GENERAL ASSEMBLY

# PYTHON NUMPY PANDAS

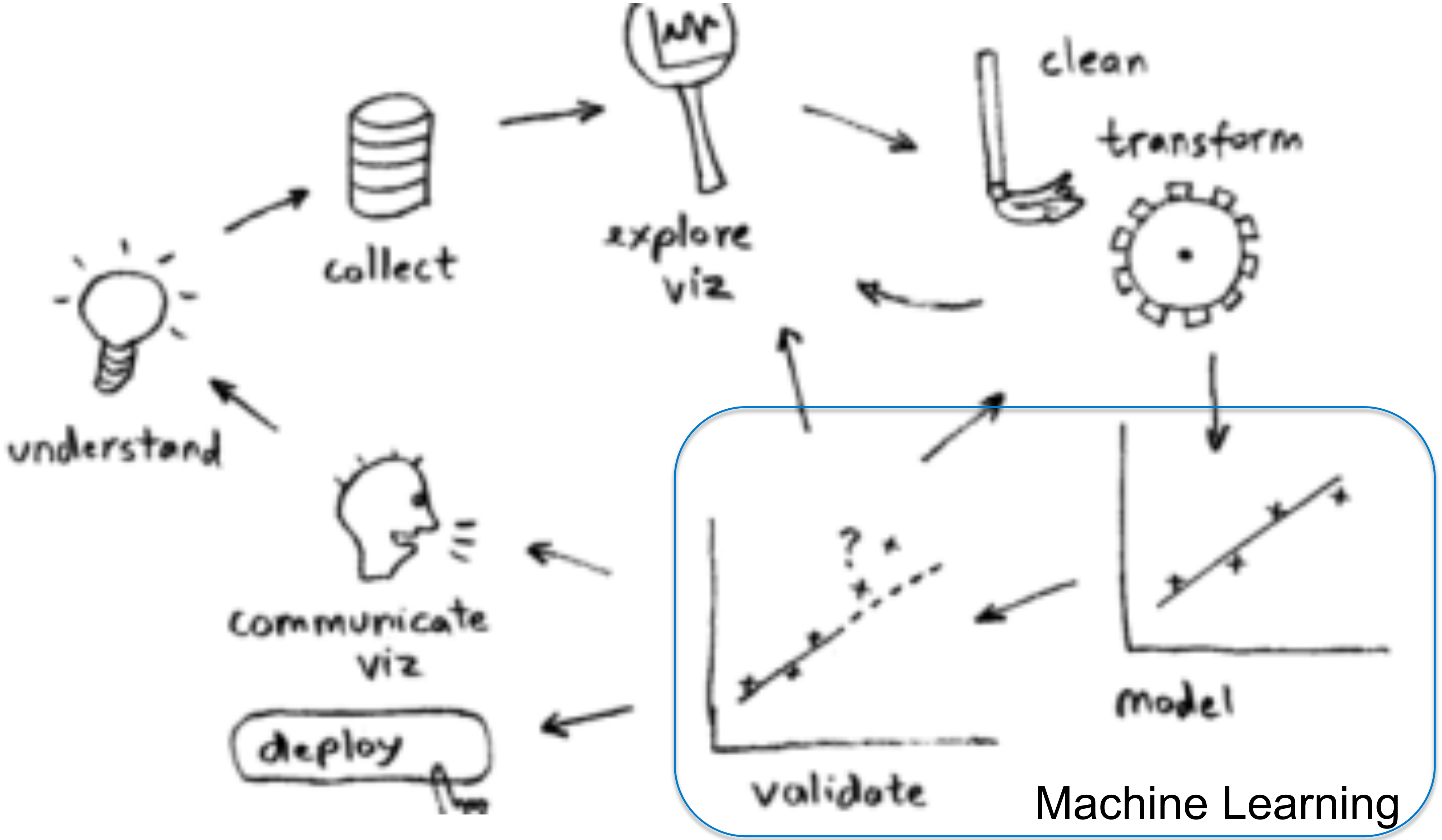
Brian Chung

Whats cool in ML? https://www.youtube.com/watch?v=qv6UVOQ0F44
http://nn.cs.utexas.edu/downloads/papers/stanley.ec02.pdf

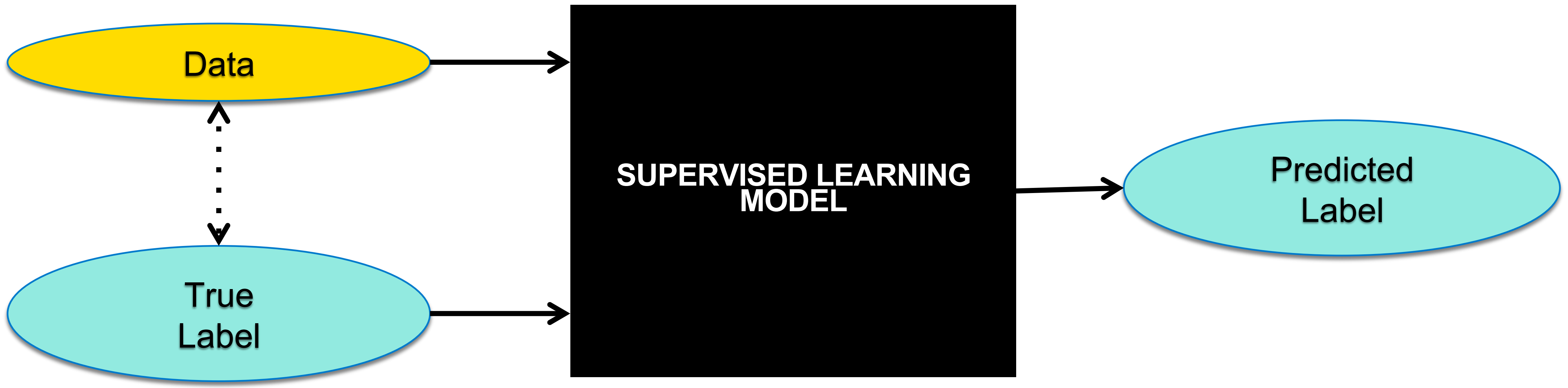
Review / Exit Tickets
- What is Data Science
- Data Science Workflow
- Machine Learning Algorithm Taxonomy

Homework overview

## DATA SCIENCE WORKFLOW

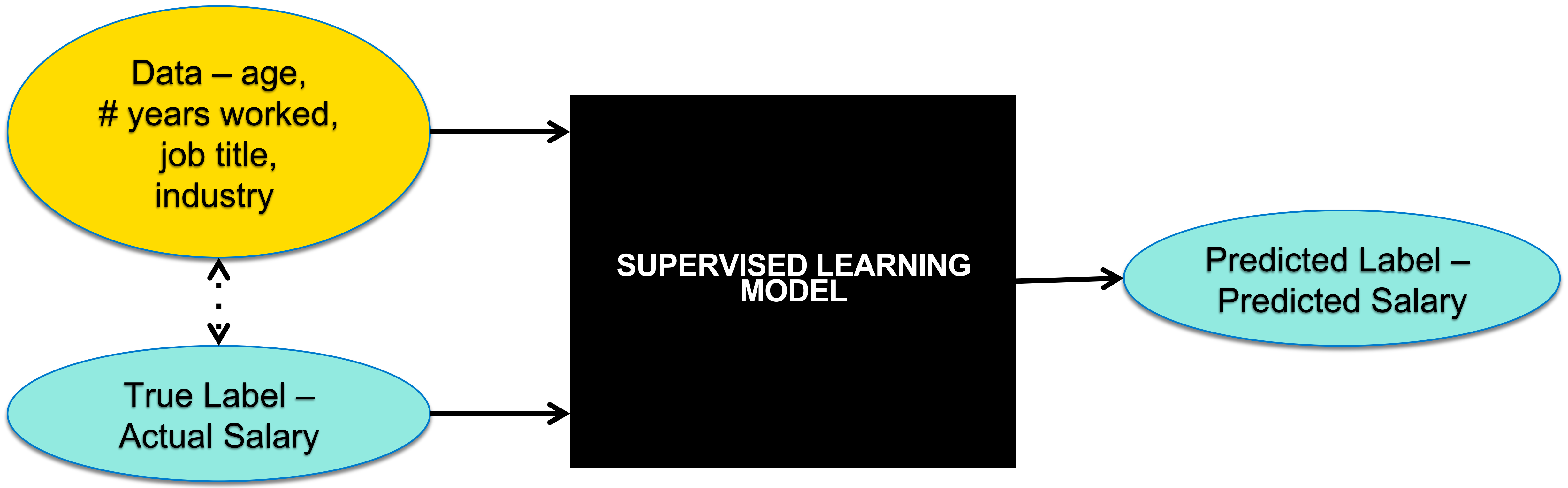


## SUPERVISED LEARNING



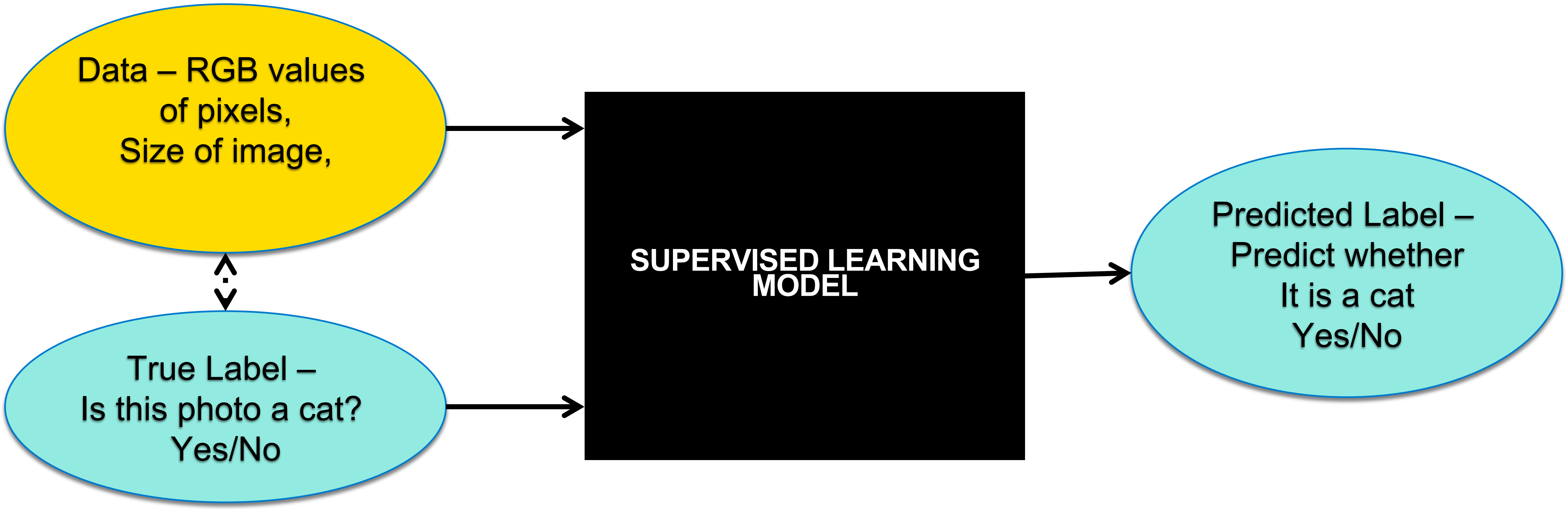
**Supervised Learning** – Data examples have associated ***labels*** that are used in building a model for **prediction or classification**

## SUPERVISED LEARNING – SALARY EXAMPLE



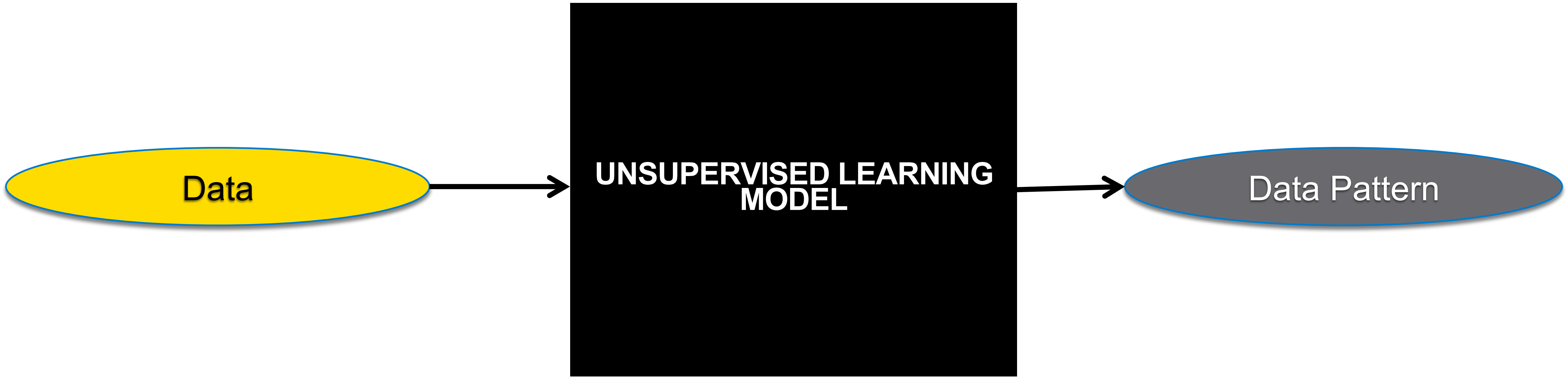
**Supervised Learning** – Data examples have associated ***labels*** that are used in building a model for **prediction or classification**

## SUPERVISED LEARNING – CAT DETECTION EXAMPLE



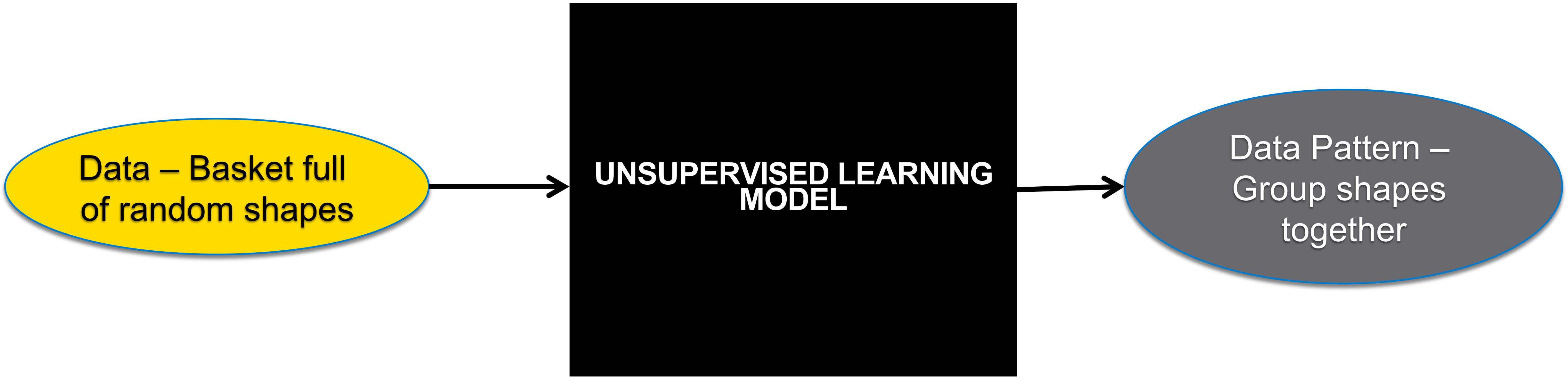
**Supervised Learning** – Data examples have associated ***labels*** that are used in building a model for **prediction or classification**

## UNSUPERVISED LEARNING



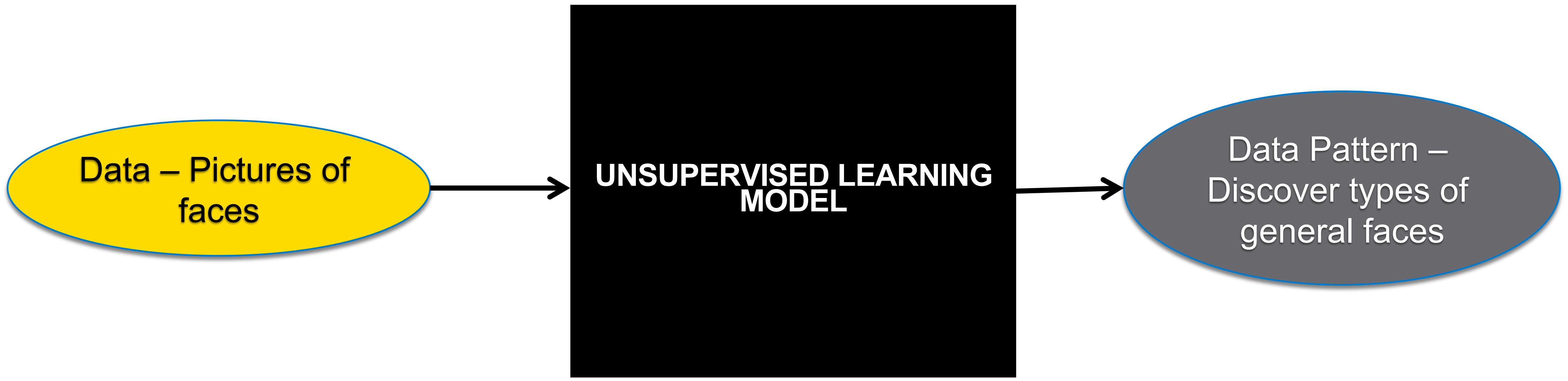
**Supervised Learning** – Data examples do NOT have associated labels that are input into the model. The model does not predict labels, but discovers ***patterns***

## UNSUPERVISED LEARNING


Data – Basket full of random shapes
UNSUPERVISED LEARNING MODEL
Data Pattern – Group shapes together

**Supervised Learning** – Data examples do NOT have associated labels that are input into the model. The model does not predict labels, but discovers ***patterns***

## UNSUPERVISED LEARNING


Data – Pictures of faces
UNSUPERVISED LEARNING MODEL
Data Pattern – Discover types of general faces

**Supervised Learning** – Data examples do NOT have associated labels that are input into the model. The model does not predict labels, but discovers ***patterns***

Homework overview

- Homeworks are graded as
    - 0 (did not complete/not satisfactory)
    - 1 (semi satisfactory, or late)
    - 2 (well done, few errors)
- You must complete 80% of homeworks to pass

## AGENDA

Goals for the session:

- Python refresher (I do, you do)
- Statistics and Linear Algebra with Numpy (we do)
- Pandas tutorial

## INTRO TO PYTHON

Q: What is Python?

A: An **open source**, **high level**, **dynamic scripting** language

## INTRO TO PYTHON

Q: What is Python?

A: An **open source**, **high level**, **dynamic scripting** language

**Open Source:** It’s free! (Looking at you MSVC++…)

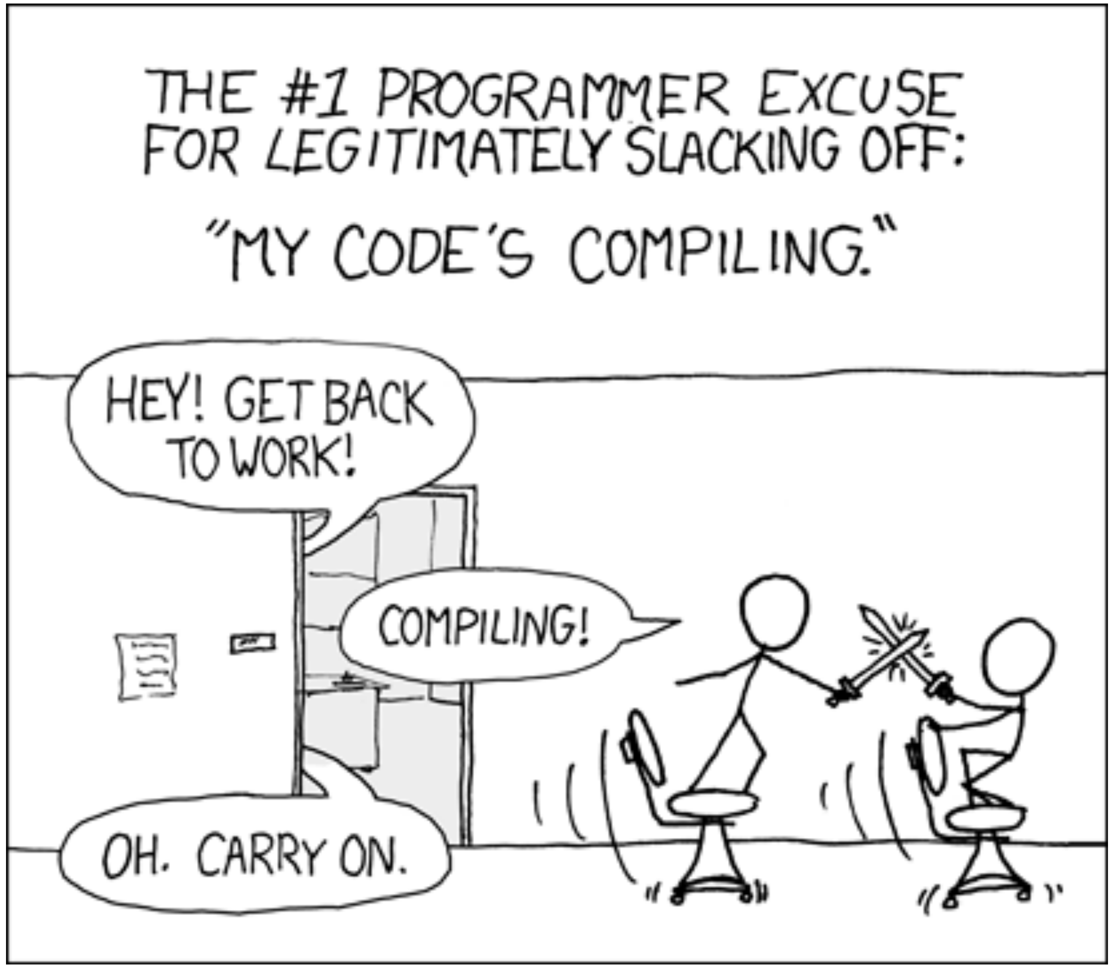
## INTRO TO PYTHON

Q: What is Python?

A: An **open source**, **high level**, **dynamic scripting** language

**Open Source:** It's free! (Looking at you MSVC++…)

**High Level**: The code is interpreted at runtime, no need to compile like C++



## INTRO TO PYTHON

Q: What is Python?

A: An **open source**, **high level**, **dynamic scripting** language

**Open Source:** It’s free! (Looking at you MSVC++…)

**High Level**: The code is interpreted at runtime, no need to compile like C++

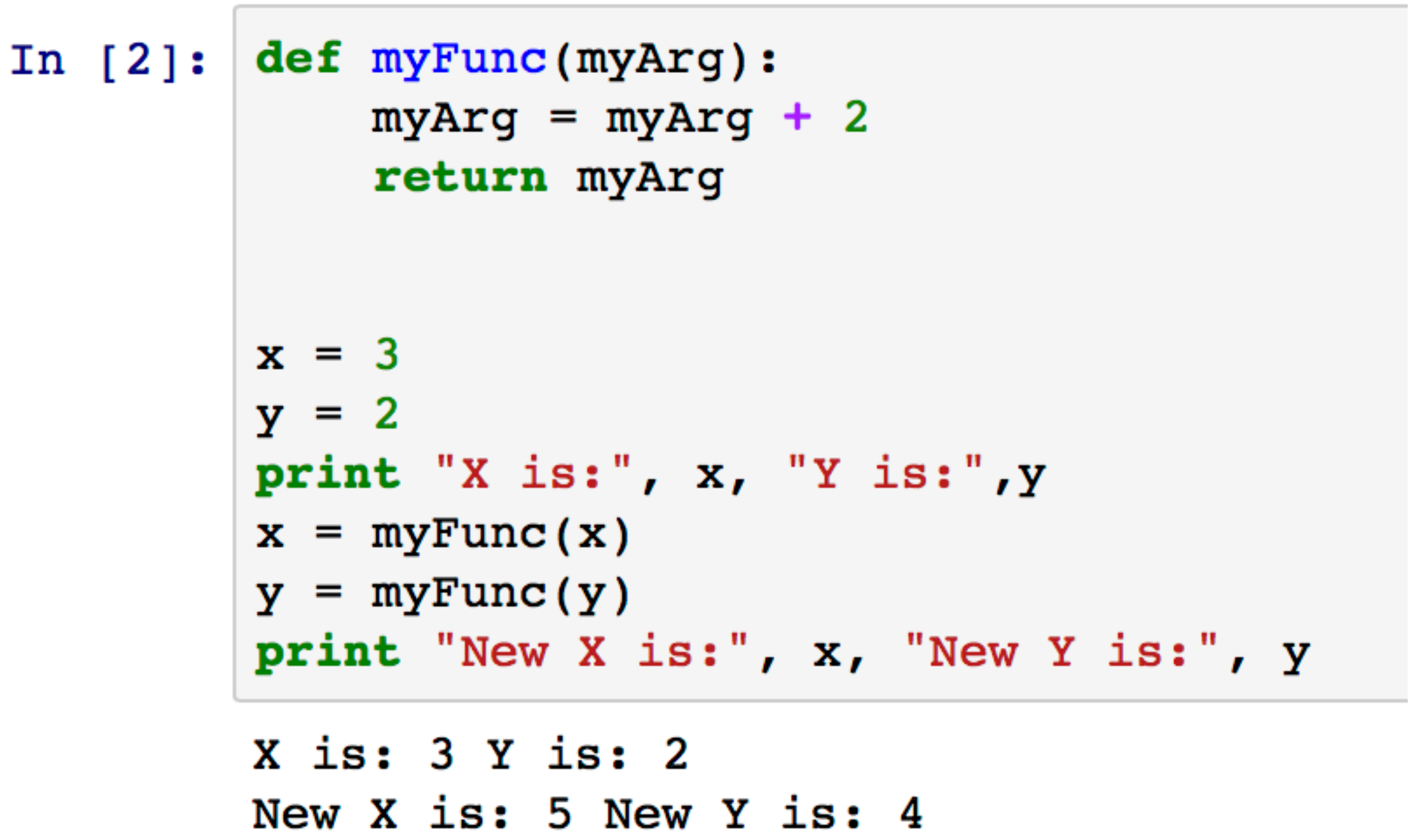
**Dynamic**: Free flow, can add/modify program, change types of variables at runtime

## PYTHON PROGRAMMING PARADIGMS

Python supports multiple programming paradigms:

- Imperative programming ala C
- Object Oriented Programming ala Java
- Functional Programming ala Haskell to a certain extent

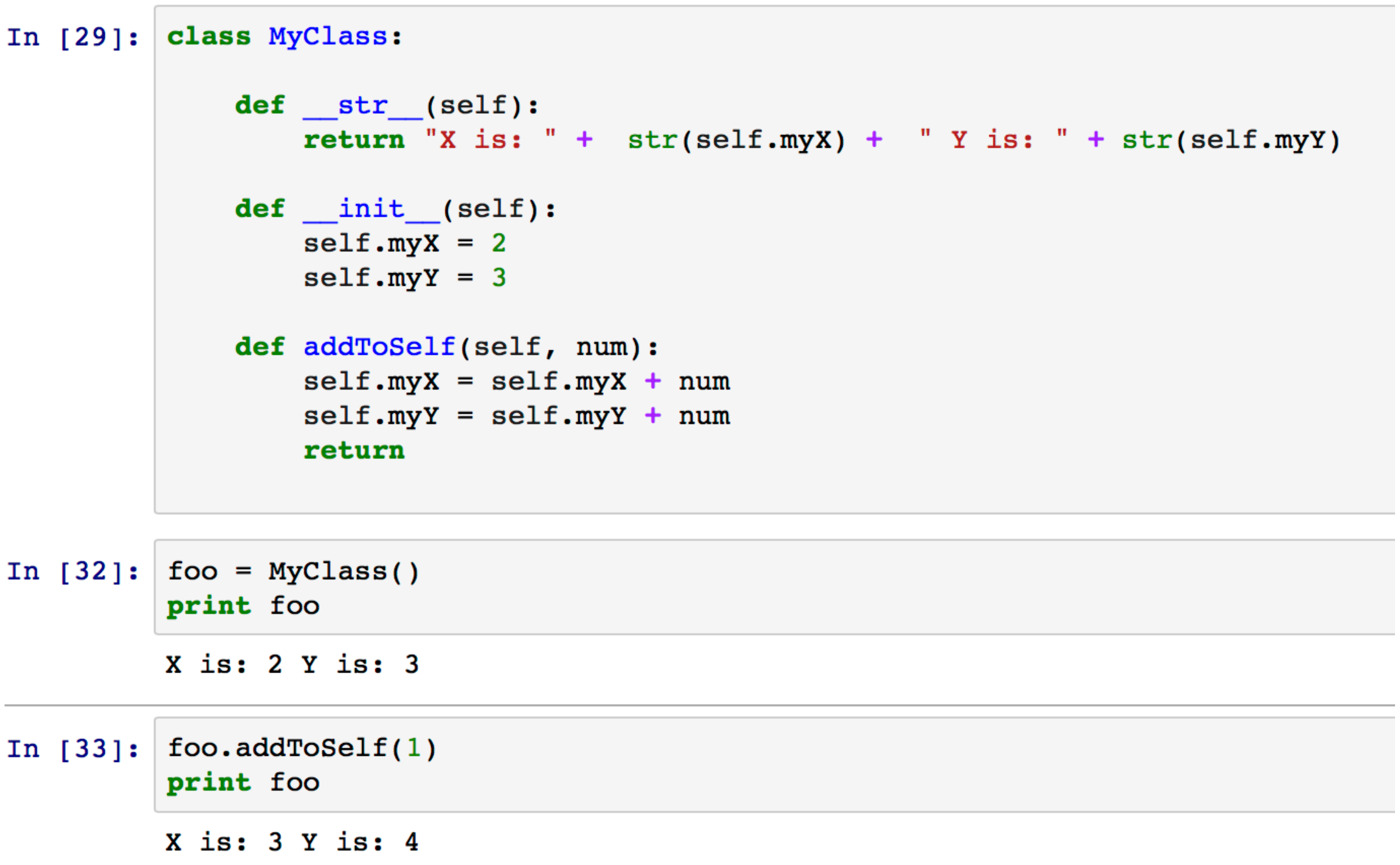
## IMPERATIVE PROGRAMMING IN PYTHON



```
In [2]: def myFunc(myArg):
            myArg = myArg + 2
            return myArg


        x = 3
        y = 2
        print "X is:", x, "Y is:",y
        x = myFunc(x)
        y = myFunc(y)
        print "New X is:", x, "New Y is:", y
```

```
X is: 3 Y is: 2
New X is: 5 New Y is: 4
```

## OBJECT ORIENTED PROGRAMMING IN PYTHON



In [29]:
```
class MyClass:

    def __str__(self):
        return "X is: " +  str(self.myX) +  " Y is: " + str(self.myY)

    def __init__(self):
        self.myX = 2
        self.myY = 3

    def addToSelf(self, num):
        self.myX = self.myX + num
        self.myY = self.myY + num
        return
```

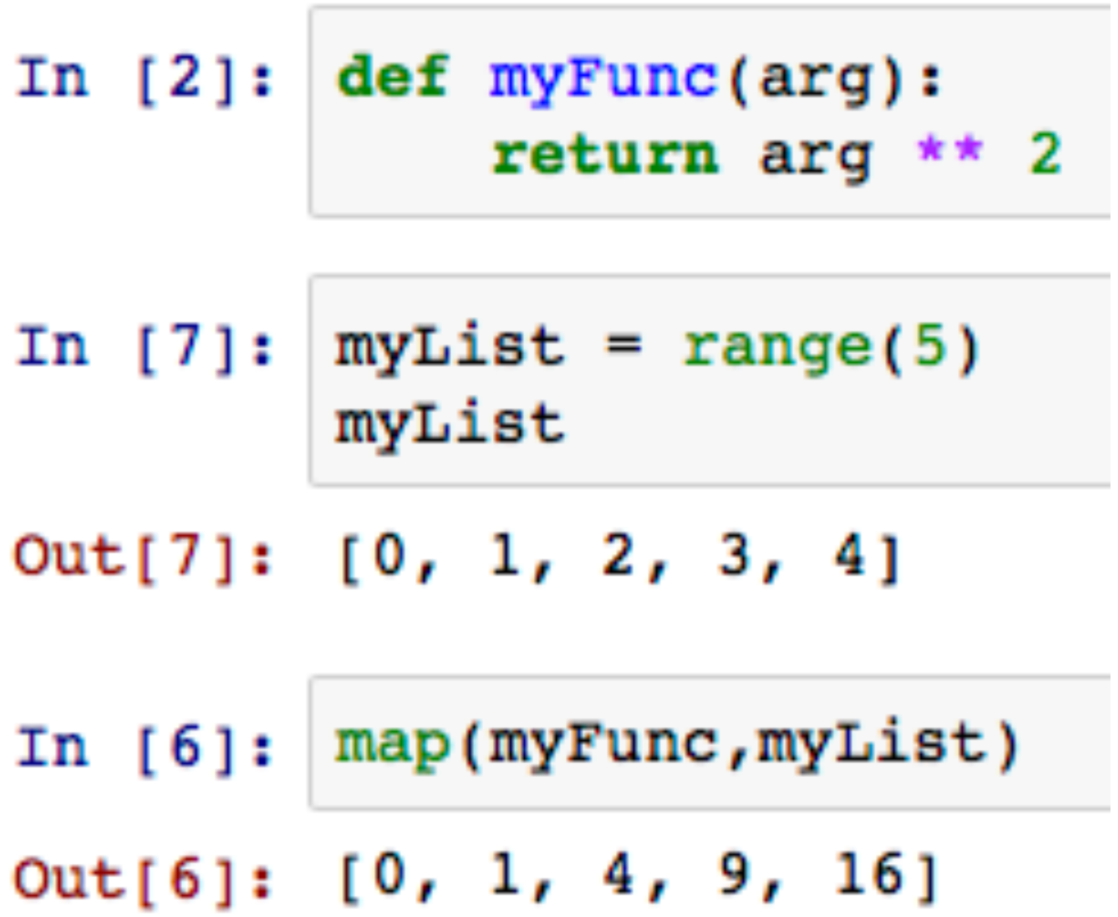
In [32]:
```
foo = MyClass()
print foo
```

```
X is: 2 Y is: 3
```

In [33]:
```
foo.addToSelf(1)
print foo
```

```
X is: 3 Y is: 4
```

## FUNCTIONAL PROGRAMMING IN PYTHON



```
In [2]: def myFunc(arg):
            return arg ** 2

In [7]: myList = range(5)
        myList

Out[7]: [0, 1, 2, 3, 4]

In [6]: map(myFunc,myList)

Out[6]: [0, 1, 4, 9, 16]
```

## PYTHON STRENGTHS

- Very easy to read (like pseudocode)
- Typically there is a ‘Pythonic’ way to do things (clear and compact)

**C++**

```
template <typename T>
T myFunc(T x)
{
  x = x + 2;
  return x;
}

vector<int> myVector = {0, 1, 2, 3, 4 };
for(unsigned x = 0; x < myVector.size(); ++x) {
  myVector.at(x) = myFunc(myVector.at(x));
}
```

**Python (list comprehension)**

```
[x + 2 for x in range(5)]
```

## PYTHON STRENGTHS

- Very easy to read (like pseudocode)
- Typically there is a ‘Pythonic’ way to do things (clear and compact)
- Lots of included functions and tools within the Python Standard Library (zip, enumerate, lambda, map, etc.). ***“Python comes with batteries installed”***

## PYTHON STRENGTHS

- Very easy to read (like pseudocode)
- Typically there is a ‘Pythonic’ way to do things (clear and compact)
- Lots of included functions and tools within the Python Standard Library (zip, enumerate, lambda, map, etc.). ***“Python comes with batteries installed”***
- Huge community!

django

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

NumPy

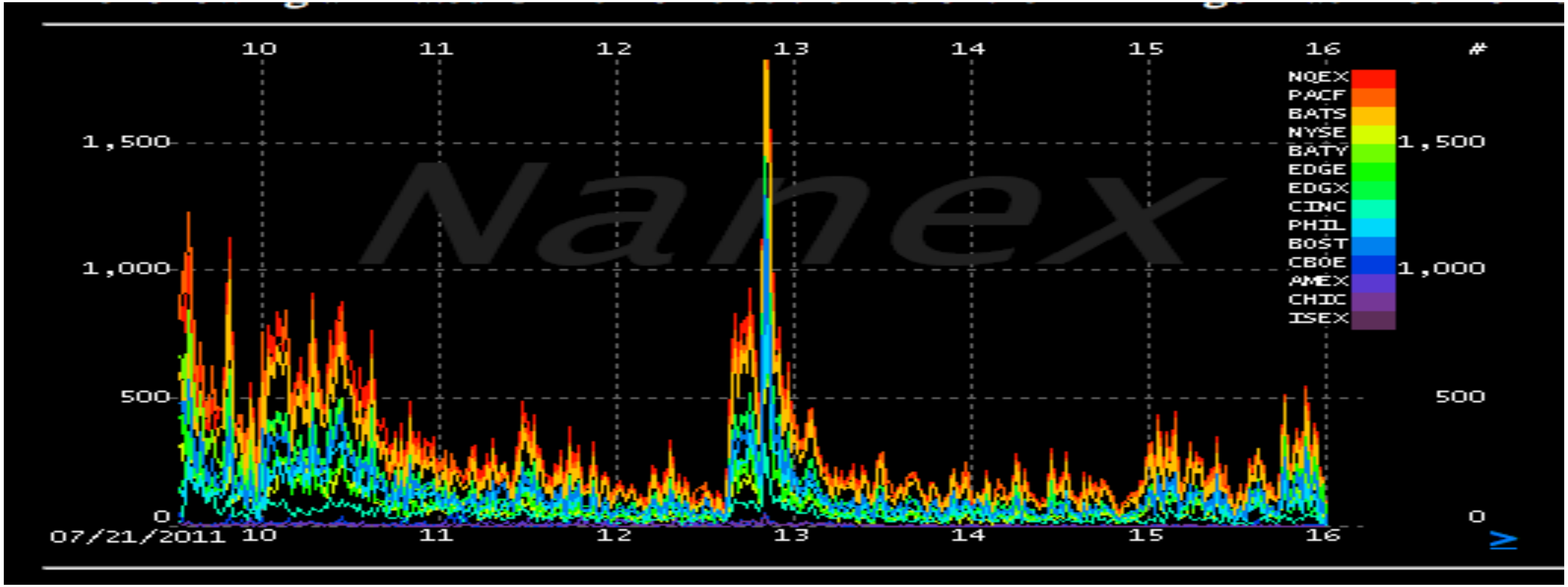
scikits learn machine learning in Python

## PYTHON STRENGTHS

- Very easy to read (like pseudocode)
- Typically there is a ‘Pythonic’ way to do things (clear and compact)
- Lots of included functions and tools within the Python Standard Library (zip, enumerate, lambda, map, etc.). ***“Python comes with batteries installed”***
- Huge community!
- Unified design philosophy (thanks to Guido Van Rossum)

## PYTHON WEAKNESSES

- Slow…relatively speaking



## PYTHON WEAKNESSES

- Slow…relatively speaking
- Minor syntactic and language gotchas (i.e. Pass by reference, default args, tabs,etc.)

http://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

## PYTHON WEAKNESSES

- Slow…relatively speaking
- Minor syntactic and language gotchas (i.e. Pass by reference, default args, tabs,etc.)
- Not exactly parallel! Threading doesn’t work the way you would think it would (i.e. Two Python threads != two OS threads)

- https://wiki.python.org/moin/GlobalInterpreterLock

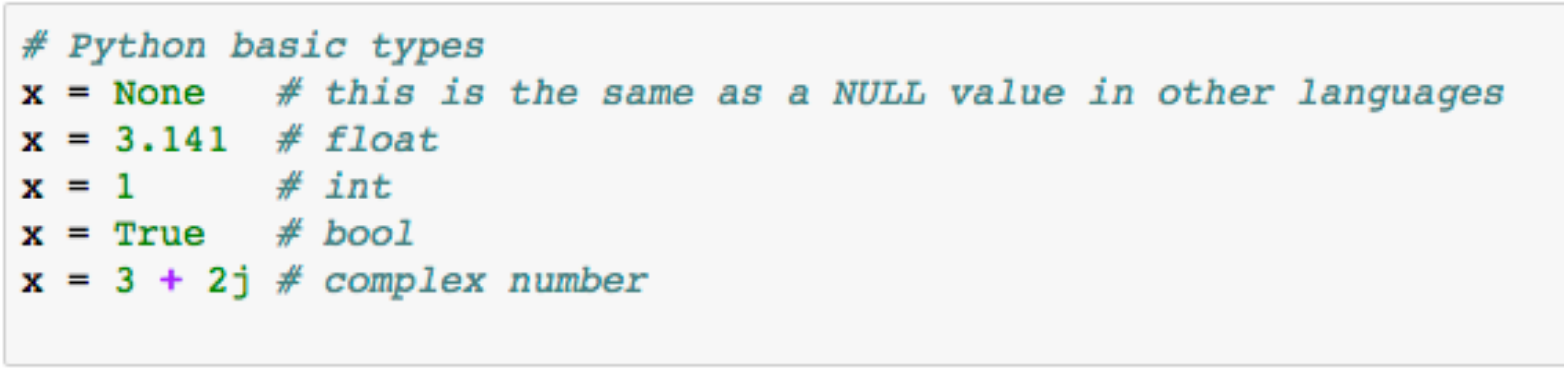
## PYTHON WEAKNESSES

- Slow…relatively speaking
- Minor syntactic and language gotchas (i.e. Pass by reference, default args, tabs,etc.)
- Not exactly parallel! Threading doesn’t work the way you would think it would (i.e. Two Python threads != two OS threads)
- Difficult to work with for data science...until numpy came around ;)

## BASIC PYTHON DATA TYPES

The most basic data structure is the **None** type. This is the equivalent of *NULL* in other languages. (None == None will evaluate to true)

There are four basic numerical types: **int, float, bool, complex**

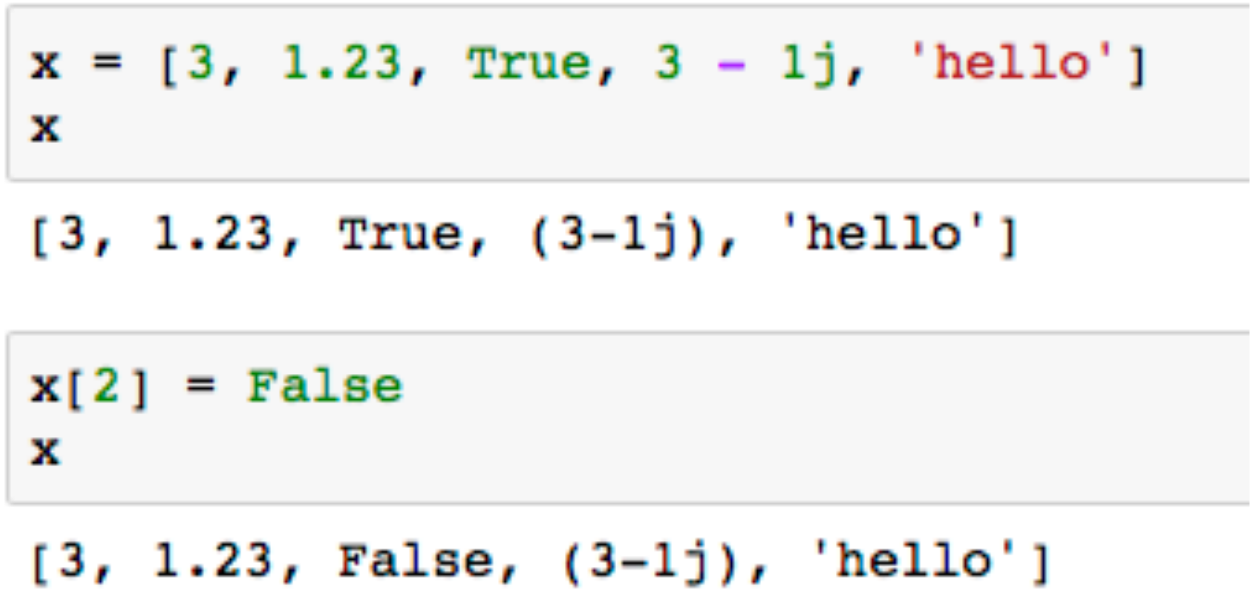


```
# Python basic types
x = None    # this is the same as a NULL value in other languages
x = 3.141   # float
x = 1       # int
x = True    # bool
x = 3 + 2j  # complex number
```

## BASIC PYTHON DATA TYPES: LISTS

The next basic data type is the **list** (similar to a vector or dynamic array)

A list is an ordered collection of **mutable** elements and **arbitrary** data types, denoted with the **[ ]** operator.



```
x = [3, 1.23, True, 3 - 1j, 'hello']
x
```

```
[3, 1.23, True, (3-1j), 'hello']
```

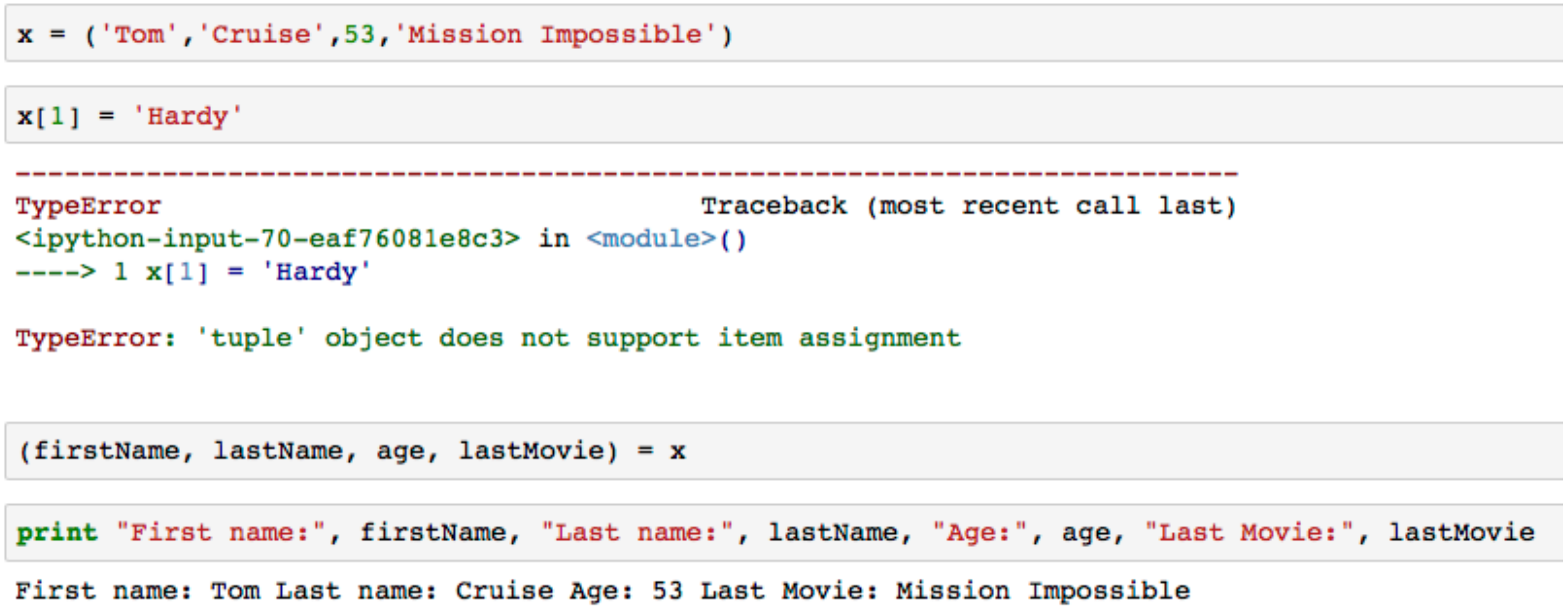
```
x[2] = False
x
```

```
[3, 1.23, False, (3-1j), 'hello']
```

## BASIC PYTHON DATA TYPES: TUPLES

Tuples are similar to lists, but they are **immutable**

Tuples are frequently used behind the scenes and by programmers for assigning and operations called *tuple packing/unpacking*



```
x = ('Tom','Cruise',53,'Mission Impossible')
```

```
x[1] = 'Hardy'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-70-eaf76081e8c3> in <module>()
----> 1 x[1] = 'Hardy'

TypeError: 'tuple' object does not support item assignment
```

```
(firstName, lastName, age, lastMovie) = x
```

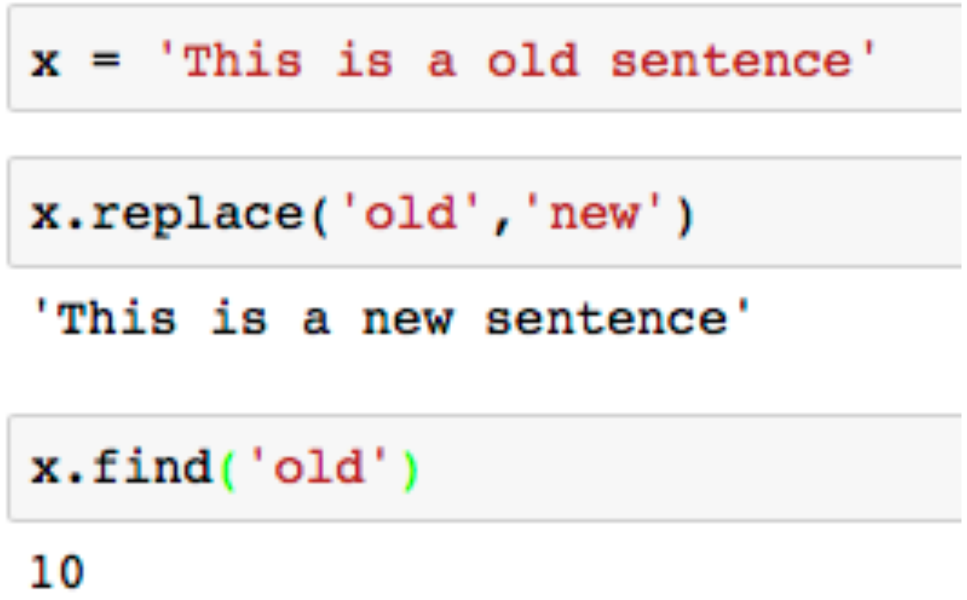
```
print "First name:", firstName, "Last name:", lastName, "Age:", age, "Last Movie:", lastMovie
```

```
First name: Tom Last name: Cruise Age: 53 Last Movie: Mission Impossible
```

## BASIC PYTHON DATA TYPES: STRINGS

Strings are **immutable arrays** of characters (note that there is not char type)

Strings support slicing and indexing operations and have many string specific operations as well



```
x = 'This is a old sentence'
```

```
x.replace('old','new')
```

```
'This is a new sentence'
```

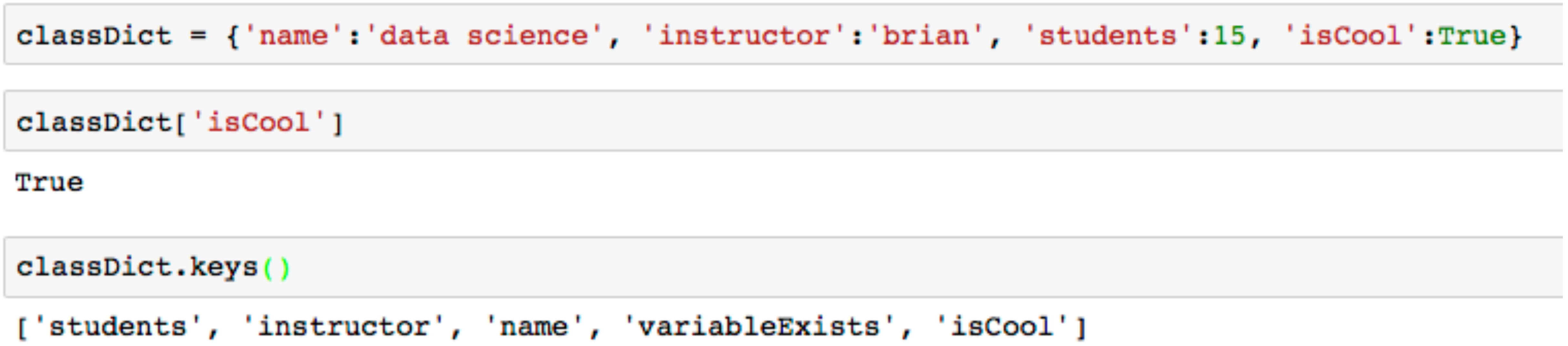
```
x.find('old')
```

```
10
```

## BASIC PYTHON DATA TYPES: DICTIONARIES

Hash tables are represented in Python as the **dict** (dictionary) type through key, value pairs

The key-value pairs, and dictionary keys must be immutable



```
classDict = {'name':'data science', 'instructor':'brian', 'students':15, 'isCool':True}
```

```
classDict['isCool']
```

```
True
```

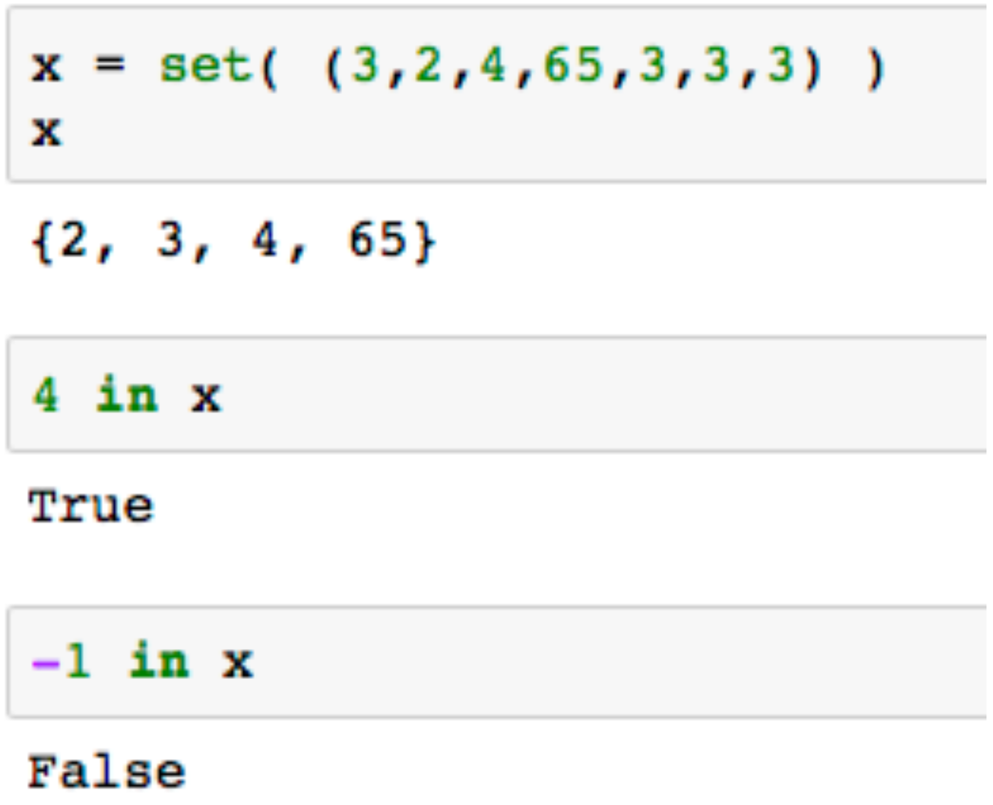
```
classDict.keys()
```

```
['students', 'instructor', 'name', 'variableExists', 'isCool']
```

## BASIC PYTHON DATA TYPES: SETS

Unique collections of **unordered** values are represented as **sets** in Python.

Sets are useful for checking for membership and ensuring uniqueness



```
x = set( (3,2,4,65,3,3,3) )
x
```

```
{2, 3, 4, 65}
```

```
4 in x
```

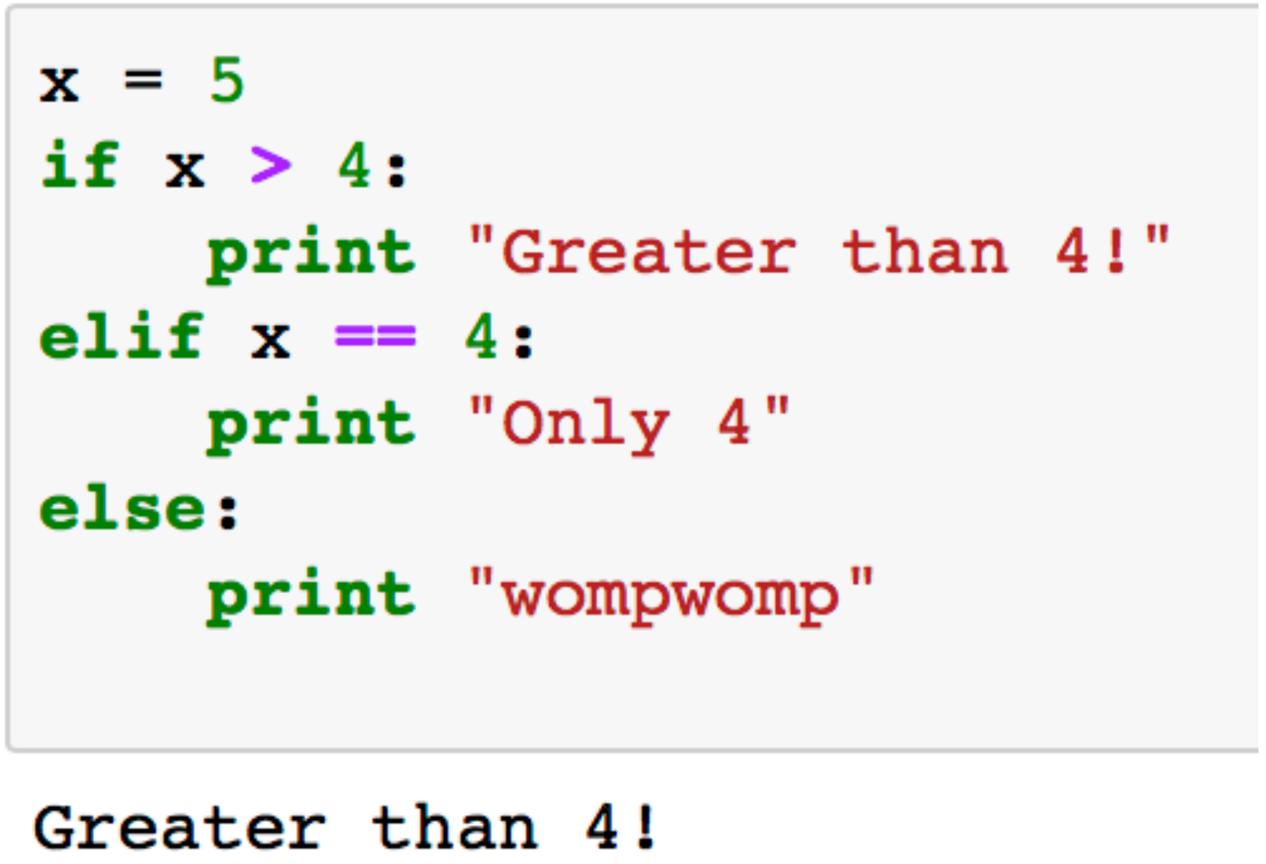
```
True
```

```
-1 in x
```

```
False
```

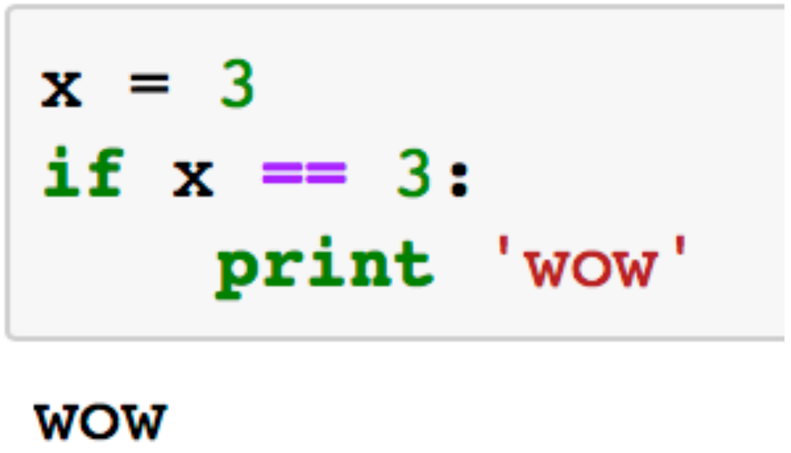
## LOOPS: IF/ELIF/ELSE

If/elif/else loops let us branch our code depending on the conditions



```
x = 5
if x > 4:
    print "Greater than 4!"
elif x == 4:
    print "Only 4"
else:
    print "wompwomp"
```

```
Greater than 4!
```

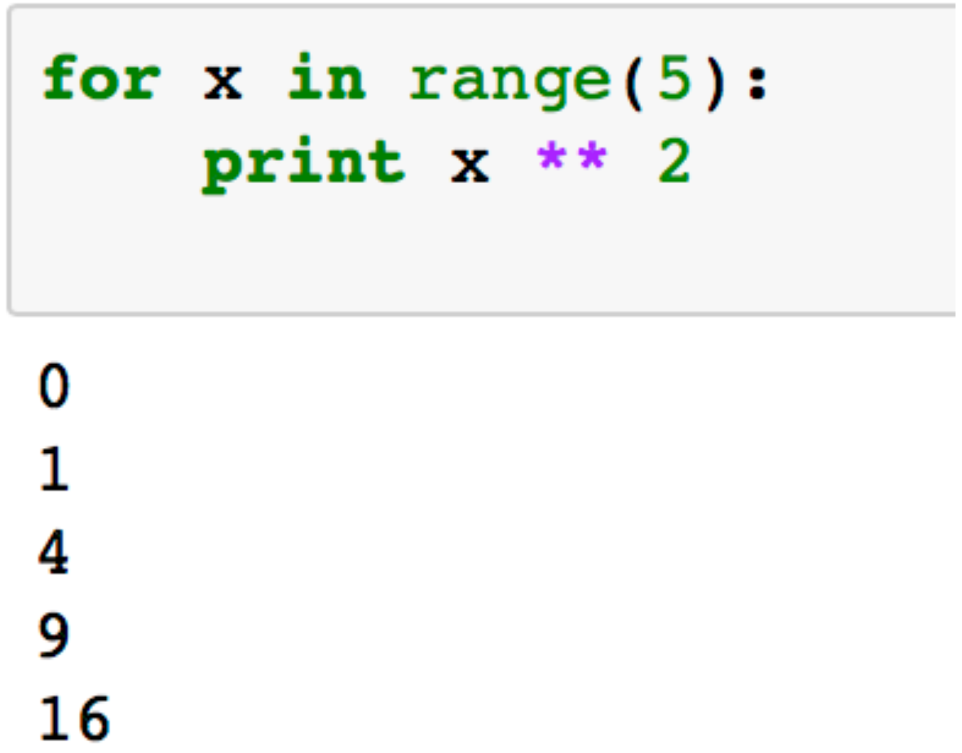


```
x = 3
if x == 3:
    print 'wow'
```

```
wow
```

## LOOPS: FOR

For loops repeat blocks of code



```
for x in range(5):
    print x ** 2
```

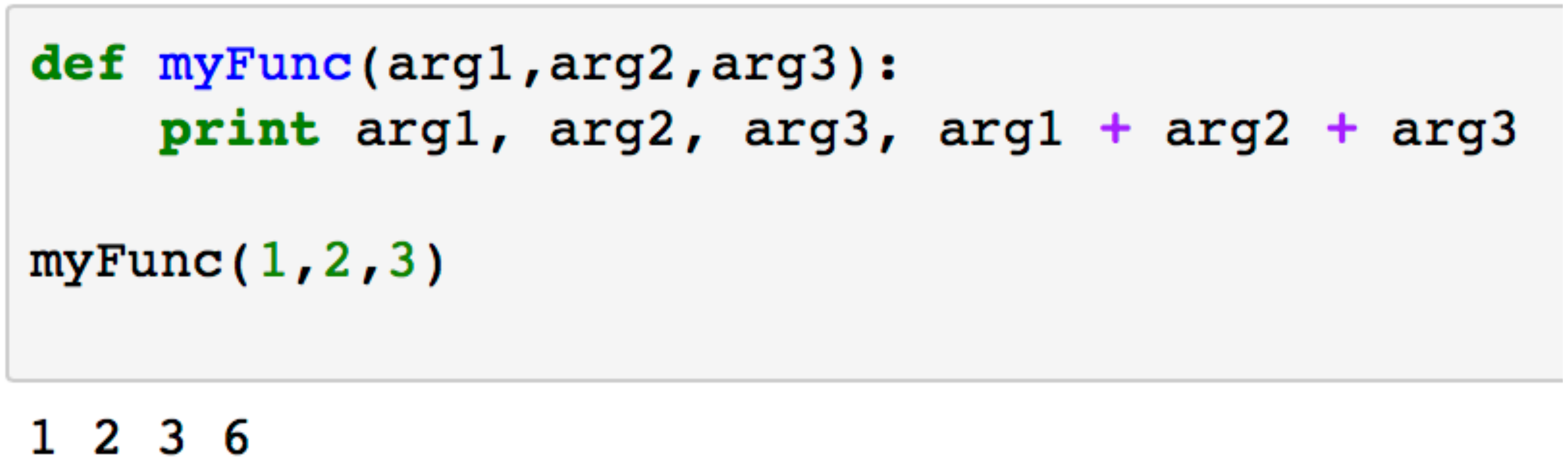
```
0
1
4
9
16
```

## FUNCTIONS

**Functions** can be defined anywhere (but must be defined before using it)

They begin with the **def** keyword and take in a number of (or no) arguments

Remember indentation!!



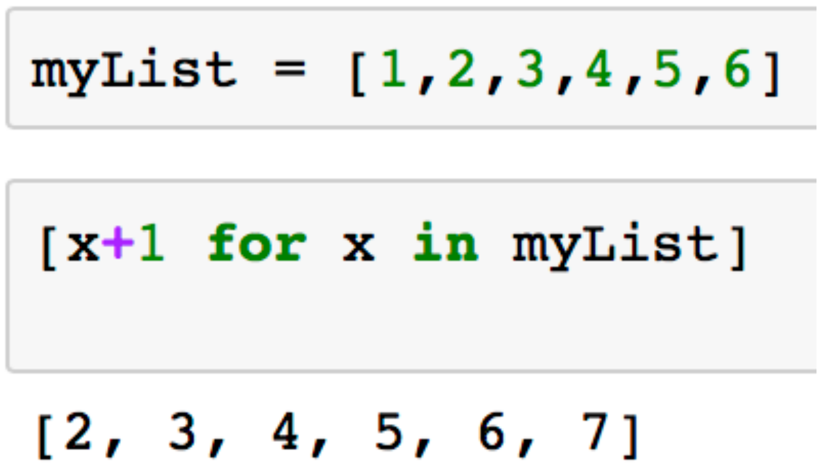
```
def myFunc(arg1,arg2,arg3):
    print arg1, arg2, arg3, arg1 + arg2 + arg3

myFunc(1,2,3)
```

```
1 2 3 6
```

## LIST COMPREHENSION

We can use a powerful construct called a **list comprehension** to create lists

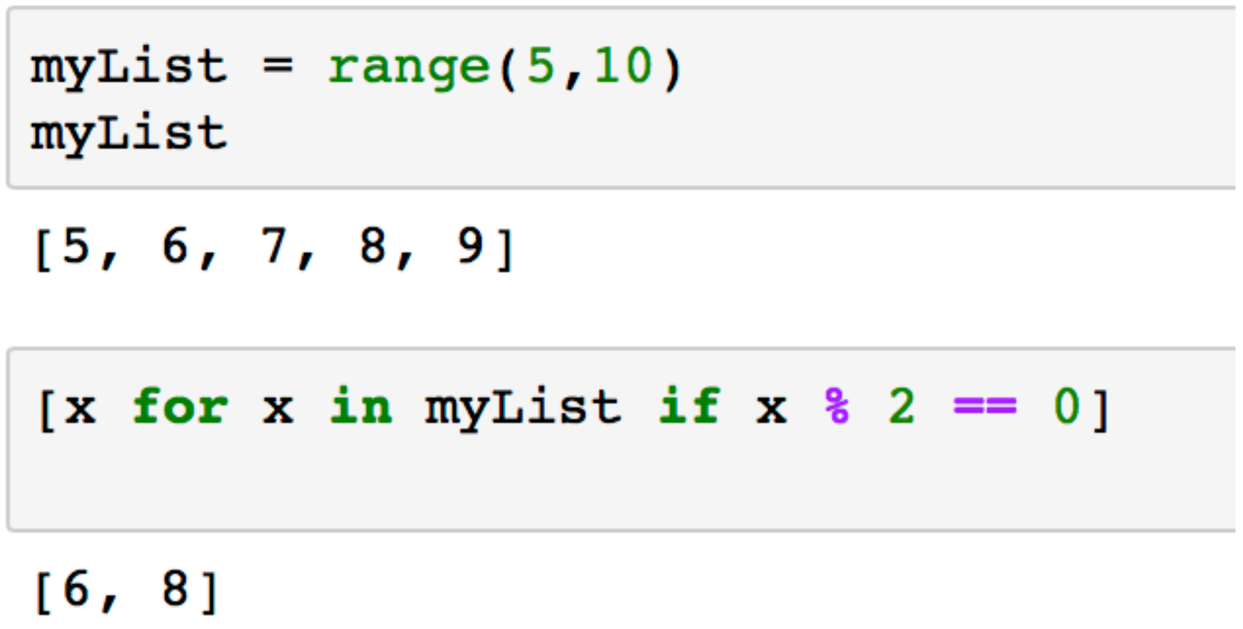
**[ ReturnValue Loop (Conditional) ]**



```
myList = [1,2,3,4,5,6]
```

```
[x+1 for x in myList]
```

```
[2, 3, 4, 5, 6, 7]
```



```
myList = range(5,10)
myList
```

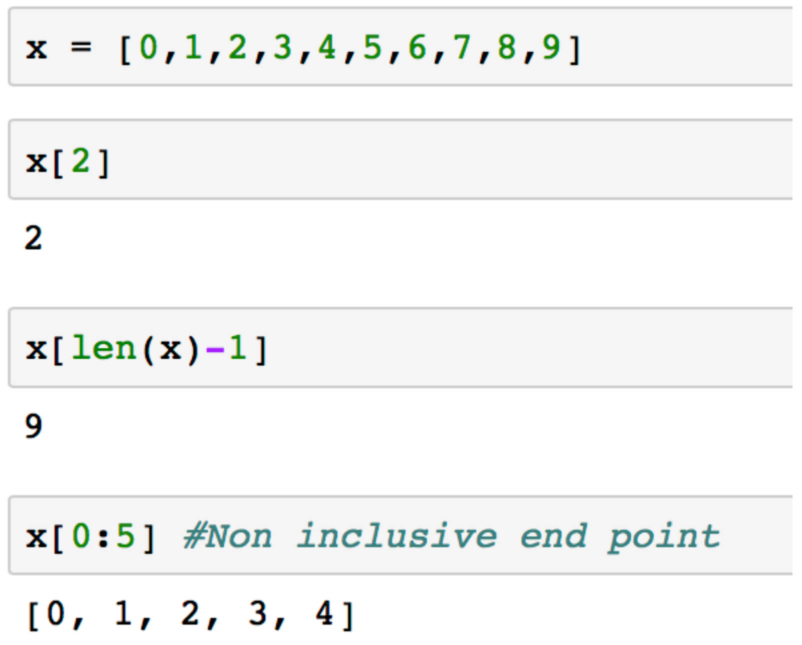
```
[5, 6, 7, 8, 9]
```

```
[x for x in myList if x % 2 == 0]
```

```
[6, 8]
```

## LIST SLICING

- Indexing into arrays works as you would expect.
- There is also a concept of ‘negative indexing’. i.e. Indexing from the rear of hte list
- However, you can also *slice* into portions of a list through the ‘:’ operator



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

```
x[2]
```

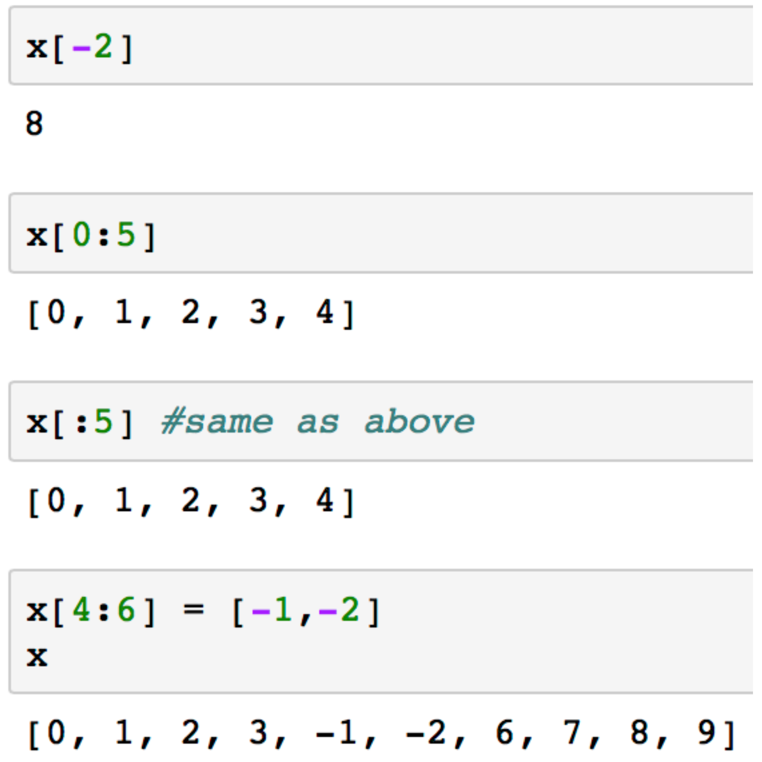
```
2
```

```
x[len(x)-1]
```

```
9
```

```
x[0:5] #Non inclusive end point
```

```
[0, 1, 2, 3, 4]
```



```
x[-2]
```

```
8
```

```
x[0:5]
```

```
[0, 1, 2, 3, 4]
```

```
x[:5] #same as above
```

```
[0, 1, 2, 3, 4]
```

```
x[4:6] = [-1,-2]
x
```

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

## QUICK INTRODUCTION TO IPYTHON NOTEBOOK

The sample ipynb is found at the github in 02_pandas folder, titled **lab_02_01_ipython.ipynb**

You can obtain the files through cloning and pulling from the Github repo OR

Find the link to the Raw file. Using curl, download this file. i.e. In OSX or linux:

```
curl https://github.com/brianchandbound/ga-ds/blob/master/02_pandas/
02_pandas.pdf
```

## EXERCISE!

Using **IPython notebook**, answer the following questions. Feel free to work in pairs:

- What is the sum of the sequence 1, 4, 9, 16, 25, … up to the 40th element?

  **sum( [ x**2 for x in range(1,41) ] )**

(Hint: You can solve this with loops or with list comprehensions. Also, **range** can take in more than one argument)

- Write out all the positive odd numbers up to 35 (hint: use the “%” operator)

  **[ x for x in range(1,36) if x % 2 != 0 ]**

- Create a list with the following sequence ‘aa’, ‘bb’, ‘cc’, ... ‘zz’.

(Hint: Use **ord**, **chr**, and **range** to help!  ord(‘a’) -> 97    chr(97) -> ‘a’)

  **[ chr(x)*2 for x in range(ord(‘a’),ord(‘z’)+1)]**

- Write a function that takes in two arguments, and checks if **both** their squares are even

  **def func(arg1, arg2): return (arg1**2 % 2 ==0) and (arg2**2  %2 == 0)**

GENERAL ASSEMBLY

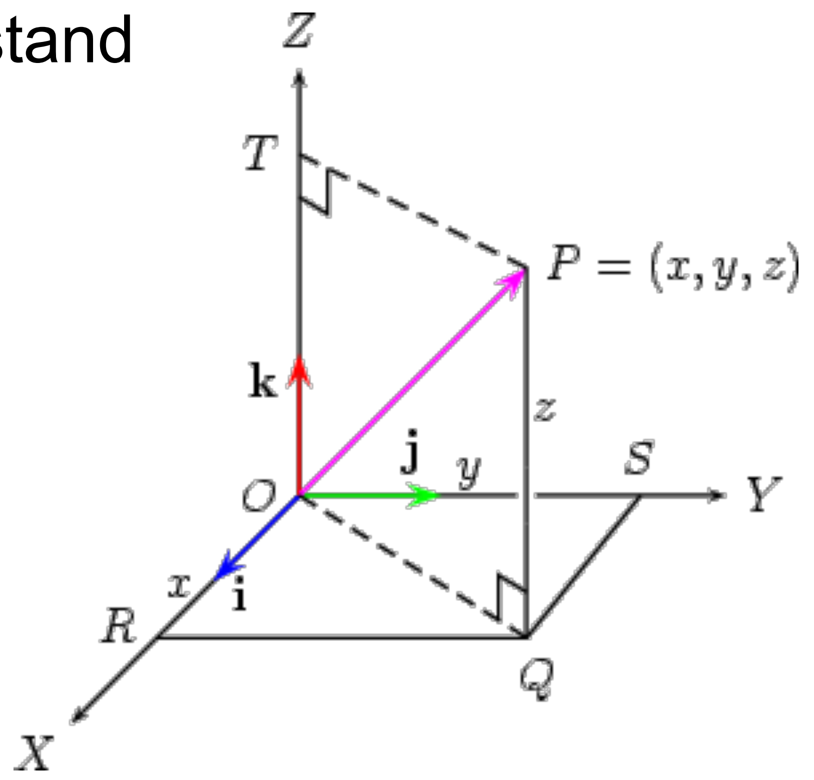
## II. LINEAR ALGEBRA WITH NUMPY

## WHY LINEAR ALGEBRA (I.E. MATRIX ALGEBRA)?

Most **Machine Learning** algorithms utilize linear algebra (matrix algebra) to represent data as well as solve for solutions

In addition, learning matrix concepts will help you understand the physical transformations in your data

Linear algebra can be defined as mathematics in **multidimensional space** and the mapping between said spaces



## WHAT IS THE MATRIX…

Matrices are an array of real numbers with **m** rows, and **n** columns.

Matrices are referred to by their size (rows x columns). This example would be a “3 by 4 matrix”

$$\begin{array}{rc} & \text{4 columns} \\ & \downarrow \\ \text{3 rows} \longrightarrow & \begin{bmatrix} 3.1 & 9 & 7 & 5 \\ 5.6 & 9 & 8.3 & -5 \\ 6 & 4 & \Pi & 0 \end{bmatrix} \end{array}$$

## WHAT IS THE MATRIX…

**Matrices** are an array of real numbers with **m** rows, and **n** columns.

Each value in a matrix is called an **entry**, and is referred to by its position in the matrix (starting with the row number and followed by the column number)

$$a_{21} \rightarrow \begin{bmatrix} 3.1 & 9 & 7 & 5 \\ 5.6 & 9 & 8.3 & -5 \\ 6 & 4 & \Pi & 0 \end{bmatrix} \leftarrow a_{23}$$

## VECTORS

**Vectors** are special kinds of matrices and only contain one dimension of real numbers. They can be referred to as matrices of size “p x 1” or “size p vectors”

3 x 1 vector
(column vector)

$$\downarrow$$

$$\begin{bmatrix} 3.4 \\ -5 \\ .23 \end{bmatrix}$$

1 x 4 vector
(row vector)

$$\downarrow$$

$$\begin{bmatrix} 1.2 & 0 & -1 & .6 \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX ADDITION

- In order to add two matrices, the size of the matrices must be exactly the same.
- Matrices are added and subtracted element-wise.
- The resulting matrix will be the same size

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 2 \\ 4 & 5 & 6 \\ 1 & 1 & 4 \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX ADDITION

- In order to add two matrices, the size of the matrices must be exactly the same.
- Matrices are added and subtracted element-wise.
- The resulting matrix will be the same size

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 2 \\ 4 & 5 & 6 \\ 1 & 1 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 \\ 0 & 10 & 13 \\ 1 & 10 & 13 \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX MULTIPLICATION BY A SCALAR

- Matrices can be multiplied by a scalar (single value)
- The resulting matrix is the same size, and each element is multiplied element-wise by the value

$$2 * \begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX MULTIPLICATION BY A SCALAR

- Matrices can be multiplied by a scalar (single value)
- The resulting matrix is the same size, and each element is multiplied element-wise by the value

$$2*\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix} = \begin{bmatrix} 2 & 8 & 4 \\ -8 & 10 & 14 \\ 0 & 18 & 18 \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX MULTIPLICATION BY A VECTOR

- Matrices can be multiplied by a vector if the matrix columns are as wide as the vector is long
- The result will always be a vector given by the # of rows in the matrix

$$\underbrace{\begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix}}_{2 \times 4} * \underbrace{\begin{bmatrix} 2 \\ 3 \\ 6 \\ 5 \end{bmatrix}}_{4 \times 1} = \underbrace{\begin{bmatrix} ? \\ ? \end{bmatrix}}_{2 \times 1}$$

## MATRIX OPERATIONS – MATRIX MULTIPLICATION BY A VECTOR

- Matrices can be multiplied by a vector if the matrix columns are as wide as the vector is long
- The result will always be a vector given by the # of rows in the matrix

$$\begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \\ 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 1*2+3*3+9*6+2*5 \\ 2*2+4*3+6*6+8*5 \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX MULTIPLICATION BY A VECTOR

- Matrices can be multiplied by a vector if the matrix columns are as wide as the vector is long
- The result will always be a vector given by the # of rows in the matrix

$$\begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \\ 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 75 \\ 92 \end{bmatrix}$$

## MATRIX OPERATIONS – MULTIPLICATION BY A MATRIX

- Matrices can be multiplied with other matrices
- The **Inner Dimensions** must be the same
- The resulting matrix size will be determined by the **Outer Dimensions**

$$\underbrace{\begin{bmatrix}1 & 4 & 2\\ -4 & 5 & 7\\ 0 & 9 & 9\end{bmatrix}}_{3 \text{ x } 3} * \underbrace{\begin{bmatrix}1 & 4\\ 2 & 0\\ 9 & 2\end{bmatrix}}_{3 \text{ x } 2}$$

Inner Dimensions

Outer Dimensions

## MATRIX OPERATIONS – MULTIPLICATION BY A MATRIX

- Matrices can be multiplied with other matrices
- The **Inner Dimensions** must be the same
- The resulting matrix size will be determined by the **Outer Dimensions**

$$\underset{3 \times 3}{\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix}} * \underset{3 \times 2}{\begin{bmatrix} 1 & 4 \\ 2 & 0 \\ 9 & 2 \end{bmatrix}} = \underset{3 \times 2}{\begin{bmatrix} ? & ? \\ ? & ? \\ ? & ? \end{bmatrix}}$$

## MATRIX OPERATIONS – MULTIPLICATION BY A MATRIX

- Matrices can be multiplied with other matrices
- The **Inner Dimensions** must be the same
- The resulting matrix size will be determined by the **Outer Dimensions**

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 4 \\ 2 & 0 \\ 9 & 2 \end{bmatrix} = \begin{bmatrix} 1+8+18 & 4+0+4 \\ -4+10+63 & -16+0+14 \\ 0+18+81 & 0+0+18 \end{bmatrix}$$

## MATRIX OPERATIONS – MULTIPLICATION BY A MATRIX

- Matrices can be multiplied with other matrices
- The **Inner Dimensions** must be the same
- The resulting matrix size will be determined by the **Outer Dimensions**

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 4 \\ 2 & 0 \\ 9 & 2 \end{bmatrix} = \begin{bmatrix} 27 & 8 \\ 69 & -2 \\ 99 & 18 \end{bmatrix}$$

## MATRIX OPERATIONS – TRANSPOSE

- **Transposing** is an operation that flips the elements of a matrix across the diagonals. For each column vector, map it onto a row vector.
- **Transpose** operations are often denoted by a “T” or an “ ‘ “ symbol
- The result of a transpose will be a matrix with flipped size (i.e. 3x4 -> 4x3)

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix}^T = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

## MATRIX OPERATIONS – TRANSPOSE

- **Transposing** is an operation that flips the elements of a matrix across the diagonals For each column vector, map it onto a row vector.
- **Transpose** operations are often denoted by a “T” or an “ ‘ “ symbol
- The result of a transpose will be a matrix with flipped size (i.e. 3x4 -> 4x3)

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & -4 & 0 \\ 4 & 5 & 9 \\ 2 & 7 & 9 \end{bmatrix}$$

## MATRIX OPERATIONS – MATRIX INVERSE

- There is no such thing as “dividing by a matrix”
- However, there is a concept of a **matrix inverse**. Conceptually, it is similar to taking an element to the -1th power
- Matrix inversion rules are more complex, require a square matrix (#rows == #columns), and the matrix must be **full rank**

$$\begin{bmatrix} 1 & 4 & 2 \\ -4 & 5 & 7 \\ 0 & 9 & 9 \end{bmatrix}^{-1} = \begin{bmatrix} -.33 & -.33 & .33 \\ .67 & .16 & -.27 \\ -.67 & -.16 & .38 \end{bmatrix}$$

## WHIRLWIND OF MATRICES…WHY DOES ALL THIS MATTER?

- We can represent the multiple dimensions of our data through matrices!
- Once our data is in matrix form, we can perform matrix operations
- Almost all of our machine learning tools use matrices of some sort

## WHIRLWIND OF MATRICES…WHY DOES ALL THIS MATTER?

- We can represent the multiple dimensions of our data through matrices!
- Once our data is in matrix form, we can perform matrix operations
- Almost all of our machine learning tools use matrices of some sort

**Name | Age | Years Worked | Distance from Chicago -> Salary**

Bob | 21 years old | 1 year worked | .1 miles away -> $35000

Jim | 30 years old | 5 years worked | 20 miles away -> $67000

Alice | 18 years old | 0 years worked | 13 miles away -> $45000

David | 45 years old | 22 years worked | 4 miles away -> $120000

Tom | 34 years old | 16 years worked | 5 miles away -> $75000

$$\xrightarrow{} \begin{array}{cc} \begin{bmatrix} 21 & 1 & .1 \\ 30 & 5 & 20 \\ 18 & 0 & 13 \\ 45 & 22 & 4 \\ 34 & 16 & 5 \end{bmatrix} & \begin{bmatrix} 35000 \\ 67000 \\ 45000 \\ 120000 \\ 75000 \end{bmatrix} \\ \text{Data} & \text{Labels} \end{array}$$

## MATRIX OPERATIONS WITH NUMPY

Let’s see how to perform all of these operations using…

**Numpy** – A scientific computing library built for python, providing high performance multidimensional array objects and tools for working with these arrays

Further tutorials: http://cs231n.github.io/python-numpy-tutorial/

## PANDAS OVERVIEW

Let’s see some more interesting tools using **pandas**

**Pandas**– A scientific computing library built for python on top of numpy, providing high performance data structures

10 minute tour of pandas: https://vimeo.com/59324550

GENERAL ASSEMBLY

Exit Tickets! DAT-1, Lesson 2, Intro To Python
--Do share whether you liked ‘watch, then code (Python portion)’ or ‘code togeth
er (Numpy portion)’ better

Homework 2 Due Dec 16 before class