maximize 2x_1 + 3x_2 - x_3
\[ x_1 + x_2 \leq 1 \]
\[ x_1 + x_2 + x_3 \leq 0 \]
\[ x_2 - 2x_3 \leq 2 \]

Text file ‘example.lp’:
maximize 2x1 + 3x2 - x3

subject to
  x1 + x2 <= 1
  x1 + x2 + x3 <= 0
  x2 - 2x3 <= 2

bounds
  x1 free
  x2 free
  x3 free

end
Python and SAGE

• SAGE is a mathematical software
• SAGE is free software
• It is based on Python

Visit http://www.sagemath.org

• Python: interpreted, dynamically typed language
• Very convenient for many applications
• Slower than C, but many critical parts of SAGE are written in C, and interfaced with Python

Visit http://www.python.org
Data types

Integer
>> a = 5
>> b = -6

String
>> a = 'text'  # or "text"
>> len(a)
    4
>> a[0]
    't'

Rational
>> a = 1 / 2
>> b = 5 / 2  # Equal to 0 in Python, and 5 / 2 in SAGE

Floating-point
>> a = 1.0
>> b = 1.5 / 3
>> c = 2 * a * (3 + b) / 44 / 55
>> c = ((2 * a * (3 + b)) / 44) / 55
Tuples: ordered lists

Tuples are ordered and immutable

```python
>> a = ()        # Empty tuple
>> a = 1,        # One element tuple
>> a = (1)       # not a tuple!
>> a = (1, 2, 3) # 3 element tuple
>> a = 1, 2, 3   # The same tuple
>> a[1]
  2
>> len(a)
  3
>> a[1] = 2
  TypeError: tuple does not support assignment!
>> sum(a), max(a), min(a)
  (6, 3, 1)
```
List: works as an array

```python
>> a = [ ]  # Empty list
>> a = [ 1, 2 ] # Two elements in the list
>> a.append(3)
>> a  
[ 1, 2, 3 ]
>> len(a)
3
>> a[0] = 5  # List becomes [ 5, 2, 3 ]
>> a[-1]
3
>> a = 3 * [ 5 ]
[ 5, 5, 5 ]
>> a = [ 5, "teste" ]
>> a = [ 1, 2, 3 ]
>> sum(a), max(a), min(a)
6, 3, 1
```
Data types: dictionary

Dictionary: efficient dictionary implementation

```python
>> a = { }  # Empty dictionary
>> a = { "Alice": 1432, "Bob": 1717 }  # Social security numbers
>> "Alice" in a
   True
>> "Richard" in a
   False
>> a["Alice"]
   1432
>> a["Richard"]
   KeyError
>> a["Alice"] = 10
>> a["Richard"] = 5
>> a
   { "Alice": 10, "Bob": 1717, "Richard": 5 }
```
Some simple conditionals

```python
a = { }
if not "Richard" in a:
    print "Richard does not have a number!"
    a["Richard"] = -1
elif a["Richard"] == -1:
    print "Richard didn’t have a number last time!"
else:
    print "Richard’s number is", a["Richard"]
```

Python blocks are given by indentation!

```python
if a > b:
    do something
else:
    do something else
    if a > 2 * b:
        and yet something else
    oops! not good!
```

How would it look like in C?

```c
if (a > b) {
    do something;
}
else {
    do something else;
    if (a > 2 * b) {
        and yet something else;
    }
}
```
Looping: while

While loop
while condition:
  do something
  break  # Finishes loop
  continue  # Skips to next iteration
else:  # optional!
  what is done when condition becomes False
  If exit through break, this is ignored!

Example: is a number prime?
f = 2
while f < a:
  if a % f == 0:
    print "Number is not prime"
    break
  f += 1
else:
  print "Number is prime"
Looping: for

Use for lists:
fruit_list = [ "orange", "grape", "banana" ]
for fruit in fruit_list:
    print fruit
else:
    print "End of list"

range:
>> range(a, b, s)
   [ a + ks : a + ks < b and k = 0, 1, 2, ... ]
>> range(3)   # The same as range(0, 3, 1)
   [ 0, 1, 2 ]
>> range(1, 3)
   [ 1, 2 ]
>> range(2, 2)
   [ ]
>> range(3, 0, -1)
   [ 3, 2, 1 ]
Looping: for

Looping with range:

total = 0
for x in range(10):
    total += x
# Now total = 0 + 1 + ... + 9 = 45
# The same as sum(range(10))

In C:

```c
int total = 0;
for (int x = 0; x < 10; x++)
    total += x
```

When looping with range, use xrange

```python
fac = 1
for x in xrange(1, 10):
    fac *= x
```

s = 1
for k in range(10000000): s *= 1  # Takes 3.28s
for k in xrange(10000000): s *= 1  # Takes 2.73s
The dictionary is like a list of the keys!

```
a = { "Alice": 1, "Bob": 2 }
for name in a:
    print name, a[name]
```

**Result:**

```
Alice 1
Bob  2
```

**Other lists associated with a dictionary:**

```
>> a.keys()
    [ "Alice", "Bob" ]
>> a.values()
    [ 1, 2 ]
```
Function notation:
def func():  # No arguments!
    return "Hello"  # Return value is optional!

>> func()
    "Hello"

More complicated example:
def invert_if_bigger(a, b):
    if a > b: return b, a
    return a, b

>> invert_if_bigger(1, 2)
    (1, 2)
>> invert_if_bigger(2, 1)
    (1, 2)
Writing SAGE programs

• Write program to a file
  ▶ Use a nice editor, line Emacs in Python mode!
  ▶ This helps you with the indentation
• Load it in sage

Program "fac.sage"
def fac(p):
    x = 1
    for k in xrange(2, p + 1):
        x *= k
    return x

print fac(5)

Sage console:
>> load("fac.sage")
    120
>> fac(6)
    720
LP in SAGE

maximize $2x_1 + 3x_2 - x_3$

$x_1 + x_2 \leq 1$

$x_1 + x_2 + x_3 \leq 0$

$x_2 - 2x_3 \leq 2$

Basic setup

```python
>> p = MixedIntegerLinearProgram(maximization = True)
>> x = p.new_variable()
>> for k in xrange(1, 4): p.set_min(x[k], None)
>> p.solve()
 3.0
>> p.get_values(x)
  {1: 1.0000000000, 2: -2.2204460131e-16, 3: -1.0}
```
SAGE Graph class

Used to represent undirected graphs:

```python
>> a = { 0: [ 1, 'c' ], 'c': [ 1, 2 ] }  # Adjacency list
>> G = Graph(a)
>> G.plot()
>> G = graphs.PetersenGraph()
>> G.plot()
```
Iterating through graphs

List of vertices and the adjacency list of a vertex

```python
>> a = { 0: [ 1, 'c' ], 'c': [ 1, 2 ] }
>> G = Graph(a)
>> G.vertices()
    [ 0, 1, 'c', 2 ]
>> G[1]
    [ 0, 'c' ]
>> G['c']
    [ 0, 1, 2 ]
>> G.edges(labels = False)
    [(0, 1), (0, 'c'), (1, 'c'), ('c', 2)]
```

How to iterate:

```python
for u in G:  # You don't need to use G.vertices() here!
    print 'Here are the neighbors of', u
    for v in G[u]:
        print v
```
Exercise 8

maximize \[ \sum_{e \in E} x(e) \]
\[ \sum_{e \in \delta(u)} x(e) \leq 1 \quad \text{for all } u \in V \]
\[ x(e) \geq 0 \quad \text{for all } e \in E \]

Finds a maximum matching or concludes input graph is not bipartite

```python
def maximum_matching(G):
    p = MixedIntegerLinearProgram(maximization = True)
    x = p.new_variables()

    # Trick for quickly accessing edges
    E = {}
    for u, v in G.edges(labels = False):
        E[u, v] = E[v, u] = u, v

    # Constraints for each vertex
    for u in G:
        c = 0
        for v in G[u]: c += x[E[u, v]]
        p.add_constraint(c <= 1.0)
```

Finds a maximum matching or concludes input graph is not bipartite

def maximum_matching(G):
    (...)
    # Objective function
    obj = 0
    for e in G.edges(labels = False): obj += x[e]
    p.set_objective(obj)

    # Solves and analyzes solution
    p.solve()
    sol = p.get_values(x)
    M = [ ]  # This is the matching we return

    for e in G.edges(labels = False):
        if sol[e] > 1e-10 and sol[e] < 1 - 1e-10:
            raise ValueError('G is not bipartite!')
        elif sol[e] > 1 - 1e-10:
            M.append(e)

    return M