Numerical Python

randomness

CS101 Lecture #17

2016-11-21

Administrivia

Administrivia

- Homework #7 will be out later this week.

Administrivia

- Homework #7 will be out later this week.
- Midterm #2 is Next Thursday, Nov. 30 (in class) .

Warmup Quiz

Question #1

```
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

$$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 4 & 8 \end{pmatrix}$$

C

$$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 3 & 6 \end{pmatrix}$$

Question #1

```
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

$$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 4 & 8 \end{pmatrix}$$

C

$$\begin{pmatrix} 0 & 1 & 2 \\ 0 & 2 & 4 \\ 0 & 3 & 6 \end{pmatrix} \star$$

Randomness

Randomness

- A philosophical excursus: what is randomness?
- What are some sources of genuine randomness?

Randomness

- A philosophical excursus: what is randomness?
- What are some sources of genuine randomness?
- Consider the following two sequences:

$$7\,8\,5\,3\,9\,8\,1\,6\,3\,3\,9\,7\,4\,4\,8\,3\,0\,9\,6\,1\,5\,6\,6\,0\,8\,4\,\ldots$$

$$+1, -\frac{1}{3}, +\frac{1}{5}, -\frac{1}{7}, -\frac{1}{9}, -\frac{1}{11}, +\frac{1}{13}, -\frac{1}{15}, \cdots$$

Randomness

- A philosophical excursus: what is randomness?
- What are some sources of genuine randomness?
- Consider the following two sequences:

$$7\,8\,5\,3\,9\,8\,1\,6\,3\,3\,9\,7\,4\,4\,8\,3\,0\,9\,6\,1\,5\,6\,6\,0\,8\,4\ldots$$

$$+1, -\frac{1}{3}, +\frac{1}{5}, -\frac{1}{7}, -\frac{1}{9}, -\frac{1}{11}, +\frac{1}{13}, -\frac{1}{15}, \cdots$$

- These are derived from the same rule $(\pi/4)$—but only one seems "random" to us.

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.
- It is designed to draw random numbers following some distribution.
- Dozens of distributions are available—let's see a few.

randint

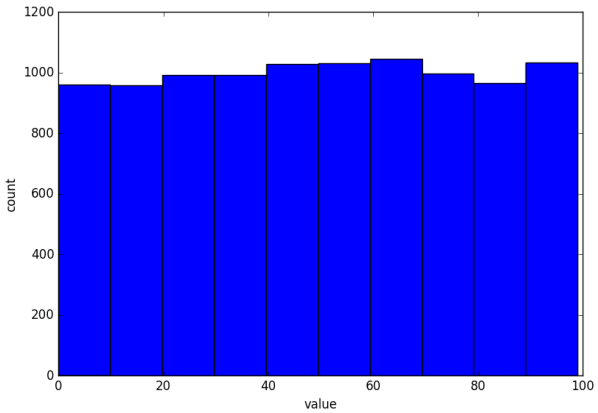
- `randint` returns a random (pseudorandom) integer in a range (which works the same as `range`).

```
np.random.randint(10)  # random int, [0,10)
np.random.randint(1,7) # random int, [1, 7)
np.random.randint(0,10,size=(5,5)) # in array
```

hist

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

```
x = np.random.randint(0,100,size=(10000,1))
plt.hist(x)
plt.show()
```



Example

- Number guessing (a game for the easily entertained):

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

Example

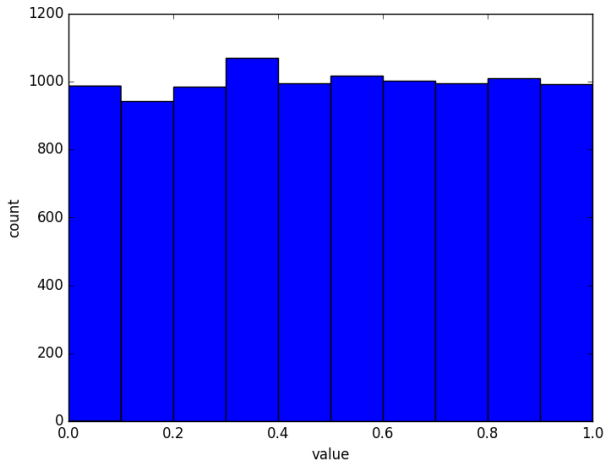
- Number guessing (a game for the easily entertained):

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

uniform

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

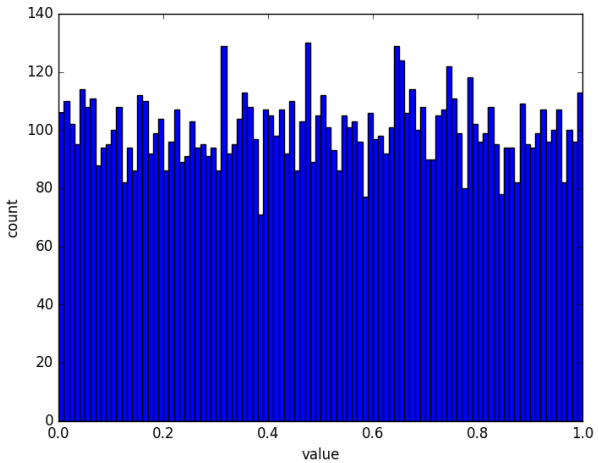
```
np.random.uniform()  # random number, [0,1)
np.random.uniform( size=(4,3) ) # in array
x = np.random.uniform( size=(10000,1) )
plt.hist(x)
plt.show()
```



uniform

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

```
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.

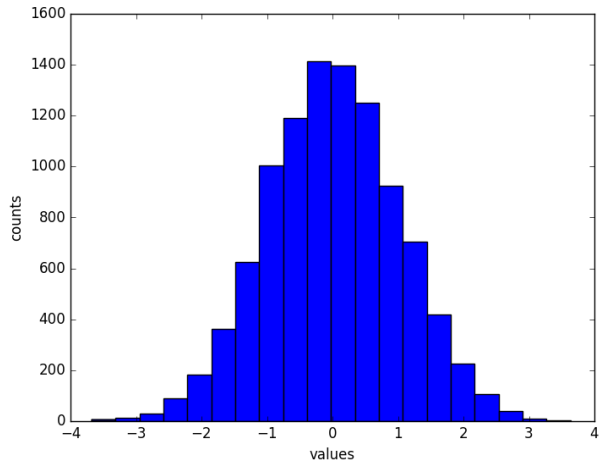
```
np.random.randn()     # random normal number
np.random.randn() + 1.0  # mean 1.0
(np.random.randn()) * 4  # std 4.0
```

- how to create a normal distribution of mean -10 and standard deviation 0.2?

randn

- `randn` returns a random number selected from the *normal* distribution with mean 0 and variance 1.

```
x = np.random.randn( 10000 )
plt.hist(x,bins=20)
plt.show()
```



choice

- `choice` randomly samples from a one dimensional *array* or *list* (rather than the first dimension of the array).

```
x = ['red','orange','yellow','green','blue']
np.random.choice(x) # samples a color from x
```

choice

- `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).

```
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

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

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

- The foregoing code shuffles a deck of cards.

Question

Which of the following will *not* reproduce the behavior of a six-sided die?

A
```
d = np.random.randn( 6 ) + 1
```

B
```
x = np.arange( 1,7 )
d = np.random.choice( x )
```

C
```
d = np.random.randint( 6 )+1
```

D
```
x = np.random.uniform( 0,6 )
d = int(d) + 1
```

Question

Which of the following will *not* reproduce the behavior of a six-sided die?

A
```
d = np.random.randn( 6 ) + 1
```

B
```
x = np.arange( 1,7 )
d = np.random.choice( x )
```

C
```
d = np.random.randint( 6 )+1
```

D
```
x = np.random.uniform( 0,6 )
d = int(d) + 1
```

More example

- Our first toy example was pretty lame. What else can we do?
- Example: Sentence Generator

Sentence generator

```
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 names and verbs have a slightly different structure
# than adj and nouns
```

Sentence generator

```
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 + '.'
```

Random walk

```
import numpy as np
import matplotlib.pyplot as plt

x = np.zeros( ( 100,1 ) )
y = np.zeros( ( 100,1 ) )
```

Random walk

```
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()
```