Numerical Python - NumPy Package

Computer Programming for Data Science 7CCSMCMP

Isabel Sassoon

Department of Informatics
King's College London
isabel.sassoon@kcl.ac.uk

October, 2016

Topics

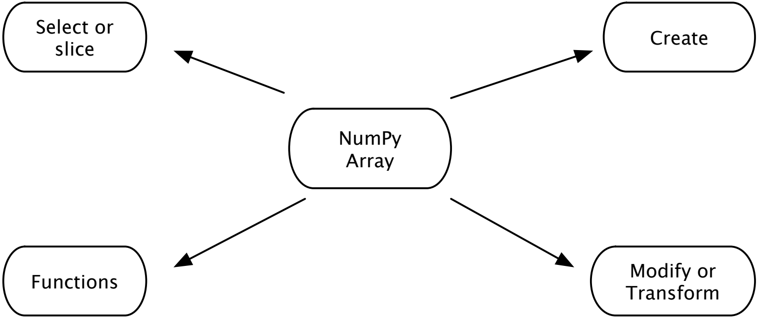
NumPy

- **Part 1:**
  - **Overview of NumPy**
  - **Creating an array**
  - **Indexing and slicing arrays**
- Part 2:
  - Operations on arrays
  - Sorting arrays
  - Writing and reading arrays from and to disk
  - Generating random number arrays
  - Histogram in NumPy

NumPy Overview

- Fast multidimensional arrays
- Speed of C and developer friendliness of Python
- Why are arrays needed?
  - Handle large sequences of data
  - To process many data points at the same time - matrix maths and regression for example
  - Numpy handled arrays faster than standard python lists and tuples
- in order to use the library: `import numpy as np`

NumPy Highlights



Select or slice

Create

NumPy Array

Functions

Modify or Transform

About arrays

An array is the central structure of NumPy

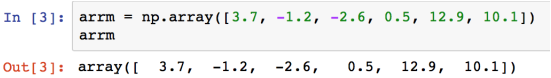
- Each element is of the same type
- The rank of an array is the number of dimensions
- The length of each dimension do not have to be the same

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

Ways to create Arrays

An array can be created:

- Manual input



```
In [3]: arrm = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
        arrm

Out[3]: array([  3.7,  -1.2,  -2.6,   0.5,  12.9,  10.1])
```

- Using lists
- Functions are available to create arrays of sequences, zeros and ones

Creating an array

Creating an array with lists:

```
data1=[6,7.5,8,0,1]
arr1=np.array(data1)
arr1
```

Results in this array:

```
array([ 6. ,  7.5,  8. ,  0. ,  1. ])
```

An array of zeros can be created using:

```
np.zeros(8)
```

This results in:

```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

Creating an array (contd...)

There are more ways of creating an array:

```
np.ones(4)
np.arange(15)
```

What will these generate?

Creating an array (contd...)

There are more ways of creating an array:

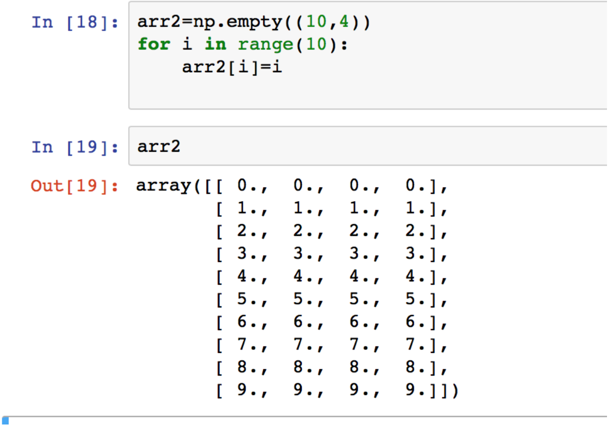
```
np.ones(4)
np.arange(15)
```

These will generate the following arrays:

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

Using recursion to populate an array

Populating an array using recursion:



```
In [18]: arr2=np.empty((10,4))
         for i in range(10):
             arr2[i]=i
```

```
In [19]: arr2
```

```
Out[19]: array([[ 0.,  0.,  0.,  0.],
                [ 1.,  1.,  1.,  1.],
                [ 2.,  2.,  2.,  2.],
                [ 3.,  3.,  3.,  3.],
                [ 4.,  4.,  4.,  4.],
                [ 5.,  5.,  5.,  5.],
                [ 6.,  6.,  6.,  6.],
                [ 7.,  7.,  7.,  7.],
                [ 8.,  8.,  8.,  8.],
                [ 9.,  9.,  9.,  9.]])
```

Multidimensional Arrays

Arrays can have more than just one dimension, for example:

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2=np.array(data2)
arr2
```

arr2 will look like:

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

Using size and shape:

```
arr2.shape
(2, 4)

arr2.size
8
```

Array Data Type

The data type of the array:

```
arr=np.arange(8)
arr.dtype
```

Will return in this case:

```
dtype('int64')
```

Specifying the type when generating the array

```
arr2 = np.array([1, 2, 3], dtype=np.int32)
arr2.dtype
```

will result in this:

```
dtype('int32')
```

Array Indexing and Slicing

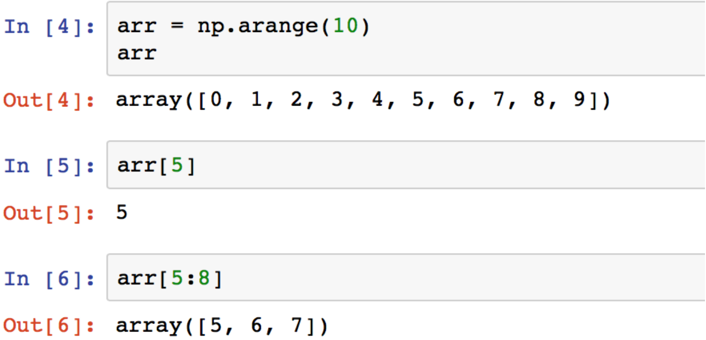
Select a subset or individual elements from a one dimensional array:

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

What do the selections above return?

Array Indexing and Slicing contd.

The results of the indexing from the previous slide:



```
In [4]:  arr = np.arange(10)
         arr

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

In [5]:  arr[5]

Out[5]:  5

In [6]:  arr[5:8]

Out[6]:  array([5, 6, 7])
```

Array Indexing and Slicing

Assign a value to a slice:

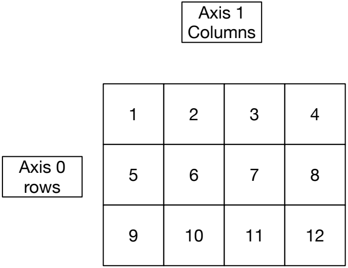
```
arr[5:8] = 12
arr
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

A slice is a view of the array, so any changes made to a slice are propagated to the source array.

Indexing and Slicing Multidimensional arrays

Lets look at an array created as follows:

```
arr2d = np.array([[1, 2, 3,4], [5, 6, 7,8], [9, 10,
    11,12]])
```



Axis 1
Columns

Axis 0
rows

1	2	3	4
5	6	7	8
9	10	11	12

A two dimensional array

The positions of each element in the two dimensional array are (row,column):

Axis 1
Columns

	1 (0,0)	2 (0,1)	3 (0,2)	4 (0,3)
Axis 0 rows	5 (1,0)	6 (1,1)	7 (1,2)	8 (1,3)
	9 (2,0)	10 (2,1)	11 (2,2)	12 (2,3)

Individual elements can be accessed:

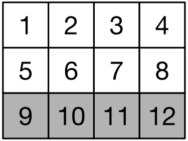
```
arr2d[0, 2]
3
```

Two dimensional array indexing and slicing example

1	2	3	4
5	6	7	8
9	10	11	12

```
arr2d[2,:]
```

This results in this shaded area of the array:



1	2	3	4
5	6	7	8
9	10	11	12

Two dimensional array indexing and slicing example

1	2	3	4
5	6	7	8
9	10	11	12

```
arr2d[:,:2]
```

What will this return?

1. First 2 columns, all rows
2. Last 2 rows, all columns
3. Last 2 columns, all rows

Two dimensional array indexing and slicing example

1	2	3	4
5	6	7	8
9	10	11	12

```
arr2d[:,:2]
```

1	2	3	4
5	6	7	8
9	10	11	12

Answer: First two columns, all rows.

3D arrays

Assuming a 3D array is generated:

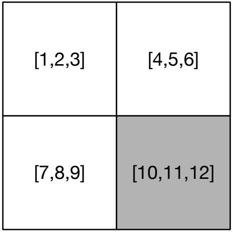
```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9],
    [10, 11, 12]]])
```

[1,2,3]	[4,5,6]
[7,8,9]	[10,11,12]

The shape of this array is :(2, 2, 3) i.e. 2 * 2* 3.

3D arrays

In order to select the following from the 3D array:



[1,2,3]	[4,5,6]
[7,8,9]	[10,11,12]

Then refer to it:

```
arr3d[1,1,::]
arr3d[1,1,]
```

3D arrays

If later indices are omitted, the returned object will be a lower-dimensional array consisting of all the data along the higher dimensions.

```
arr3d[0]
```

Will result in a 2*3 array:

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

NumPy - Summary of Part 1

- Overview of Numpy
- Methods to create an array: manual inputs, using lists, using `zeros, ones`, random and recursion.
- Attributes of arrays: shape, number of dimensions and data types
- Indexing and slicing: selections, 2d and 3d