Homework #9 is due Monday, Oct. 31.
Homework #10 is due Monday, Nov. 7. (I’ll post soon.)
Midterm #2 is Monday, Nov. 14 from 7–8 p.m.
x = np.zeros((3,3))
for i in range(3):
    for j in range(3):
        x[i,j] = i*j + j

A = 
\[
\begin{pmatrix}
0 & 0 & 0 \\
1 & 2 & 3 \\
2 & 4 & 6 \\
\end{pmatrix}
\]

B = 
\[
\begin{pmatrix}
0 & 0 & 0 \\
0 & 2 & 4 \\
0 & 4 & 8 \\
\end{pmatrix}
\]

C = 
\[
\begin{pmatrix}
0 & 1 & 2 \\
0 & 2 & 4 \\
0 & 3 & 6 \\
\end{pmatrix}
\]
Question #1

```python
x = np.zeros((3,3))
for i in range(3):
    for j in range(3):
        x[i,j] = i*j + j
```

\[
A = \begin{pmatrix}
0 & 0 & 0 \\
1 & 2 & 3 \\
2 & 4 & 6
\end{pmatrix}
\quad B = \begin{pmatrix}
0 & 0 & 0 \\
0 & 2 & 4 \\
0 & 4 & 8
\end{pmatrix}
\quad C = \begin{pmatrix}
0 & 1 & 2 \\
0 & 2 & 4 \\
0 & 3 & 6
\end{pmatrix}^*
\]
Randomness
A philosophical excursus: what is randomness?
A philosophical excursus: what is randomness?
What are some sources of true randomness?
A philosophical excursus: what is randomness?  
What are some sources of true randomness?  
Consider the following two sequences:

\[
\begin{align*}
7 & \quad 8 & \quad 5 & \quad 3 & \quad 9 & \quad 8 & \quad 1 & \quad 6 & \quad 3 & \quad 3 & \quad 9 & \quad 7 & \quad 4 & \quad 4 & \quad 8 & \quad 3 & \quad 0 & \quad 9 & \quad 6 & \quad 1 & \quad 5 & \quad 6 & \quad 6 & \quad 0 & \quad 8 & \quad 4 & \quad \ldots
\end{align*}
\]

\[
\begin{align*}
+1, & -\frac{1}{3}, +\frac{1}{5}, -\frac{1}{7}, -\frac{1}{9}, -\frac{1}{11}, +\frac{1}{13}, -\frac{1}{15}, \ldots
\end{align*}
\]
A philosophical excursus: what is randomness?
What are some sources of true randomness?
Consider the following two sequences:

7 8 5 3 9 8 1 6 3 3 9 7 4 4 8 3 0 9 6 1 5 6 6 0 8 4 ...

+1, −1/3, +1/5, −1/7, −1/9, −1/11, +1/13, −1/15, ...

These are derived from the same rule (π/4)—but one seems “random” to us.
Pseudorandom numbers come from computer formulae.
Randomness

- Pseudorandom numbers come from computer formulae.
- The formula uses a seed (often the system clock time) to start the sequence.
Pseudorandom numbers come from computer formulae.

The formula uses a seed (often the system clock time) to start the sequence.

It then returns a new number unpredictable to you (but predictable to the formula!) each time you query the function.
Pseudorandom numbers come from computer formulae. The formula uses a seed (often the system clock time) to start the sequence. It then returns a new number unpredictable to you (but predictable to the formula!) each time you query the function. NumPy uses the Mersenne twister, based on prime number distributions (but you don’t need to know this).
Randomness

- Pseudorandom numbers come from computer formulae.
- The formula uses a seed (often the system clock time) to start the sequence.
- It then returns a new number unpredictable to you (but predictable to the formula!) each time you query the function.
- NumPy uses the Mersenne twister, based on prime number distributions (but you don’t need to know this).
- Dozens of distributions are available—let’s see a few.
randint

- randint returns a random (pseudorandom) integer in a range (which works the same as range).

np.random.randint( 10 )  # random int, [0,10)
randint

randint returns a random (pseudorandom) integer in a range (which works the same as range).

```python
np.random.randint( 10 )  # random int, [0,10)
np.random.randint( 1,7 )  # random int, [1, 7)
```
randint

- randint returns a random (pseudorandom) integer in a range (which works the same as range).

```python
np.random.randint(10)  # random int, [0,10)
np.random.randint(1,7)  # random int, [1, 7)
np.random.randint(0,10, size=(5,5))  # in array
```
hist (MatPlotLib) creates a histogram.

Histograms plot the number of times a value occurs in a data set.
hist

- hist (MatPlotLib) creates a histogram.
- Histograms plot the number of times a value occurs in a data set.

```python
x = np.random.randint(0,100,size=(10000,1))
plt.hist(x)
plt.show()
```
- `hist` (MatPlotLib) creates a histogram.
- Histograms plot the number of times a value occurs in a data set.

```python
x = np.random.randint(0,100,size=(10000,1))
plt.hist(x)
plt.show()
```
Number guessing (a game for the easily entertained):

```python
import numpy as np
number = np.random.randint(10)+1
guess = input( 'Guess the number between 1 and 10:' )
while guess != number:
    guess = input( 'Nope. Try again:' )
print( 'You did it. Hooray.' )
```
Number guessing (a game for the easily entertained):

```python
import numpy as np
number = np.random.randint(10)+1
guess = input( 'Guess the number between 1 and 10:' )
while int( guess ) != number:
    guess = input( 'Nope. Try again:' )
print( 'You did it. Hooray.' )
```
uniform

- uniform returns a random float in the range [0,1).

```python
np.random.uniform()  # random number, [0,1)
```
uniform

- uniform returns a random float in the range [0, 1).

```python
np.random.uniform()  # random number, [0,1)
np.random.uniform(size=(4,3))  # in array
```
uniform

uniform returns a random float in the range [0, 1).

```python
np.random.uniform()       # random number, [0,1)
np.random.uniform( size=(4,3) ) # in array
x = np.random.uniform( size=(10000,1) )
plt.hist(x)
plt.show()
```
uniform

- uniform returns a random float in the range [0,1).

```python
np.random.uniform()       # random number, [0,1)
np.random.uniform( size=(4,3) )  # in array
x = np.random.uniform( size=(10000,1) )
plt.hist(x)
plt.show()
```
uniform

- uniform returns a random float in the range $[0, 1)$.

```python
np.random.uniform()  # random number, [0,1)
np.random.uniform(size=(4,3))  # in array
x = np.random.uniform(size=(10000,1))
plt.hist(x, bins=100)
plt.show()
```
Uniform returns a random float in the range $[0, 1)$.

```python
np.random.uniform()  # random number, [0,1)
np.random.uniform(size=(4,3))  # in array
x = np.random.uniform(size=(10000,1))
plt.hist(x, bins=100)
plt.show()
```
**randn**

- **randn** returns a random number selected from the **normal** distribution with mean 0 and variance 1.
- (Variance is the square of standard deviation.)
randn returns a random number selected from the normal distribution with mean 0 and variance 1.
(Variance is the square of standard deviation.)

```python
np.random.randn()  # random normal number
```

```python
np.random.randn() + 1.0  # mean 1.0
```

```python
(np.random.randn()) * 4  # variance 4.0
```
• `randn` returns a random number selected from the normal distribution with mean 0 and variance 1.
  (Variance is the square of standard deviation.)

```python
np.random.randn()  # random normal number
np.random.randn() + 1.0  # mean 1.0
```
**randn**

- **randn** returns a random number selected from the normal distribution with mean 0 and variance 1.
- (Variance is the square of standard deviation.)

```python
np.random.randn()  # random normal number
np.random.randn() + 1.0  # mean 1.0
(np.random.randn()) * 4  # variance 4.0
```
`randn` returns a random number selected from the normal distribution with mean 0 and variance 1.

```python
x = np.random.randn(10000)
plt.hist(x,bins=20)
plt.show()
```
\textit{randn} returns a random number selected from the normal distribution with mean 0 and variance 1.

\begin{verbatim}
x = np.random.randn(10000)
plt.hist(x,bins=20)
plt.show()
\end{verbatim}
choice randomly samples a one-dimensional array (rather, the first dimension of the array).

```python
x = [ 'red', 'orange', 'yellow', 'green', 'blue' ]
np.random.choice(x)  # random color
```
choice randomly samples a one-dimensional array but can do so without replacement.

```python
x = np.arange(1,53)
c = np.random.choice(x, size=5, replace=False)
```

The foregoing code draws five cards from a deck (no repeat cards allowed).
choice randomly samples a one-dimensional array but can do so without replacement.

Replacement means the difference between pulling a card from a deck and putting it back before drawing again (or not).
choice randomly samples a one-dimensional array but can do so without replacement.

Replacement means the difference between pulling a card from a deck and putting it back before drawing again (or not).

```python
x = np.arange(1,53)
c = np.random.choice( x, size=5, replace=False )
```
choice randomly samples a one-dimensional array but can do so without replacement.

Replacement means the difference between pulling a card from a deck and putting it back before drawing again (or not).

```python
import numpy as np
x = np.arange(1,53)
c = np.random.choice(x, size=5, replace=False)
```

The foregoing code draws five cards from a deck (no repeat cards allowed).
shuffle randomly reorders an array in place.

What is its return type?
shuffle randomly reorders an array in place.

What is its return type?

```python
x = np.arange(1, 53)
np.random.shuffle(x)
```
shuffle randomly reorders an array in place.
What is its return type?

```python
x = np.arange(1, 53)
np.random.shuffle(x)
```

The foregoing code shuffles a deck of cards.
Which of the following will **not** reproduce the behavior of a six-sided die in `c`?

A  \[ c = \text{np.random.randn}(6) + 1 \]

B  \[ x = \text{np.arange}(1,7) \\
\quad c = \text{np.random.choice}(x) \]

C  \[ c = \text{np.random.randint}(6)+1 \]

D  \[ d = \text{np.random.uniform}(0,6) \\
\quad c = \text{int}(d) + 1 \]
Which of the following will not reproduce the behavior of a six-sided die in c?

A  \[ c = \text{np.random.randn}(6) + 1 \]

B  \[ x = \text{np.arange}(1,7) \]
    \[ c = \text{np.random.choice}(x) \]

C  \[ c = \text{np.random.randint}(6)+1 \]

D  \[ d = \text{np.random.uniform}(0,6) \]
    \[ c = \text{int}(d) + 1 \]
Our first toy example was pretty lame. What else can we do?
Our first toy example was pretty lame. What else can we do?
Example: Mad Libs
import numpy as np

adjs = []
for line in open('adjectives.txt').readlines():
    adjs.append( line.strip() )

names = []
for line in open('names.txt').readlines():
    names.append( line.strip().split(',', ',') )

verbs = []
for line in open('verbs.txt').readlines():
    verbs.append( line.strip().split(',', ',') )

nouns = []
for line in open('nouns.txt').readlines():
    nouns.append( line.strip() )

# note that names and verbs have a slightly different structure than adj and nouns
adj1 = adjs[np.random.randint(len(adjs))]
noun1 = nouns[np.random.randint(len(nouns))]
name = names[np.random.randint(len(names))]
verb = verbs[np.random.randint(len(verbs))]
adj2 = adjs[np.random.randint(len(adjs))]
noun2 = nouns[np.random.randint(len(nouns))]

phrase = adj1.title() + ' ' + noun1 + ' ' + name[0] + ' was so ' + adj2 + ' that ' + name[1] + ' ' + verb[1] + ' a ' + noun2 + '.'
Our first toy example was pretty lame. What else can we do?
- Example: Mad Libs
- Random walk
import numpy as np
import matplotlib.pyplot as plt

x = np.zeros((100,1))
y = np.zeros((100,1))
for i in range(1, len(x)):
    dir = np.random.randint(4)
    if dir == 0:
        x[i] = x[i-1]
        y[i] = y[i-1] + 1
    if dir == 1:
        x[i] = x[i-1] + 1
        y[i] = y[i-1]
    if dir == 2:
        x[i] = x[i-1]
        y[i] = y[i-1] - 1
    if dir == 3:
        x[i] = x[i-1] - 1
        y[i] = y[i-1]

plt.plot(x, y)
plt.show()
Our first toy example was pretty lame. What else can we do?
- Example: Mad Libs
- Random walk
- Think of others: games, for instance.
- Also, scientific applications (quantum mechanics).