

PSYC 027: Scientific Computing for Psychology

3 October 2019

Professor Youssef Ezzyat
McCabe Library 306
T/Th 9:55-11:10

Numpy and Scipy

- Numpy = numerical python
 - Core library for mathematical/scientific computing in Python
- Scipy = scientific Python
 - Extended library for mathematical/scientific computing
 - Relies on Numpy

```
import numpy as np
```



Why Numpy?

- Provides an easy to use data structure (the Numpy array) that can represent multi-dimensional data
- Optimizes calculations over large arrays

Why Numpy?

- Provides a large library of functions/methods for performing numerical operations

```
In [23]: import numpy as np
```

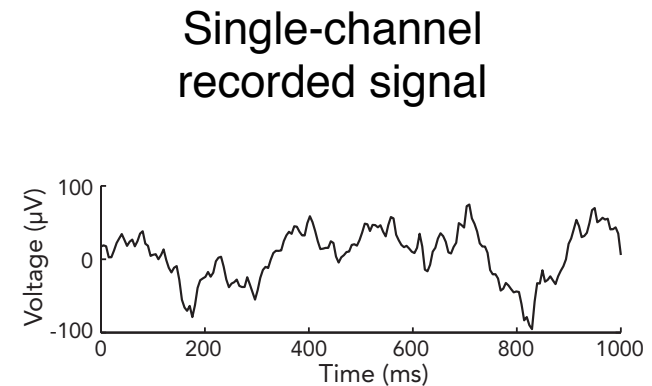
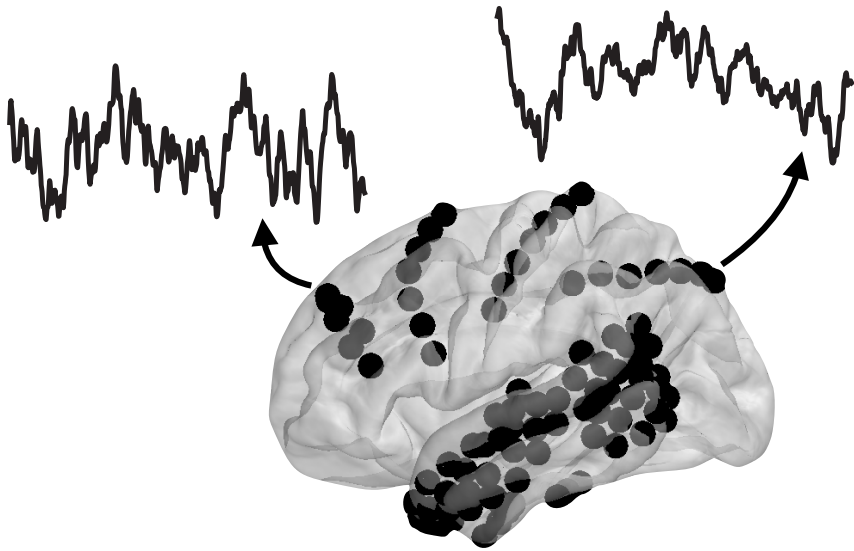
```
In [25]:
```

```
log10  
log1p  
log2  
logaddexp  
logaddexp2  
logical_and  
logical_not  
logical_or  
logical_xor  
logspace
```

```
In [ ]: np.
```

What kinds of data?

- Multidimensional behavioral and/or neural data



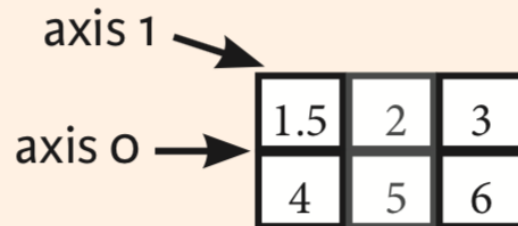
Numpy

- Primary numpy object is a multidimensional array containing objects all of the same type

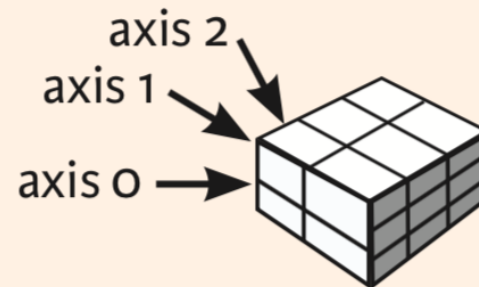
1D array



2D array



3D array



**N-D
arrays**

...

Numpy

- Creating numpy arrays

```
In [7]: # Creating a numpy array of floats (e.g. decimal numbers)  
arr = np.array([1, 2, 3, 4], dtype='float')  
print(type(arr))  
arr
```

```
<class 'numpy.ndarray'>
```

```
Out[7]: array([1., 2., 3., 4.])
```

Numpy

- Unlike Python lists, the values within numpy arrays must be of the same type
- If you mix types, numpy will try to 'upcast' values

```
arr = np.array([1, 2.1, 3, 4])  
print(type(arr))  
arr
```

```
<class 'numpy.ndarray'>
```

```
Out[11]: array([1. , 2.1, 3. , 4. ])
```


Numpy

- Creating ranges is straightforward with numpy

```
In [19]: print(np.arange(0,1,.1))  
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

Numpy

- Numpy's `arange` can be used to create decimal ranges, which are not supported by the standard python `range` function

```
In [20]: print(np.arange(0,1,.1))
```

```
range(0,1,.1)
```

```
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

```
-----  
-----  
TypeError                                Traceback (most recent call 1  
ast)  
<ipython-input-20-11fb656ba0f6> in <module>()  
      1 print(np.arange(0,1,.1))  
      2  
----> 3 range(0,1,.1)
```

```
TypeError: 'float' object cannot be interpreted as an integer
```

Numpy

- Indexing elements of a numpy array is similar to indexing elements of python lists
- For **multidimensional** arrays, you index using a set of numbers, where each position in the set corresponds to a dimension of the array

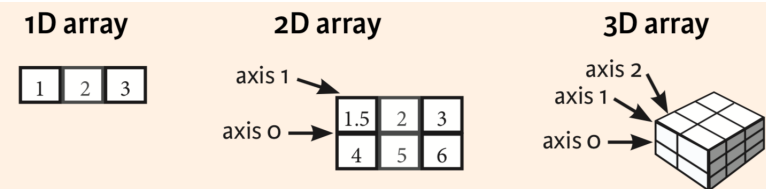
```
arr2d = np.array([[1,3,5],  
                  [2,4,6]],dtype=int)
```

```
print(arr2d)
```

```
print(arr2d[0,1])
```

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

Dim 1 Dim 2



Numpy

- Indexing elements of a numpy array is similar to indexing elements of python lists

```
arr = np.array([1, 2, 3, 4], dtype='float')  
print(arr[1])
```

2.0

Numpy

- You can also slice arrays to pull out an entire row or column (or other dimension...)

```
arr2d = np.array([[1, 2, 3, 4],  
                  [ 5, 6, 7, 8],  
                  [ 9, 10, 11, 12]])  
print(arr2d)
```

```
[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]]
```

```
print(arr2d[:,0])
```

```
[1 5 9]
```

**Grab all rows, first
column**

```
print(arr2d[1,:])
```

```
[5 6 7 8]
```

**Grab second row, all
columns**

Numpy

- Use slice notation to access sub-arrays of an array

```
array_big = np.array([[1, 2, 3, 4],  
                      [ 5, 6, 7, 8],  
                      [ 9, 10, 11, 12],  
                      [13, 14, 15, 16]])  
print(array_big)
```

```
[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]  
 [13 14 15 16]]
```

```
array_big[1:-1, 1:-1]
```

```
array([[ 6,  7],  
       [10, 11]])
```

```
: array_big[1:-1, 1:-1] = -1  
array_big
```

```
: array([[ 1,  2,  3,  4],  
         [ 5, -1, -1,  8],  
         [ 9, -1, -1, 12],  
         [13, 14, 15, 16]])
```

Numpy

- All of the indexing on the previous slides are 'views'
- If you want to **copy** data into a new object, use `.copy()`

```
arr2d = np.array([[1, 2, 3, 4],  
                  [ 5, 6, 7, 8],  
                  [ 9, 10, 11, 12]])  
print(arr2d)
```

```
[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]]
```

```
arr2d_slice = arr2d[1,:].copy()  
arr2d[1,1] = 25 # Changing the original array wont' change the copy  
  
print(arr2d)  
print('\n')  
print(arr2d_slice)
```

```
[[ 1  2  3  4]  
 [ 5 25  7  8]  
 [ 9 10 11 12]]
```

```
[5 6 7 8]
```

Numpy

- Numpy also has specialized methods for creating arrays filled with specific values

```
np.zeros((3, 3), dtype=float)
```

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

```
np.full([3,3], 5, dtype=int)
```

```
array([[5, 5, 5],  
       [5, 5, 5],  
       [5, 5, 5]])
```