Administrivia
Homework #10 is due Friday, Nov. 11.
Midterm #2 is Monday, Nov. 14 from 7–8 p.m.
Warmup Quiz
def fact( n ):
    if n <= 1:
        return 1
    else:
        ???

Which line of code correctly makes fact return the factorial $n$!?

A return fact( n - 1 ) * fact( n )
B return fact( n - 1 ) * n
C return ( n - 1 ) * n
D return fact( n - 2 ) * n
def fact( n ):  
    if n <= 1:  
        return 1  
    else:  
        ???

Which line of code correctly makes fact return the factorial n!?

A return fact( n - 1 ) * fact( n )
B return fact( n - 1 ) * n ⋆
C return ( n - 1 ) * n
D return fact( n - 2 ) * n
Randomness Refresher
- `randint( start,end,size=tuple )`
- `uniform( start,end,size=tuple )`
- `randn( d0,d1,d2,... )`
- Note that the interfaces for each are slightly different.
```python
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  

---

Randomness Refresher
Question #2

\[ x = \text{np.random.randint( 0,10, size=(1000,1) )} \]
\[ \text{plt.hist( } x \text{ )} \]
\[ \text{plt.show()} \]

What is a possible output of this code?

A

B

C

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

```python
import itertools

a = [1, 2, 3, 4]
for x in itertools.combinations(a, 2):
    print(x)
```
- **product**: replace nested for loops.
- Can use `repeat=n` argument as well.

```python
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)
```
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'
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'
import itertools

max_value = 0.0
max_set = None
for i in range(n):
    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
Brute-force search of a password:

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

chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789'
for pair in itertools.product( chars, repeat=3 ) :
    pair = ''.join( pair )
    if check_password( pair ) :
        print( pair )
```
Brute-force search of a password:

\[
2 \times n(\text{alphabet}) + n(\text{digits}) + n(\text{special}) \\
= 2 \times 26 + 10 + \{24-32\} \\
= \{86-94\}
\]

per letter! This gets very big very quickly!
A note on HW10

- In hw10 we run many simulations.
- A good approach:
  - Create a 2D array for the state variables.
  - Each row tracks a different simulation (angle).
  - Each column tracks one time step.
  - (You can transpose these as well, but be consistent.)
# Parameters of simulation
n = 1000  # number of data points to plot
m = 20  # number of balls to drop
start = 0.0  # start time of simulation
d = 2.0  # ending time of simulation
g = -9.8  # acceleration of gravity

# State variable initialization
t = np.linspace(start,end,n+1)  # time in seconds
y = np.zeros((m,n+1),dtype=np.float64)  # height in meters
v = np.zeros((m,n+1),dtype=np.float64)  # velocity in m/s

for i in range(m):
    y[i][0]=i+1
for i in range(m): # ball number
    for j in range(1,n+1): # time number
        if y[i][j-1]>0:
            y[i,j] = y[i,j-1] + v[i,j-1] * (t[j]-t[j-1])
            v[i,j] = v[i,j-1] + g * (t[j]-t[j-1])
        else:
            y[i,j] = 0
            v[i,j] = 0

plt.plot( y.transpose() )
plt.show()