numpy: Introduction

Let's import the `numpy` module.

In [1]:

```
import numpy as np
```

In [2]:

```
n = 10  # CHANGE ME
a1 = list(range(n))
a2 = np.arange(n)

if n <= 10:
    print(a1)
    print(a2)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0 1 2 3 4 5 6 7 8 9]
```

In [3]:

```
%timeit [i**2 for i in a1]
```

```
1000000 loops, best of 3: 1.41 µs per loop
```

In [4]:

```
%timeit a2**2
```

```
1000000 loops, best of 3: 521 ns per loop
```

Numpy Arrays: much less flexible, but:

- much faster
- less memory

Ways to create a numpy array:

- Casting from a list

In [5]:

```
np.array([1,2,3])
```

Out[5]:

```
array([1, 2, 3])
```

- linspace

In [6]:

```
np.linspace(-1, 1, 10)
```

Out[6]:

```
array([-1.        , -0.77777778, -0.55555556, -0.33333333, -0.111111
11,
        0.11111111,  0.33333333,  0.55555556,  0.77777778,  1.
])
```

- zeros

In [7]:

```
np.zeros((10,10), np.float64)
```

Out[7]:

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

Operations on arrays propagate to all elements:

In [8]:

```
a = np.array([1.2, 3, 4])
b = np.array([0.5, 0, 1])
```

Addition, multiplication, power, .. are all elementwise:

In [10]:

```
a+b
```

Out[10]:

```
array([ 1.7,  3. ,  5. ])
```

In [11]:

```
a*b
```

Out[11]:

```
array([ 0.6,  0. ,  4. ])
```

In [12]:

```
a**b
```

Out[12]:

```
array([ 1.09544512,  1.        ,  4.        ])
```

Matrix multiplication is `np.dot(A, B)` for two 2D arrays.

Numpy arrays have two (most) important attributes:

In [13]:

```
a = np.random.rand(5, 4, 3)
a.shape
```

Out[13]:

```
(5, 4, 3)
```

In [14]:

```
a.dtype
```

Out[14]:

```
dtype('float64')
```

Other `dtypes` include `np.complex64`, `np.int32`, ...