

INTRO TO DATA SCIENCE

PYTHON AND LINEAR ALGEBRA REVIEW

RECAP

LAST TIME:

I. DATA SCIENCE

II. DATA SCIENTIST

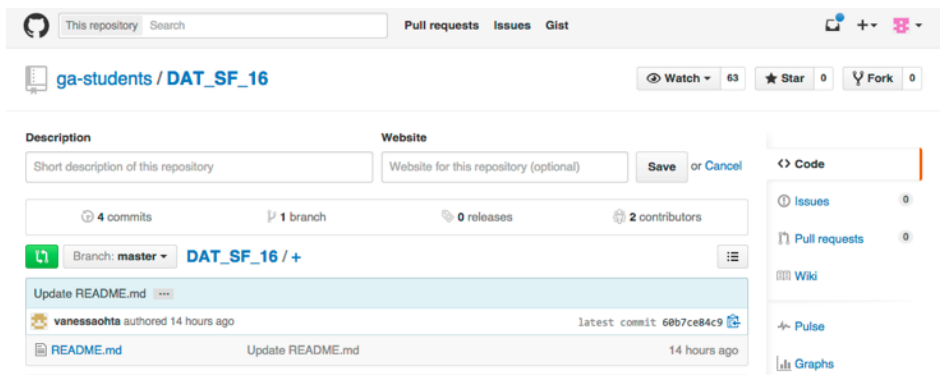
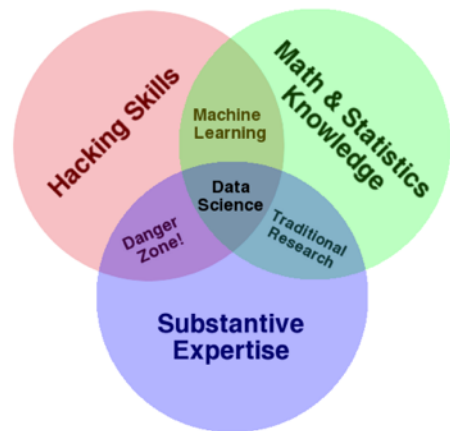
III. DATA MINING WORKFLOW

IV. GIT & GITHUB

EXERCISES:

V. I-PYTHON NOTEBOOK INTRO

QUESTIONS?



QUESTIONS?

WHAT WAS THE MOST INTERESTING THING YOU LEARNT?

WHAT WAS THE HARDEST TO GRASP?

I. PYTHON REVIEW

II. JUPYTER NOTEBOOK REVIEW

III. LINEAR ALGEBRA REVIEW

EXERCISES:

IV. PYTHON

V. NUMPY AND PANDAS IN IPYTHON

- **REVIEW HOW TO USE PYTHON OBJECTS LIKE: VARIABLES, STRINGS, LISTS, DISCTIONARIES, FUNCTIONS, CLASSES, MODULES**
- **REVIEW LINEAR ALGEBRA CONCEPTS LIKE: VECTOR, MATRIX, DOT PRODUCT**
- **BE ABLE TO USE THEM IN PYTHON**

INTRO TO PYTHON

PYTHON REVIEW

Q: What is Python?

A: An open source, high-level, dynamic scripting language.

- open source: *free! (both binaries and source files)*
- high-level: *interpreted (not compiled)*
- dynamic: *things that would typically happen at compile time happen at runtime instead (eg, dynamic typing)*
- scripting language: *“middle-weight”*

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than **right** now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.

Q: Why Anaconda python?

- *All in one place distribution of Python and commonly used libraries*
- *Easier to install and manage*
- *Ensures compatibility of libraries with each other*

Lets write a list of:

- *python data types*
- *python control flow statements*
- *misc python useful commands*

INTRO TO PYTHON

PYTHON DATA STRUCTURES

The most basic data structure is the None type. This is the equivalent of NULL in other languages.

Basic numeric types:

1. int ($< 2^{63}$) / long ($\geq 2^{63}$)*
* on 64-bit OS X/Linux, `sys.maxint = 2**63-1`
2. float (a “decimal”)
3. bool (True/False) or (1/0)
4. complex (“imaginary”)

```
>>> type(None)
<type 'NoneType'>
>>> type(1)
<type 'int'>
>>> type(2.5)
<type 'float'>
>>> type(True)
<type 'bool'>
>>> type(2+3j)
<type 'complex'>
```

*Array type, implemented in Python as a **list**.*

- *zero-base numbered, ordered collection of elements*
- *elements of arbitrary type.*
- *mutable (can be changed in-place)*

```
>>> a = [1, 'b', True]
>>> a[2]
True
>>> a[1] = 'aa'
>>> a
[1, 'aa', True]
```

Tuples: *immutable arrays of arbitrary elements.*

```
>>> x = (1, 'a', 2.5)
>>> x[0]
1
>>> x[0]='b'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> a,b = (1,2)
>>> a
1
```

Tuples are frequently used behind the scenes in a special type of variable assignment called tuple packing/unpacking.

The string type

- *immutable ordered array of characters (note there is no char type).*
- *support slicing and indexing operations like arrays*
- *have many other string-specific functions as well*

String processing is one area where Python excels.

dictionary *type*

- *Associative arrays (or hash tables)*
- *unordered collections of key-value pairs*
- *keys must be immutable*

```
>>> this_class={'subject':'Data Science','location':'501 Folsom',  
'duration':11,'has_begun':True}  
>>> this_class['subject']  
'Data Science'  
>>> this_class['has_begun']  
True
```


Sets

- *unordered mutable collections of distinct elements*
- *useful for checking membership of an element*
- *useful for ensuring element uniqueness*

```
>>> y = set([1,1,2,3,5,8])  
>>> y  
set([8, 1, 2, 3, 5])
```

file object

e.g open connection to a file

```
>>> with open('output_file.txt','w') as f:  
...     f.write('test')
```

note the "with" statement context manager, which automatically closes the file handle when it goes out of scope.

PYTHON CONTROL FLOW

if-else allows to execute alternative statements based on conditions

```
>>> x, y = False, False
>>> if x :
...     Print 'x is True'
... elif y :
...     Print 'y is True'
... else :
...     Print 'Neither...'
...
Neither...
```

while loop *executes while a given condition evaluates to True*

```
>>> x = 0
>>> while (x < 3) :
...     print 'HELLO!'
...     x += 1
...
HELLO!
HELLO!
HELLO!
```

for loop *executes a block of code for a range of values*

```
>>> for k in range(4) :  
...     print k**2  
...  
0  
1  
4  
9
```

The object that a for loop iterates over is called (appropriately) an iterable.

try-except block

```
>>> try:
...     print undefined_variable
... except :
...     print 'An Exception has been caught'
...
An Exception has been caught
```

useful for catching and dealing with errors, also called exception handling.

custom functions

```
>>> def x_minus_3(x) :  
...     return x - 3  
...  
>>> x_minus_3(12)  
9
```

NOTE: Functions can optionally return a value with a return statement (as this example does).

Functions arguments as inputs, and these arguments can be provided in two ways:

1) as positional arguments:

```
>>> def f(x,y) :  
...     return x - y  
...  
>>> f(4,2)  
2  
>>> f(2,4)  
-2
```

2) as keyword arguments:

```
>>> def g(arg1=10, arg2=20) :  
...     return arg1 / float(arg2)  
...  
>>> g()  
0.5  
>>> g(1,20)  
0.05  
>>> g(arg2=100)  
0.1
```

Classes *with member attributes and functions*:

```
>>> from math import pi
>>>
>>> class Circle() :
...     def __init__(self, r=1) :
...         self.radius = r
...     def area(self) :
...         return pi * (self.radius ** 2)
...
>>> c=Circle(4)
>>> c.radius
4
>>> c.area()
50.26548245743669
>>> 3.141592653589793 * 4 * 4
50.26548245743669
```

***import** statement to load libraries and functions:*

```
>>> import math
>>> math.pi
3.141592653589793
>>> from math import sin
>>> sin(math.pi/2)
1.0
>>> from math import *
>>> print e, log10(1000), cos(pi)
2.71828182846 3.0 -1.0
```

The three methods differ with respect to the interaction with the local namespace.

Comments are very important to make your code readable to others

```
# break when msg timestamp passes t_end
try:
    if created >= t_end:
        break

# if created DNE, keep going
except Exception as details:
    print details
    pass
```

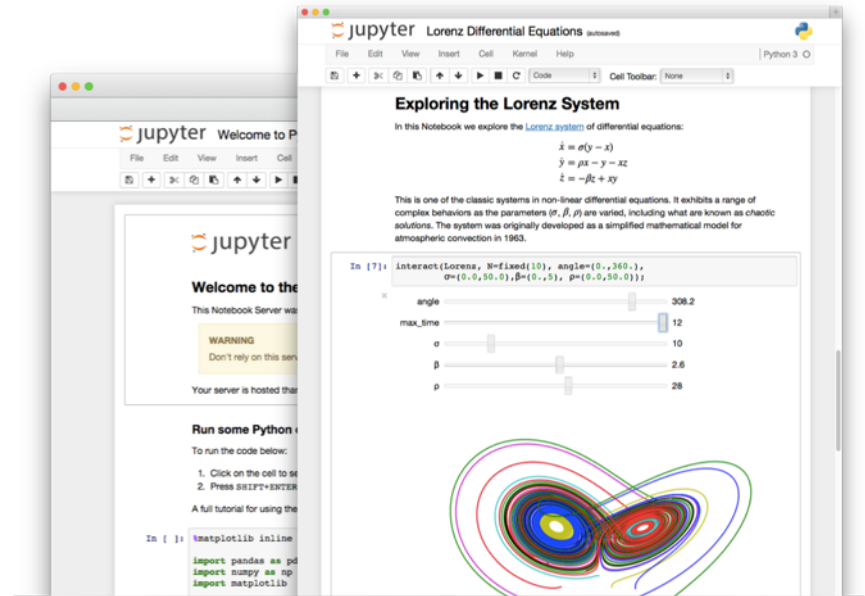
INTRO TO PYTHON

JUPYTER NOTEBOOK REVIEW

The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.

Uses include:

- *data cleaning and transformation*
- *numerical simulation*
- *statistical modeling*
- *machine learning*
- *and much more.*



Jupyter Architecture

The Jupyter Notebook is based on a set of open standards for interactive computing. Think HTML and CSS for interactive computing on the web. These open standards can be leveraged by third party developers to build customized applications with embedded interactive computing.



Notebook
Document Format

[Learn more](#)



Interactive
Computing Protocol

[Learn more](#)



The Kernel

[Learn more](#)



Jupyter Architecture

The Jupyter Notebook is based on a set of open standards for interactive computing. Think HTML and CSS for interactive computing on the web. These open standards can be leveraged by third party developers to build customized applications with embedded interactive computing.

The Notebook Document Format

Jupyter Notebooks are an open document format based on JSON. They contain a complete record of the user's sessions and embed code, narrative text, equations and rich output.

[Go back](#)

Interactive Computing Protocol

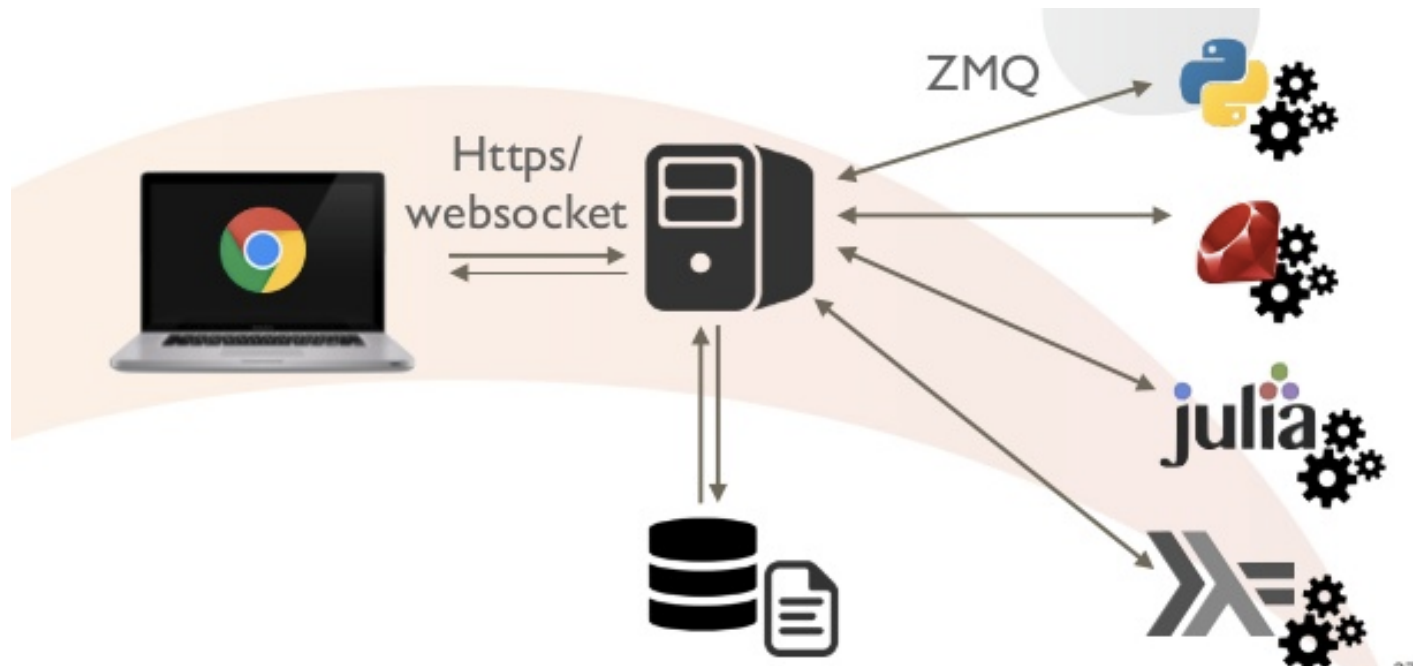
The Notebook communicates with computational Kernels using the Interactive Computing Protocol, an open network protocol based on JSON data over ZMQ and WebSockets.

[Go back](#)

The Kernel

Kernels are processes that run interactive code in a particular programming language and return output to the user. Kernels also respond to tab completion and introspection requests.

[Go back](#)





Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▾



<input type="checkbox"/>	<input type="checkbox"/>	A small blue house icon representing the home directory.
<input type="checkbox"/>	A small blue folder icon.	homework
<input type="checkbox"/>	A small blue folder icon.	labs
<input type="checkbox"/>	A small blue folder icon.	lectures
<input type="checkbox"/>	A small blue document icon.	README.md



50+ kernels available!

<https://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages>

Files

Running

Clusters

Select items to perform actions on them.

Upload

New ▾



homework



labs



lectures



README.md

Text File

Folder

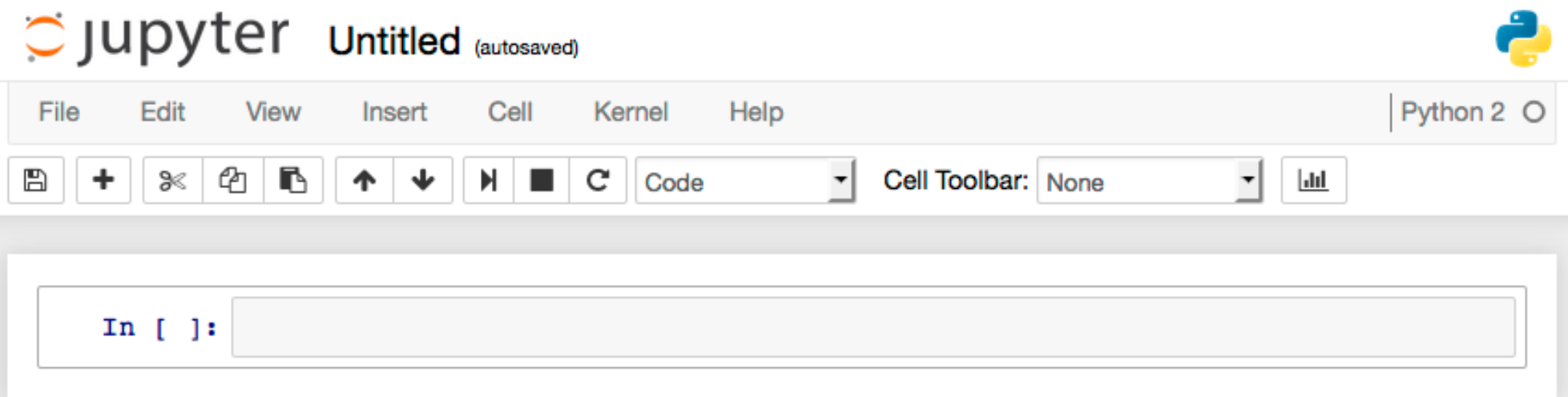
Terminal

Notebooks

Julia 0.4.0

Python 2

iTorch



The image shows the Jupyter Notebook interface. At the top, the Jupyter logo is followed by the text "Untitled (autosaved)" and the Python logo. Below this is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, and Help. To the right of the menu bar is a dropdown menu showing "Python 2". Below the menu bar is a toolbar with icons for saving, adding a new cell, cutting, copying, pasting, undo, redo, and a cell type dropdown menu currently set to "Code". To the right of the toolbar is a "Cell Toolbar" dropdown menu set to "None" and a button with a bar chart icon. The main area of the notebook is a large text input field with the prompt "In []:" followed by a large empty box for code.

jupyter Untitled (autosaved) Python 2

File Edit View Insert Cell Kernel Help

Save Add Cut Copy Paste Undo Redo Code Cell Toolbar: None

In []:

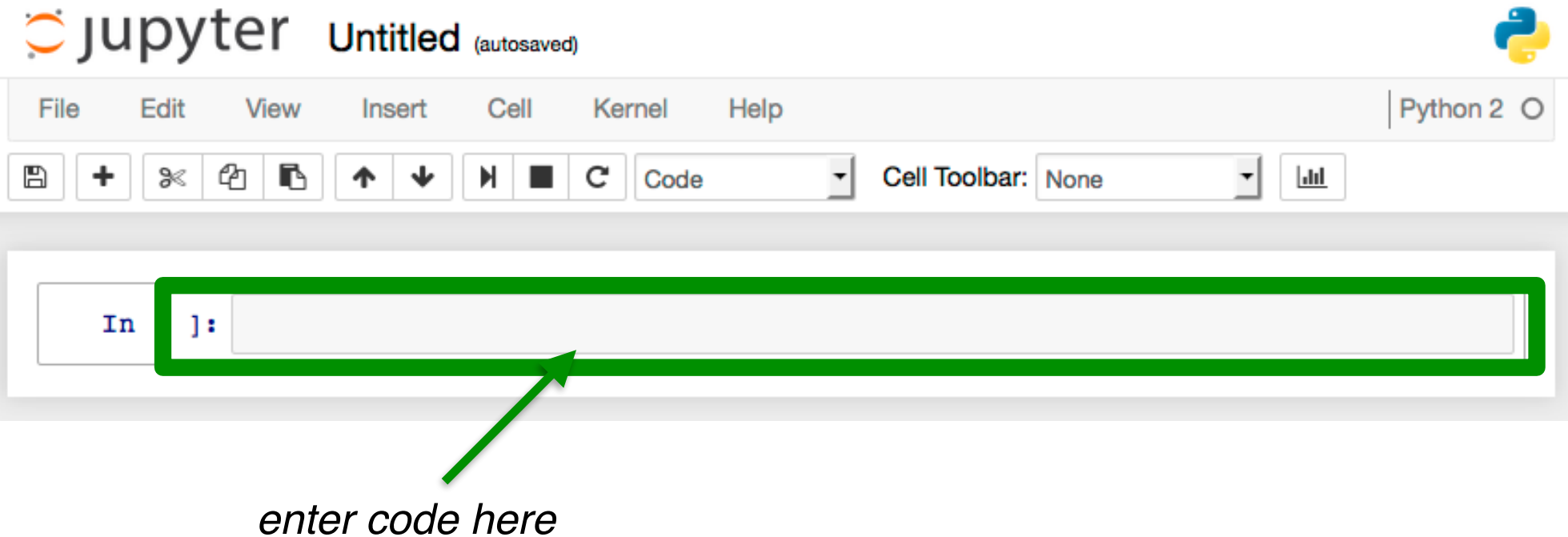
jupyter Untitled (autosaved)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

Python 2

In []:



The image shows the Jupyter Notebook interface. At the top, the title bar displays the Jupyter logo, the text "jupyter", and "Untitled (autosaved)". To the right is the Python logo. Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, and Help. On the far right of the menu bar is a dropdown menu showing "Python 2". Below the menu bar is a toolbar with icons for saving, adding a new cell, cutting, copying, pasting, undo, redo, and a cell type dropdown menu currently set to "Code". To the right of the toolbar is a "Cell Toolbar" dropdown menu set to "None" and a "Run" button. The main area of the notebook shows a single code cell. The cell has a label "In" on the left and a prompt "]: " followed by a large text input area. This input area is highlighted with a thick green rectangular border. A green arrow points from the text "enter code here" at the bottom towards the green-bordered input area.

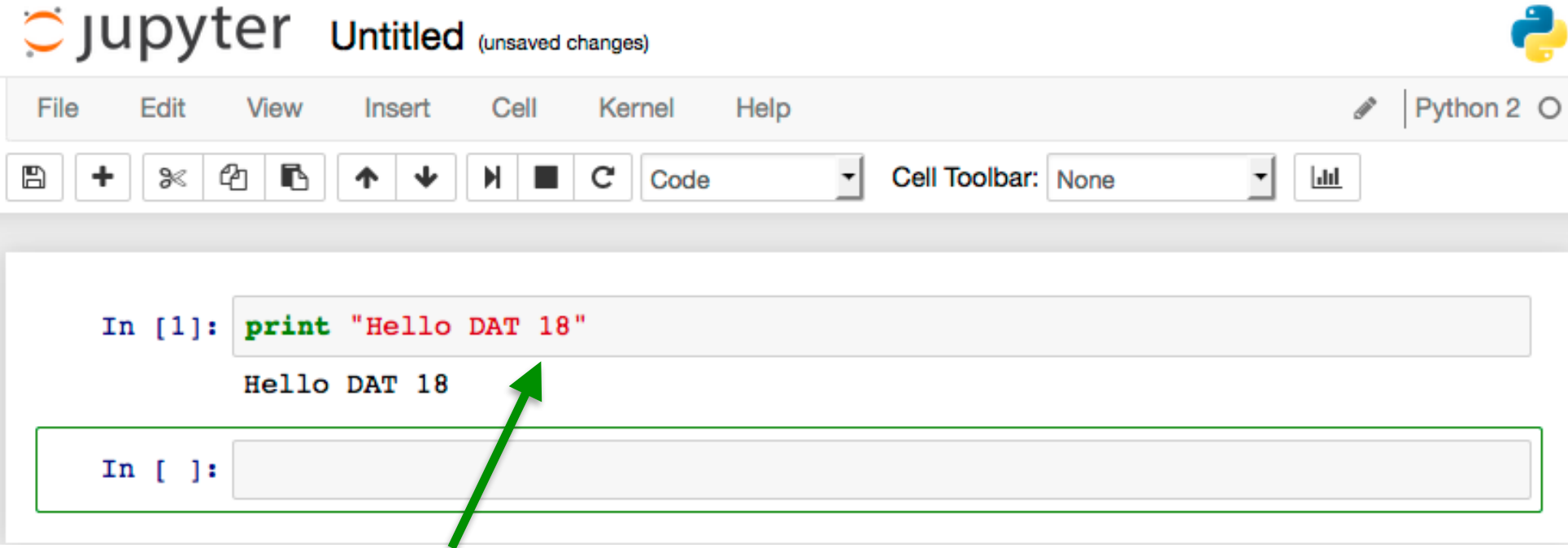
jupyter Untitled (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Save Add Cut Copy Paste Undo Redo Code Cell Toolbar: None Run

In]:

enter code here



The screenshot displays the Jupyter Notebook interface. At the top, the title bar shows the Jupyter logo, the text "jupyter", and the document name "Untitled (unsaved changes)". Below the title bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, and Help. To the right of the menu bar is a toolbar with icons for saving, adding, deleting, and duplicating cells, as well as navigation and execution controls. The main area shows a code cell with the input "In [1]:" followed by the code `print "Hello DAT 18"`. Below the code, the output "Hello DAT 18" is displayed. A green arrow points from the text "Shift + Enter runs code and returns results" at the bottom of the image to the execution button (a square icon) in the toolbar. Below the first cell is an empty code cell with the input "In []:".

jupyter Untitled (unsaved changes)

File Edit View Insert Cell Kernel Help

Python 2

Code

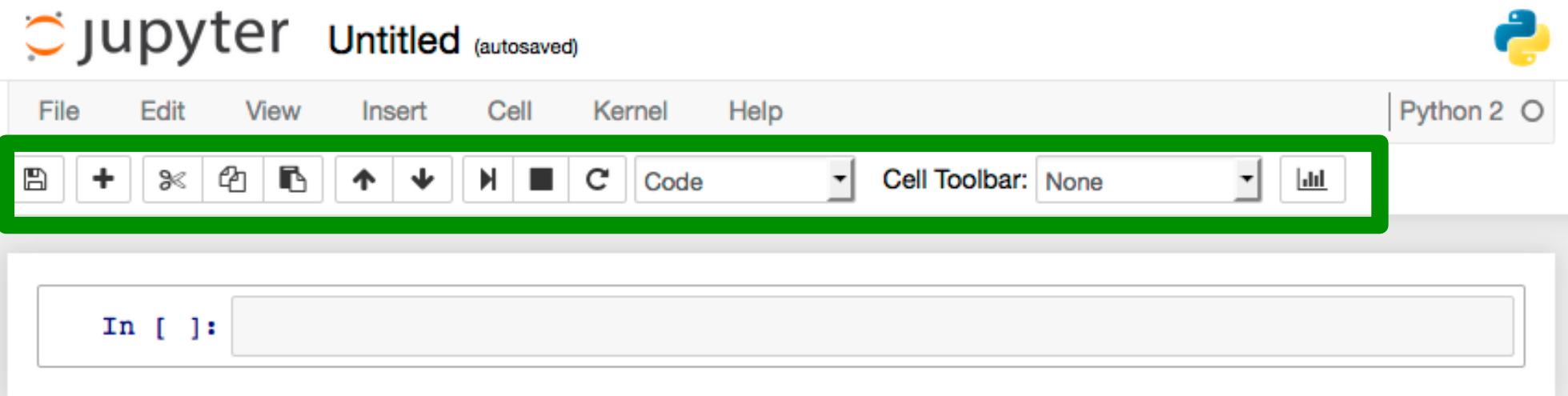
Cell Toolbar: None

In [1]: `print "Hello DAT 18"`

Hello DAT 18

In []:

Shift + Enter runs code and returns results



The screenshot displays the Jupyter Notebook interface. At the top, the title bar shows the Jupyter logo, the text "jupyter", and the document name "Untitled (autosaved)". To the right of the title bar is the Python logo and the text "Python 2". Below the title bar is a menu bar with the following items: File, Edit, View, Insert, Cell, Kernel, and Help. Below the menu bar is a toolbar, which is highlighted with a green border. The toolbar contains the following icons from left to right: a save icon (floppy disk), a plus icon (new), a scissors icon (cut), a document icon (copy), a document icon (paste), an up arrow icon, a down arrow icon, a play icon (run), a square icon (stop), a circular arrow icon (refresh), a dropdown menu currently showing "Code", a "Cell Toolbar:" dropdown menu currently showing "None", and a bar chart icon (output). Below the toolbar is a code input area with the prompt "In []:" followed by a large text input field.

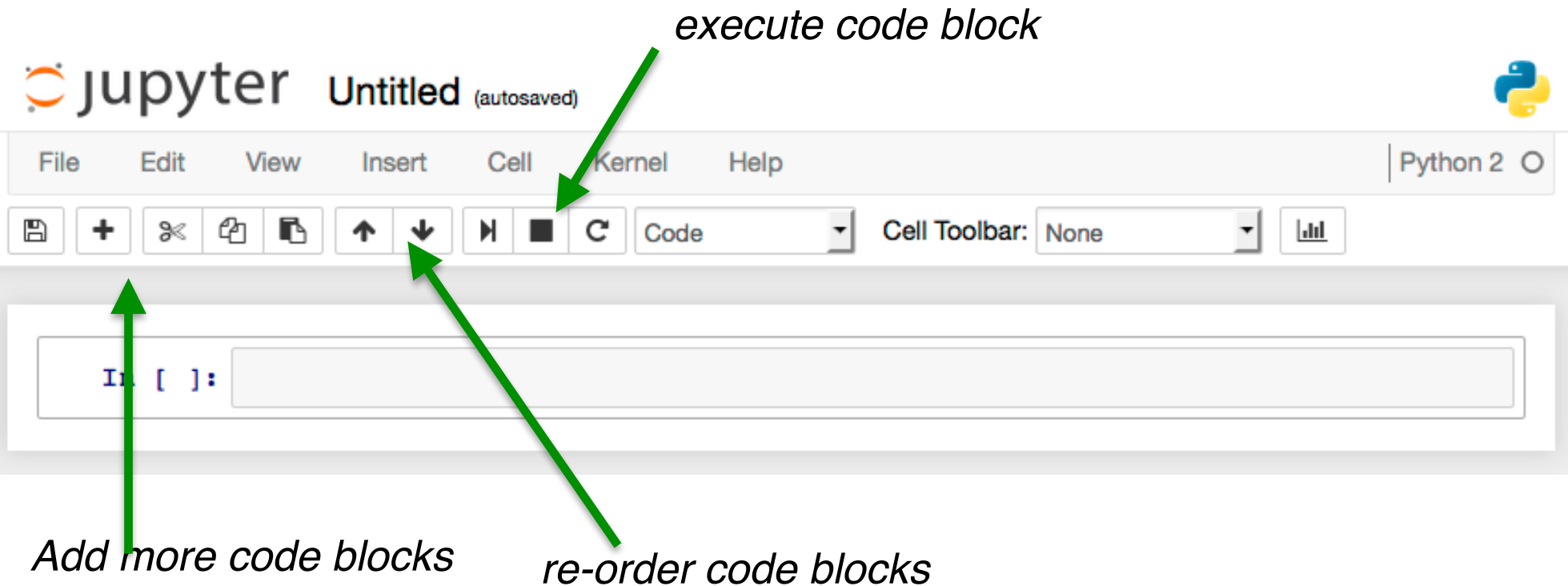
jupyter Untitled (autosaved)

Python 2

File Edit View Insert Cell Kernel Help

Save New Cut Copy Paste Up Down Run Stop Refresh Code Cell Toolbar: None Output

In []:





The image shows the Jupyter Notebook interface. At the top, the title bar displays the Jupyter logo, the text "Jupyter", and "Untitled (autosaved)". To the right of the title bar is the Python logo and the text "Python 2". Below the title bar is a menu bar with the following items: File, Edit, View, Insert, Cell, Kernel, and Help. Below the menu bar is a toolbar with the following icons: a save icon, a plus icon, a scissors icon, a copy icon, a paste icon, an up arrow icon, a down arrow icon, a play icon, a square icon, and a refresh icon. To the right of the toolbar is a dropdown menu showing "Code" and a "Cell Toolbar: None" dropdown. Below the toolbar is a code editor area with a prompt "In []:" followed by a text input field. Three green arrows point to specific icons in the toolbar: one points to the plus icon, another points to the down arrow icon, and a third points to the square icon.

execute code block

Add more code blocks

re-order code blocks

 **jupyter** Untitled (autosaved) 

File Edit View Insert Cell Kernel Help Python 2

Cell Toolbar: None

In [1]: `print "Hello"`
Hello DAT 18

In []:

Run
Run and Select Below
Run and Insert Below
Run All
Run All Above
Run All Below
Cell Type
Current Output
All Output

run multiple cells

jupyter Untitled (autosaved)



File Edit View Insert Cell Kernel Help

Python 2



Cell Toolbar: None



Interrupt

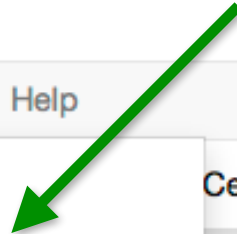
Restart

Reconnect

Change kernel ▶

```
In [1]: print "Hello DAT 18"
Hello DAT 18
```

```
In [ ]:
```

restart kernel

DATA FORMAT, ACCESS & TRANSFORMATION

QUESTIONS?

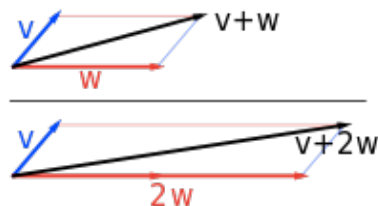
INTRO TO DATA SCIENCE

LINEAR ALGEBRA

Linear algebra is the branch of mathematics concerning **vector spaces** and **linear mappings** between such spaces.

Linear algebra is the branch of mathematics concerning **vector spaces** and **linear mappings** between such spaces.

A **vector space** (also called a linear space) is a collection of objects called vectors, which may be added together and multiplied ("scaled") by numbers, called scalars in this context.



Linear algebra is the branch of mathematics concerning **vector spaces** and **linear mappings** between such spaces.

A **vector space** (also called a linear space) is a collection of objects called vectors, which may be added together and multiplied ("scaled") by numbers, called scalars in this context.

A **linear mapping** is a mapping $V \rightarrow W$ between spaces that preserves the operations of addition and scalar multiplication.

Two operations are defined in a vector space

Two operations are defined in a vector space

ADDITION:

takes any two vectors **v** and **w** and outputs a third vector **v + w**

$$\mathbf{z} = \mathbf{v} + \mathbf{w}$$

Two operations are defined in a vector space

ADDITION:

takes any two vectors **\mathbf{v}** and **\mathbf{w}** and outputs a third vector **$\mathbf{v} + \mathbf{w}$**

$$\mathbf{z} = \mathbf{v} + \mathbf{w}$$

SCALAR MULTIPLICATION:

takes any scalar a and any vector **\mathbf{v}** and outputs a new vector **$a\mathbf{v}$**

$$\mathbf{z} = a\mathbf{v}$$

PROPERTY

MEANING

Associativity of addition

$$u + (v + w) = (u + v) + w$$

u, v and w are vectors in V , and a and b are scalars in F .

PROPERTY

MEANING

Associativity of addition

$$u + (v + w) = (u + v) + w$$

Commutativity of addition

$$u + v = v + u$$

u, v and w are vectors in V , and a and b are scalars in F .

PROPERTY	MEANING
Associativity of addition	$u + (v + w) = (u + v) + w$
Commutativity of addition	$u + v = v + u$
Identity element of addition	There exists an element $0 \in V$, called the zero vector , such that $v + 0 = v$ for all $v \in V$.

u, v and w are vectors in V , and a and b are scalars in F .

PROPERTY	MEANING
Associativity of addition	$u + (v + w) = (u + v) + w$
Commutativity of addition	$u + v = v + u$
Identity element of addition	There exists an element $0 \in V$, called the zero vector , such that $v + 0 = v$ for all $v \in V$.
Inverse elements of addition	For every $v \in V$, there exists an element $-v \in V$, called the additive inverse of v , such that $v + (-v) = 0$

u, v and w are vectors in V , and a and b are scalars in F .

PROPERTY	MEANING
Associativity of addition	$u + (v + w) = (u + v) + w$
Commutativity of addition	$u + v = v + u$
Identity element of addition	There exists an element $0 \in V$, called the zero vector , such that $v + 0 = v$ for all $v \in V$.
Inverse elements of addition	For every $v \in V$, there exists an element $-v \in V$, called the additive inverse of v , such that $v + (-v) = 0$
Distributivity (scalar and vector)	$a(u + v) = au + av$ $(a + b)v = av + bv$

u, v and w are vectors in V , and a and b are scalars in F .

PROPERTY	MEANING
Associativity of addition	$u + (v + w) = (u + v) + w$
Commutativity of addition	$u + v = v + u$
Identity element of addition	There exists an element $0 \in V$, called the zero vector , such that $v + 0 = v$ for all $v \in V$.
Inverse elements of addition	For every $v \in V$, there exists an element $-v \in V$, called the additive inverse of v , such that $v + (-v) = 0$
Distributivity (scalar and vector)	$a(u + v) = au + av$ $(a + b)v = av + bv$
Compatibility of multiplication	$a(bv) = (ab)v$

u, v and w are vectors in V , and a and b are scalars in F .

PROPERTY	MEANING
Associativity of addition	$u + (v + w) = (u + v) + w$
Commutativity of addition	$u + v = v + u$
Identity element of addition	There exists an element $0 \in V$, called the zero vector , such that $v + 0 = v$ for all $v \in V$.
Inverse elements of addition	For every $v \in V$, there exists an element $-v \in V$, called the additive inverse of v , such that $v + (-v) = 0$
Distributivity (scalar and vector)	$a(u + v) = au + av$ $(a + b)v = av + bv$
Compatibility of multiplication	$a(bv) = (ab)v$
Identity element of scalar multiplication	$1v = v$

u, v and w are vectors in V , and a and b are scalars in F .

A linear transformation T between two vector spaces V and W is compatible with scalar multiplication and vector addition:

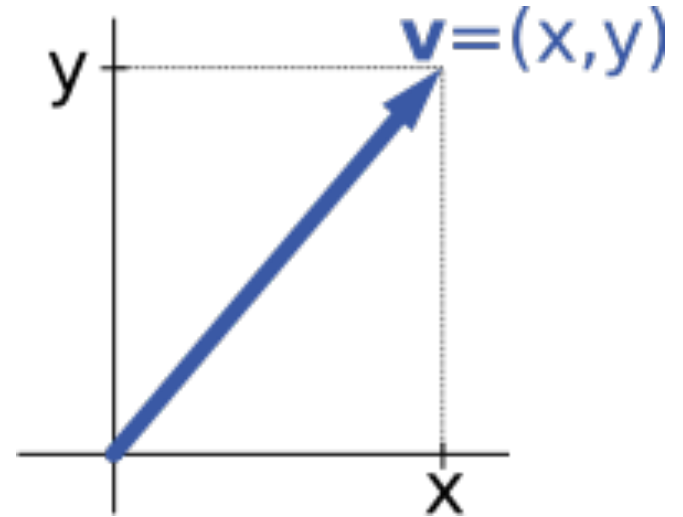
$$T: V \rightarrow W$$

satisfies:

$$T(a \mathbf{u} + b \mathbf{v}) = a T(\mathbf{u}) + b T(\mathbf{v})$$

Vectors can be represented by a list of numbers, their coordinates:

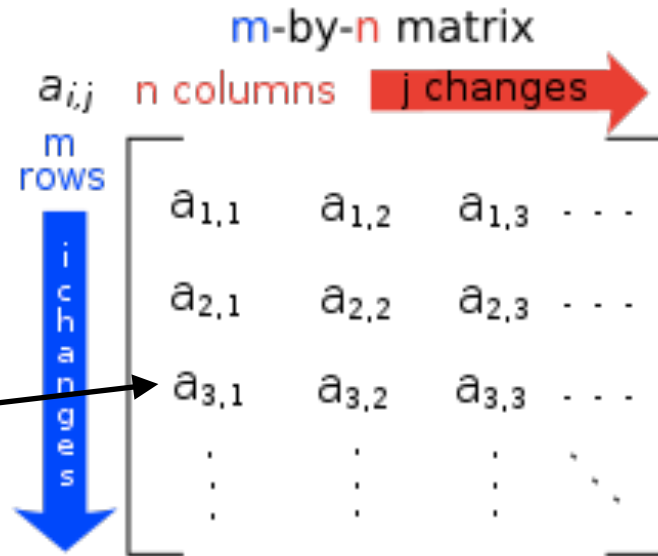
(give me some examples of vector quantities)



Linear mappings can be represented by **matrices**

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

Each value in a matrix is called an entry.



The size of a matrix is defined by the number of rows and columns.

Examples:

Name	Size	Example
Row vector	$1 \times n$	$\begin{bmatrix} 3 & 7 & 2 \end{bmatrix}$
Column vector	$n \times 1$	$\begin{bmatrix} 4 \\ 1 \\ 8 \end{bmatrix}$
Square matrix	$n \times n$	$\begin{bmatrix} 9 & 13 & 5 \\ 1 & 11 & 7 \\ 2 & 6 & 3 \end{bmatrix}$

Rule 1!

Matrices can be added together only when they are the same size. If they are not the same size, their sum is **undefined**.

$$[1 \ 3 \ 9 \ 2] + [2 \ 5 \ 9 \ 4] = [3 \ 8 \ 18 \ 6]$$

Rule 1!

Matrices can be added together only when they are the same size. If they are not the same size, their sum is **undefined**.

$$[1 \ 3 \ 9 \ 2] + [2 \ 5 \ 9 \ 4] = [3 \ 8 \ 18 \ 6]$$

$$[8 \ 72 \ 3 \ 1] + [17 \ 55 \ 3 \ 10] = ?$$

Rule 2!

Matrices can be multiplied by a scalar (single entity) value.

Each value in the matrix is multiplied by the scalar value.

$$[1 \ 3 \ 9 \ 2] * 3 = [3 \ 9 \ 27 \ 6]$$

$$[8 \ 72 \ 3 \ 1] * 2 = ?$$

Rule 3!

Matrices and vectors can be multiplied together given that the matrix columns are as wide as the vector is long.

What shape will the result take?

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

2×4 4×1

Rule 3!

Matrices and vectors can be multiplied together given that the matrix columns are as wide as the vector is long.

The result will always be a vector.

$$\begin{array}{ccc} \begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix} & * & \begin{bmatrix} 2 \\ 3 \\ 6 \\ 5 \end{bmatrix} = \begin{array}{c} (2 + 9 + 54 + 10) \\ (4 + 12 + 36 + 40) \end{array} = \begin{bmatrix} 75 \\ 92 \end{bmatrix} \\ 2 \times 4 & & 4 \times 1 \qquad \qquad \qquad 2 \times 1 \end{array}$$

Rule 4!

Matrices can be multiplied together using the same rules that we have from matrix-vector multiplication.

What shape will the result take?

$$\begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix} * \begin{bmatrix} 2 & 1 \\ 3 & 2 \\ 6 & 0 \\ 5 & 4 \end{bmatrix} = ?$$

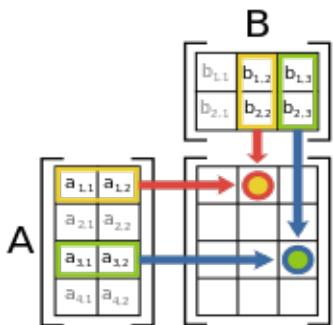
Rule 4!

Matrices can be multiplied together using the same rules that we have from matrix–vector multiplication.

The result will always be a matrix.

$$\begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix} * \begin{bmatrix} 2 & 1 \\ 3 & 2 \\ 6 & 0 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} (2 + 9 + 54 + 10) & (1 + 6 + 0 + 8) \\ (4 + 12 + 36 + 40) & (2 + 8 + 0 + 32) \end{bmatrix}$$

$$\begin{matrix} = 75 & = 15 \\ = 92 & = 42 \end{matrix}$$



Rule 4!

Matrices can be multiplied together using the same rules that we have from matrix–vector multiplication.

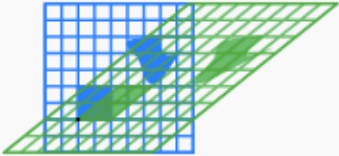
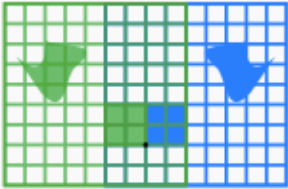
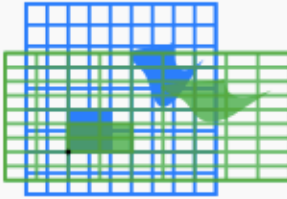
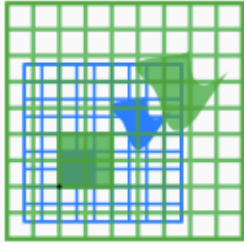
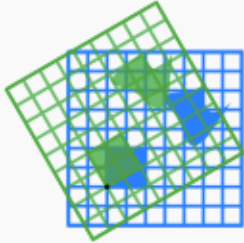
The result will always be a matrix.

IMPORTANT NOTE:

Matrix multiplication is **NOT COMMUTATIVE**. The order of matrix multiplication **DOES** matter. The number of columns of the first matrix must match the number of rows of the second matrix.

Here are some examples of operations in a 2D vector space with the corresponding matrix.

Each point in this space is represented by the vector of its coordinates $P = (x, y)$

Horizontal shear with $m=1.25$.	Horizontal flip	Squeeze mapping with $r=3/2$	Scaling by a factor of $3/2$	Rotation by $\pi/6^R = 30^\circ$
$\begin{bmatrix} 1 & 1.25 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 3/2 & 0 \\ 0 & 2/3 \end{bmatrix}$	$\begin{bmatrix} 3/2 & 0 \\ 0 & 3/2 \end{bmatrix}$	$\begin{bmatrix} \cos(\pi/6^R) & -\sin(\pi/6^R) \\ \sin(\pi/6^R) & \cos(\pi/6^R) \end{bmatrix}$
				

MATRICES

LINKS

[https://en.wikipedia.org/wiki/Matrix_\(mathematics\)](https://en.wikipedia.org/wiki/Matrix_(mathematics))

<http://mathworld.wolfram.com/Matrix.html>

<http://ed.ted.com/lessons/how-to-organize-add-and-multiply-matrices-bill-shillito>

INTRO TO DATA SCIENCE

NUMPY & PANDAS LAB