CS 42 Today

Midterm 1: DUE 9AM TOMORROW!

Allowed:
- 1 8.5x11 sheet of paper with your own writing/work.
  (Printed copies of your code done for this class OK)
- A computer for typing your solutions
  • Running of code NOT OK

Can you recognize these sorting algorithms?

http://sortvis.org/

What do the following have in common:

I'd love to get it all sorted out.
Functioning in Python

Some basic, built-in functions:

- **abs**: absolute value
- **max**: of lists
- **min**: of lists
- **sum**: creates lists
- **range**: creates lists
- **round**: ???

These change data from one type to another:

- **bool**
- **float**
- **int**
- **long**
- **list**
- **str**

These are the most important:

**help**  **dir**
Functioning in Python

Far more are available in separate files, or modules:

```python
import math
math.sqrt(1764)
dir(math)
```

accesses `math.py`'s functions

lists all of `math.py`'s functions

```python
from math import *
pi
sin(pi/2)
```
same, but without typing `math` all of the time…
Functioning in Python

# my own function!
defdbl(x):
    """returns double its input, x""
    return2*x
# my own function!

def dbl( x ):
    """ returns double its input, x """
    return 2*x

Some of Python's baggage…

**Docstrings**

They become part of python's built-in help system! With each function be sure to include one that

(1) describes overall what the function does, and
(2) explains what the inputs mean/are

**Keywords**

def starts the function
return stops it immediately and sends back the return value

**Comments**

They begin with #
Functioning in Python

```python
def undo(s):
    """ this "undoes" its string input, s """
    return 'de' + s

>>> undo('caf')
>>> undo(undo('caf'))
```
if, elif, else...

def foo(x):
    """This function demonstrates the use of if, elif, and else"""
    if x > 0 and x < 42:
        return "Small"
    elif x >= 42 and x < 100:
        return "Nice!"
    elif 100 <= x < 200:
        return "Big"
    else:
        print "That was one nasty number!"
        return "Yuck!"

Notice how lines with the same level of indentation are in the same code block!
Range

>>> range(5)
[0, 1, 2, 3, 4]

>>> range(5, 10)
[5, 6, 7, 8, 9]

>>> range(5, 20, 2)
[5, 7, 9, 11, 13, 15, 17, 19]
Two kinds of \texttt{for} loops

\begin{itemize}
\item \textbf{Element-based Loops}
\end{itemize}

\begin{align*}
\text{sum} &= 0 \\
\text{for } x \text{ in } L: \\
& \quad \text{sum }+= x
\end{align*}

\begin{itemize}
\item \textbf{Index-based Loops}
\end{itemize}

\begin{align*}
\text{sum} &= 0 \\
\text{for } i \text{ in } L: \\
& \quad \text{sum }+= i
\end{align*}
## Mutable vs. Immutable data

<table>
<thead>
<tr>
<th>Changeable types:</th>
<th>Unchangeable types:</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary</td>
<td>tuple</td>
</tr>
<tr>
<td>list</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>float</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
</tbody>
</table>

What's a dictionary? I guess I'll have to look it up!
Functions and (immutable) Variables

def fact(a):
    """non-recursive factorial"""
    result = 1
    while a > 0:
        result *= a
        a -= 1
    return result

>>> x = 5
>>> y = fact(x)
>>> x
??
def main()
    """ calls conform """
    print " Welcome to Conformity, Inc. "
    fav = 7
    conform(fav)
    print " My favorite number is", fav

def conform(fav)
    """ sets input to 42 """
    fav = 42
    return fav
"Pass By Value"

```python
def main():
    """ calls conform ""
    print " Welcome to Conformity, Inc. "

    fav = 7
    conform(fav)

    print " My favorite number is", fav

def conform(fav):
    """ sets input to 42 ""
    fav = 42
    return fav
```

"Pass by value" means that data is **copied** when sent to a method.
Functions and (immutable) Variables

The fine print: This is not the "truth"… but it's close enough

def swap(a, b):
    temp = a
    a = b
    b = temp

>>> x = 5
>>> y = 10
>>> swap(x, y)
>>> print x, y
??

x
y
a
b
temp
Passing *lists* by value...

def main()
    """ calls conform2 """
    print " Welcome to Conformity, Inc. "
fav = [ 7, 11 ]
conform2(fav)
print " My favorite numbers are", fav

def conform2(fav)
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42

What gets passed by value here?
Functions and Mutable Types

```python
def swap(L, i1, i2):
    temp = L[i1]
    L[i1] = L[i2]
    L[i2] = temp

>>> MyL = [2, 3, 4, 1]
>>> swap(MyL, 0, 3)
>>> print MyL
??
```
Reference vs. Value

**Mutable** types:
- dictionary
- list

**Unmutable** types:
- tuple
- string
- float
- int
- bool

L = [5, 42, 'hi']

L = 42

Reference, Pointer, id

Whee!
Passing *lists* by value...

def main():
    """ calls conform2 """
    print "Welcome to Conformity, Inc."
    fav = [7, 11]
    conform2(fav)
    print "My favorite numbers are", fav

def conform2(fav):
    """ sets all of fav to 42 """
    fav[0] = 42
    fav[1] = 42
The conclusion

You can change the contents of lists in functions that take those lists as input.

(Actually, lists or any mutable objects)

Those changes will be visible everywhere.

(Immutable objects are safe, however)
Lists vs. Dictionaries

Lists are not perfect...

If I had a dictionary, I guess I could look up what it was!
Lists vs. Dictionaries

Lists are not perfect...

You can't choose what to name data.

L[0], L[1], ...

If I had a dictionary, I guess I could look up what it was!
Lists vs. Dictionaries

Lists are not perfect...

You can't choose what to name data.

L[0], L[1], ...

You have to start at 0.

L[1988] = 'dragon'

L[1989] = 'snake'
Lists vs. Dictionaries

Lists are not perfect…

You can't choose what to name data.

L[0], L[1], …

You have to start at 0.

L[1988] = 'dragon'
L[1989] = 'snake'

Some operations can be slow for big lists …

if 'dragon' in L:
Lists vs. Dictionaries

In Python a **dictionary** is a set of **key-value** pairs.

```python
>>> d = {}
>>> d[1988] = 'dragon'
>>> d[1989] = 'snake'
>>> d
>>> d[1988]
'dragon'
>>> d[1987]
key error
```

It's a list where the index can be **any immutable-type key**.
Lists vs. Dictionaries

In Python a **dictionary** is a set of **key - value** pairs.

```python
>>> d = {}
creates an empty dictionary, d
>>> d[1988] = 'dragon'
1988 is the key
'dragon' is the value
>>> d[1989] = 'snake'
1989 is the key
'snake' is the value
>>> d

Anyone seen this before?

>>> d[1988]
'dragon'

>>> d[1987]
key error

Retrieve data as with lists...

or almost!

It's a list where the index can be **any immutable-type key**.
More on dictionaries

Dictionaries have lots of built-in methods:

```python
>>> d.keys() # get all keys
[1989, 1988]
>>> d.has_key(1988) # check if a key is present
True
>>> d.has_key(1969)
False
>>> d.pop(1988) # delete a key (and its value)
'dragon'
```

They don't seem moronic to me!
A challenge...

```python
prov = { 'BC': 0, 'AB': 0, ... }

def provinceChallenge( prov ):
    """ prov is a dictionary of Canada's provinces
    -- the challenge is to name them all! """

    while 0 in prov.values():
        guess = raw_input("Name a province: ")
        if prov.has_key( guess ) == False:
            print 'Try again...'
        elif prov[guess] == 0:
            print 'Yes!'
            prov[guess] += 1
        else:
            print 'Already guessed...'

    print 'Phew!'
```

help?!
1. Change this code so that it keeps track of you *how many times* you've guessed each item.

2. Then change it so it keeps track of all incorrect guesses in a single dictionary entry

```python
def provinceChallenge( prov ):

    while 0 in prov.values():
        guess = raw_input("Guess: ")

        if prov.has_key( guess ) == False:
            print 'Try again...'  

        elif prov[guess] == 0:
            print 'Yes!'  
            prov[guess] += 1

        else:
            print 'Already guessed...'
```

"Quiz"
def provinceChallenge( prov ):

    while '0' in prov.values():
        guess = raw_input("Guess: ")
        if prov.has_key( guess ) == False:
            print 'Try again...'

    elif prov[guess] == 0:
        print 'Yes!'
        prov[guess] += 1

    else:
        print 'Already guessed...'

• first, count guesses
• then, track incorrect provinces
Lists are *mutable* objects.
(So are dictionaries.)

```python
>>> L = [2,1,3]
>>> L.sort()
>>> L
[1,2,3]
```

Strings are *immutable* objects.

```python
>>> s = 'string'
>>> s.replace('st','',)
'ring'
>>> s
'string'
```

*L* has changed.

L has changed.

no return value

returns a NEW string

s has NOT changed
Why Sort?
Sorting Algorithms

the *fruit flies* of complexity theory...

**Checksort**
check permutations until sorted

Check:  
\[
\begin{array}{c}
3 & 1 & 2 \\
3 & 2 & 1 \\
1 & 2 & 3 \\
1 & 3 & 2 \\
2 & 1 & 3 \\
2 & 3 & 1 \\
\end{array}
\]

"bogosort"

---

**SlowSort**!
pull #s from a hat
*with* replacement...

---

Slow:  
\[
\begin{array}{c}
1 & 1 & 1 \\
1 & 1 & 2 \\
1 & 1 & 3 \\
1 & 2 & 1 \\
1 & 2 & 2 \\
1 & 2 & 3 \\
1 & 3 & 1 \\
1 & 3 & 2 \\
1 & 3 & 3 \\
2 & 1 & 1 \\
2 & 1 & 2 \\
2 & 1 & 3 \\
2 & 2 & 1 \\
2 & 2 & 2 \\
2 & 2 & 3 \\
2 & 3 & 1 \\
2 & 3 & 2 \\
2 & 3 & 3 \\
3 & 1 & 1 \\
3 & 1 & 2 \\
3 & 1 & 3 \\
3 & 2 & 1 \\
\end{array}
\]

---

Bogosort
Bogosort is a particularly ineffective sorting algorithm.
def checkSort(L):
    ''' sort the list by permuting it and checking for sorted
    returns a sorted version of L. Does not modify L'''
    currentPermList = range(len(L))
    sortedList = copy(L)
    while not isSorted(sortedList):
        updatePermList(currentPermList)
        for i in range(len(L)):
            sortedList[i] = L[currentPermList[i]]
    return sortedList

def isSorted(L):
    ''' check to see if a list is sorted '''
    if len(L) == 0:
        return True
    for i in range(len(L)-1):
        if L[i+1] < L[i]:
            return False
    return True
Sorting Algorithms: Can we do better?

Yes!

Fun for all ages
## Sorting Algorithms

The fruit flies of complexity theory...

<table>
<thead>
<tr>
<th>Minsort</th>
<th>7</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>5</th>
<th>0</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>choose the min each iteration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An algorithm of a similar sort...

**Insertion Sort**

insert elements in order to the left

7 | 4 | 3 | 2 | 1 | 5 | 0 | 6

vs. minsort?
better than \(O(N^2)\)?