# Python: beyond the basics

Albert DeFusco

September 19, 2013

# Numpy/Scipy Arrays

- Homogeneous data types
- Contiguous memory layout
- Mathematical operations are much faster than lists
- Fixed length but mutable
- Multidimensional; row major
- For loops are slow
  - Built-in operations make use of compiled code
  - Array operations are auto-vectorized

http://docs.scipy.org/doc/numpy/reference/index.html

```
>>> import numpy as np
>>> help(np.<type/method>)
```

# Array Creation

```
import numpy as np
```

- Manual

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

- Creators
  - Zero

    ```
    np.zeros(N,M)
    ```

  - Ranges

    ```
    np.arange(start,stop,increment)
    np.linspace(start,stop,n)
    ```

    $\text{start} <= x <= \text{stop}$ for $n$ elements
  - Identity

    ```
    np.ones(N,M)
    ```

  - Unit diagonal

    ```
    np.eye(N,M)
    ```

  - Diagonal

    ```
    np.diag(<1d-array>)
    ```

  - Casting

    ```
    np.array([i*0.5 for i in x])
    ```

  - Filling

    ```
    a=np.empty((3,3));a.fill(42)
    ```

  - Tiling

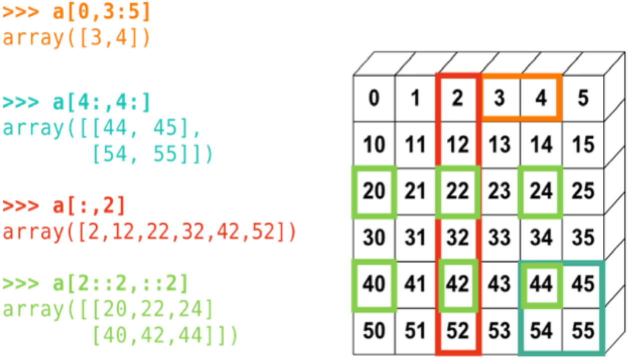
    ```
    a=np.tile(np.arange(0,3),(3,1))
    ```

# Array types

- The default is double precision

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> a.dtype
dtype(’int64’)
>>> a = np.array([1.,2.,3.])
>>> a.dtype
dtype(’float64’)
>>> a = np.ones(3,dtype=np.int32)
>>> a.dtype
dtype(’int32’)
```

# Array Slicing



```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 10 | 11 | 12 | 13 | 14 | 15 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 30 | 31 | 32 | 33 | 34 | 35 |
| 40 | 41 | 42 | 43 | 44 | 45 |
| 50 | 51 | 52 | 53 | 54 | 55 |

http://scipy-lectures.github.io/intro/numpy/array_object.html#indexing-and-slicing

# Array operations

- Rshape

  ```
  x.reshape(3,3)
  ```

- Transpose

  ```
  x.T
  ```

- Elementwise operations

  ```
  x**2
  x * y
  x + 1
  x > y
  np.sin(x)
  ```

- Matrix multiplication

  ```
  x.dot(y)
  ```

- Masking creates copies

  ```
  x[ x % 3== 0]
  ```

- Avoid loops
  - The above operations are vectorized and very efficient

# Views

Slicing and other operations create a view, not a copy

```
> python
>>> import numpy as np
>>> x = np.array([[1.,2.],[3.,4.]])
>>> x
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> y = x.T
>>> z = x.T.copy()
>>> y
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> x[1] = 10.
>>> x
array([[  1.,   2.],
       [ 10.,  10.]])
>>> y
array([[  1.,  10.],
       [  2.,  10.]])
>>> z
array([[  1.,  2.],
       [  3.,  4.]])
```

# Arrays without loops

1. Can you compute $\pi$ faster?
   - Begin by reading `pydoc numpy.lib`
   - Can you do it without loops?
   - I got a factor of 20 speed-up over last week's answer

# Linear Algebra

```
pydoc scipy.linalg
```

# Array exercises

1. Devise a generic Hückel solver for butadiene
    - How would you build the matrix?
    - How do you find the solution?
      $\alpha = 0$ and $\beta = 1$

$$\begin{bmatrix} \alpha - E & \beta & 0 & 0 \\ \beta & \alpha - E & \beta & 0 \\ 0 & \beta & \alpha - E & \beta \\ 0 & 0 & \beta & \alpha - E \end{bmatrix}$$