Numerical Python

optimization

CS101 Lecture #18

2016-11-23

Randomness Refresher

Randomness refresher

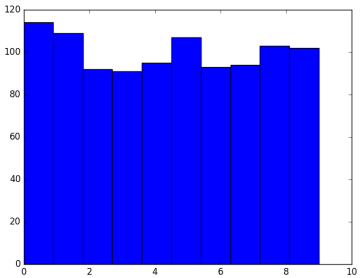
- `randint( start,end,size=tuple )`
- `uniform( start,end,size=tuple )`
- `randn(d1,d2,..)` #specify size of output
- Note that the interfaces for each are slightly different.

Question #1

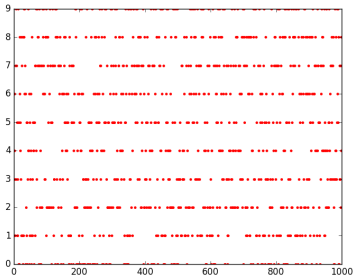
```
x = np.random.randint( 0,10, size=(1000,1) )
plt.hist( x )
plt.show()
```

What is a possible output of this code?

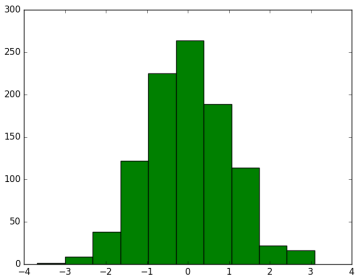
A



B



C

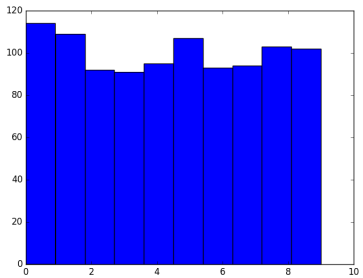


Question #1

```
x = np.random.randint( 0,10, size=(1000,1) )
plt.hist( x )
plt.show()
```

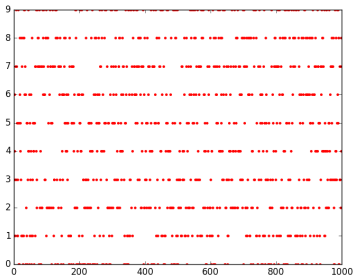
What is a possible output of this code?

A

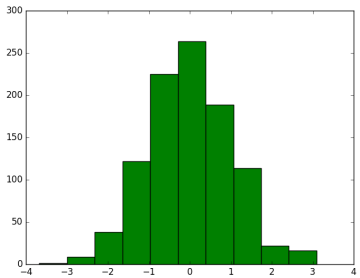


★

B



C

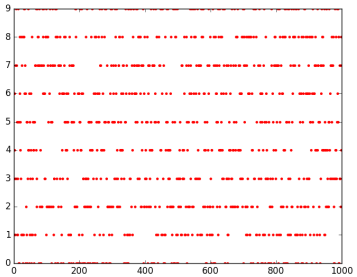


Question #2

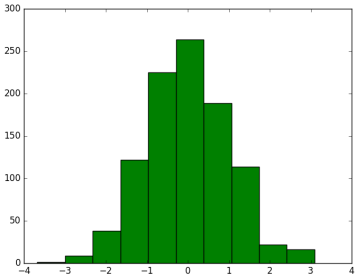
```
x = np.random.uniform( size=(1000,1) )
plt.plot( x, 'c.' )
plt.ylim( (-1,2) )
plt.show()
```

What is a possible output of this code?

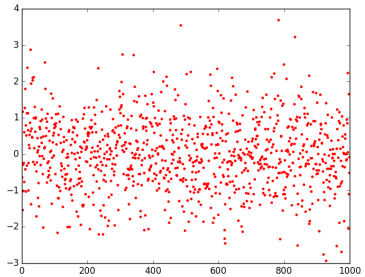
A



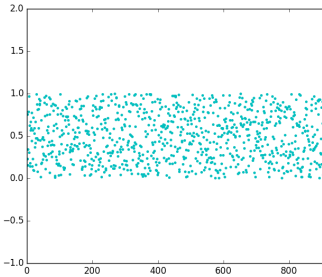
B



C



D

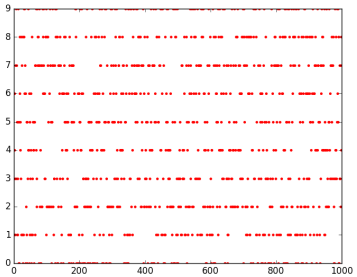


Question #2

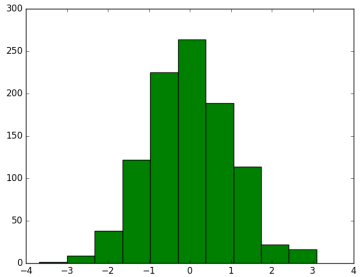
```
x = np.random.uniform( size=(1000,1) )
plt.plot( x, 'c.' )
plt.ylim( (-1,2) )
plt.show()
```

What is a possible output of this code?

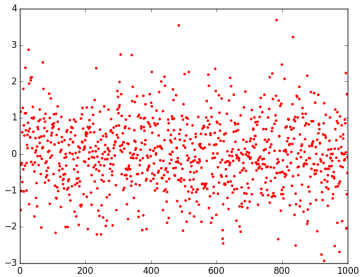
A



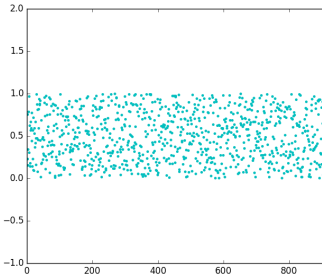
B



C



D



★

http://matplotlib.org/api/colors_api.html

Optimization

Optimization

On vacation, you purchase a range of n souvenirs of varying weight and value. When it comes time to pack, you find that your bag has a weight limit of 50 pounds. What is the best set of items to take on the flight?

Optimization

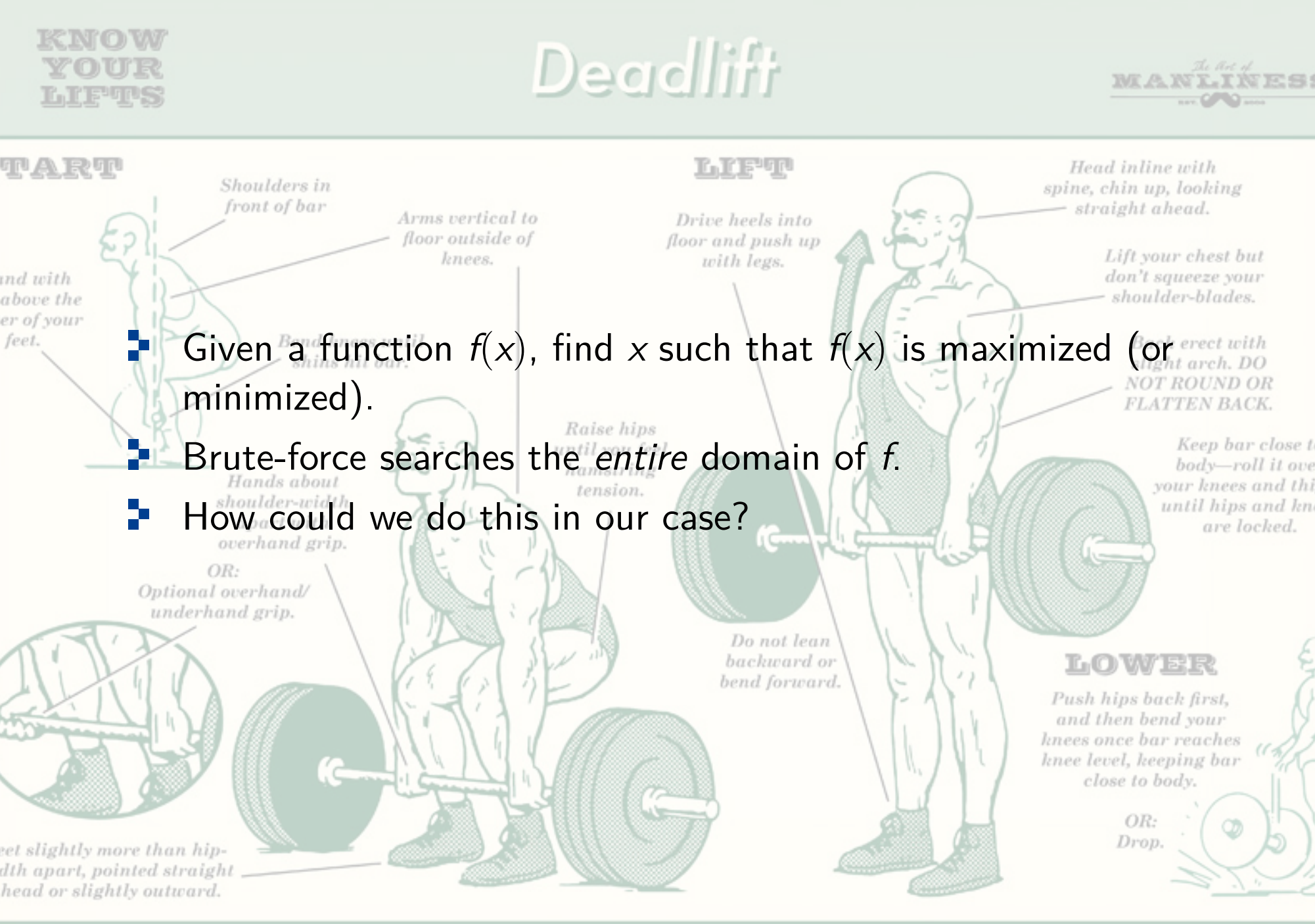
- Given a function $f(x)$, find x such that $f(x)$ is maximized (or minimized).
- The goal is to search the domain for the optimal x yielding the optimal $f(x)$.
- Many clever techniques exist, but we'll start with a naïve approach.

Setup

```
import numpy as np

n = 10
items   = list(range(n))
weights = np.random.uniform( size=(n,1) )*50
values  = np.random.uniform( size=(n,1) )*100
```

Setup

```
def f( wts, vals ):
    total_weight = 0
    total_value = 0

    for i in range( len( wts ) ):
        total_weight += wts[ i ]
        total_value  += vals[ i ]

    if total_weight >= 50:
        return 0
    else:
        return total_value
```



- Given a function $f(x)$, find x such that $f(x)$ is maximized (or minimized).
- Brute-force searches the *entire* domain of f.
- How could we do this in our case?

Optimization

- Two useful functions from `itertools` to keep in mind:
  - `combinations`: provide all subsets of size `n`.
  - `product`: replace nested `for` loops.

Optimization

- `combinations`: provide all subsets of size `n`.

```
import itertools

a = [ 1,2,3,4 ]
for x in itertools.combinations( a,2 ):
    print( x )
```

Optimization

- `product`: replace nested `for` loops.
- Can use `repeat=n` argument as well.

```
import itertools

a = [ 1,2,3,4 ]
b = [ 'g','h','i' ]
for x in itertools.product( a,b ):
    print( x )
for x in itertools.product( a, repeat=3 ):
    print( x )
```

Question #3

```
x = 'ABCD'
z = 'XYZ'

for a in itertools.product( x,y ):
    print( ' '.join( a ) )
```

Which of the following is *not* printed?

A `'A X'`

B `'B D'`

C `'C X'`

D `'D Z'`

Question #4

```
x = 'ABCD'
z = 'XYZ'

for a in itertools.product( x,y ):
    print( ' '.join( a ) )
```

Which of the following is *not* printed?

A `'A X'`

B `'B D'` ⋆

C `'C X'`

D `'D Z'`

Setup

```
import itertools

max_value = 0.0
max_set = None
for i in range(1,n+1):
    for set in itertools.combinations( items,i ):
        wts  = []
        vals = []
        for item in set:
            wts.append( weights[ item ] )
            vals.append( values[ item ] )
        value = f( wts,vals )
        if value > max_value:
            max_value = value
            max_set = set
```

Optimization

- Brute-force search of a password:

```
def check_password( pwd ):
    if pwd == 'pas':
        return True
    else:
        return False

chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'+\
        'abcdefghijklmnopqrstuvwxyz0123456789'
for pair in itertools.product( chars, repeat=3 ):
    pair = ''.join( pair )
    if check_password( pair ):
        print( pair )
```

Optimization

- Brute-force search of a password:

$$\begin{aligned} & 2 \times n(\text{alphabet}) + n(\text{digits}) + n(\text{special}) \\ = \; & 2 \times 26 + 10 + \{24\text{–}32\} \\ = \; & \{86\text{–}94\} \end{aligned}$$

per letter! This gets very big very quickly!