`numpy`: Numerical Python

"Duck" typing makes Python slow

- Duck Typing
  - If it looks like a duck, then it is a duck.
  - a.k.a. dynamic typing
  - Dynamic typing requires lots of metadata around a variable.

- Solution: `numpy` data structures
  - Data structures, as objects, that have a single type and continuous storage.
  - Common functionality with implementation in C.

How slow is Python?

- Add 1 to a million numbers
  - Use `timeit`

```
%timeit [i+1 for i in range(1000000)]
```

```
110 ms ± 13.1 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
import numpy
%timeit numpy.arange(1000000) + 1
```

```
1.22 ms ± 121 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Universal functions

- Universal functions are vectorized functions that operate on arrays in an element-by-element fashion.
- Arithmetic operators (`+`, `-`, `/`, `*`, `**`) are overloaded to work in an element-by-element fashion.

Another speed comparison:

```
import math
%timeit [math.sin(i) ** 2 for i in range(1000000)]
```

```
322 ms ± 40.7 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
import numpy
%timeit numpy.sin(numpy.arange(1000000)) ** 2
```

```
25.3 ms ± 2.51 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Creating numpy arrays

numpy offers several built-in functions for creating arrays

```
import numpy
x = numpy.array([2,3,11])
x = numpy.array([[1,2.],[0,0],[1+1j,2.]])
x = numpy.arange(-10,10,2, dtype=float)
x = numpy.linspace(1.,4.,6)
x = numpy.indices((3,3))
x = numpy.fromfile('foo.dat')
```

Array functions

numpy array functions for slicing, getting info, etc.

```
import numpy as np
x = np.arange(9).reshape(3,3)
x
```

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

```
x[:,0]
```

```
array([0, 3, 6])
```

```
x.shape
```

```
(3, 3)
```

```
y = x[::2, ::2]
y
```

```
array([[0, 2],
       [6, 8]])
```

Efficient and compact finite differences

```
x = np.arange(0,20,2)
y = x ** 2

dy_dx = (y[1:] - y[:-1]) / (x[1:] - x[:-1])
dy_dx
```

```
array([  2.,   6.,  10.,  14.,  18.,  22.,  26.,  30.,  34.])
```

Sophisticated broadcasting rules

```
red = np.random.rand(800,600)
blue = np.random.rand(800, 600)
green = np.random.rand(800, 600)
rgb = np.array([red, blue, green])
rgb.shape
```

```
(3, 800, 600)
```