START WITH NUMPY

NUMPY – INTRODUCTION

- NumPy is the fundamental package needed for scientific computing with Python.
- In 2005, **Travis Oliphant** created NumPy package by incorporating the features of **Numarray** into **Numeric** package.
- NumPy is often used along with packages like **SciPy** (Scientific Python) and **Mat−plotlib**(plotting library).

NUMPY – INTRODUCTION

- It contains:
  - a powerful N-dimensional array object
  - basic linear algebra functions
  - basic Fourier transforms
  - sophisticated random number capabilities
  - tools for integrating Fortran code
  - tools for integrating C/C++ code

NUMPY – INTRODUCTION

- Official documentation
  - http://docs.scipy.org/doc/
- The NumPy book
  - http://www.tramy.us/numpybook.pdf
- Example list
  - http://www.scipy.org/Numpy_Example_List_With_Doc

NUMPY – ENVIRONMENT

- Using popular Python package installer, pip.

```
pip install numpy
```

- Anaconda (https://www.anaconda.com/download/) is a free Python distribution for SciPy stack. It is also available for Linux and Mac.

NUMPY – ENVIRONMENT

- To test whether NumPy module is properly installed：

```
import numpy (import numpy as np)
```

- If it is not installed, the following error message will be displayed.

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
  import numpy
ImportError: No module named 'numpy'
```

NUMPY – NDARRAY OBJECT

- Array in python:

Type code	C Type	Minimum size in bytes
'c'	character	1
'b'	signed integer	1
'B'	unsigned integer	1
'u'	Unicode character	2
'h'	signed integer	2
'H'	unsigned integer	2
'i'	signed integer	2
'I'	unsigned integer	2
'l'	signed integer	4
'L'	unsigned integer	4
'f'	floating point	4
'd'	floating point	8

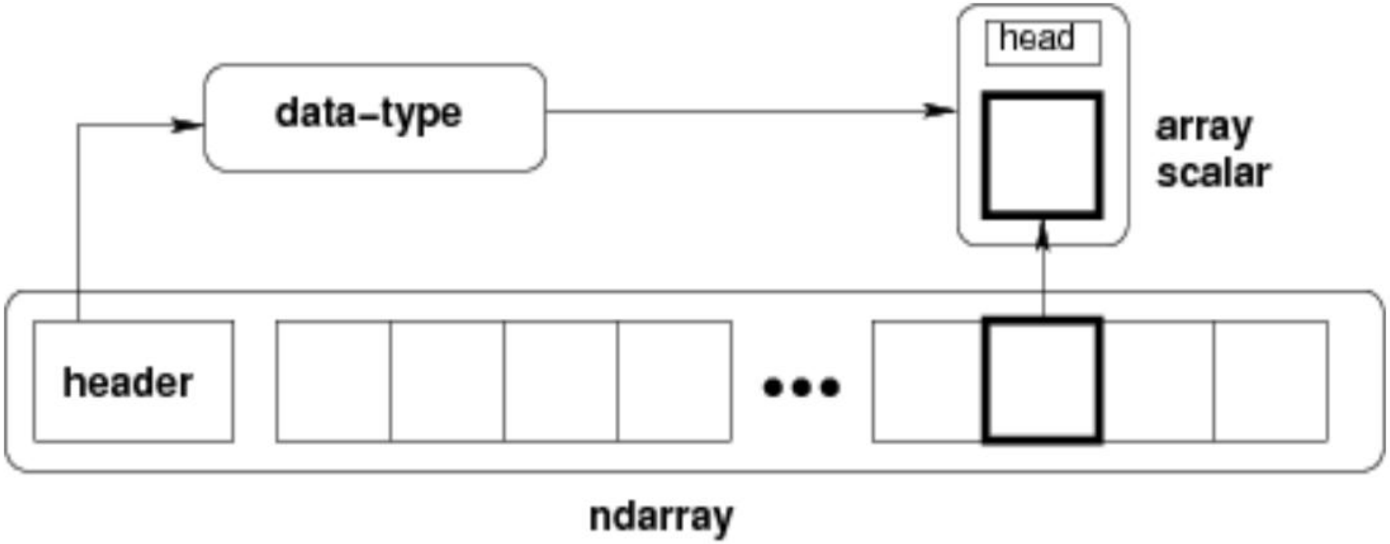
```
from array import *
myarray=array("I")
myarray.append(3)
myarray.pop()
myarray.remove(X)
num=myarray[0]
myarray.insert(3,10)
myarray.reverse()
```

NUMPY – NDARRAY OBJECT

- The most important object，**N-dimensional array** type called **ndarray**.
    - The collection of items of the **same type**.
    - Using a **zero-based index**.
    - Items in an it takes the **same size** of block in the memory.
    - Each element is an object of data-type object (called **dtype**).

NUMPY – NDARRAY OBJECT

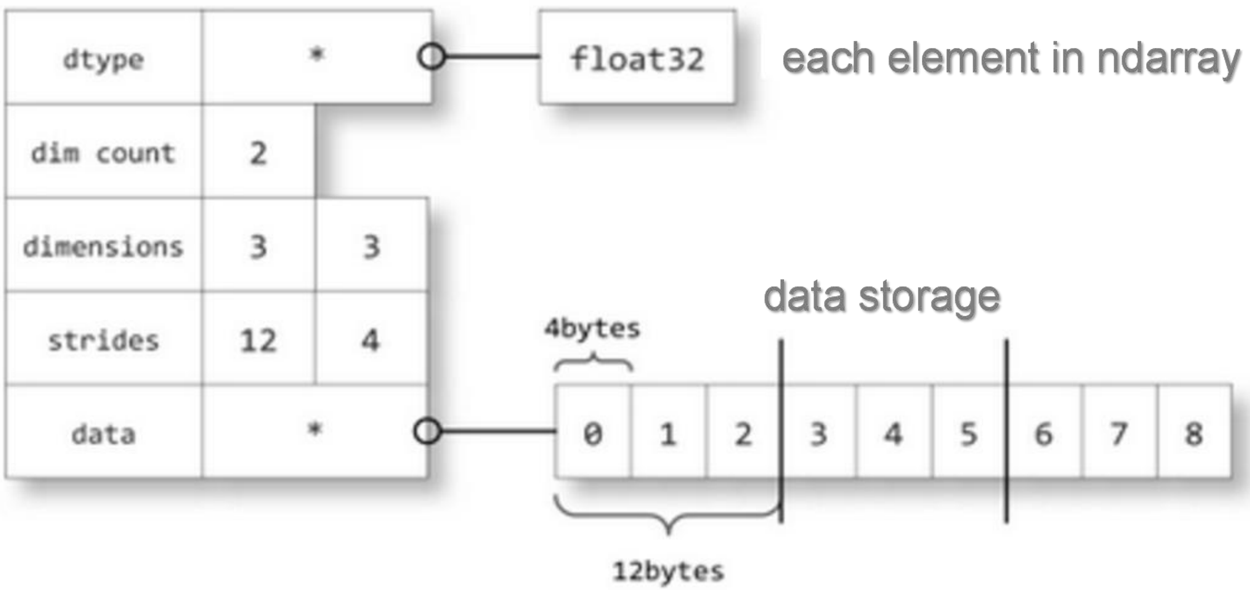
- A relationship between ndarray, data type object (dtype) and array scalar type:



NUMPY – NDARRAY OBJECT

- For example:

```
from numpy import np
a = np.array([[0,1,2],[3,4,5],[6,7,8]],dtype=np.float32)
```



NUMPY – NDARRAY OBJECT

- The basic ndarray is **created** using an array function in NumPy as follows:

```
numpy.array
numpy.array(object, dtype=None, copy=True, order=None,
subok=False, ndmin=0)
```

- Examples:

```
import numpy as np
a=np.array([1,2,3])
print a
```

- The output is: `[1, 2, 3]`

NUMPY – NDARRAY OBJECT

object	Any object exposing the array interface method returns an array, or any (nested) sequence
dtype	Desired data type of array, optional
copy	Optional. By default (true), the object is copied
order	C (row major) or F (column major) or A (any) (default)
subok	By default, returned array forced to be a base class array. If true, sub-classes passed through
ndimin	Specifies minimum dimensions of resultant array

NUMPY – NDARRAY OBJECT

jupyter ndarray_tst Last Checkpoint: 几秒前 (autosaved)

File Edit View Insert Cell Kernel Widgets Help

In [4]:

```python
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print (a)
```

```
[[1 2]
 [3 4]]
```

In [5]:

```python
# minimum dimensions

import numpy as np
a=np.array([1, 2, 3,4,5], ndmin=2)
print (a)
```

```
[[1 2 3 4 5]]
```

In [6]:

```python
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype=complex)
print(a)
```

```
[1.+0.j 2.+0.j 3.+0.j]
```

NUMPY – DATA TYPES

- The table shows different scalar data types defined in NumPy.

Data Types	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)

NUMPY – DATA TYPES

uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

NUMPY – DATA TYPES

- Each built-in data type has **a character code** that uniquely identifies it.

'i': (signed) integer	'u': unsigned integer	'f': floating-point
'b': boolean	'c': complex-floating point	'm': timedelta
'M': datetime	'O': (Python) objects	'S', 'a': (byte-)string
'U': Unicode	'V': raw data (void)	

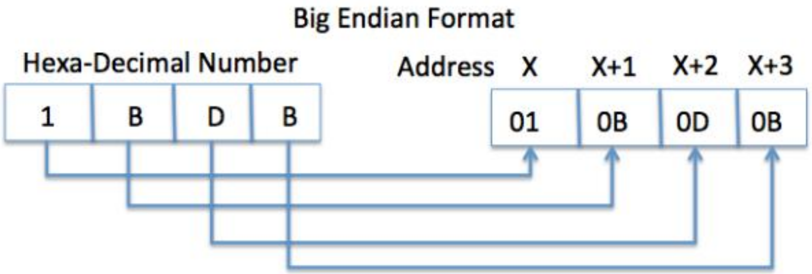
NUMPY – DATA TYPES

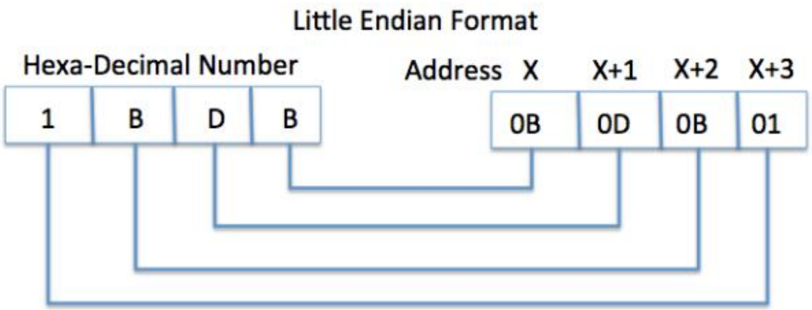
- A **data type object** describes fixed block of memory corresponding to an array, depending on the following aspects:
  - Type of data (integer, float or Python object)
  - Size of data
  - Byte order (little-endian or big-endian)
  - In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field.
  - If data type is a subarray, its shape and data type

NUMPY – DATA TYPES

NOTE:

- the **byte order** is decided by prefixing **'<'** or **'>'** to data type.**'<'** means little endian while **'>'** means big-endian.





NUMPY – DATA TYPES

- Construct a **data type object**:

```
numpy.dtype(object, align, copy)
```

  - **Object**: To be converted to data type object
  - **Align**: If true, adds padding to the field to make it similar to C-struct
  - **Copy**: Makes a new copy of dtype object. If false, the result is reference to built-in

NUMPY – DATA TYPES

- Construct a **data type object**:

```
numpy.dtype(object, align, copy)
```

  - **Object**: To be converted to data type object
  - **Align**: If true, adds padding to the field to make it similar to C-struct
  - **Copy**: Makes a new copy of dtype object. If false, the result is reference to built-in

NUMPY – DATA TYPES

In [8]:
```
import numpy as np
dt=np.dtype(np.int32)
print(dt)
```

```
int32
```

In [9]:
```
import numpy as np
dt = np.dtype('i4')
print (dt)
```

#int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4',etc.

```
int32
```

In [10]:
```
import numpy as np
dt = np.dtype('>i4')
print (dt)
```

```
>i4
```

NUMPY – DATA TYPES

- The use of structured data type：

```
In [12]: # # first create structured data type
         import numpy as np
         dt = np.dtype([('age',np.int8)])
         print (dt)

         [('age', 'i1')]
```

```
In [13]: # # now apply it to ndarray object
         import numpy as np
         dt = np.dtype([('age',np.int8)])
         a = np.array([(10,),(20,),(30,)], dtype=dt)
         print (a)

         [(10,) (20,) (30,)]
```

```
In [15]: # file name can be used to access content of age column
         import numpy as np
         dt = np.dtype([('age',np.int8)])
         a = np.array([(10,),(20,),(30,)], dtype=dt)
         print (a['age'])

         [10 20 30]
```

NUMPY – DATA TYPES

- A structured data type called **student** with a **string field** 'name',an **integer field** 'age' and a **float field** 'marks'.

```
In [16]: import numpy as np
         student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
         print (student)

[('name', 'S20'), ('age', 'i1'), ('marks', '<f4')]
```

```
In [17]: import numpy as np
         student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
         a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype=student)
         print (a)

[(b'abc', 21, 50.) (b'xyz', 18, 75.)]
```

NUMPY – ARRAY ATTRIBUTES

- **ndarray.shape :** return array **dimensions** or **resize** the array.

In [19]:
```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print (a.shape)
```

```
(2, 3)
```

In [20]:
```
# this resizes the ndarray
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
a.shape=(3,2)
print (a)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

In [21]:
```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print (b)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

NUMPY – ARRAY ATTRIBUTES

- **ndarray.ndim :** returns the number of array dimensions

```
In [23]: # this is one dimensional array
         import numpy as np
         a = np.arange(24)
         a.ndim

Out[23]: 1

In [24]: # now reshape it
         b = a.reshape(2,4,3)
         print (b)
         # b is having three dimensions

[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]
  [ 9 10 11]]

 [[12 13 14]
  [15 16 17]
  [18 19 20]
  [21 22 23]]]

In [25]: b.ndim

Out[25]: 3
```

NUMPY – ARRAY ATTRIBUTES

- **numpy.itemsize：** returns the length of each element of array in bytes.

```
In [26]: # dtype of array is int8 (1 byte)
         import numpy as np
         x = np.array([1,2,3,4,5], dtype=np.int8)
         print (x.itemsize)

         1
```

```
In [27]: # dtype of array is now float32 (4 bytes)
         import numpy as np
         x = np.array([1,2,3,4,5], dtype=np.float32)
         print (x.itemsize)

         4
```

NUMPY – ARRAY ATTRIBUTES

- **numpy.flags**：return the following attributes

C_CONTIGUOUS (C)	The data is in a single, C-style contiguous segment
F_CONTIGUOUS (F)	The data is in a single, Fortran-style contiguous segment
OWNDATA (O)	The array owns the memory it uses or borrows it from another object
WRITEABLE (W)	The data area can be written to. Setting this to False locks the data, making it read-only
ALIGNED (A)	The data and all elements are aligned appropriately for the hardware
UPDATEIFCOPY (U)	This array is a copy of some other array. When this array is deallocated, the base array will be updated with the contents of this array

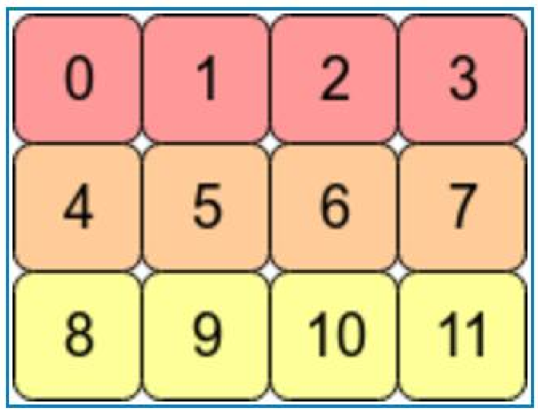
NUMPY – ARRAY ATTRIBUTES

- **numpy.flags**：return the following attributes

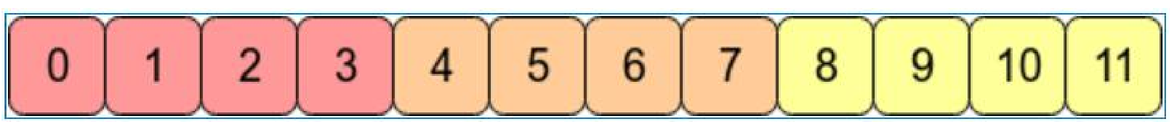
C_CONTIGUOUS (C)	The data is in a single, C-style contiguous segment
F_CONTIGUOUS (F)	The data is in a single, Fortran-style contiguous segment
OWNDATA (O)	The array owns the memory it uses or borrows it from another object
WRITEABLE (W)	The data area can be written to. Setting this to False locks the data, making it read-only
ALIGNED (A)	The data and all elements are aligned appropriately for the hardware
UPDATEIFCOPY (U)	This array is a copy of some other array. When this array is deallocated, the base array will be updated with the contents of this array

NUMPY – ARRAY ATTRIBUTES

- Consider the 2D array `arr = np.arange(12).reshape(3,4)`. It looks like this:



- the values of `arr` are stored like this (C contiguous) :



NUMPY – ARRAY ATTRIBUTES

- Transposing the array with `arr.T` , it is Fortran contiguous.

0	4	8
1	5	9
2	6	10
3	7	11

- Column-wise operations are usually faster than row-wise operations.

NUMPY – ARRAY ATTRIBUTES

In [28]:
```
import numpy as np
x = np.array([1,2,3,4,5])
print (x.flags)
```

```
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
WRITEBACKIFCOPY : False
UPDATEIFCOPY : False
```

NUMPY – ARRAY CREATION ROUTINES

- **numpy.empty:** It creates an uninitialized array of specified shape and dtype.

```
numpy.empty(shape, dtype=float, order='C')
```

Shape	Shape of an empty array in int or tuple of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

NUMPY – ARRAY CREATION ROUTINES

In [30]:
```
import numpy as np
x = np.empty([3,2], dtype=int)
print (x)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

In [31]:
```
import numpy as np
x = np.empty([3,2], dtype=int)
print (x)
```

```
[[         0 1072693248]
 [         0 1073741824]
 [         0 1074266112]]
```

NUMPY – ARRAY CREATION ROUTINES

- **numpy.zeros:** Returns a new array of specified size, filled with zeros.

```
numpy.zeros(shape, dtype=float, order='C')
```

Shape	Shape of an empty array in int or sequence of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

NUMPY – ARRAY CREATION ROUTINES

In [32]:
```
# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print (x)
```

```
[0. 0. 0. 0. 0.]
```

In [33]:
```
import numpy as np
x = np.zeros((5,), dtype=np.int)
print (x)
```

```
[0 0 0 0 0]
```

In [34]:
```
# custom type
import numpy as np
x = np.zeros((2,2), dtype=[('x', 'i4'), ('y', 'i4')])
print(x)
```

```
[[(0, 0) (0, 0)]
 [(0, 0) (0, 0)]]
```

NUMPY – ARRAY CREATION ROUTINES

- **numpy.ones**

```
numpy.ones(shape, dtype=None, order='C')
```

In [35]:

```
# array of five ones. Default dtype is float
import numpy as np
x = np.ones(5)
print (x)
```

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

NUMPY – ARRAY FROM EXISTING DATA

- **numpy.asarray:** convert Python sequence into ndarray.

```
numpy.asarray(a, dtype=None, order=None)
```

a	Input data in any form such as list, list of tuples, tuples, tuple of tuples or tuple of lists
dtype	By default, the data type of input data is applied to the resultant ndarray
order	C (row major) or F (column major). C is default

NUMPY – ARRAY FROM EXISTING DATA

In [37]:
```
# dtype is set
import numpy as np
x = [1,2,3]
a = np.asarray(x, dtype=float)
print (a)
```

```
[1. 2. 3.]
```

In [39]:
```
# ndarray from list of tuples
import numpy as np
x = [(1,2,3),(4,5)]
a = np.asarray(x)
print (a)
```

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

NUMPY – ARRAY FROM EXISTING DATA

- **numpy.frombuffer:** interprets a buffer as one-dimensional array. Any object that exposes the buffer interface is used as parameter to return an ndarray.

```
numpy.frombuffer(buffer, dtype=float, count=-1, offset=0)
```

buffer	Any object that exposes buffer interface
dtype	Data type of returned ndarray. Defaults to float
count	The number of items to read, default -1 means all data
offset	The starting position to read from. Default is 0

NUMPY – ARRAY FROM EXISTING DATA

In [41]:
```
import numpy as np
s = 'Hello World'
a = np.frombuffer(s, dtype='S1')
print (a)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-41-a974618e1386> in <module>()
      1 import numpy as np
      2 s = 'Hello World'
----> 3 a = np.frombuffer(s, dtype='S1')
      4 print (a)

AttributeError: 'str' object has no attribute '__buffer__'
```

In [44]:
```
import numpy as np
a=np.frombuffer(b'hello world',dtype='S1')
print(a)
```

```
[b'h' b'e' b'l' b'l' b'o' b' ' b'w' b'o' b'r' b'l' b'd']
```

In [45]:
```
import numpy as np
s = b'Hello World'
a = np.frombuffer(s, dtype='S1')
print (a)
```

```
[b'H' b'e' b'l' b'l' b'o' b' ' b'W' b'o' b'r' b'l' b'd']
```

In PY3, the default string type is unicode. The b is used to create and display bytestrings.

NUMPY – ARRAY FROM EXISTING DATA

- **numpy.fromiter**:build an one-dimensional ndarray object from any iterable object.

```
numpy.fromiter(iterable, dtype, count=-1)
```

```
In [46]: # create list object using range function
         import numpy as np
         list = range(5)
         print (list)

range(0, 5)

In [47]: # obtain iterator object from list
         import numpy as np
         list = range(5)
         it = iter(list)
         # use iterator to create ndarray
         x = np.fromiter(it, dtype=float)
         print (x)

[0. 1. 2. 3. 4.]
```

NUMPY – ARRAY FROM NUMERICAL RANGES

- **numpy.arange :** returns an ndarray object containing evenly spaced values within a given range.

```
numpy.arange(start, stop, step, dtype)
```

start	The start of an interval. If omitted, defaults to 0
stop	The end of an interval (not including this number)
step	Spacing between values, default is 1
dtype	Data type of resulting ndarray. If not given, data type of input is used

NUMPY – ARRAY FROM NUMERICAL RANGES

In [48]:
```
import numpy as np
x = np.arange(5)
print (x)
```
```
[0 1 2 3 4]
```

In [49]:
```
import numpy as np
# dtype set
x = np.arange(5, dtype=float)
print (x)
```
```
[0. 1. 2. 3. 4.]
```

In [50]:
```
# start and stop parameters set
import numpy as np
x = np.arange(10,20,2)
print (x)
```
```
[10 12 14 16 18]
```

NUMPY – ARRAY FROM NUMERICAL RANGES

- **numpy.linspace：** similar to arange() function，In this function, the number of evenly spaced values between the interval is specified.

```
numpy.linspace(start, stop, num, endpoint, retstep,
dtype)
```

start	The starting value of the sequence
stop	The end value of the sequence, included in the sequence if endpoint set to true
num	The number of evenly spaced samples to be generated. Default is 50
endpoint	True by default, hence the stop value is included in the sequence. If false, it is not included
retstep	If true, returns samples and step between the consecutive numbers
dtype	Data type of output **ndarray**

```
In  [57]: import matplotlib.pyplot as plt
N = 8
y = np.zeros(N)
x1 = np.linspace(0, 10, N, endpoint=True)
x2 = np.linspace(0, 10, N, endpoint=False)
%matplotlib inline
plt.plot(x1, y, 'o')
#[<matplotlib.lines.Line2D object at 0x...>]
plt.plot(x2, y + 0.5, 'o')
#[<matplotlib.lines.Line2D object at 0x...>]
plt.ylim([-0.5, 1])
(-0.5, 1)
plt.show()
```