



ROI TRAINING
MAXIMIZE YOUR TRAINING INVESTMENT

Python for Data Scientists

CHAPTER 2: NUMPY ESSENTIALS: ARRAYS AND VECTORIZED COMPUTATION

Chapter Objectives

In this chapter, we will introduce:

- Universal functions: fast element-wise array functions
- Data processing using arrays
- File input and output with arrays
- Linear algebra
- Random number generation

Chapter Concepts

Array Functions

Data Processing

File Input Output

Linear Algebra

Random Numbers

Chapter Summary

NumPy ndarray

- ◆ `ndarray` is a N-dimensional array object
 - Fast, flexible container for large data sets in Python
- ◆ Easiest way to create an array is to use the `array` function
 - Accepts any sequence-like object and produces `ndarray`

```
import numpy as np

data = [1, 2, 3, 4]

array1 = np.array(data)

array1
array([1, 2, 3, 4])
```

Access numpy library

Create numpy array

NumPy ndarray (continued)

- Nested sequences, e.g., list of lists are converted to a multi-dimensional array

```
data2 = [[1,2,3,4],[5,6,7,8]]

array2 = np.array(data2)

array2
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

- Data type is inferred from array data used
 - Stored in property `dtype`

```
array2.dtype
dtype('int64')
```

Other Functions for Creating Arrays

- Other functions are provided for creating arrays
 - `zeros` – creates array of 0's
 - `ones` creates array of 1's
 - `empty` creates uninitialized array

```
np.zeros(4)  
array([ 0.,  0.,  0.,  0.])  
  
np.ones((2,4))  
array([[ 1.,  1.,  1.,  1.],  
       [ 1.,  1.,  1.,  1.]])  
  
x = np.empty((2,4))  
x[:] = 0  
array([[ 0.,  0.,  0.,  0.],  
       [ 0.,  0.,  0.,  0.]])
```

No guarantee
elements will be 0
so do this to
guarantee it

Data Types for ndarrays

- ◆ `dtype` is a special object that defines type of data in array
 - Can be set when creating array
- ◆ Full set of data types can be found at
 - <https://docs.scipy.org/doc/numpy/user/basics.types.html>

```
array1 = np.array([1,2,3,4,5], dtype=np.float64)
```

```
array1
```

```
array([ 1.,  2.,  3.,  4.,  5.])
```

```
array1.dtype
```

```
dtype('float64')
```

Specify type of data

Exercise 2.1: Array Creation



➡ Please Open Day2-Labs notebook and complete the first lab

Chapter Concepts

Array Functions

Data Processing

File Input Output

Linear Algebra

Random Numbers

Chapter Summary

Operations Between Arrays

- Arrays allow operations on elements without writing loops
 - Usually called *vectorization*
- Arithmetic operations between equal sized arrays applies the operation element to element

```
array1 = np.array([[1,2,3],[4,5,6]])
```

```
array1
```

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

```
array1+array1
```

```
array([[ 2,  4,  6],  
       [ 8, 10, 12]])
```

Standard arithmetic
operators

Operations Between Arrays and Scalars

- ➡ Operations on arrays with scalars propagate the value to each element

```
array1 = np.array([[1.,2.,3.],[4.,5.,6.]])
```

```
array1
```

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

```
1/array1
```

```
array([[ 1.          ,  0.5          ,  0.33333333],  
       [ 0.25         ,  0.2         ,  0.16666667]])
```

Operation with scalar

```
array1*3
```

```
array([[ 3.,  6.,  9.],  
       [12., 15., 18.]])
```

Indexing and Slicing

- One-dimensional arrays are similar to Python lists
- Values applied to a slice are propagated (broadcasted) to the entire selection

```
array1 = np.arange(12)
array1
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

array1[2]
2

array1[2:5] ————— Array slice
array([2, 3, 4])

array1[2:5] = 99 ————— Value is broadcast
                        to selection
array1
array([ 0,  1, 99, 99, 99,  5,  6,  7,  8,  9, 10, 11])
```

Higher Dimensional Arrays

- In multi-dimensional arrays, elements can be accessed
 - Recursively
 - Comma-separated lists

```
array2d = np.array([[1,2],[3,4],[5,6]])  
array2d[1]  
array([3, 4])
```

```
array2d[1][0]  
3
```

Recursive index access

```
array2d[1,0]  
3
```

Comma-separated access

Higher Dimensional Arrays (continued)

- In multi-dimensional arrays, if later indices are omitted, returned objects are lower-dimensional arrays
- Consider the following 2 x 2 x 4 array

```
array3d
```

```
array([[[ 1,  2,  3,  4],  
        [ 5,  6,  7,  8]],  
       [[ 9, 10, 11, 12],  
        [13, 14, 15, 16]]])
```

```
array3d[1]
```

```
array([[ 9, 10, 11, 12],  
        [13, 14, 15, 16]])
```

Returns 2 x 4 array


Transposing Arrays

- Transposing returns a view of underlying data without copying data
- Reshape function will change dimensionality of array

```
array = np.arange(20).reshape((4,5))
```

```
array
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19]])
```



Reshape to a 4 x 5 array

Transposing Arrays (continued)

- ◆ Arrays have the transpose method and the `T` attribute
- ◆ `T` can be used to transpose axis

array

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14],  
       [15, 16, 17, 18, 19]])
```

array.T

```
array([[ 0,  5, 10, 15],  
       [ 1,  6, 11, 16],  
       [ 2,  7, 12, 17],  
       [ 3,  8, 13, 18],  
       [ 4,  9, 14, 19]])
```

Transpose array

Mathematical and Statistical Methods

- A number of mathematical functions that compute statistics about a complete array along an axis are available
- Can be called on the array or calling top-level NumPy functions

```
array = np.random.randn(2,4)
array
array([[ -0.25417597,  0.34146309,  2.52982769,  1.45988811],
       [ 1.50491216, -0.49649801,  0.72392736, -0.24452797]])

array.mean()
0.69560205905268047

np.mean(array)
0.69560205905268047

array.mean(axis=0)
array([ 0.62536809, -0.07751746,  1.62687753,  0.60768007])
```

Can specify axis for computation

Exercise 2.2: Array Basic Operations



➡ Please Open Day2-Labs notebook and complete the second lab

Chapter Concepts

Array Functions

Data Processing

File Input Output

Linear Algebra

Random Numbers

Chapter Summary

File Input and Output with Arrays

- NumPy can load and save data from disk in text or binary format
 - By default, files are written in an uncompressed binary format
 - File extension `.npy`

```
array1 = np.arange(10)

np.save('array1.npy', array1)

array2 = np.load('array1.npy')

array2
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Extension added if not
provided explicitly

File Archives

- ◆ Multiple arrays can be saved to an archive file using `np.savez()`
 - `np.load()` will return dictionary style object
 - ◆ Each array is loaded lazily

```
array1 = np.arange(10)
```

```
array2 = 2 * array1
```

```
np.savez('array_archive.npz', data_set_1=array1,  
data_set_2=array2)
```

```
archive = np.load('array_archive.npz')
```

```
archive['data_set_1']  
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Data loaded lazily

```
archive['data_set_2']  
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Saving and Loading Text Files

◆ NumPy provides `loadtxt()` and `savetxt()` to read and write text files

```
array2d = np.array([[1,2,3],[4,5,6]])

np.savetxt('array_data.txt',array2d, delimiter=',')

array2d_from_file = np.loadtxt('array_data.txt',
delimiter=',')

array2d_from_file

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

Data loaded lazily

Chapter Concepts

Array Functions

Data Processing

File Input Output

Linear Algebra

Random Numbers

Chapter Summary

Linear Algebra

- ✦ With NumPy, multiplying two two-dimensional arrays with `*` is an element-wise product, not a matrix dot product
- ✦ The function `dot` provides matrix dot product

```
array1 = np.array([[1,2,3],[4,5,6]])
```

```
array2 = np.array([[1,2,3],[4,5,6]])
```

```
array_multiply = array1 * array2
```

```
array_multiply
```

```
array([[ 1,  4,  9],  
       [16, 25, 36]])
```

Result is element-wise multiplication

Linear Algebra (continued)

```
array1 = np.array([[1,2,3],[4,5,6]])
```

```
array2 = np.array([[1,2],[3,4],[5,6]])
```

```
array_dot_product = array1.dot(array2)
```

```
array_dot_product
```

```
array([[22, 28],  
       [49, 64]])
```

Dot product of array: array1 . array2

numpy.linalg

- Has a standard set of matrix decompositions
 - E.g., `inv()`, `dot()`, `solve()` etc.
- Documentation found at:
 - <https://docs.scipy.org/doc/numpy/reference/routines.linalg.html>

```
array1 = np.array([[.1, .2, .3], [.4, .5, .6], [.7, .8, .9]])
```

```
inv(array1)
```

```
array([[ -6.74335773e+15,  1.34867155e+16, -6.74335773e+15],  
       [  1.34867155e+16, -2.69734309e+16,  1.34867155e+16],  
       [ -6.74335773e+15,  1.34867155e+16, -6.74335773e+15]])
```

Chapter Concepts

Array Functions

Data Processing

File Input Output

Linear Algebra

Random Numbers

Chapter Summary

Random Number Generation

- ◆ `numpy.random` provides functions for generating arrays
 - From many kinds of probability distributions
 - ◆ Normal
 - ◆ Uniform
 - ◆ Poisson
 - ◆ Many more

```
print (np.random.normal(5, 2, 9)) # mean = 5, std = 2  
array([6.81532146, 3.64397936, 6.68626991, 6.24245039,  
2.74427372, 6.35545999, 3.19515877, 1.83536618, 2.59710754])
```

```
print (np.random.uniform(1, 100, 8)) # low = 1, high = 100  
array([88.54188937, 21.03845531, 12.20124916, 64.99097202,  
49.20289727, 33.33857889, 46.44605034, 31.57050879])
```

```
print (np.random.poisson(10, 10)) # 10 numbers averaging to 10  
array([ 3,  8,  8, 12, 13, 14, 11, 10, 14, 11])
```

Chapter Concepts

Array Functions

Data Processing

File Input Output

Linear Algebra

Random Numbers

Chapter Summary

Chapter Summary

In this chapter, we have introduced:

- ◆ Universal functions: fast element-wise array functions
- ◆ Data processing using arrays
- ◆ File input and output with arrays
- ◆ Linear algebra
- ◆ Random number generation