NumPy

PROF. JACQUES SAVOY
UNIVERSITY OF NEUCHATEL

NumPy

Package (in Python) to manipulate vectors, matrix, and having many mathematical functions.

NumPy = Numerical Python is *de facto* a standard library.

Useful to manipulate large dataset.

Yes, you can do the same computation without NumPy... but with more effort.

```
>>> import numpy as np
```

Creating Objects

```
>>> a = np.array([1.0, 0.5, 0.33, 0.25, 0.2])
>>> a        # all elements must be of the same type (not the case with list in Python)
   array([1.  , 0.5 , 0.33, 0.25, 0.2 ])
>>> a = np.array([0, 1, 1, 2, 3, 5], dtype='int32')  #by default int64
>>> print(a.dtype)
   int32
>>> a
   array([ 1,  3,  6, 10, 15])     # clearly integer
>>> a = a.astype('float32')    # Convert into float or into  float64
>>> print(a.dtype)
   float32
>>> a
   array([0., 1., 1., 2., 3., 5.], dtype=float32)    # now float
```

Creating Objects

```
>>> a = np.array([0, 1, 1, 2, 3, 5])     # create a vector
>>> print(a.ndim)    # dimensions are important in NumPY
   1
>>> a = np.array([[0, 1, 2], [1, 0, 2], [2, 1, 0]]) # create a matrix
>>> print(a.ndim)
   2
>>> print(a.shape)    # the size in each dimension
    (3, 3)    # two values = two dimensions
>>> a = np.array([[[1, 3, 3], [2, 5, 2]], [[2, 3, 7], [4, 5, 9]]])
>>> print(a.ndim)
   3
```

Creating Objects

```
>>> print(np.zeros((3, 5)))      # because growing and shrinking are expensive
   [[0. 0. 0. 0. 0.]
    [0. 0. 0. 0. 0.]
    [0. 0. 0. 0. 0.]]
>>> print(np.zeros(10))
   [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
>>> print(np.ones((3, 4), dtype='int64'))
   [[1 1 1 1]
    [1 1 1 1]
    [1 1 1 1]]
>>> print(np.empty((3, 2)))  # not fully sure about the content of each cell
   [[0.0e+000 4.9e-324]
    [4.9e-324 9.9e-324]
    [1.5e-323 2.5e-323]]
```

Creating Objects

```
>>> np.random.seed(1365473)    # set the starting point of the random generation
>>> print(np.random.random_sample(5))    # full it with random values
   [0.63923325 0.06306471 0.2839203  0.71309518 0.06293769]

>>> print(np.random.random_sample((2, 3)))
   [[0.55002934 0.34723758 0.05313465]
    [0.66523458 0.09903012 0.68039935]]
>>> a = np.arange(0, 2, 0.25)     # all values between two limits and a step
>>> print(a)
   [0.   0.25 0.5  0.75 1.   1.25 1.5  1.75]
```

Indexing and Slicing Arrays

```
>>> a = np.arange(10)    # ten values from 0 to 9 (range)
>>> a
   array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> print(a[5])    # similar to Python
   5
>>> print(a[3:8])
   [3 4 5 6 7]
>>> a[-1:]     # the last element
   array([9])
>>> a[-2:]     # the last two elements
   array([8, 9])
```

Indexing and Slicing Arrays

```
>>> a
   array([[ 0,  1,  2],
          [11,  10, 12],
          [22,  21, 20]])
>>> a[1,2]     # extract a specific element  (index starts at 0)
  12
>>> a[2,0]      # extract a specific element
   22
>>> a[1, :]   # extract a sequence of elements
 array([11, 10, 12])

>>> a[:2, 0:3]   # extract from the first two rows, the first three elements
   array([[ 0,  1,  2],
          [11, 10, 12]])
```

Indexing and Slicing Arrays

```
>>> a
   array([[ 0,  1,  2],
          [11, 10, 12],
          [22, 21, 20]])

>>> a[(0,2), :]   # extract from the first and third row, all items
 array([[ 0,  1,  2],
        [22, 21, 20]])
```

Boolean Vector

```
>>> numbers = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
>>> print([number for number in numbers if number < 10])
   [0, 1, 1, 2, 3, 5, 8]
>>> numbers = np.array(numbers)
>>> print(numbers[numbers < 10])
   [0 1 1 2 3 5 8]
>>> print(numbers < 10)
   [ True  True  True  True  True  True  True False False False False]
```

Computing with Arrays

```
>>> numbers = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
>>> print([number * 10 for number in numbers])
   [0, 10, 10, 20, 30, 50, 80, 130, 210, 340, 550]
>>> numbers = np.array(numbers)
>>> print(numbers * 10)
   [  0  10  10  20  30  50  80 130 210 340 550]
```

Computing with Arrays

```
>>> np.random.seed(1365473)    # set the starting point of the random generation
>>> numbers = np.random.random_sample(100000)
>>> print(sum(numbers))
    50048.55835134195
>>> print(numbers.sum())    # equivalent to np.sum(numbers)
    50048.55835134195
```

Timing

```
>>> import timeit
>>> import_module = "import random"
>>> testcode = '''
 ... def test():
 ...     return random.randint(10, 100)   # a random value between 10 and 100
 ... '''
>>> print(timeit.repeat(stmt=testcode, setup=import_module))
   [0.06437887100000239, 0.05907883500000821, 0.05818968499998789,
      0.05817810499996094, 0.05733122600000229]
```

Timing

```
>>> import_module = "import random; numpy as np; numbers = np.random.random_sample(1000000)"
>>> testcode = '''
def test():
    sum(numbers)    # strategy A (Python)
'''
>>> print(timeit.repeat(stmt=testcode, setup=import_module))
   [0.06105757700061076, 0.05761055100083467 6, 0.056453124001563 99,
     0.05573070199898211, 0.05890514799830271]
>>> testcode = '''
def test():
    numbers.sum()    # strategy B (NumPy)
'''
>>> print(timeit.repeat(stmt=testcode, setup=import_module))
  [0.05956420700022136, 0.06062703599673114, 0.05995739800346 2004,
    0.05892276899976423, 0.05644585600020946]
```

Computing with Arrays

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([2, 4, 6])
>>> print(a * b)
   [ 2  8 18]
>>> a = np.array([1, 2, 3])
>>> print(a * 2)
   [2 4 6]
```

Computing with Arrays

```
>>> a
   array([[ 0,  1,  2],
          [11, 10, 12],
          [22, 21, 20]])

>>> a.sum()
   99                    # apply the sum function on the entire matrix

>>> a.sum(axis=0)    # apply the sum function on each column (axis = 0)
   array([33, 32, 34])

>>> a.sum(axis=1)    # apply the sum function on each row (axis = 1)
 array([ 3, 33, 63])
```

Computing with Arrays

```
>>> a.min()
   99                    # apply the min function on the entire matrix
>>> a.min(axis=1)    # apply the sum function on each row (axis = 1)
   array([0, 10, 20])
>>> a.mean(axis=0)     # apply the sum function on each column (axis = 0)
   array([11.          , 10.66666667, 11.33333333])
```