Numerical Python

Brute-Force Solution

Announcements

quiz: `quiz16` due on Tues 12/11

lab: `lab` on Fri 15/11

hw: `hw08` due 13/11

`exam02` this wed `13th Nov 8pm`

`lec 06 to 13` and related stuff

CompE @ LTE 102/103
ME @ LTE 201/202
EE and CE @ LTW 102/103

Roadmap

- **Week 1 (9/9)**: L0 Intro; L1 PY - Intro
- **Week 2 (16/9)**: L2 PY – Data& Bool; L3 PY – if & Loops
- **Week 3 (23/9)**: L4 PY - Lists; L5 PY - Funcs
- **30/9**: L6 PY - Files; X
- **9/10**: L7 PY - Muta
- **12/10**: Exam 01
- **Week 6 (14/10)**: L8 PY - Dict; L9 PY - IO
- **Week 7 (21/10)**: L10 PY - Numpy; L11 PY - Plot
- **Week 8 (28/10)**: L12 PY - State; L13 PY - Rand
- **Week 9 (4/11)**: L14 PY – De-bug; L15 PY - Eqn
- **Week 10 (11/11)**: L16 Brute; Exam 02
- **Week 11 (18/11)**: L17 PY - Opt II; L18 PY - Mod
- **Week 12 (25/11)**: L19 PY – Sym I; L20 PY – Sym II
- **Week 13 (2/12)**: L21 PY – Sym III; L22 ML - Intro
- **Week 14 (9/12)**: L23 ML - IO; L24 ML - Poly
- **Week 15 (16/12)**: L25 ML - Stats; L26 ML - Fit
- **Week 16 (23/12)**: Review
- **Week 17 (30/12)**: Finals

Objectives

A. Apply a brute-force (comprehensive) search to solve problems relying on multiple dependent variables, with or without constraints.

B. Understand some of the tools available with `itertools` to obtain `permutations and combinations` of items in a container.

Solving Equations Recap

Did you use this to solve your math

For `scipy.optimize.newton( f, x0 )`,

`f` is a function

```
def thisIsHowYouDoIt(x):
    return x**2 + 4*x - 1
```

`x0` is the initial guess number, say `x0` = 5

To run, type:

```
scipy.optimize.newton(thisIsHowYouDoIt, 5)
```

Solving eqns - scipy.optimize

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

Solving eqns - scipy.optimize

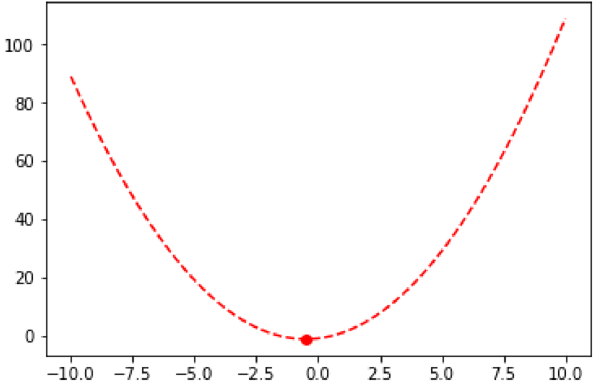
We can also find minima using `scipy.optimize.fmin( f,x0 )`.

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

Solving eqns - scipy.optimize

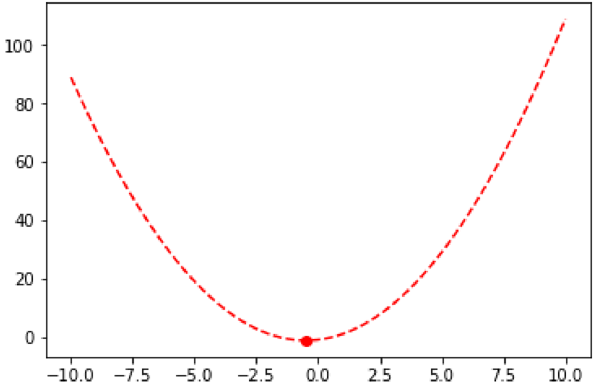
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.optimize

def f( x ):
    return x**2 + x - 1

x = np.linspace( -10,10,1000 )
xstar = scipy.optimize.fmin( f, x0=3 )
# or
# xstar = scipy.optimize.fmin( f, 3 )

plt.plot( x,f( x ),'r--', xstar,f( xstar ),'ro' )
plt.show()
```

Solving eqns - scipy.optimize



How does this code decide to stop? How does the computer know it has reacheed the minimum??

Solving eqns - scipy.optimize



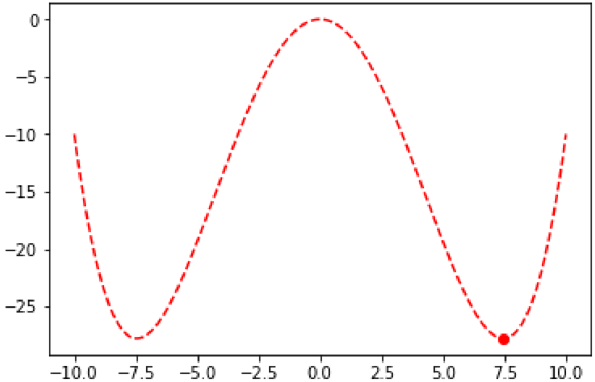
How does this code decide to stop? How does the computer know it has reacheed the minimum??

`Comparing the difference between the current and last value with the TOLERANCE !`

Solving eqns - scipy.optimize

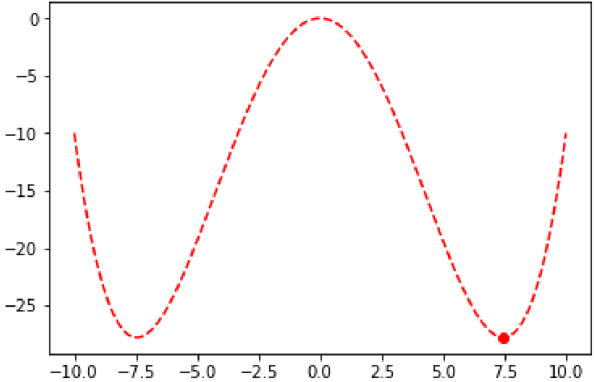
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.optimize

def f( x ):
    return 9e-3*x**4 - x**2

x = np.linspace( -10,10,1000 )
xstar = scipy.optimize.fmin( f, x0=3 )

plt.plot( x,f( x ),'r--', xstar,f( xstar ),'ro' )
plt.show()
```

Solving eqns - scipy.optimize



How do we get the value of the other minima?

Solving eqns - scipy.optimize

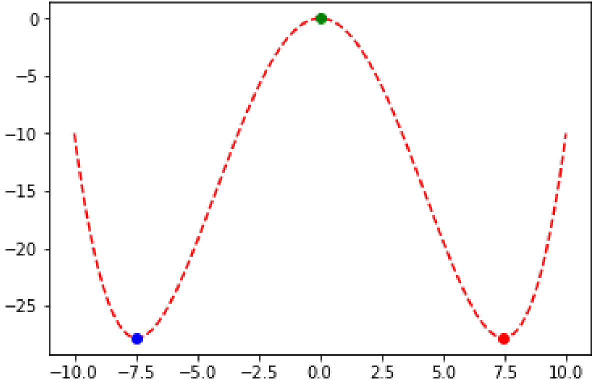


How do we get the value of the other minima?
Change x0 !

Solving eqns - scipy.optimize

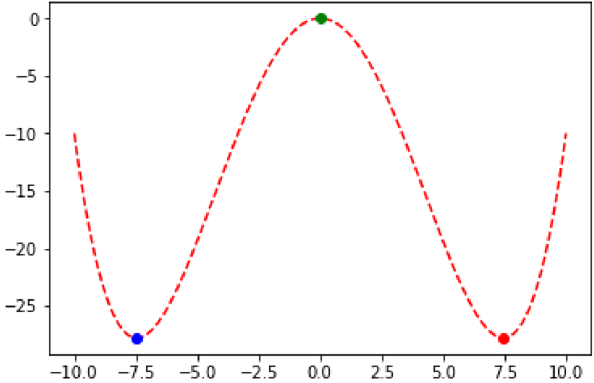
```
import matplotlib.pyplot as plt
import numpy as np
import scipy.optimize

def f( x ):
    return 9e-3*x**4 - x**2

x = np.linspace( -10,10,1000 )
xstar1 = scipy.optimize.fmin( f,  0.1 )
xstar2 = scipy.optimize.fmin( f,  -0.1 )
xstar3 = scipy.optimize.fmin( f,  -0.0001 )

plt.plot( x,f( x ),'r--', xstar1,f( xstar1 ),'ro',
            xstar2,f( xstar2 ),'bo',
            xstar3,f( xstar3 ),'go' )
plt.show()
```

Solving eqns - scipy.optimize



Why does `fmin` give us a maxima at the green dot?

Solving eqns - scipy.optimize



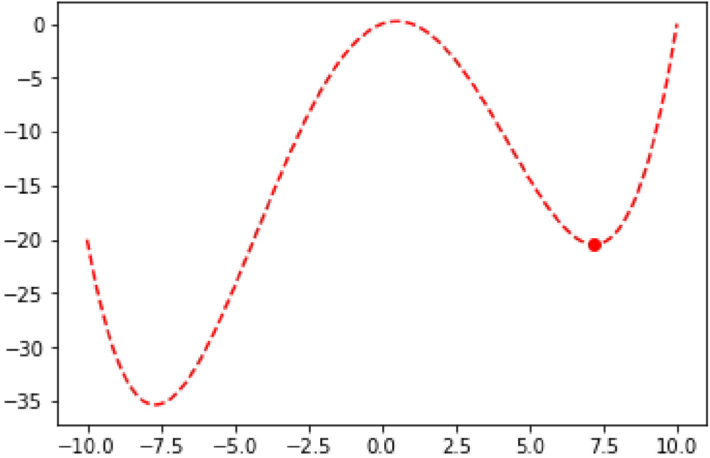
Why does `fmin` give us a maxima at the green dot?

`It does not! It stops there as the difference between the current and last values is smaller than the TOLERANCE !`

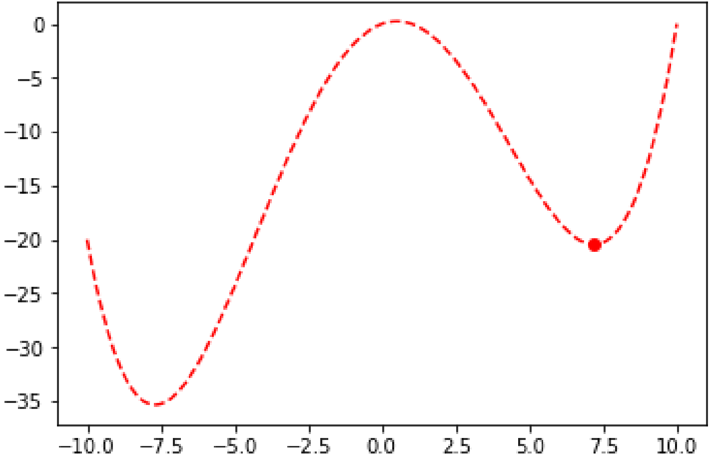
Solving eqns - scipy.optimize

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

def f( x ):
    return 9e-3*x**4 - x**2 + x

x = np.linspace( -10,10,1000 )
xstar1 = scipy.optimize.fmin( f, x0=1 )

plt.plot( x,f( x ),'r--', xstar1,f( xstar1 ),'ro' )
plt.show()
```

Solving eqns - scipy.optimize



Solving eqns - scipy.optimize



Did we get the global minima or the local minina?

`Plot to see what is happening!`

Optimization

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 kg. What is the best set of items to take on the flight?

What is your goal?

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 kg. What is the best set of items to take on the flight?

What is your goal?

`To maximize the value of souvenirs and minimize the weight! Trying to optimize what you can carry!`

Optimization

Given a function $f(\underline{x})$, find $\underline{x} = \underline{x}^*$ such that $f(\underline{x}^*)$ is maximized (or minimized).

The goal is to search all $\underline{x}$ to find a $\underline{x}^*$ which yields the optimal $f(\underline{x}^*)$.

Many clever techniques exist, but we'll start with a naïve approach, i.e., Brute-force Method.

Create the problem: Setup

```
import numpy as np
np.random.seed( 101 )

#number of souvenirs that you bought
#and hope to take back
n = 10
items    = list( range( n ) )

# weight of item
weights = np.random.uniform( size=(n,) ) * 50

# value of item => $
values  = np.random.uniform( size=(n,) ) * 100
```

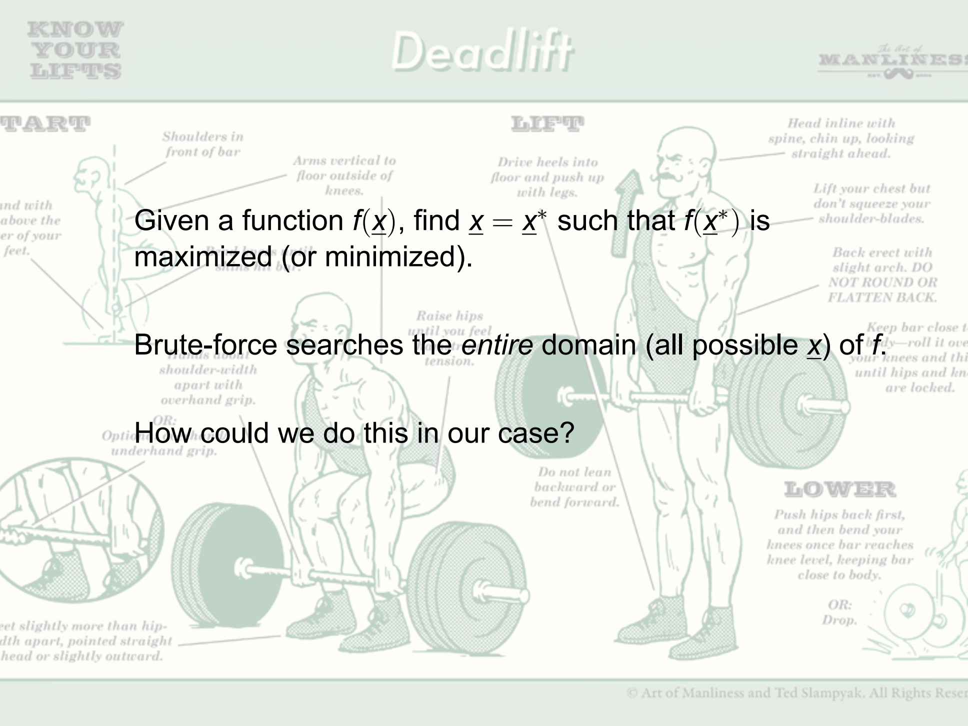
Decision code

```
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
```

Decision code

```
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
```

How to select all the possibilities so that this decision code can calculate?



Given a function $f(\underline{x})$, find $\underline{x} = \underline{x}^*$ such that $f(\underline{x}^*)$ is maximized (or minimized).

Brute-force searches the *entire* domain (all possible $\underline{x}$) of f.

How could we do this in our case?

Optimization

Two useful functions from the `itertools` module:

A. `combinations`: provide all subsets of size `n`.
B. `product`: replace nested `for` loops.

Optimization - combinations

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

Order of the entries is maintained

```
import itertools

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

Optimization - combinations

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

Order of the entries is maintained

```
import itertools

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

(1, 2)
(1, 3)
(1, 4)
(2, 3)
(2, 4)
(3, 4)

Optimization - product 1

`product`: replace nested `for` loops.

Can use `repeat=n` argument as well.

Order of the entries is maintained

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

Optimization - product 1

`product`: replace nested `for` loops.

Can use `repeat=n` argument as well.

Order of the entries is maintained

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

(1, 'g')
(1, 'h')
(1, 'i')
(2, 'g')
...
(4, 'i')

Optimization - product 2

`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, repeat=3 ):
    print( x )
```

Optimization - product 2

`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, repeat=3 ):
    print( x )
```

(1, 1, 1)
(1, 1, 2)
(1, 1, 3)
(1, 1, 4)
(1, 2, 1)
...

Question 1

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

Are they the same?

Question 1

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

Are they the same?
Ans:
Combination takes from one list and combines the different items.
Product take from many lists (including itself again). In both commands, the order of the items is maintained.
Go test in python

Question 2

```
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'`

Question 2

```
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'` ⋆

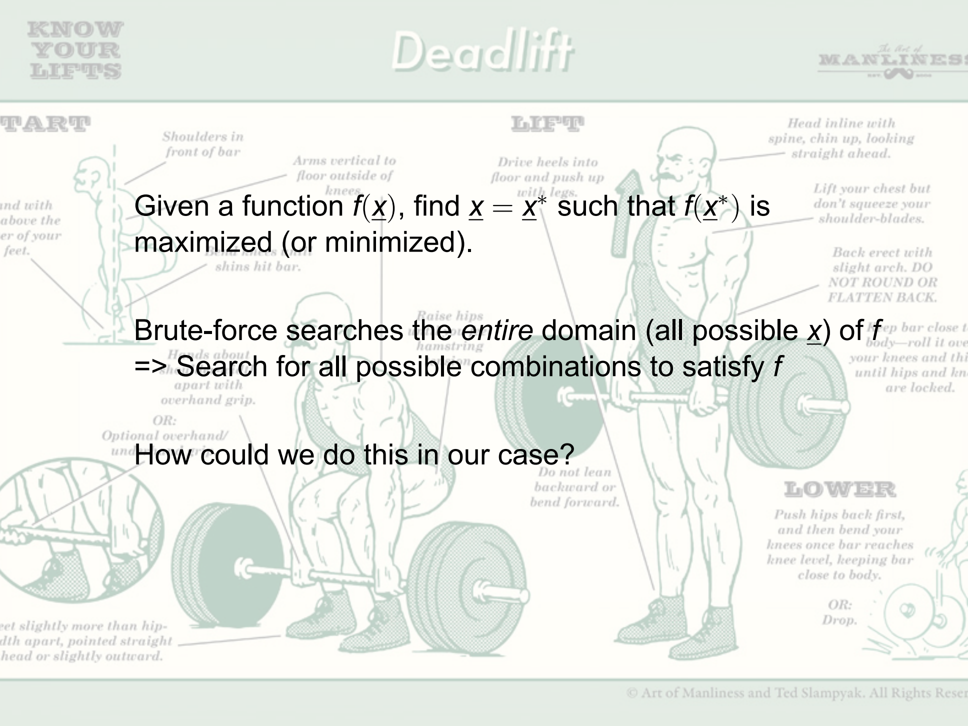
C `'C X'`

D `'D Z'`

Given a function $f(\underline{x})$, find $\underline{x} = \underline{x}^*$ such that $f(\underline{x}^*)$ is maximized (or minimized).

Brute-force searches the *entire* domain (all possible $\underline{x}$) of f
=> Search for all possible combinations to satisfy f

How could we do this in our case?



Setup 1

```
import numpy as np
np.random.seed( 101 )

#number of souvenirs that you bought
#and hope to take back
n = 10
items   = list( range( n ) )

# weight of item
weights = np.random.uniform( size=(n,) ) * 50

# value of item => $
values  = np.random.uniform( size=(n,) ) * 100
```

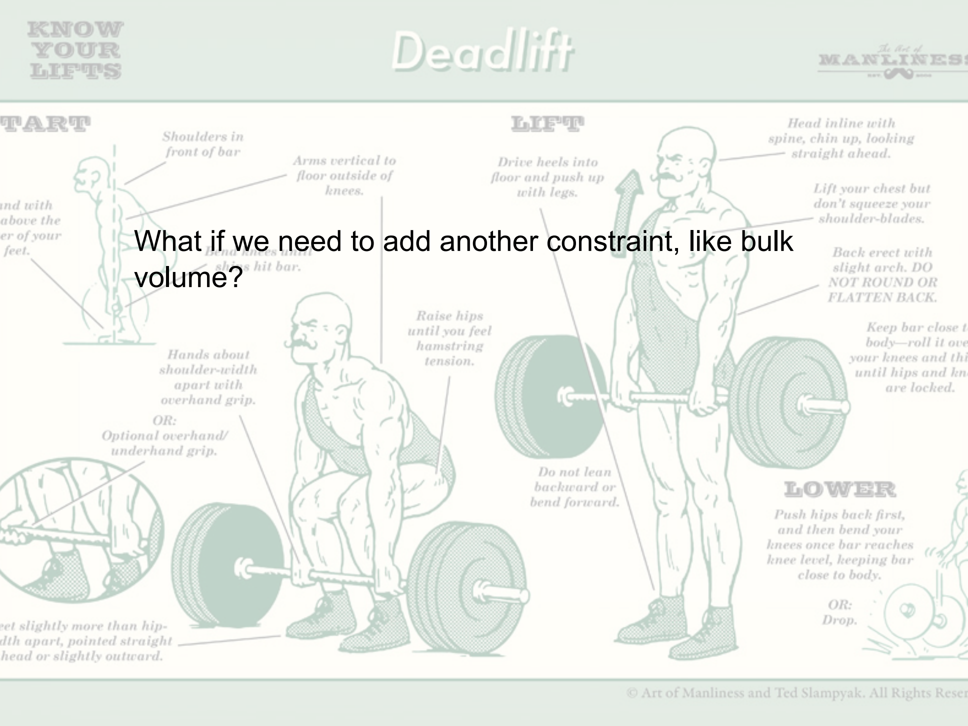
Setup 2

```
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
```

Decision code

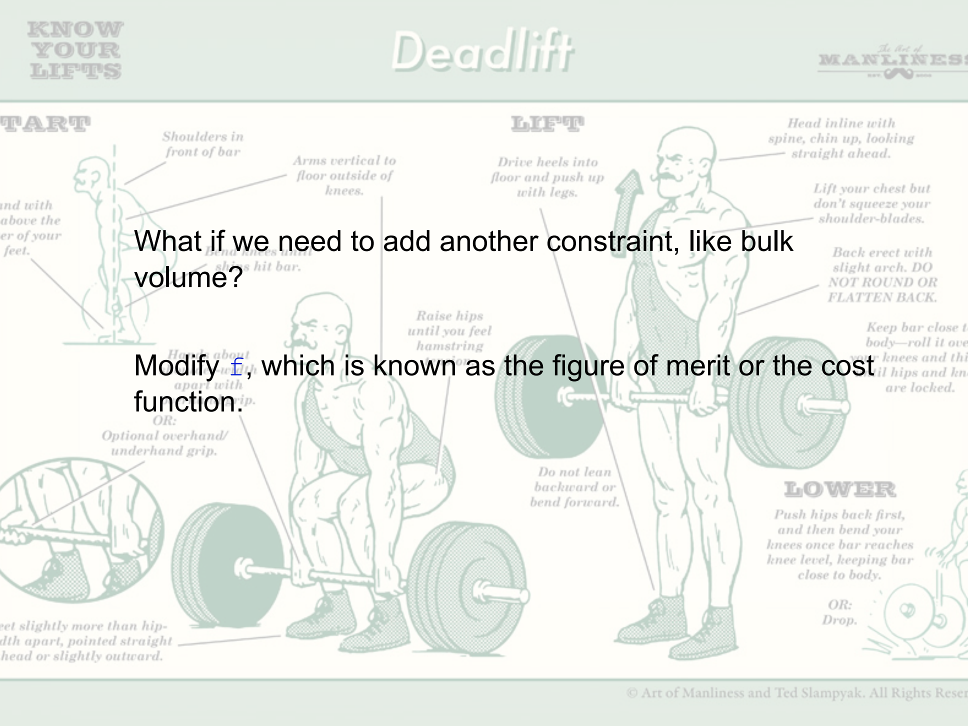
```
def f( wts, vals ):
    total_weight = 0
    total_value = 0

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

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

What if we need to add another constraint, like bulk volume?



What if we need to add another constraint, like bulk volume?

Modify `f`, which is known as the figure of merit or the cost function.



Another Problem: Pwd search

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:

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

possibilities per letter! This gets very big very quickly!

Optimization of Big Problems

When things get too big,

- Many optimization problems might take many, many, many years to solve
- Use supercomputer
- Use clever algorithm e.g., consider symmetry
- Simplify the problem: Get approximately correct solutions

Summary

A. Optimization solver - simplest using Brute force
B. `import itertools`
C. `combinations` and `product`
D. Be careful of using Brute force when too many combinations/products!!!