GENERAL ASSEMBLY

INTRO TO DATA SCIENCE

PYTHON AND LINEAR ALGEBRA REVIEW

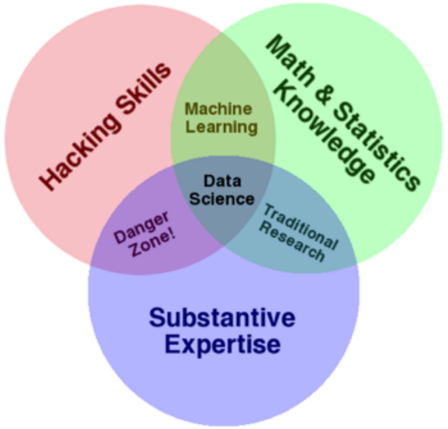
LAST TIME:

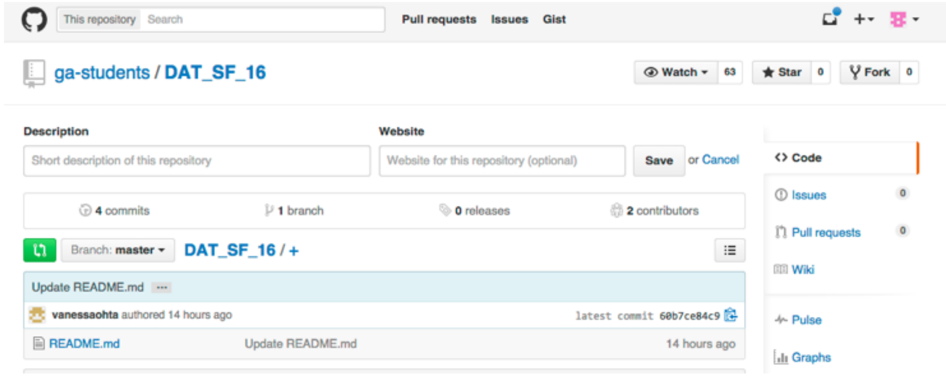
I. DATA SCIENCE
II. DATA SCIENTIST
III. DATA MINING WORKFLOW
IV. GIT & GITHUB

EXERCISES:

V. I-PYTHON NOTEBOOK INTRO

QUESTIONS?





ga-students / DAT_SF_16

Description: Short description of this repository
Website: Website for this repository (optional)

4 commits 1 branch 0 releases 2 contributors

Branch: master DAT_SF_16 / +

Update README.md ...
vanessaohta authored 14 hours ago latest commit 60b7ce84c9

README.md Update README.md 14 hours ago

QUESTIONS?

WHAT WAS THE MOST INTERESTING THING YOU LEARNT?

WHAT WAS THE HARDEST TO GRASP?

I. PYTHON REVIEW

II. LINEAR ALGEBRA REVIEW

EXERCISES:

III. PYTHON

IV. NUMPY AND PANDAS

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

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”*

PEP 20: THE ZEN OF PYTHON

```
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.
```

http://legacy.python.org/dev/peps/pep-0020/

Lets write a list of:

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

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 (≥ 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’>
```

http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-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]
```

http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

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*.*

http://docs.python.org/2/library/stdtypes.html#numeric-types-int-float-long-complex

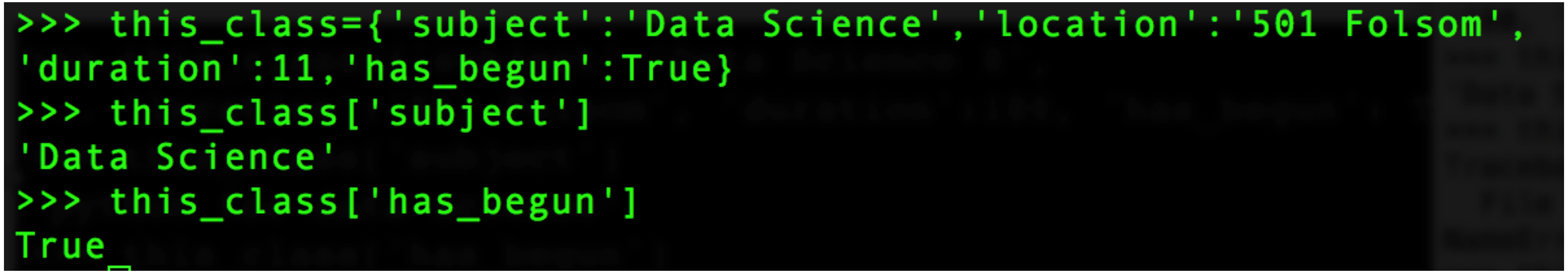
The string *type*

- *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
```

http://docs.python.org/2/tutorial/datastructures.html#dictionaries

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])
```

http://docs.python.org/2/library/sets.html 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.*

http://docs.python.org/2/library/stdtypes.html#file-objects

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

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**:*

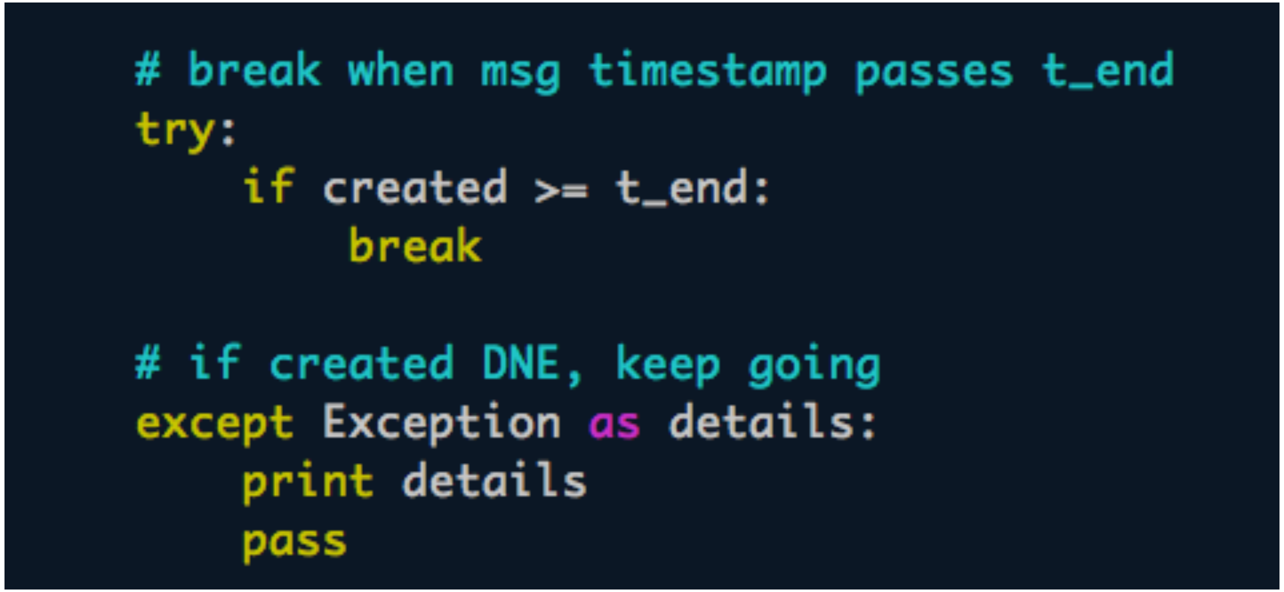
```
>>> from math import pi
>>>
>>> class Circle() :
...     def __init__(self, r=1) :
...         self.radius = r
...     def area(self) :
...         return pi * (self.radius ** 2)
...
>>> c=Circle(4)
>>> c.radius
4
>>> c.area()
50.26548245743669
>>> 3.141592653589793 * 4 * 4
50.26548245743669
```

***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
```

QUESTIONS?

LINEAR ALGEBRA

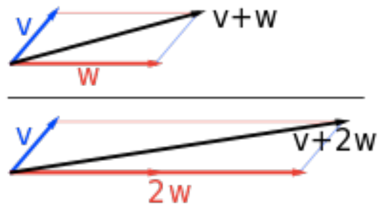
LINEAR ALGEBRA DEFINITIONS

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

https://en.wikipedia.org/wiki/Linear_algebra

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.



https://en.wikipedia.org/wiki/Vector_space

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 → W between spaces that preserves the operations of addition and scalar multiplication.

https://en.wikipedia.org/wiki/Linear_map

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 **v** and **w** and outputs a third vector **v + w**

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

SCALAR MULTIPLICATION:
takes any scalar *a* and any vector **v** and outputs a new vector *a***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 ∈ V, called the **zero vector**, such that v + 0 = v for all v ∈ V.
Inverse elements of addition	For every v ∈ V, there exists an element −v ∈ 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.

PROPERTIES OF A VECTOR SPACE OPERATIONS

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 ∈ V, called the **zero vector**, such that v + 0 = v for all v ∈ V.
Inverse elements of addition	For every v ∈ V, there exists an element −v ∈ 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:

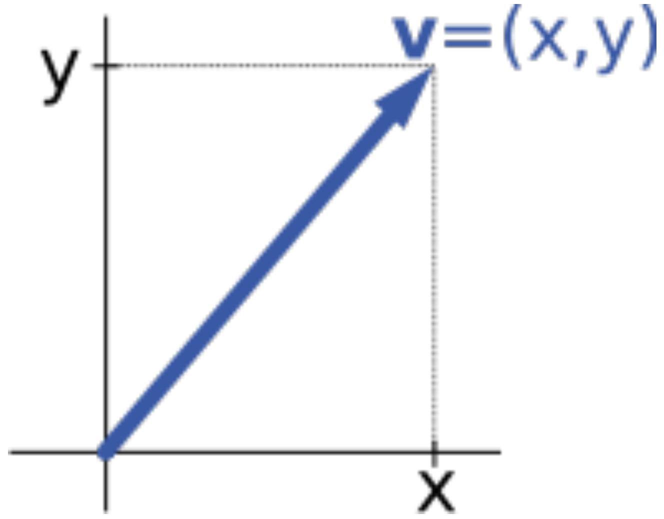
$$T: V \longrightarrow W$$

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:

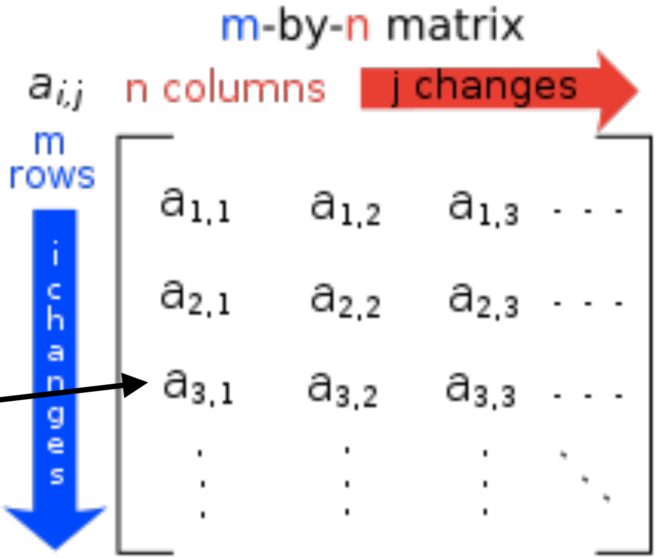
(give me some examples of vector quantities)



Linear mappings can be represented by **matrices**

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

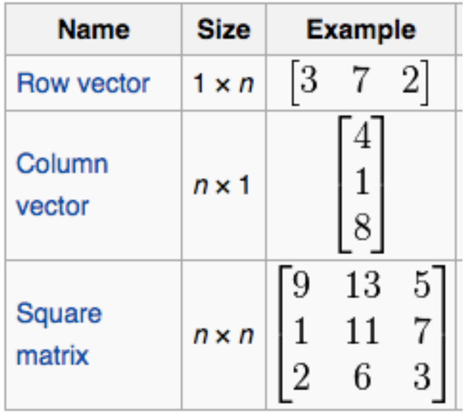
Each value in a matrix is called an entry.


m-by-n matrix
$a_{i,j}$
n columns
j changes
m rows
i changes
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ . . .
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$ . . .
$a_{3,1}$ $a_{3,2}$ $a_{3,3}$ . . .

MATRIX SIZE

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	$n \times n$	$\begin{bmatrix} 9 & 13 & 5 \\ 1 & 11 & 7 \\ 2 & 6 & 3 \end{bmatrix}$

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

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?

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

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.

$$\underset{2x4}{\begin{bmatrix} 1 & 3 & 9 & 2 \\ 2 & 4 & 6 & 8 \end{bmatrix}} * \underset{4x1}{\begin{bmatrix} 2 \\ 3 \\ 6 \\ 5 \end{bmatrix}} = \begin{matrix} (2+\ \ 9+54+10) \\ (4+12+36+40) \end{matrix} = \underset{2x1}{\begin{bmatrix} 75 \\ 92 \end{bmatrix}}$$

Rule 4!

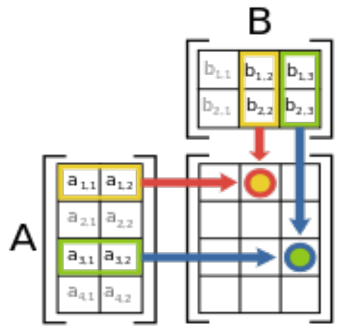
Matrices can be multiplied together using the same rules that we have from matrix-vector multiplication.
What shape will the result take?

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

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.

$$\begin{bmatrix}1&3&9&2\\2&4&6&8\end{bmatrix} * \begin{bmatrix}2&1\\3&2\\6&0\\5&4\end{bmatrix} = \begin{bmatrix} (2+9+54+10) & (1+6+0+8)\\ =75 & =15 \\ (4+12+36+40) & (2+8+0+32)\\ =92 & =42\end{bmatrix}$$



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.

IMPORTANT NOTE:

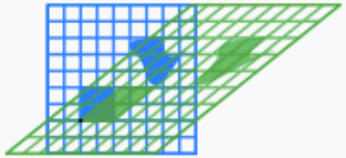
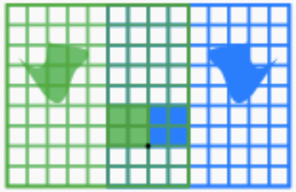
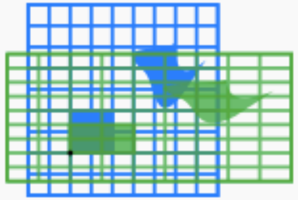
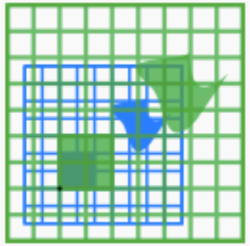
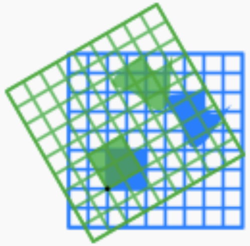
Matrix multiplication is NOT COMMUTATIVE. The order of matrix multiplication DOES matter. The number of columns of the first matrix must match the number of rows of the second matrix.

https://en.wikipedia.org/wiki/Matrix_(mathematics)#Matrix_multiplication

MATRIX TRANSFORMATIONS EXAMPLES

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 π/6^R = 30°*
$\begin{bmatrix}1 & 1.25\\0 & 1\end{bmatrix}$	$\begin{bmatrix}-1 & 0\\0 & 1\end{bmatrix}$	$\begin{bmatrix}3/2 & 0\\0 & 2/3\end{bmatrix}$	$\begin{bmatrix}3/2 & 0\\0 & 3/2\end{bmatrix}$	$\begin{bmatrix}\cos(\pi/6^R) & -\sin(\pi/6^R)\\\sin(\pi/6^R) & \cos(\pi/6^R)\end{bmatrix}$
				

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

MATRICES

LINKS

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

NUMPY & PANDAS LAB