

Logistics

- Midterm:
 - 3 hours
 - Content: things before this week (exercises)
 - 12pm – 11:59pm Wed

Lecture 7. Python Primer – Part 7

Chao Chen

Stony Brook University

Nov. 1, 2022

Main Function and Arguments

- In a file, the main code block starts with `if __name__ == '__main__':`
- When executed -- this is the starting point
- Function definitions are provided before the main function and can be called inside main (what if after?)
- Code outside the main block and not a function definition
 - also gets executed, but this is a bad habit. Should avoid.
 - Rule-of-thumb: each code line is either in a function definition or in the main

```
chaochen$ python3 counting-example.py 1 12 13 1 5 1 7
```

sys.argv[0]

sys.argv[1]

sys.argv[2]

sys.argv[3]

Counting-Example.py

```
import sys
From utilities import count
def func1(args..):
    .....
def func1(args..):
    .....
if __name__ == '__main__':
    .....
```

Main Function and Functions

- In a file, the main code block starts with `if __name__ == '__main__':`
- Call functions from default module
- Call self-defined functions
- Call count function in utility.py
from utility **import** count
- Can also do
import utility
utility.count(...)
- Note: any code not in main/function def in utility.py will be executed when importing.
- To avoid this, define the main code of utility.py
 - Only run when utility.py is executed)
Used for testing the utility.py functions
 - Will not run when importing utility
- **Rule of thumb:** anything not in a function should be in the main code block

counting-example.py

```
import sys
From utilities import count
def func1(args..):
    .....
def func1(args..):
    .....

if __name__ == '__main__':
    sys.exit()
    sys.argv

    func1(...)
    func2(...)

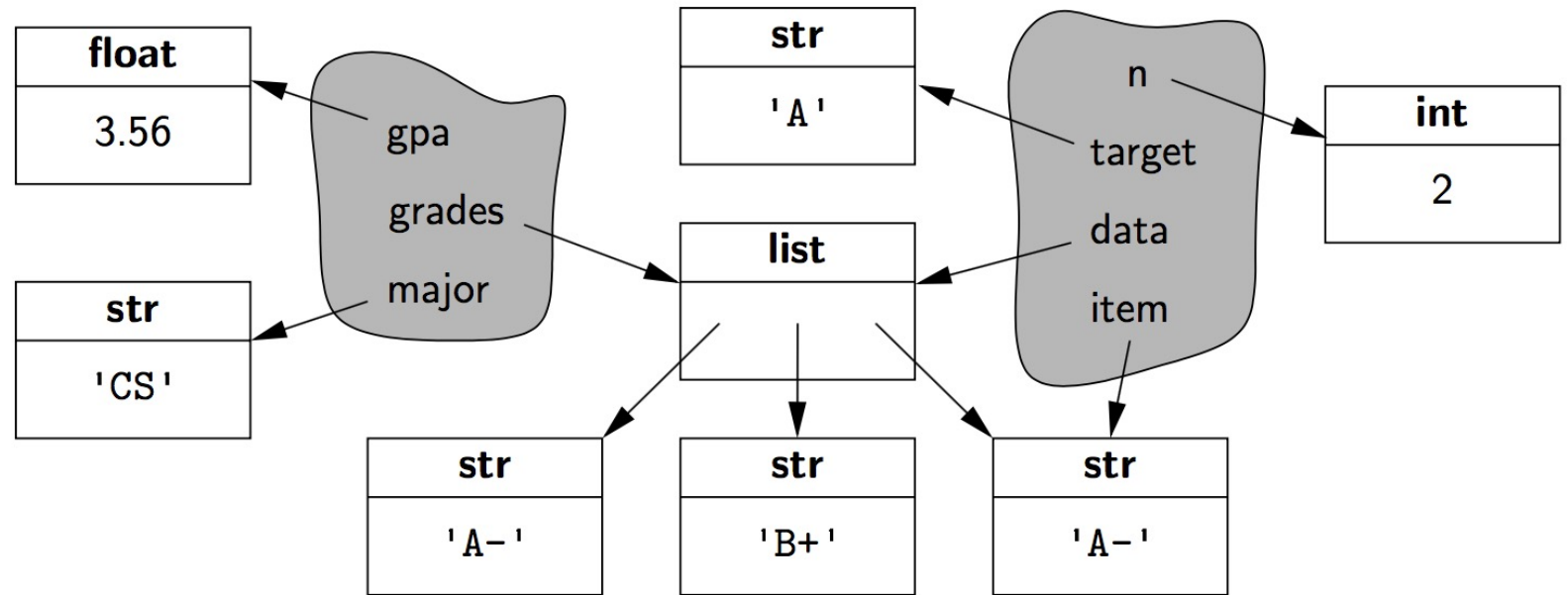
    count(...)
```

Scopes and Namespaces

- A name (identifier) is associated with an object within a **scope (namespace)**
 - Global scope; local to a function call

```
def count(data, target):  
    n = 0  
    for item in data:  
        if item == target:  
            n += 1  
    return n
```

- prizes = count(grades, 'A')



Useful tip:

In the interactive environment: use `who` to find currently available variables.

Scopes and Namespaces

- Checking variables within current scope:
 - locals() – local scope variables
 - globals() -- global scope
 - vars() and dir() like locals, but can pass in an object
 - vars() for objects with specific attributes only.
- Check out (can be outdated):
- <https://stackoverflow.com/questions/980249/difference-between-dir-and-vars-keys-in-python>

First-class Objects

- **first-class objects:** instances of a type that can be assigned to an identifier, passed as a parameter, or returned by a function.
 - Built-in classes (types)
 - Functions
 - Classes
 - Example 1:

```
scream = print # define scream as an alias for print function  
scream('Hello') # uses is like print
```
 - Example 2:

```
max(a, b, key=abs)
```

Random Module: Generating Random Numbers

- import random
s = [random.randint(3,10) for i in range(10)]
- random.seed -- making the random sequence repeatable
- Example

Syntax	Description
seed(hashable)	Initializes the pseudo-random number generator based upon the hash value of the parameter
random()	Returns a pseudo-random floating-point value in the interval [0.0, 1.0).
randint(a,b)	Returns a pseudo-random integer in the closed interval $[a, b]$.
randrange(start, stop, step)	Returns a pseudo-random integer in the standard Python range indicated by the parameters.
choice(seq)	Returns an element of the given sequence chosen pseudo-randomly.
shuffle(seq)	Reorders the elements of the given sequence pseudo-randomly.

Random Module: Generating Random Numbers

- **Pseudo-random number generator:** uses a deterministic formula to generate the next number in a sequence based upon one or more past numbers that it has generated.
 - $\text{next} = (a * \text{current} + b) \% n$ --- a , b , and n carefully chosen
 - Mersenne Twister
https://en.wikipedia.org/wiki/Mersenne_Twister
- In deployment:
`import time`
`random.seed(time.time())`
- In testing/debugging: use `random.seed(100)`
Pick a fixed seed, so the random sequence can be repeated.
- Exercises:
 - Implement `randint` using `random`
 - Given $P(x=0) = 0.25$, $P(x=1) = 0.5$, $P(x=2) = 0.25$, design a random generator which samples according to P
`myrandint(P)`

From uniform to normal distribution (box-muller)

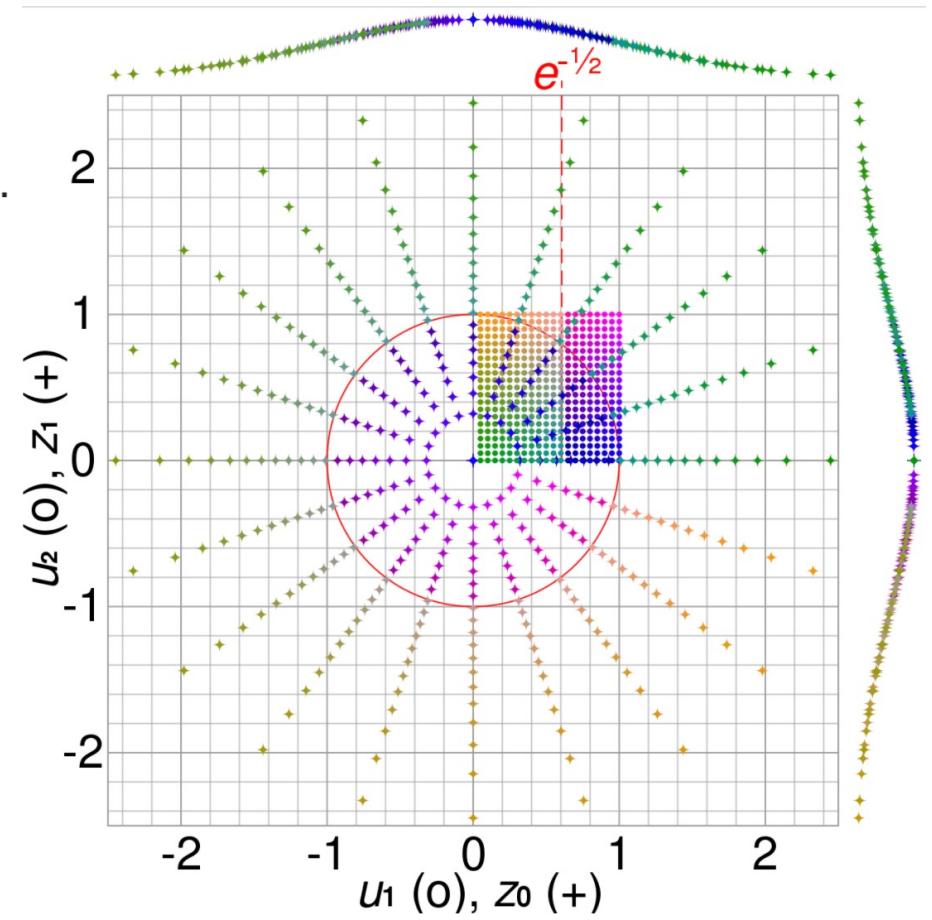
Suppose U_1 and U_2 are independent samples chosen from the uniform distribution on the **unit interval** (0, 1). Let

$$Z_0 = R \cos(\Theta) = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$$

and

$$Z_1 = R \sin(\Theta) = \sqrt{-2 \ln U_1} \sin(2\pi U_2).$$

Then Z_0 and Z_1 are **independent** random variables with a **standard normal distribution**.



Pseudorandom

```
In [238]: samples = np.random.normal(size=(4, 4))
```

```
In [239]: samples
```

```
Out[239]:
```

```
array([[ 0.5732,  0.1933,  0.4429,  1.2796],  
       [ 0.575 ,  0.4339, -0.7658, -1.237 ],  
       [-0.5367,  1.8545, -0.92  , -0.1082],  
       [ 0.1525,  0.9435, -1.0953, -0.144 ]])
```

Pseudorandom

- Motivation -- efficiency

```
In [240]: from random import normalvariate
```

```
In [241]: N = 1000000
```

```
In [242]: %timeit samples = [normalvariate(0, 1) for _ in range(N)]  
1.77 s +- 126 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

```
In [243]: %timeit np.random.normal(size=N)  
61.7 ms +- 1.32 ms per loop (mean +- std. dev. of 7 runs, 10 loops each)
```

Pseudorandom

- Same seed -- same randomized sequence

```
In [244]: np.random.seed(1234)
```

The data generation functions in `numpy.random` use a global random seed. To avoid global state, you can use `numpy.random.RandomState` to create a random number generator isolated from others:

```
In [245]: rng = np.random.RandomState(1234)
```

```
In [246]: rng.randn(10)
```

```
Out[246]:
```

```
array([ 0.4714, -1.191 ,  1.4327, -0.3127, -0.7206,  0.8872,  0.8596,  
       -0.6365,  0.0157, -2.2427])
```

Exceptions

- Unexpected situations: wrong input type, wrong values, etc.
Still want the program to proceed, not stopping
- **exceptions (errors)**: objects that are **raised** (or **thrown**) by code that encounters an unexpected circumstance.
- A raised error may be **caught** by a surrounding context that “handles” the exception in an appropriate fashion.
- If uncaught, an exception causes the interpreter to stop executing the program and to report an appropriate message to the console.
- Examples:
 - use of an undefined identifier in an expression causes a NameError,
 - If object foo does not support a member called bar, foo.bar() will generate an AttributeError.
 - ValueError if calling int('abc')
 - TypeError if abs('hello')
 - IndexError for wrongly indexing a list/tuple, KeyError for dict

Exceptions

- *Catching the exception:* **try:**
 fp = open('sample.txt')
except IOError **as** e:
 print('Unable to open the file:', e)
- Example code: counting-example.py
- *Raising an exception in your own code (argument – error message):*

```
def sqrt(x):  
    if not isinstance(x, (int, float)):   
        raise TypeError('x must be numeric')  
    elif x < 0:  
        raise ValueError('x cannot be negative')  
    # do the real work here...
```

Exceptions

- Particularly useful for dealing with **unpredictable** user input
- Use **except**: to process any unprocessed exception
- Use **finally** in the end to execute with or without exception

```
age = -1                                # an initially invalid choice
while age <= 0:
    try:
        age = int(input('Enter your age in years: '))
        if age <= 0:
            print('Your age must be positive')
    except ValueError:
        print('That is an invalid age specification')
    except EOFError:
        print('There was an unexpected error reading input.')
    raise                                # let's re-raise this exception
```


Exceptions (common types)

Class	Description
Exception	A base class for most error types
AttributeError	Raised by syntax <code>obj.foo</code> , if <code>obj</code> has no member named <code>foo</code>
EOFError	Raised if “end of file” reached for console or file input
IOError	Raised upon failure of I/O operation (e.g., opening file)
IndexError	Raised if index to sequence is out of bounds
KeyError	Raised if nonexistent key requested for set or dictionary
KeyboardInterrupt	Raised if user types ctrl-C while program is executing
NameError	Raised if nonexistent identifier used
StopIteration	Raised by <code>next(iterator)</code> if no element; see Section 1.8
TypeError	Raised when wrong type of parameter is sent to a function
ValueError	Raised when parameter has invalid value (e.g., <code>sqrt(-5)</code>)
ZeroDivisionError	Raised when any division operator used with 0 as divisor

Questions (exercises)

- `myTuple = ((2,3,4), [2,3,4])`

What will happen if: `myTuple[1] = 5` or `myTuple[1][1] = 5` or `myTuple[0][1] = 5`

Questions (exercises)

- R-1.1** Write a short Python function, `is_multiple(n, m)`, that takes two integer values and returns `True` if n is a multiple of m , that is, $n = mi$ for some integer i , and `False` otherwise.
- R-1.2** Write a short Python function, `is_even(k)`, that takes an integer value and returns `True` if k is even, and **False** otherwise. However, your function cannot use the multiplication, modulo, or division operators.
- R-1.3** Write a short Python function, `minmax(data)`, that takes a sequence of one or more numbers, and returns the smallest and largest numbers, in the form of a tuple of length two. Do not use the built-in functions `min` or `max` in implementing your solution.

Questions (exercises)

R-1.6 Write a short Python function that takes a positive integer n and returns the sum of the squares of all the odd positive integers smaller than n .

Questions (exercises)

- R-1.8** Python allows negative integers to be used as indices into a sequence, such as a string. If string s has length n , and expression $s[k]$ is used for index $-n \leq k < 0$, what is the equivalent index $j \geq 0$ such that $s[j]$ references the same element?
- C-1.13** Write a pseudo-code description of a function that reverses a list of n integers, so that the numbers are listed in the opposite order than they were before, and compare this method to an equivalent Python function for doing the same thing.
- C-1.15** Write a Python function that takes a sequence of numbers and determines if all the numbers are different from each other (that is, they are distinct).

Questions (exercises)

C-1.28 The *p-norm* of a vector $v = (v_1, v_2, \dots, v_n)$ in n -dimensional space is defined as

$$\|v\| = \sqrt[p]{v_1^p + v_2^p + \dots + v_n^p}.$$

For the special case of $p = 2$, this results in the traditional *Euclidean norm*, which represents the length of the vector. For example, the Euclidean norm of a two-dimensional vector with coordinates $(4, 3)$ has a Euclidean norm of $\sqrt{4^2 + 3^2} = \sqrt{16 + 9} = \sqrt{25} = 5$. Give an implementation of a function named `norm` such that `norm(v, p)` returns the p -norm value of v and `norm(v)` returns the Euclidean norm of v . You may assume that v is a list of numbers.

Use `math.pow()` <https://docs.python.org/3/library/math.html>

Try implementing using list comprehension

Still not fast enough --- use `numpy`

Questions (exercises)

- C-1.21** Write a Python program that repeatedly reads lines from standard input until an EOFError is raised, and then outputs those lines in reverse order (a user can indicate end of input by typing ctrl-D).
- P-1.32** Write a Python program that can simulate a simple calculator, using the console as the exclusive input and output device. That is, each input to the calculator, be it a number, like 12.34 or 1034, or an operator, like + or =, can be done on a separate line. After each such input, you should output to the Python console what would be displayed on your calculator.

Questions (exercises)

- P-1.35** The *birthday paradox* says that the probability that two people in a room will have the same birthday is more than half, provided n , the number of people in the room, is more than 23. This property is not really a paradox, but many people find it surprising. Design a Python program that can test this paradox by a series of experiments on randomly generated birthdays, which test this paradox for $n = 5, 10, 15, 20, \dots, 100$.
- P-1.36** Write a Python program that inputs a list of words, separated by white-space, and outputs how many times each word appears in the list. You need not worry about efficiency at this point, however, as this topic is something that will be addressed later in this book.

Linear Algebra

- Vectors
 - A one dimensional array.
 - If not specified, assume x is a column vector.
- Matrices
 - Higher dimensional array.
 - Typically denoted with capital letters.
 - n rows by m columns

$$\mathbf{X} = \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix}$$

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,m-1} \\ a_{1,0} & a_{1,1} & & a_{1,m-1} \\ \vdots & & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,m-1} \end{pmatrix}$$

Transposition

- **Transposing** a matrix swaps columns and rows.

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix}$$

$$\mathbf{x}^T = (x_0 \quad x_1 \quad \dots \quad x_{n-1})$$

Transposition

- **Transposing** a matrix swaps columns and rows.

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,m-1} \\ a_{1,0} & a_{1,1} & & a_{1,m-1} \\ \vdots & & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \dots & a_{n-1,m-1} \end{pmatrix}$$
$$A^T = \begin{pmatrix} a_{0,0} & a_{1,0} & \dots & a_{n-1,0} \\ a_{0,1} & a_{1,1} & & a_{1,m-1} \\ \vdots & & \ddots & \vdots \\ a_{0,m-1} & a_{1,m-1} & \dots & a_{n-1,m-1} \end{pmatrix}$$

Addition

- Matrices can be added to themselves iff they have the same dimensions.
 - A and B are both n-by-m matrices.

$$A + B = \begin{pmatrix} a_{0,0} + b_{0,0} & a_{0,1} + b_{0,1} & \dots & a_{0,m-1} + b_{0,m-1} \\ a_{1,0} + b_{1,0} & a_{1,1} + b_{1,1} & & a_{1,m-1} + b_{1,m-1} \\ \vdots & & \ddots & \vdots \\ a_{n-1,0} + b_{n-1,0} & a_{n-1,1} + b_{n-1,1} & \dots & a_{n-1,m-1} + b_{n-1,m-1} \end{pmatrix}$$

$$\begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} + \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} 7 & 8 \\ 5 & -3 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 8 \\ 4 & 6 \end{bmatrix} - \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} -1 & 8 \\ 3 & 15 \end{bmatrix}$$

Hadamard Product

- Element-wise product (like addition)
 - A and B are both n-by-m matrices.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \circ \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11} b_{11} & a_{12} b_{12} & a_{13} b_{13} \\ a_{21} b_{21} & a_{22} b_{22} & a_{23} b_{23} \\ a_{31} b_{31} & a_{32} b_{32} & a_{33} b_{33} \end{pmatrix}$$

- Multiplies with a scalar

$2 \times \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 2 & -18 \end{bmatrix}$ $-(2) = -2$

$-\begin{bmatrix} 2 & -4 \\ 7 & 10 \end{bmatrix} = \begin{bmatrix} -2 & 4 \\ -7 & -10 \end{bmatrix}$

Multiplication

- To multiply two matrices, the **inner dimensions** must be the same.
 - An n-by-m matrix can be multiplied by an m-by-k matrix
 - Outcome: n-by-k matrix

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

$$(1, 2, 3) \cdot (7, 9, 11) = 1 \times 7 + 2 \times 9 + 3 \times 11 = 58$$

Multiplication

A diagram illustrating the calculation of the element 64 in the product matrix. The first matrix is $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ and the second is $\begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix}$. The product is $\begin{bmatrix} 58 & 64 \\ \end{bmatrix}$. A yellow oval highlights the first row of the first matrix (1, 2, 3). A yellow oval highlights the second column of the second matrix (8, 10, 12). A yellow arrow points from the 1 in the first row to the 8 in the second column. Another yellow arrow points from the 2 in the first row to the 10 in the second column. A third yellow arrow points from the 3 in the first row to the 12 in the second column. The result 64 in the product matrix is also highlighted with a yellow oval.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ \end{bmatrix}$$

$$(1, 2, 3) \cdot (8, 10, 12) = 1 \times 8 + 2 \times 10 + 3 \times 12 = 64$$

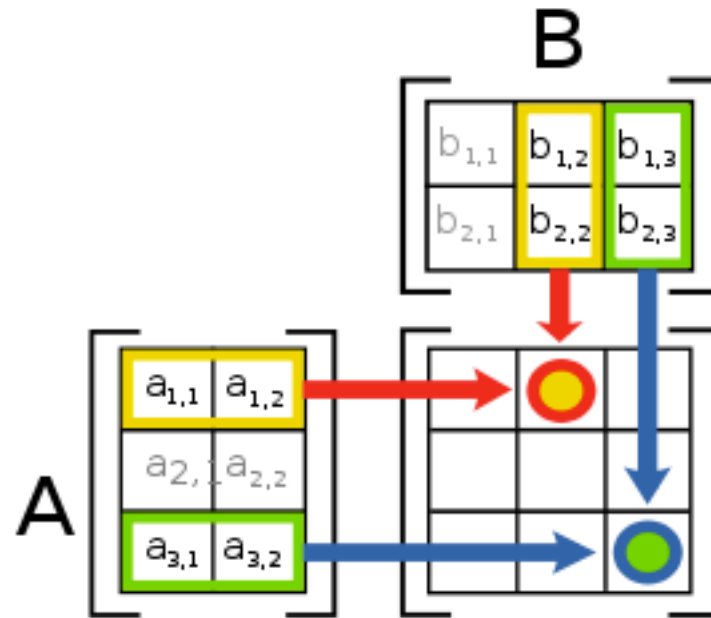
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

Multiplication

- To multiply two matrices, the **inner dimensions** must be the same.
 - An n-by-m matrix can be multiplied by an m-by-k matrix

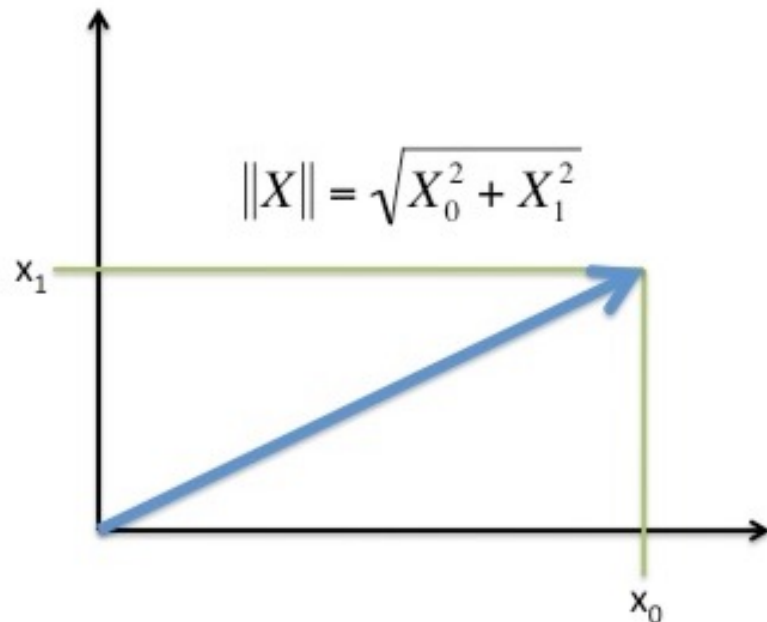
$$AB = C$$

$$c_{ij} = \sum_{k=0}^m a_{ik} * b_{kj}$$



Norm

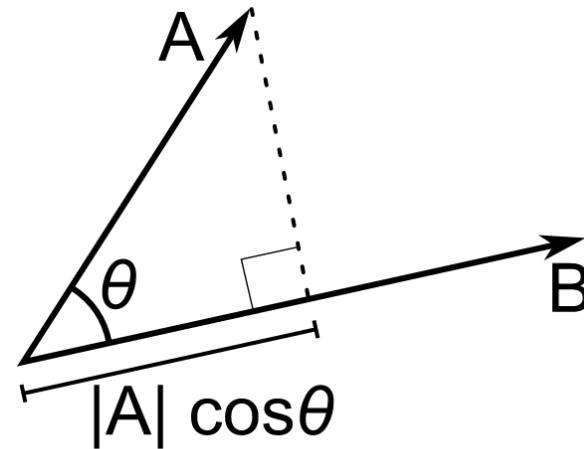
- The **norm** of a vector, \mathbf{x} , represents the Euclidean length of a vector.



$$\|\mathbf{x}\| = \sqrt{\sum_{i=0}^{n-1} x_i^2}$$
$$= \sqrt{x_0^2 + x_1^2 + \dots + x_{n-1}^2}$$

Operations on Vectors

- $\mathbf{u} + \mathbf{v}$, $\mathbf{u} - \mathbf{v}$, $k\mathbf{u}$
- Dot product: $\mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \langle \mathbf{u}, \mathbf{v} \rangle = ?$
- Norm: $\|\mathbf{u}\|^2 = \langle \mathbf{u}, \mathbf{u} \rangle$
- Geometric views
 - $\langle \mathbf{u}, \mathbf{v} \rangle = \|\mathbf{u}\| \|\mathbf{v}\| \cos(\theta)$
 - $|\langle \mathbf{u}, \mathbf{v} \rangle| \leq \|\mathbf{u}\| \|\mathbf{v}\|$
- Q: projection of \mathbf{u} on \mathbf{v} direction?



Transposition

- Transposition rules

$$\begin{aligned}(\mathbf{A} + \mathbf{B})^T &= \mathbf{A}^T + \mathbf{B}^T \\ (\mathbf{AB})^T &= \mathbf{B}^T \mathbf{A}^T\end{aligned}$$

- Sanity check: dimensions ($\mathbf{A}^T \mathbf{B}^T$)

Distributive/Commutative Law,

- $A(B+C) = AB + AC$
 - Matrix/vector/scalar
- $(A+B)(C+D) = ?$
- $||u-v|| = ?$
- Commutative law:
 - Only if inner product: $\langle u,v \rangle = \langle v,u \rangle$
 - Also scalar $kA = Ak$

Identity Matrix

- Matrices are invariant under multiplication by the identity matrix.

$$AI = A$$

$$IA = A$$

- What if A is $m \times n$?

NumPy – a convenient module for matrix/vector

One of the reasons NumPy is so important for numerical computations in Python is because it is designed for efficiency on large arrays of data. There are a number of reasons for this:

- NumPy internally stores data in a contiguous block of memory, independent of other built-in Python objects. NumPy's library of algorithms written in the C language can operate on this memory without any type checking or other overhead. NumPy arrays also use much less memory than built-in Python sequences.
- NumPy operations perform complex computations on entire arrays without the need for Python for loops.

First example

To give you an idea of the performance difference, consider a NumPy array of one million integers, and the equivalent Python list:

```
In [7]: import numpy as np
```

```
In [8]: my_arr = np.arange(1000000)
```

```
In [9]: my_list = list(range(1000000))
```

Now let's multiply each sequence by 2:

```
In [10]: %time for _ in range(10): my_arr2 = my_arr * 2  
CPU times: user 20 ms, sys: 50 ms, total: 70 ms  
Wall time: 72.4 ms
```

```
In [11]: %time for _ in range(10): my_list2 = [x * 2 for x in my_list]  
CPU times: user 760 ms, sys: 290 ms, total: 1.05 s  
Wall time: 1.05 s
```

NumPy-based algorithms are generally 10 to 100 times faster (or more) than their pure Python counterparts and use significantly less memory.

ndarray – vectors/matrices

```
In [12]: import numpy as np
```

```
# Generate some random data
```

```
In [13]: data = np.random.randn(2, 3)
```

```
In [14]: data
```

```
Out[14]:
```

```
array([[ -0.2047,  0.4789, -0.5194],  
       [ -0.5557,  1.9658,  1.3934]])
```

```
In [15]: data * 10
```

```
Out[15]:
```

```
array([[ -2.0471,  4.7894, -5.1944],  
       [ -5.5573, 19.6578, 13.9341]])
```

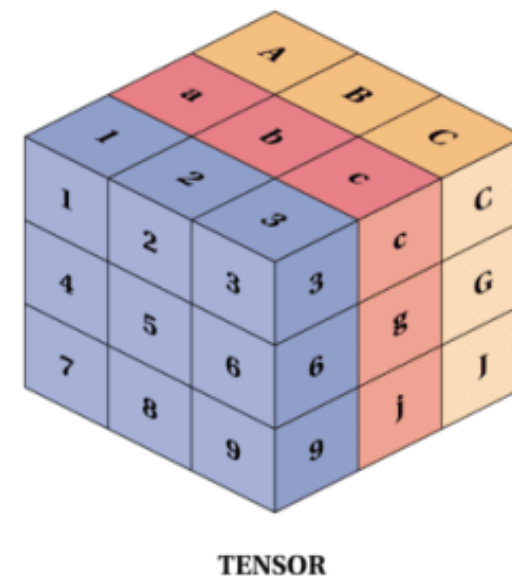
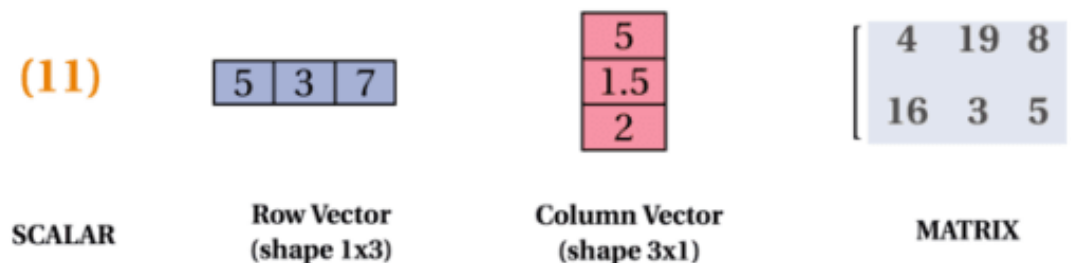
```
In [16]: data + data
```

```
Out[16]:
```

```
array([[ -0.4094,  0.9579, -1.0389],  
       [ -1.1115,  3.9316,  2.7868]])
```


ndarray – vectors/matrices

- Nddarray
 - 1d -- vector
 - 2d -- matrix
 - 3d – tensor
 - 4d -- ...



- Same dtype for all elements (diff from list)

```
In [17]: data.shape  
Out[17]: (2, 3)
```

```
In [18]: data.dtype  
Out[18]: dtype('float64')
```



Whenever you see “array,” “NumPy array,” or “ndarray” in the text, with few exceptions they all refer to the same thing: the ndarray object.

Creation of ndarrays

- Creation from list, list of lists

```
In [19]: data1 = [6, 7.5, 8, 0, 1]
```

```
In [20]: arr1 = np.array(data1)
```

```
In [21]: arr1
```

```
Out[21]: array([ 6. ,  7.5,  8. ,  0. ,  1. ])
```

- ndim, shape, dtype

```
In [27]: arr1.dtype
```

```
Out[27]: dtype('float64')
```

```
In [28]: arr2.dtype
```

```
Out[28]: dtype('int64')
```

```
In [22]: data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
```

```
In [23]: arr2 = np.array(data2)
```

```
In [24]: arr2
```

```
Out[24]:  
array([[1, 2, 3, 4],  
       [5, 6, 7, 8]])
```

```
In [25]: arr2.ndim
```

```
Out[25]: 2
```

```
In [26]: arr2.shape
```

```
Out[26]: (2, 4)
```

Creation of ndarrays (cont'd)

- All zeros, all ones, identity matrix, range, random matrix

```
In [29]: np.zeros(10)
```

```
Out[29]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

```
In [30]: np.zeros((3, 6))
```

```
Out[30]:
```

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

`arange` is an array-valued version of the built-in Python `range` function:

```
In [32]: np.arange(15)
```

```
Out[32]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

Creation of ndarrays (cont'd)

- All zeros, all ones, identity matrix, range, random matrix

```
In [129]: arr = np.random.randn(6, 3)
```

```
In [130]: arr
```

```
Out[130]:
```

```
array([[ -0.8608,  0.5601, -1.2659],  
       [ 0.1198, -1.0635,  0.3329],  
       [-2.3594, -0.1995, -1.542 ],  
       [-0.9707, -1.307 ,  0.2863],  
       [ 0.378 , -0.7539,  0.3313],  
       [ 1.3497,  0.0699,  0.2467]])
```

Table 4-1. Array creation functions

Function	Description
<code>array</code>	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
<code>asarray</code>	Convert input to ndarray, but do not copy if the input is already an ndarray
<code>arange</code>	Like the built-in <code>range</code> but returns an ndarray instead of a list
<code>ones</code> , <code>ones_like</code>	Produce an array of all 1s with the given shape and dtype; <code>ones_like</code> takes another array and produces a ones array of the same shape and dtype
<code>zeros</code> , <code>zeros_like</code>	Like <code>ones</code> and <code>ones_like</code> but producing arrays of 0s instead
<code>empty</code> , <code>empty_like</code>	Create new arrays by allocating new memory, but do not populate with any values like <code>ones</code> and <code>zeros</code>
<code>full</code> , <code>full_like</code>	Produce an array of the given shape and dtype with all values set to the indicated “fill value” <code>full_like</code> takes another array and produces a filled array of the same shape and dtype
<code>eye</code> , <code>identity</code>	Create a square $N \times N$ identity matrix (1s on the diagonal and 0s elsewhere)

Table 4-2. NumPy data types

Type	Type code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit (1 byte) integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 64-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point; compatible with C float
float64	f8 or d	Standard double-precision floating point; compatible with C double and Python float object
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers represented by two 32, 64, or 128 floats, respectively
bool	?	Boolean type storing True and False values
object	O	Python object type; a value can be any Python object
string_	S	Fixed-length ASCII string type (1 byte per character); for example, to create a string dtype with length 10, use 'S10'
unicode_	U	Fixed-length Unicode type (number of bytes platform specific); same specification semantics as string_ (e.g., 'U10')

Arithmetic

```
In [51]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])
```

```
In [52]: arr
```

```
Out[52]:
```

```
array([[ 1.,  2.,  3.],  
       [ 4.,  5.,  6.]])
```

```
In [53]: arr * arr
```

```
Out[53]:
```

```
array([[ 1.,  4.,  9.],  
       [16., 25., 36.]])
```

Element-wise multiplication
Note: not matrix multiplication

```
In [54]: arr - arr
```

```
Out[54]:
```

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

Arithmetic (cont'd)

```
In [55]: 1 / arr
```

```
Out[55]:
```

```
array([[ 1.      ,  0.5     ,  0.3333 ],  
       [ 0.25    ,  0.2     ,  0.1667 ]])
```

```
In [56]: arr ** 0.5
```

```
Out[56]:
```

```
array([[ 1.      ,  1.4142 ,  1.7321 ],  
       [ 2.      ,  2.2361 ,  2.4495 ]])
```


Arithmetic (cont'd)

```
In [57]: arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
```

```
In [58]: arr2
```

```
Out[58]:
```

```
array([[ 0.,  4.,  1.],  
       [ 7.,  2., 12.]])
```

```
In [59]: arr2 > arr
```

```
Out[59]:
```

```
array([[False,  True, False],  
       [ True, False,  True]], dtype=bool)
```

Indexing

```
In [60]: arr = np.arange(10)
```

```
In [61]: arr
```

```
Out[61]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [62]: arr[5]
```

```
Out[62]: 5
```

```
In [63]: arr[5:8]
```

```
Out[63]: array([5, 6, 7])
```

```
In [64]: arr[5:8] = 12
```

```
In [65]: arr
```

```
Out[65]: array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
```

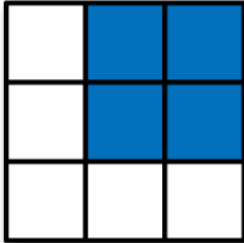
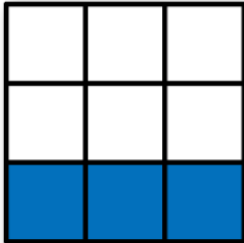
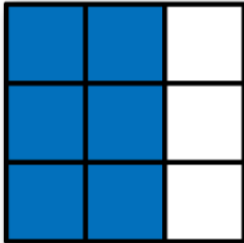
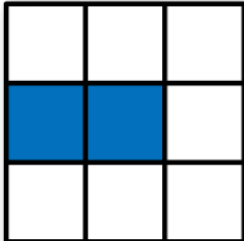
	Expression	Shape
	<code>arr[:2, 1:]</code>	<code>(2, 2)</code>
	<code>arr[2]</code> <code>arr[2, :]</code> <code>arr[2:, :]</code>	<code>(3,)</code> <code>(3,)</code> <code>(1, 3)</code>
	<code>arr[:, :2]</code>	<code>(3, 2)</code>
	<code>arr[1, :2]</code> <code>arr[1:2, :2]</code>	<code>(2,)</code> <code>(1, 2)</code>

Figure 4-2. Two-dimensional array slicing

Transposing (Stopped 10/31)

```
In [126]: arr = np.arange(15).reshape((3, 5))
```

```
In [127]: arr
```

```
Out[127]:
```

```
array([[ 0,  1,  2,  3,  4],  
       [ 5,  6,  7,  8,  9],  
       [10, 11, 12, 13, 14]])
```

```
In [128]: arr.T
```

```
Out[128]:
```

```
array([[ 0,  5, 10],  
       [ 1,  6, 11],  
       [ 2,  7, 12],  
       [ 3,  8, 13],  
       [ 4,  9, 14]])
```

Universal function (*ufunc*)

- A function that performs element-wise operations on data in ndarrays.

```
In [137]: arr = np.arange(10)
```

```
In [138]: arr
```

```
Out[138]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [139]: np.sqrt(arr)
```

```
Out[139]:
```

```
array([ 0.      ,  1.      ,  1.4142,  1.7321,  2.      ,  2.2361,  2.4495,  
        2.6458,  2.8284,  3.      ])
```

```
In [140]: np.exp(arr)
```

```
Out[140]:
```

```
array([  1.      ,  2.7183,  7.3891, 20.0855, 54.5982,  
        148.4132, 403.4288, 1096.6332, 2980.958 , 8103.0839])
```

Binary ufunc

```
In [141]: x = np.random.randn(8)
```

```
In [142]: y = np.random.randn(8)
```

```
In [143]: x
```

```
Out[143]:
```

```
array([-0.0119,  1.0048,  1.3272, -0.9193, -1.5491,  0.0222,  0.7584,  
       -0.6605])
```

```
In [144]: y
```

```
Out[144]:
```

```
array([ 0.8626, -0.01  ,  0.05  ,  0.6702,  0.853  , -0.9559, -0.0235,  
       -2.3042])
```

```
In [145]: np.maximum(x, y)
```

```
Out[145]:
```

```
array([ 0.8626,  1.0048,  1.3272,  0.6702,  0.853  ,  0.0222,  0.7584,  
       -0.6605])
```

Binary ufunc (cont'd)

```
In [146]: arr = np.random.randn(7) * 5
```

```
In [147]: arr
```

```
Out[147]: array([-3.2623, -6.0915, -6.663 ,  5.3731,  3.6182,  3.45  ,  5.0077])
```

```
In [148]: remainder, whole_part = np.modf(arr)
```

```
In [149]: remainder
```

```
Out[149]: array([-0.2623, -0.0915, -0.663 ,  0.3731,  0.6182,  0.45  ,  0.0077])
```

```
In [150]: whole_part
```

```
Out[150]: array([-3., -6., -6.,  5.,  3.,  3.,  5.])
```

Table 4-3. Unary ufuncs

Function	Description
abs, fabs	Compute the absolute value element-wise for integer, floating-point, or complex values
sqrt	Compute the square root of each element (equivalent to <code>arr ** 0.5</code>)
square	Compute the square of each element (equivalent to <code>arr ** 2</code>)
exp	Compute the exponent e^x of each element
log, log10, log2, log1p	Natural logarithm (base e), log base 10, log base 2, and $\log(1 + x)$, respectively
sign	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
ceil	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
floor	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
rint	Round elements to the nearest integer, preserving the dtype
modf	Return fractional and integral parts of array as a separate array
isnan	Return boolean array indicating whether each value is NaN (Not a Number)
isfinite, isinf	Return boolean array indicating whether each element is finite (non- <code>inf</code> , non-NaN) or infinite, respectively
cos, cosh, sin, sinh, tan, tanh	Regular and hyperbolic trigonometric functions
arccos, arccosh, arcsin, arcsinh, arctan, arctanh	Inverse trigonometric functions
logical_not	Compute truth value of <code>not</code> x element-wise (equivalent to <code>~arr</code>).

Table 4-4. Binary universal functions

Function	Description
<code>add</code>	Add corresponding elements in arrays
<code>subtract</code>	Subtract elements in second array from first array
<code>multiply</code>	Multiply array elements
<code>divide</code> , <code>floor_divide</code>	Divide or floor divide (truncating the remainder)
<code>power</code>	Raise elements in first array to powers indicated in second array
<code>maximum</code> , <code>fmax</code>	Element-wise maximum; <code>fmax</code> ignores NaN
<code>minimum</code> , <code>fmin</code>	Element-wise minimum; <code>fmin</code> ignores NaN
<code>mod</code>	Element-wise modulus (remainder of division)
<code>copysign</code>	Copy sign of values in second argument to values in first argument
<code>greater</code> , <code>greater_equal</code> , <code>less</code> , <code>less_equal</code> , <code>equal</code> , <code>not_equal</code>	Perform element-wise comparison, yielding boolean array (equivalent to infix operators <code>></code> , <code>>=</code> , <code><</code> , <code><=</code> , <code>==</code> , <code>!=</code>)
<code>logical_and</code> , <code>logical_or</code> , <code>logical_xor</code>	Compute element-wise truth value of logical operation (equivalent to infix operators <code>&</code> , <code> </code> , <code>^</code>)

Matrix multiplication

```
In [223]: x = np.array([[1., 2., 3.], [4., 5., 6.]])
```

```
In [224]: y = np.array([[6., 23.], [-1, 7], [8, 9]])
```

```
In [225]: x
```

```
Out[225]:  
array([[ 1.,  2.,  3.],  
       [ 4.,  5.,  6.]])
```

```
In [226]: y
```

```
Out[226]:  
array([[ 6., 23.],  
       [-1.,  7.],  
       [ 8.,  9.]])
```

```
In [227]: x.dot(y)
```

```
Out[227]:  
array([[ 28.,  64.],  
       [ 67., 181.]])
```

`x.dot(y)` is equivalent to `np.dot(x, y)`:

```
In [228]: np.dot(x, y)
```

```
Out[228]:  
array([[ 28.,  64.],  
       [ 67., 181.]])
```

- Python tends to abuse the operator.
- May be different from the linear algebra meaning.
- Recommend to use `matmul` instead.
- Check out details in

<https://numpy.org/doc/stable/reference/generated/numpy.dot.html>

More about linear algebra

Table 4-7. Commonly used numpy.linalg functions

Function	Description
<code>diag</code>	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
<code>dot</code>	Matrix multiplication
<code>trace</code>	Compute the sum of the diagonal elements
<code>det</code>	Compute the matrix determinant
<code>eig</code>	Compute the eigenvalues and eigenvectors of a square matrix
<code>inv</code>	Compute the inverse of a square matrix
<code>pinv</code>	Compute the Moore-Penrose pseudo-inverse of a matrix
<code>qr</code>	Compute the QR decomposition
<code>svd</code>	Compute the singular value decomposition (SVD)
<code>solve</code>	Solve the linear system $Ax = b$ for x , where A is a square matrix
<code>lstsq</code>	Compute the least-squares solution to $Ax = b$

Pseudorandom

```
In [238]: samples = np.random.normal(size=(4, 4))
```

```
In [239]: samples
```

```
Out[239]:
```

```
array([[ 0.5732,  0.1933,  0.4429,  1.2796],  
       [ 0.575 ,  0.4339, -0.7658, -1.237 ],  
       [-0.5367,  1.8545, -0.92  , -0.1082],  
       [ 0.1525,  0.9435, -1.0953, -0.144 ]])
```

Pseudorandom

- Motivation -- efficiency

```
In [240]: from random import normalvariate
```

```
In [241]: N = 1000000
```

```
In [242]: %timeit samples = [normalvariate(0, 1) for _ in range(N)]  
1.77 s +- 126 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

```
In [243]: %timeit np.random.normal(size=N)  
61.7 ms +- 1.32 ms per loop (mean +- std. dev. of 7 runs, 10 loops each)
```

Pseudorandom

- Same seed -- same randomized sequence

```
In [244]: np.random.seed(1234)
```

The data generation functions in `numpy.random` use a global random seed. To avoid global state, you can use `numpy.random.RandomState` to create a random number generator isolated from others:

```
In [245]: rng = np.random.RandomState(1234)
```

```
In [246]: rng.randn(10)
```

```
Out[246]:
```

```
array([ 0.4714, -1.191 ,  1.4327, -0.3127, -0.7206,  0.8872,  0.8596,  
       -0.6365,  0.0157, -2.2427])
```

More about data analysis

- Pandas
- Visualization

Achieve Computations without Loops

- Loop is very expensive in matlab/python
 - Avoid using it as much as possible
 - In python:
 - numpy matrix operations, list comprehension
 - <http://www.jesshamrick.com/2012/04/29/the-demise-of-for-loops/>
 - <http://codereview.stackexchange.com/questions/38580/fastest-way-to-iterate-over-numpy-array>

Code Demo: Shannon entropy computation

- Entropy: the optimal code length to encode a distribution

	M	F
bk	7	9
gr	11	2
bl	3	2

Count

	M	F
bk	7/34	9/34
gr	11/34	2/34
bl	3/34	2/34

Probability

- Total = 7 + 9 + 11 + 2 + 3 + 2 = 34

- $P(\text{Gender} = \text{M}, \text{Eye Color} = \text{bk}) = 7 / 34$

- Entropy:

$$H(P) = - P(\text{M}, \text{bk}) \log_2 P(\text{M}, \text{bk}) - P(\text{F}, \text{bk}) \log_2 P(\text{F}, \text{bk})$$

-

$$= \sum_{i,j} -P_{ij} \log_2 P_{ij}$$

$$= -(7/34) \log (7/34) - (9/34) \log (9/34) - ...$$

The end