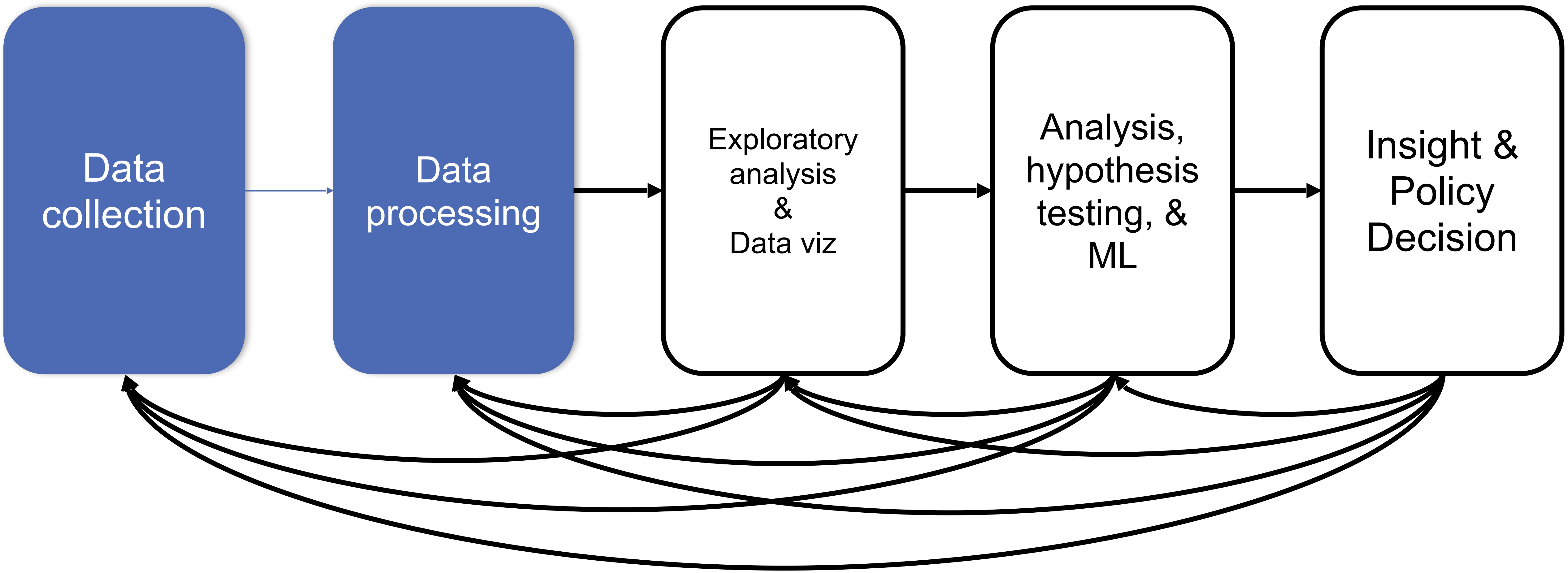
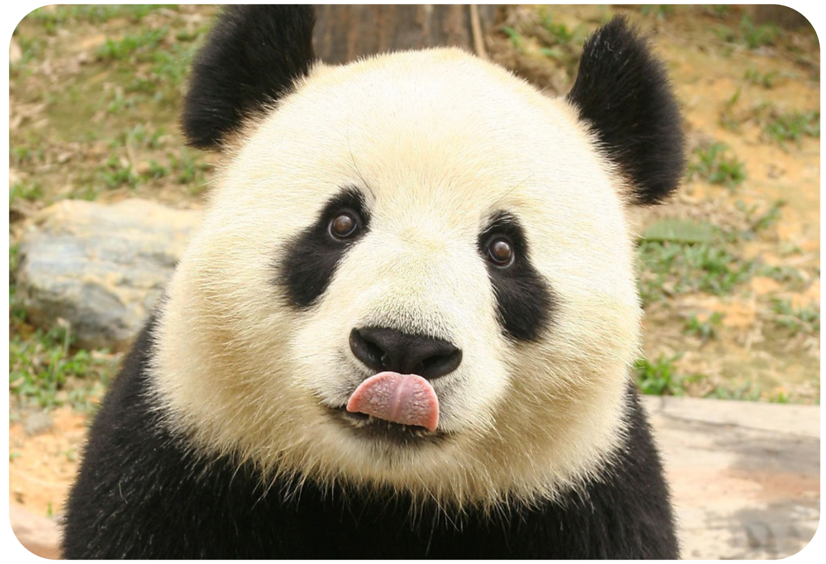
NEXT



Data collection
Data processing
Exploratory analysis & Data viz
Analysis, hypothesis testing, & ML
Insight & Policy Decision



NUMPY, SCIPY, AND DATAFRAMES

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

DATA MANIPULATION AND COMPUTATION

Data Science == manipulating and computing on data

Large to very large, but somewhat "structured" data

We will see several tools for doing that this semester

Thousands more out there that we won't cover

Need to learn to shift thinking from:

Imperative code to manipulate data structures

to:

Sequences/pipelines of operations on data

Should still know how to implement the operations themselves, especially for debugging performance

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Indexing
Slicing/subsetting
Filter
'map' → apply a function to every element
'reduce/aggregate' → combine values to get a single scalar (e.g., sum, median)

Given two vectors: **Dot and cross products**

One-dimensional Arrays, Vectors

0.1	2	3.2	6.5	3.4	4.1

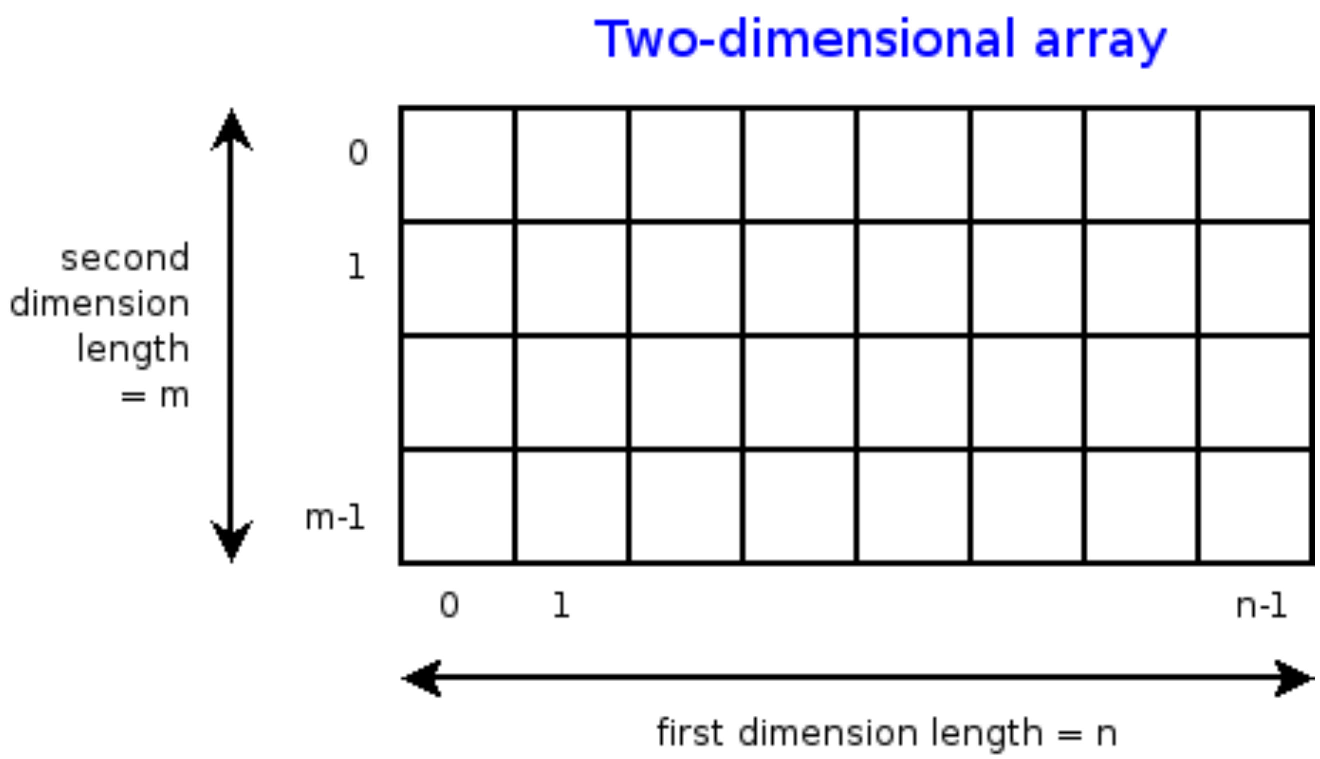
"data"	"representation"	"i.e."

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

n-dimensional arrays



Indexing
Slicing/subsetting
Filter
'map' → apply a function to every element
'reduce/aggregate' → combine values across a row or a column (e.g., sum, average, median etc..)

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

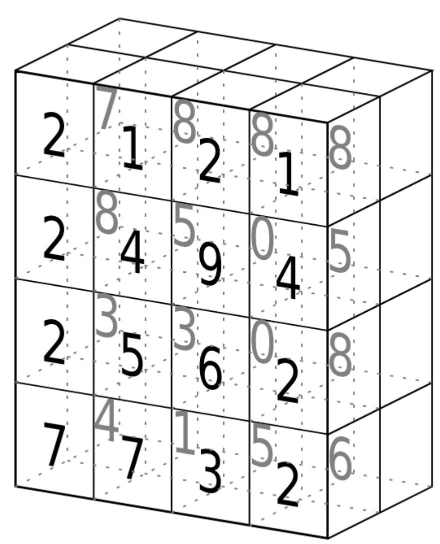
DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Matrices, Tensors

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

tensor of dimensions [6,4]
(matrix 6 by 4)



tensor of dimensions [4,4,2]

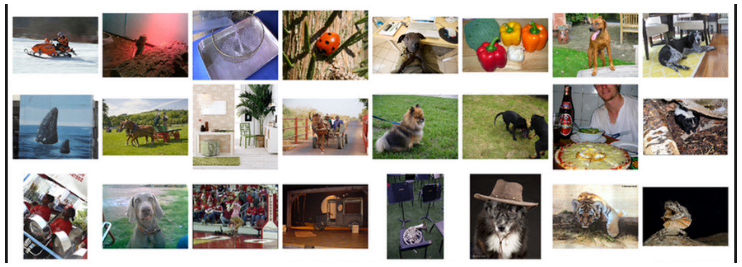
n-dimensional array operations
+
Linear Algebra
- **Matrix/tensor multiplication**
- **Transpose**
- **Matrix-vector multiplication**
- **Matrix factorization**

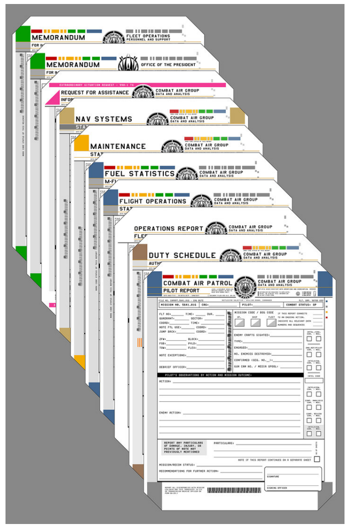
2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Sets: of Objects





Sets: of (Key, Value Pairs)

(juexu@cs.umd.edu,(email1, email2,…))
(nayeem@cs.umd.edu,(email3, email4,…))

Filter
Map
Union

Reduce/Aggregate

Given two sets, **Combine/Join** using "keys"

Group and then aggregate

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Tables/Relations == Sets of Tuples

company	division	sector	tryint
00nil_Combined_Company	00nil_Combined_Division	00nil_Combined_Sector	14625
apple	00nil_Combined_Division	00nil_Combined_Sector	10125
apple	hardware	00nil_Combined_Sector	4500
apple	hardware	business	1350
apple	hardware	consumer	3150
apple	software	00nil_Combined_Sector	5625
apple	software	business	4950
apple	software	consumer	675
microsoft	00nil_Combined_Division	00nil_Combined_Sector	4500
microsoft	hardware	00nil_Combined_Sector	1890
microsoft	hardware	business	855
microsoft	hardware	consumer	1035
microsoft	software	00nil_Combined_Sector	2610
microsoft	software	business	1215
microsoft	software	consumer	1395

Filter rows or columns

“Join” two or more relations

“Group” and “aggregate” them

Relational Algebra formalizes some of them

Structured Query Language (SQL)

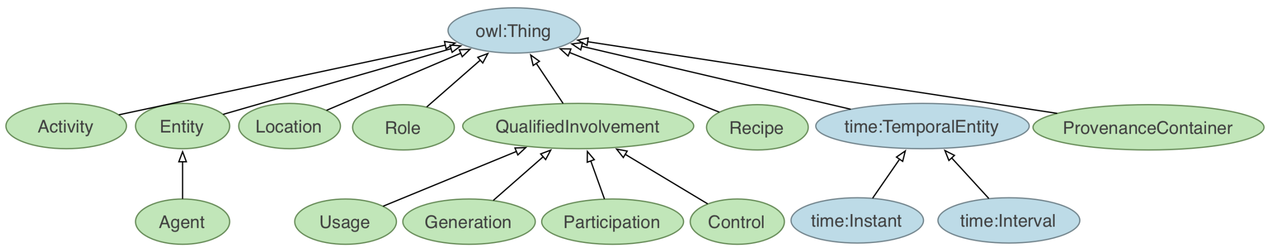
Many other languages and constructs, that look very similar

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data

Hierarchies/Trees/Graphs



"Path" queries

Graph Algorithms and Transformations

Network Science

Somewhat more ad hoc and special-purpose

Changing in recent years

2. **Data Processing Operations**, which take one or more datasets as input and produce one or more datasets as output

DATA MANIPULATION AND COMPUTATION

1. **Data Representation**, i.e., what is the natural way to think about given data
2. **Data Processing Operations**, which take one or more datasets as input and produce

- **Why?**
  - Allows one to think at a higher level of abstraction, leading to simpler and easier-to-understand scripts
  - Provides ”independence” between the abstract operations and concrete implementation
  - Can switch from one implementation to another easily

- **For performance debugging, useful to know how they are implemented and rough characteristics**

NEXT COUPLE OF CLASSES

1. **NumPy: Python Library for Manipulating nD Arrays**
   Multidimensional Arrays, and a variety of operations including Linear Algebra

2. **Pandas: Python Library for Manipulating Tabular Data**
   Series, Tables (also called **DataFrames**)
   Many operations to manipulate and combine tables/series

3. **Relational Databases**
   Tables/Relations, and SQL (similar to Pandas operations)

NEXT COUPLE OF CLASSES

1. **NumPy: Python Library for Manipulating nD Arrays**
   Multidimensional Arrays, and a variety of operations including Linear Algebra

2. **Pandas: Python Library for Manipulating Tabular Data**
   Series, Tables (also called **DataFrames**)
   Many operations to manipulate and combine tables/series

3. **Relational Databases**
   Tables/Relations, and SQL (similar to Pandas operations)

NUMERIC & SCIENTIFIC APPLICATIONS

Number of third-party packages available for numerical and scientific computing

These include:

- NumPy/SciPy – numerical and scientific function libraries.
- numba – Python compiler that support JIT compilation.
- ALGLIB – numerical analysis library.
- pandas – high-performance data structures and data analysis tools.
- pyGSL – Python interface for GNU Scientific Library.
- ScientificPython – collection of scientific computing modules.

NUMPY AND FRIENDS

By far, the most commonly used packages are those in the NumPy stack. These packages include:

- NumPy: similar functionality as Matlab
- SciPy: integrates many other packages like NumPy
- Matplotlib & Seaborn – plotting libraries
- iPython via Jupyter – interactive computing
- Pandas – data analysis library
- SymPy – symbolic computation library

THE NUMPY STACK

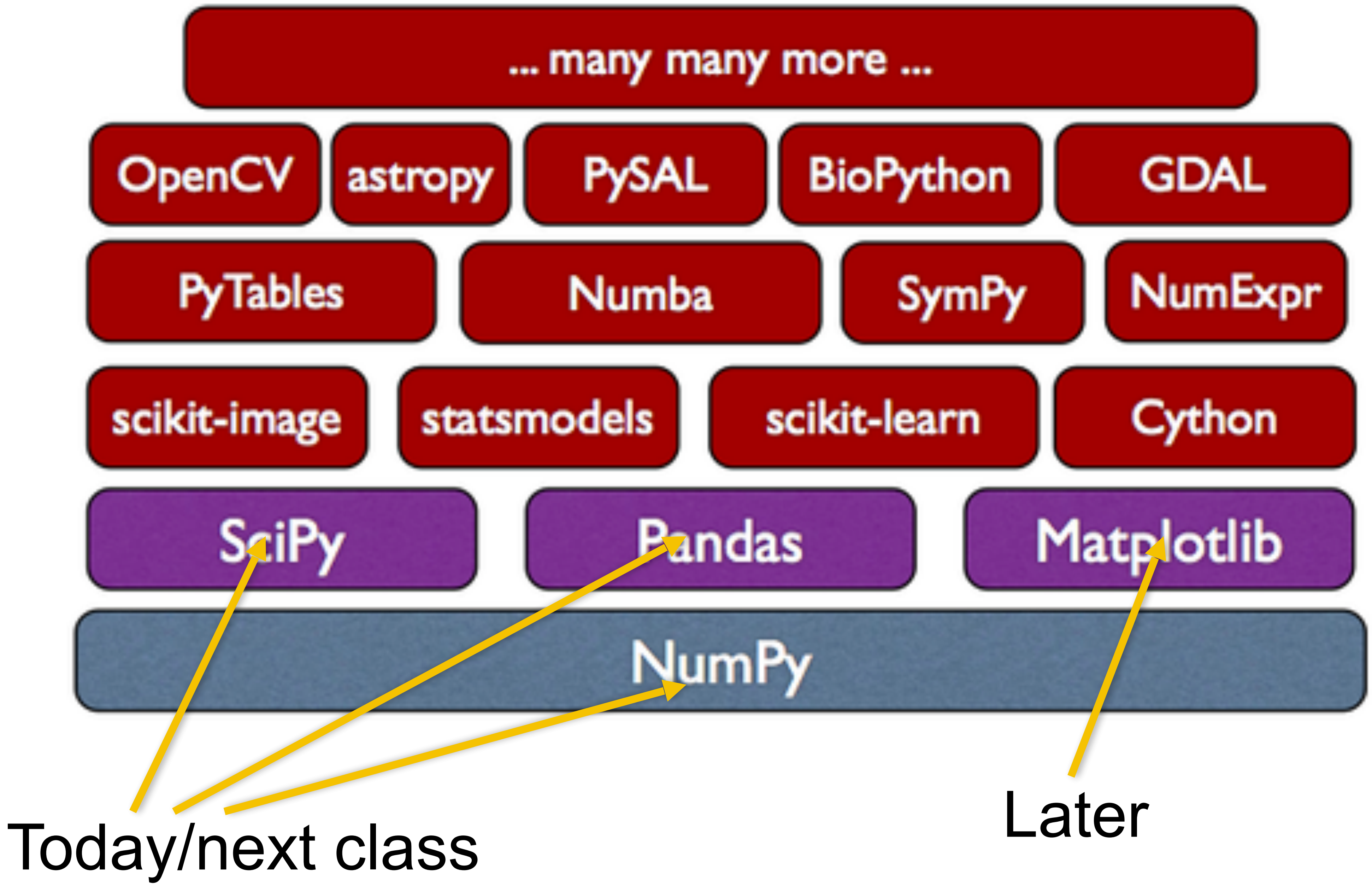


Image from Continuum Analytics

NUMPY

Among other things, NumPy contains:

- A powerful *n*-dimensional array object.
- Sophisticated (broadcasting/universal) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities, etc.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

[FSU]

NUMPY

`ndarray` object: an *n*-dimensional array of homogeneous data types, with many operations being performed in compiled code for performance

Several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size. Modifying the size means creating a new array.
- NumPy arrays must be of the same data type, but this can include Python objects – may not get performance benefits
- More efficient mathematical operations than built-in sequence types.

[FSU]

NUMPY DATATYPES

Wider variety of data types than are built-in to the Python language by default.

Defined by the `numpy.dtype` class and include:

- intc (same as a C integer) and intp (used for indexing)
- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float16, float32, float64
- complex64, complex128
- bool_, int_, float_, complex_ are shorthand for defaults.

These can be used as functions to cast literals or sequence types, as well as arguments to NumPy functions that accept the `dtype` keyword argument.

NUMPY DATATYPES

```
>>> import numpy as np
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
>>> z
array([0, 1, 2], dtype=uint8)
>>> z.dtype
dtype('uint8')
```

NUMPY ARRAYS

There are a few mechanisms for creating arrays in NumPy:

- Conversion from other Python structures (e.g., lists, tuples)
  - Any sequence-like data can be mapped to a ndarray
- Built-in NumPy array creation (e.g., `arange`, `ones`, `zeros`, etc.)
  - Create arrays with all zeros, all ones, increasing numbers from 0 to 1 etc.
- Reading arrays from disk, either from standard or custom formats (e.g., reading in from a CSV file)

[FSU]

NUMPY ARRAYS

In general, any numerical data that is stored in an array-like container can be converted to an `ndarray` through use of the `array()` function. The most obvious examples are sequence types like lists and tuples.

```
>>> x = np.array([2,3,1,0])

>>> x = np.array([2, 3, 1, 0])

>>> x = np.array([[1,2.0],[0,0],(1+1j,3.)])

>>> x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j],
[ 1.+1.j, 3.+0.j]])
```

NUMPY ARRAYS

Creating arrays from scratch in NumPy:

- `zeros(shape)` – creates an array filled with 0 values with the specified shape. The default `dtype` is `float64`.

```
>>> np.zeros((2, 3))
array([[ 0., 0., 0.], [ 0., 0., 0.]])
```

- `ones(shape)` – creates an array filled with 1 values.
- `arange()` – like Python's built-in `range`

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])
>>> np.arange(2, 3, 0.2)
array([ 2. , 2.2, 2.4, 2.6, 2.8])
```

[FSU]

NUMPY ARRAYS

`linspace()`– creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values.

```
>>> np.linspace(1., 4., 6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

`random.random(shape)` – creates arrays with random floats over the interval [0,1).

```
>>> np.random.random((2,3))
array([[ 0.75688597, 0.41759916, 0.35007419],
       [ 0.77164187, 0.05869089, 0.98792864]])
```

NUMPY ARRAYS

Printing an array can be done with the print

- statement (Python 2)
- function (Python 3)

```
>>> import numpy as np
>>> a = np.arange(3)
>>> print(a)
[0 1 2]
>>> a
array([0, 1, 2])
>>> b = np.arange(9).reshape(3,3)
>>> print(b)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> c = np.arange(8).reshape(2,2,2)
>>> print(c)
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

[FSU]

INDEXING

Single-dimension indexing is accomplished as usual.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```

```
>>> x.shape = (2,5) # now x is 2-dimensional array
>>> x[1,3]
8
>>> x[1,-1]
9
```

INDEXING

Using fewer dimensions to index will result in a subarray:

```
>>> x = np.arange(10)
>>> x.shape = (2,5)
>>> x[0]
array([0, 1, 2, 3, 4])
```

This means that `x[i, j] == x[i][j]` but the second method is less efficient.

INDEXING

Slicing is possible just as it is for typical Python sequences:

```
>>> x = np.arange(10)
>>> x[2:5]
array([2, 3, 4])
>>> x[:-7]
array([0, 1, 2])
>>> x[1:7:2]
array([1, 3, 5])
>>> y = np.arange(35).reshape(5,7)
>>> y[1:5:2,::3]
array([[ 7, 10, 13], [21, 24, 27]])
```

ARRAY OPERATIONS

Basic operations apply element-wise. The result is a new array with the resultant elements.

```
>>> a = np.arange(5)
>>> b = np.arange(5)
>>> a+b
array([0, 2, 4, 6, 8])
>>> a-b
array([0, 0, 0, 0, 0])
>>> a**2
array([ 0,  1,  4,  9, 16])
>>> a>3
array([False, False, False, False,  True], dtype=bool)
>>> 10*np.sin(a)
array([ 0., 8.41470985, 9.09297427, 1.41120008,
-7.56802495])
>>> a*b
array([ 0,  1,  4,  9, 16])
```

ARRAY OPERATIONS

Since multiplication is done element-wise, you need to specifically perform a dot product to perform matrix multiplication.

```
>>> a = np.zeros(4).reshape(2,2)
>>> a
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> a[0,0] = 1
>>> a[1,1] = 1
>>> b = np.arange(4).reshape(2,2)
>>> b
array([[0, 1],
       [2, 3]])
>>> a*b
array([[ 0.,  0.],
       [ 0.,  3.]])
>>> np.dot(a,b)
array([[ 0.,  1.],
       [ 2.,  3.]])
```

ARRAY OPERATIONS

There are also some built-in methods of `ndarray` objects.

Universal functions which may also be applied include `exp`, `sqrt`, `add`, `sin`, `cos`, etc.

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.68166391, 0.98943098,
0.69361582],
       [ 0.78888081, 0.62197125,
0.40517936]])
>>> a.sum()
4.1807421388722164
>>> a.min()
0.4051793610379143
>>> a.max(axis=0)
array([ 0.78888081, 0.98943098,
0.69361582])
>>> a.min(axis=1)
array([ 0.68166391, 0.40517936])
```

ARRAY OPERATIONS

An array shape can be manipulated by a number of methods.

`resize(size)` will modify an array in place.

`reshape(size)` will return a copy of the array with a new shape.

```
>>> a =
np.floor(10*np.random.random((3,4)))
>>> print(a)
[[ 9.  8.  7.  9.]
 [ 7.  5.  9.  7.]
 [ 8.  2.  7.  5.]]
>>> a.shape
(3, 4)
>>> a.ravel()
array([ 9.,  8.,  7.,  9.,  7.,  5.,  9.,  7.,
8.,  2.,  7.,  5.])
>>> a.shape = (6,2)
>>> print(a)
[[ 9.  8.]
 [ 7.  9.]
 [ 7.  5.]
 [ 9.  7.]
 [ 8.  2.]
 [ 7.  5.]]
>>> a.transpose()
array([[ 9.,  7.,  7.,  9.,  8.,  7.],
       [ 8.,  9.,  5.,  7.,  2.,  5.]])
```