> data.mean()
3.6999999999999997
>>> data.std()
3.083828789021855
>>> data.var()
9.51
```

Broadcasting

- Carry out operations between an array and a single number

Terminal

```
>>> a = np.array([1,2,3,4,5,6])
>>> a * 2
array([ 2,  4,  6,  8, 10, 12])
>>> a / 2
array([0.5, 1. , 1.5, 2. , 2.5, 3. ])
>>> a + 2
array([3, 4, 5, 6, 7, 8])
>>> a - 2
array([-1,  0,  1,  2,  3,  4])
```

Terminal

```
>>> b =
np.array([[1,2,3],
          [4,5,6]])
>>> b + 2
array([[3, 4, 5],
       [6, 7, 8]])
```

Broadcasting (cont.)

- Can broadcast arrays onto other arrays with compatible dimensions
- Dimensions are compatible if:
  - they are equal, or
  - one of them is 1

Terminal

```
>>> a =
np.array([[2,7,4],[3,9,4]])
>>> b = np.arange(3)
>>> a + b
array([[ 2,  8,  6],
       [ 3, 10,  6]])
```

Terminal

```
>>> b = np.arange(4)
>>> a + b
Traceback (most recent call
last):
  File "<stdin>", line 1, in
<module>
ValueError: operands could not
be broadcast together with
shapes (2,3) (4,)
```

Array Manipulation

Adding elements to an array

`np.concatenate((a,b))`

- Concatenates array b onto the end of array a
- Pass arrays to be concatenated in as a tuple
- *Note*: This operation re-allocates a new array and copies the contents of *a* and *b* into it, so it can be quite slow for large arrays

Terminal

```
>>> a = np.array([2,7,9,4])
>>> b = np.array([3,7])
>>> np.concatenate((a,b))
array([2, 7, 9, 4, 3, 7])
```

Sorting arrays

`np.sort(arr)`

- Sort an array in increasing order
- Can also optionally specify an axis (just like sum, min, max, etc.)
  - Sorts each subarray along that axis
  - Sorting a multidimensional array defaults to sorting along its last axis

Terminal

```
>>> arr = np.array([1, 5, 3, 2, 7])
>>> np.sort(arr)
array([1, 2, 3, 5, 7])
```

Terminal

```
>>> arr2
array([[27, 13,  6],
       [12,  9, 23],
       [72, 18, 12]])
>>> np.sort(arr2, axis=1)
array([[ 6, 13, 27],
       [ 9, 12, 23],
       [12, 18, 72]])
>>> np.sort(arr2, axis=0)
array([[12,  9,  6],
       [27, 13, 12],
       [72, 18, 23]])
```

Sorting in place

`arr.sort()`

- Instead of using the np.sort(...) function, just calling ".sort()" on an array will sort it in place
  - Once again, optionally specify the axis

Terminal

```
>>> arr = np.array([32, 19, 4, 9])
>>> arr.sort()
>>> arr
array([ 4,  9, 19, 32])
```

Sorting indices

`np.argsort(arr)`

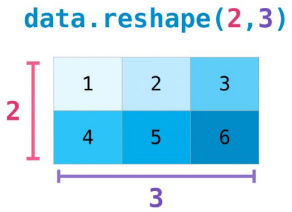
- Returns the indices of the sorted array instead of the actual sorted array
- Can be useful if you want to maintain a correspondence between two arrays and sort one of them

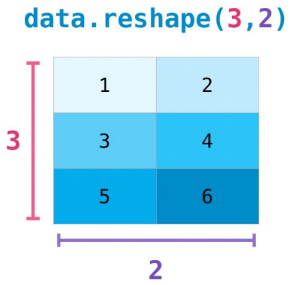
Terminal

```
>>> item_ids = np.array([100, 150, 200, 250, 300])
>>> item_prices = np.array([1.17, 3.50, 2.21, 10.05, 5.99])
>>> idx = np.argsort(item_prices)
>>> idx
array([0, 2, 1, 4, 3])
>>> item_prices[idx]
array([ 1.17,  2.21,  3.5 ,  5.99, 10.05])
>>> item_ids[idx]
array([100, 200, 150, 300, 250])
```

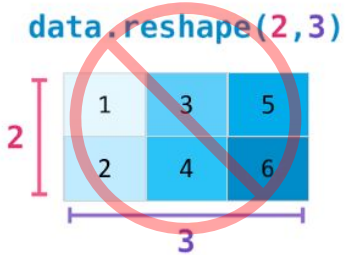
Reshaping arrays

- change the shape of the data without changing the actual data
- Maintains the same number of elements (size) in a new shape





Note: you cannot control the order in which the elements are shaped into the new array. (e.g. example on right is not possible)



Reshaping arrays

`arr.reshape(newshape)`

- Reshapes an array into the specified shape
- Accepts the new shape as an integer or tuple
- Value error if old and new shape do not have the same size

Terminal

```
>>> b = np.array([1,2,3,4,5,6])
>>> b.reshape(2,3)
array([[1, 2, 3],
       [4, 5, 6]])
```

Terminal

```
>>> b.reshape(3,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot reshape array of
size 6 into shape (3,3)
```

Transposing a matrix

`arr.transpose(), arr.T`

- Transposes the axes of a matrix

Terminal

```
>>> x = np.array([[1,2,3],[4,5,6]])
>>> x.transpose()
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> x.T
array([[1, 4],
       [2, 5],
       [3, 6]])
```

Flattening an array

`arr.flatten()`

- Flattens a multidimensional array into a 1D array

Terminal

```
>>> x
array([[1, 2, 3],
       [4, 5, 6]])
>>> x.flatten()
array([1, 2, 3, 4, 5, 6])
```

References

- https://numpy.org/doc/stable/user/absolute_beginners.html
- https://realpython.com/numpy-tutorial/