Solving Equations in $x$
We can also find minima using `scipy.optimize.fmin( f, x0 )`. 

```python
def f( x ):
    return x ** 2 - x

import scipy.optimize
scipy.optimize.fmin( f, 1.0 )
```
We can also find minima using `scipy.optimize.fmin(f, x0)`.

This requires you to be clever in preparing f: you may have to negate or manipulate your function.

```python
def f(x):
    return x ** 2 - x

import scipy.optimize
scipy.optimize.fmin(f, 1.0)
```
Another common application is to find a derivative of a function $f$. Which of the following approximates a derivative $f'(x)$ at a point $x$ using a finite-difference approximation?

A  def dfdx( f,x,h=1e-5 ):
   return f( x+h ) - f( x ) / h

B  def dfdx( f,x,h=1e-5 ):
   return (( f( x ) + h ) - f( x ) ) / h

C  def dfdx( f,x,h=1e-5 ):
   return ( f( x+h ) - f( x ) ) / h
Which of the following approximates a derivative \( f'(x) \) at a point \( x \)?

A  \[
def \text{dfdx}( f,x,h=1e-5 ):
    \quad \text{return } f( x+h ) - f( x ) / h
\]

B  \[
def \text{dfdx}( f,x,h=1e-5 ):
    \quad \text{return } ( ( f( x ) + h ) - f( x ) ) / h
\]

C  \[
def \text{dfdx}( f,x,h=1e-5 ):
    \quad \text{return } ( f( x+h ) - f( x ) ) / h
\]

⋆
Optimization
On vacation, you purchase a collection 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 $x = x^*$ such that $f(x^*)$ is maximized (or minimized).

The goal is to search the domain for the $x^*$ which yields the optimal $f(x^*)$.

Many clever techniques exist, but we’ll start with a naïve approach.
import numpy as np
np.random.seed(101)

n = 10
items = list(range(n))
weights = np.random.uniform(size=(n,)) * 50
values = np.random.uniform(size=(n,)) * 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 $x = x^*$ 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 the ` itertools ` module:
- ` combinations `: provide all subsets of size ` n `.  
- ` product `: replace nested for loops.
Optimization

- 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)
```
Question #4

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

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

Which of the following is \textit{not} printed?

A. 'A X'
B. 'B D'
C. 'C X'
D. 'D Z'
x = 'ABCD'
z = 'XYZ'

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

Which of the following is not printed?

A  'A X'
B  'B D'
C  'C X'
D  'D Z'
Given a function $f(x)$, find $x = x^*$ such that $f(x^*)$ is maximized (or minimized).

Brute-force searches the entire domain of $f$.

How could we do this in our case?
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
What if we need another constraint, like bulk volume?
What if we need another constraint, like bulk volume?
Modify $f$, the figure of merit.
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!