Announcements

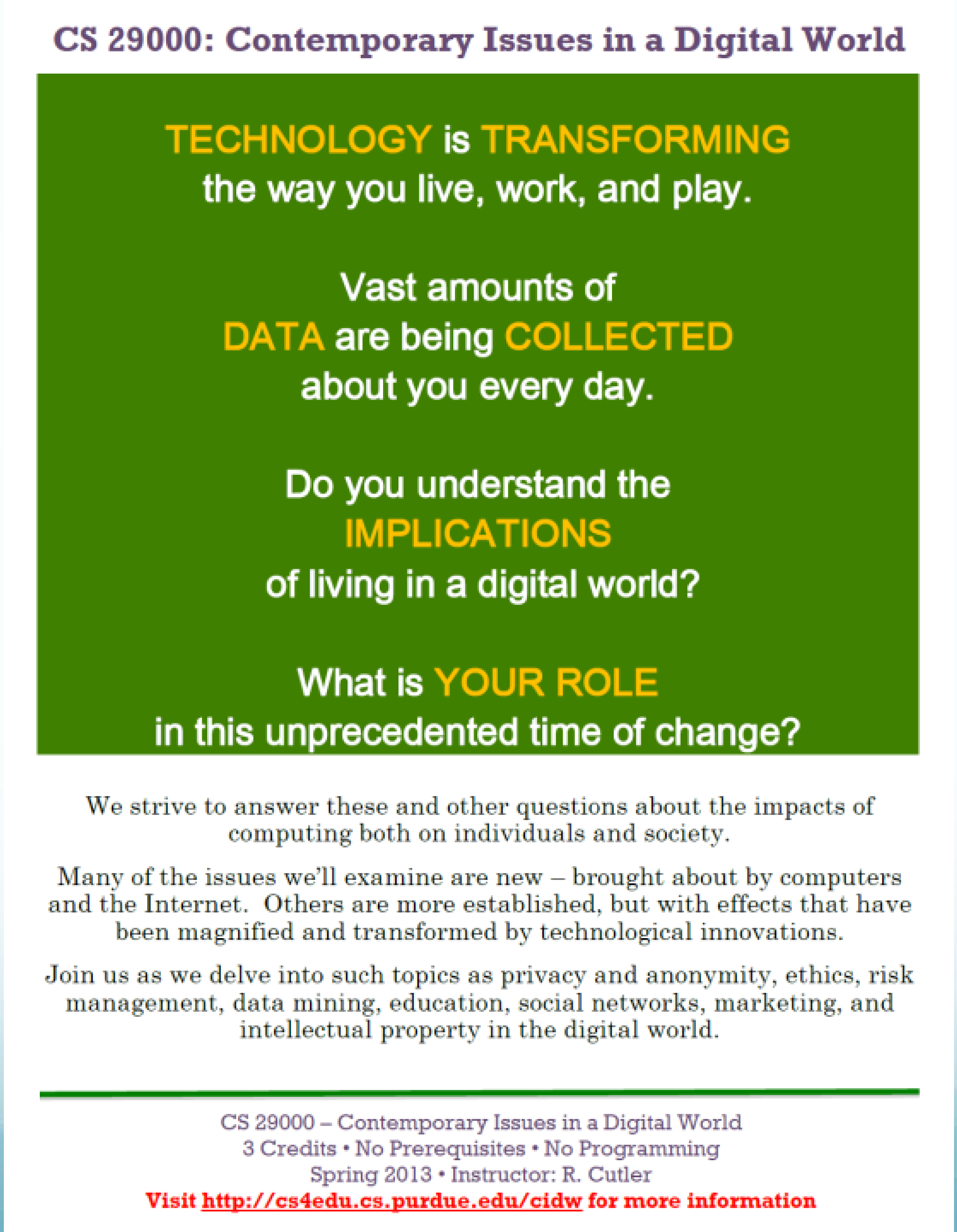
- Project 4 due Wed., Nov 7
- CQ: who has handed project 4 in as of right now?

(A) My team has turned in project 4

(B) My team has not turned in project 4

Announcements

- Project 5 coming soon; also a team project
  - This will include the brief essay question for the team course of Science
  - The team experience essays are individual
- New course CS 290 00, Spring 2013
  Contemporary Issues in a Digital World
  Instructor: Robb Cutler.
  Syllabus at http://cs4edu.cs.purdue.edu/cidw



CS 29000: Contemporary Issues in a Digital World

TECHNOLOGY is TRANSFORMING
the way you live, work, and play.

Vast amounts of
DATA are being COLLECTED
about you every day.

Do you understand the
IMPLICATIONS
of living in a digital world?

What is YOUR ROLE
in this unprecedented time of change?

We strive to answer these and other questions about the impacts of computing both on individuals and society.

Many of the issues we'll examine are new – brought about by computers and the Internet. Others are more established, but with effects that have been magnified and transformed by technological innovations.

Join us as we delve into such topics as privacy and anonymity, ethics, risk management, data mining, education, social networks, marketing, and intellectual property in the digital world.

CS 29000 – Contemporary Issues in a Digital World
3 Credits • No Prerequisites • No Programming
Spring 2013 • Instructor: R. Cutler
Visit http://cs4edu.cs.purdue.edu/cidw for more information

Covering Chapter 11

- Dictionaries
- Tuples
- Sets

Dictionaries in Python

- Useful Analogy: an actual Dictionary
- English dictionaries provide an association between a *Word* and a *Definition*
  - We use the *Word* to look up the *Definition*
  - Given a definition it would be very hard to look up the word

Dictionaries Python

- Much like a dictionary for the English language, python dictionaries create an association between a ***key*** and a ***value***
  - *Key* corresponds to a *Word* in our analogy
  - *Value* corresponds to a *Definition*

Dictionary Syntax

- A dictionary is a collection of *elements*
  - Each element is a **key:value** pair
- Just like a list is defined by [ ] a dictionary is defined by { }
- Example:

```
{ ‘key1’ :value1, ‘key2’ :value2, ‘key3’ :value3}
```

Keys

- A key can be any immutable type (we will consider two types)
  - Includes Tuples, Strings and Integers
- We use the [index] syntax to select out an element from a list, but for a dictionary we use [key]

```
A = {‘key1’:value1, ‘key2’:value2,‘key3’:value3}
print(A[‘key2’])
```

Example: Phone Book App

- phoneBook = { 'Tom' : ' 123-4567' , 'George' : '456-7890' ,
  'Dick' : '234-5678' , 'Harry' : '345-6789' }

  names are keys, phone numbers are values

```
def lookup(key):
    return phoneBook[key]
lookup( 'George' )
```

CQ: do these programs print the same thing?

1

```
A = [ 'ike' , 'mary' ,
  'marty' ]
print A[1]
```

2

```
A = {0:' ike' , 1:' mary' ,
2:' marty' }
print A[1]
```

A: yes

B: no

CQ: do these programs print the same thing?

1

```
A = [ 'mike' , 'mary' ,
 'marty' ]
print A[1]
```

2

```
A = {1:' mary' , 2:' marty' ,
0:' mike' }
print A[1]
```

A: yes

B: no

Key Differences from Lists

- Lists are ordered sequences
  - List index is implicit based on the list ordering
- Dictionaries are unordered sequences
  - Keys are specified and do not depend on order
- Lists are useful for storing ordered data, dictionaries are useful for storing *relational* data
  - Motivating example: databases!

Key Lookup: *in*

- If the key is in the dictionary we get the value
- If the key is not in the dictionary, we get an error!

- Test that the key is in the dictionary:

```
myDict = { ‘a’ :1, ‘b’ :2, ‘c’ :3}
print( ‘b’ in myDict)
print( ‘d’ in myDict)
print( ‘c’ not in myDict)
```

- This prints

```
True
False
False
```

Updating a Dictionary

- Much like a list we can *assign* to a dictionary

Abstract:

dictionary[*key*] = *newValue*

Concrete Example:

```
A = {0:' mike' , 1:' mary' ,
2:' marty' }
print (A[1])
A[1] =  'alex'
print(A[1])
```

Adding to a Dictionary

- Much like a list we can *append* to a dictionary

Abstract:

dictionary[new*Key*] = *newValue*

Concrete Example:

```
A = {0:’mike’, 1:’mary’, 2:’marty’}
print(A[1])
A[3] = ‘alex’
print(A)
    {0:’mike’, 1:’mary’, 2:’marty’,
3:’alex’}
```

Clicker Question: What is the output of this code?

```
A = {0:' mike' , 1:' mary' , 2:' marty' ,
      'marty' :2, 'mike' :0, 'mary' :1}
A[3] = 'mary'
A[ 'mary' ] = 5
A[2] = A[0] + A[1]
```

A: {'mike': 0, 'marty': 2, 3: 'mary', 'mary': 5, 2: 'mikemary', 1: 'mary', 0: 'mike'}

B: {'mike': 0, 'marty': 2, 'mary' :3, 'mary': 5, 2: 'mikemary', 1: 'mary', 0: 'mike'}

C: {'mike': 0, 'marty': 2, 'mary' :3, 'mary': 5, 2:1, 1: 'mary', 0: 'mike'}

Printing a Dictionary

```
A = {0:'mike', 1:'mary', 2:'marty’   }
for k in A:
    print(k)
```

```
Prints: 2
        1
        0
```

```
A = {0:'mike', 1:'mary', 2:'marty’   }
for k,v in A.iteritems():
    print(k, ":",  v)
```

```
Prints: 2 : marty
        1 : mary
        0 : mike
```

Character Frequency Analysis

- We can use a dictionary to calculate the number of times a particular letter occurs in a text
  - We use characters as the keys
  - The number of times that character occurs is the value
- Increment the value each time we see a character
  - Initially the value starts at 0

```
def occurrence(text):
    myDict = {}
    for char in text:
        if char in myDict:
            myDict[char] += 1
        else:
            myDict[char]=1
    return myDict

myDict = occurrence("abracadabra")
print(myDict)
```

```
{'a': 5, 'r': 2, 'b': 2, 'c': 1, 'd': 1}
```

Creating a dictionary from a list

- Python provides the dict function to create a dictionary from a list of pairs

  Example: dict([(0, 'mike'),(1, 'mary'),(2, 'marty')])

- Why do I care?
  - We can use the list creation short cuts to populate dictionaries!
  - Examples:

    ```
    dict([(x, x**2) for x in range(10)])
    dict([[x, x**2] for x in range(10)])
    ```

- Note the pair (x, x**2) – it is a tuple

Tuples

- We can create pairs in python
  - Example: `tuple = ( 'name' , 3)`
  - Such pairs are called *tuples* (see Chapter 11)
- You have used tuples in Project 4:

  `myImage.putpixel((x,y), (r,g,b))`
- Tuples support the [ ] for selecting their elements
- Tuples are immutable (like strings)

Tuples

- We can think of tuples as an immutable list
  - They are almost like lists,
  - They are indexed like lists, but
  - They do not support element assignment
- Example:

```
A = ( ‘me’ , 5, 32, ‘joe’ )
print A[0]
print A[3]
A[2] = 4      <--- this throws an error
A[1] = 4      <--- this throws an error
```

Operations on Tuples

- Concatenation:
  - (1, 2, 3) + (6, 5, 4) produces (1, 2, 3, 6, 5, 4)
  - 3*(1,2) produces (1, 2, 1, 2, 1, 2)
  - 3 * (5) produces 15
  - 3 * (5,) produces (5, 5, 5)
- Type change
  - tuple([1,2,3]) evaluates to (1, 2, 3)
  - tuple(‘abc’) evaluates to (‘a’ , ’ b’ , ’ c’)
  - tuple(12) evaluates to an error
- Argument of tuple() should be a sequence (iterable)

Why have Tuples?

- Guaranteed not to change in element value
  - Good for debugging
  - Good for optimization by Python

Sets

- Collections of immutable values
- Unordered
- Elements are unique
- Syntax:

  { expr_1, expr_2, …, expr_n }

- Elements must be "hashable"
  - Strings, tuples are ok
  - Lists and nonempty sets are not ok

Examples

- {1, 4, 9, 16, 25, 36, 49, 64, 81}
- {(1,2,3), 17, ‘oh henry’ }
- { ‘Tom’ , ‘Dick’ , ‘Harry’ }
- {(1,1), (2,4), (3,9), (4,16)}

Basic Operations

- + and * do not work
- Set operations are methods:
  - X = {1,2}
    Y = {2,3}
    X.union(Y) evaluates to {1,2,3} without changing X or Y
- More operations
  - .union(<set>)
  - .intersection(<set>)
  - .symmetric_difference(<set>)
  - .difference(<set>)

Main Set Operations

X = {1, 2, 3, 4}
Y = {3, 1, 5, 6}

X.union(Y)	evaluates to	{1, 2, 3, 4, 5, 6}
Y.union(X)	evaluates to	{1, 2, 3, 4, 5, 6}
X.intersection(Y)	evaluates to	{1, 3}
X.difference(Y)	evaluates to	{2, 4}
Y.difference(X)	evaluates to	{5, 6}

X.symmetric_difference(Y)
evaluates to {2, 4, 5, 6}

Set Membership

- Membership:
  - <expr> in <set>
  - <expr> not in <set>
- Examples

2 in {1,3,5,7}	evaluates to	False
2 not in {1,3,5,7}	evaluates to	True
5 in {1,3,5,7}	evaluates to	True
5 not in {1,3,5,7}	evaluates to	False

Nota Bene

- { } creates an empty dictionary
  - {{ }} throws a runtime error
- set() creates the empty set
  - set(set()) evaluates to set()
  - set() in set(set()) evaluates to False
- Short forms for set operations
  - X & Y is the same as X.intersection(Y)
  - X | Y is the same as X.union(Y)
  - X ^ Y is the same as X.symmetric_difference(Y)
  - X – Y is the same as X.difference(Y)

O(1) Example

```
def isOdd(list):
    return (len(list)%2 == 1)
```

```
>>> isOdd([0])
True
>>> isOdd([0,1])
False
```

Clicker Question

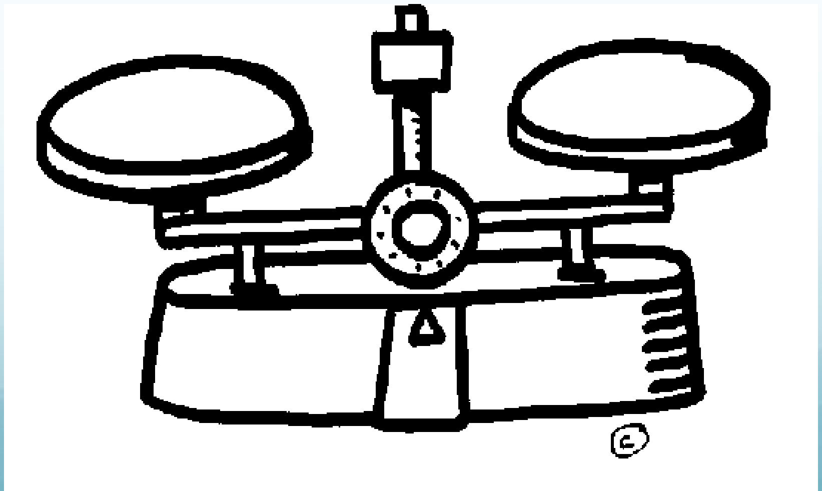
```
def getFirst(list):
    if len(list) == 0:
        return -1
    return (list[0])
```

```
>>> getFirst([])
-1
>>> getFirst([0,1,2,3])
0
>>> getFirst(["a", "b", "c"])
'a’
```

A: O(n)
B: **O(n^2)**
C: O(1)

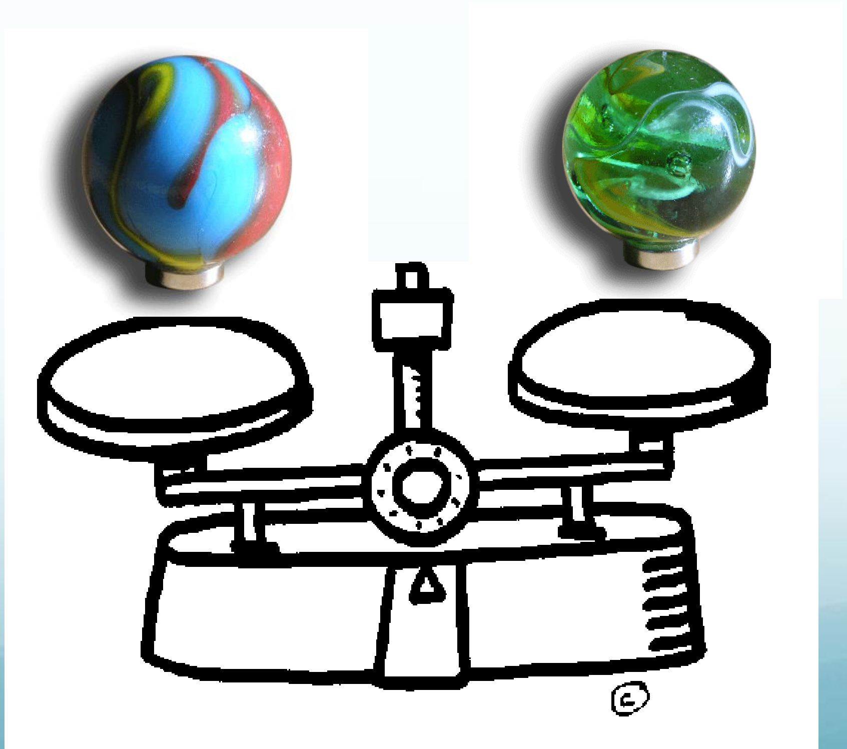
How to find an alien

- Logic Puzzle:
  - You have 9 marbles. 8 marbles weigh 1 ounce each, & one marble weighs 1.5 ounces. You are unable to determine which is the heavier marble by looking at them. How do you find the marble which weighs more?



Solution 1: Weigh one marble vs another

- What is the complexity of this solution?



Finding the Complexity

- Step 1: What is our input?
  - The marbles
- Step 2: How much work do we do per marble?
  - We weight each marble once (except one)
- Step 3: What is the total work we did?
  - 8 measurements
  - What if we had 100 marbles or 1000?

Clicker Question: What is the complexity of this algorithm?

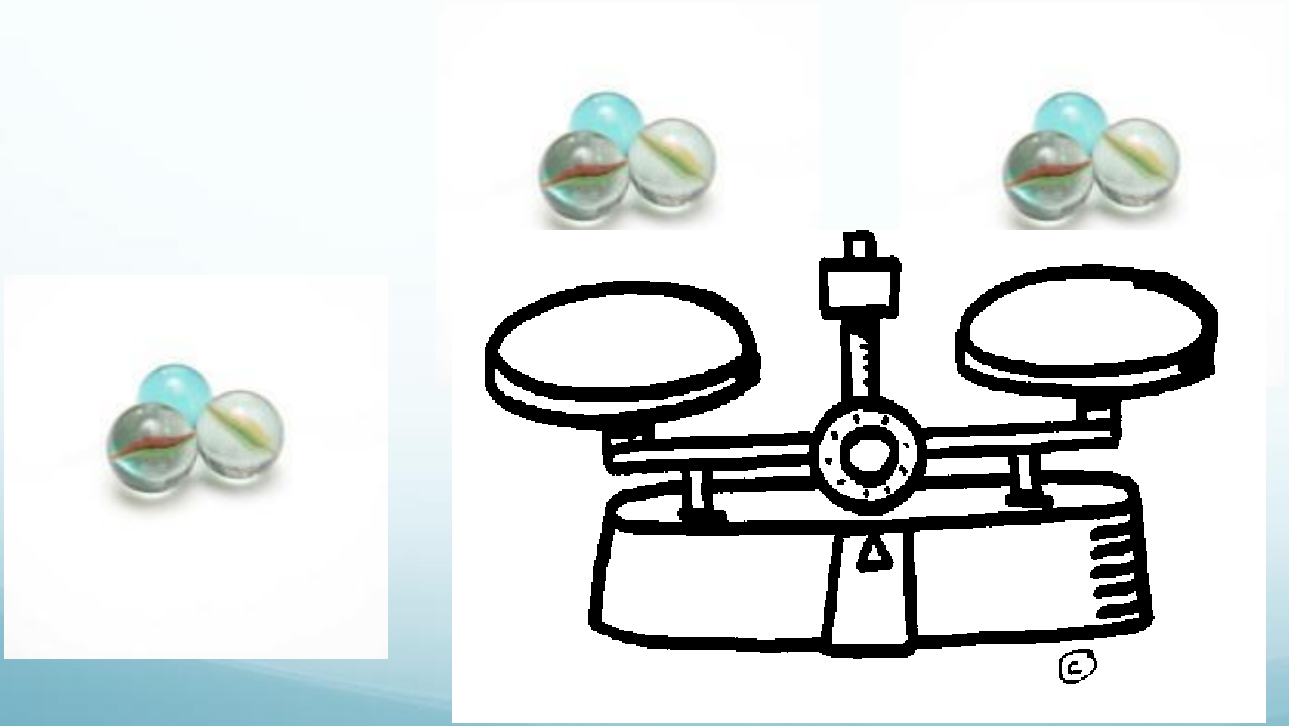
A: O(n)
B: **O(n^2)**
C: O(1)
D: O(log n)

We can do better!

- Lets pull some intuition from our search algorithm that was O(log n)
  - We want a way to eliminated ½ (or more) of the marbles with each measurement
- How might we do this?
  - What about weighing multiple marbles at once?

The Optimal Solution

- Split the marbles into three groups
  - We can then weigh two of the groups



Finding the complexity of the optimal solution

- Step 1: What is our input?
  - The marbles
- Step 2: How much work do we do per marble?
  - Logarithmic
- Step 3: What is the total work we did?
  - 2 measurements
  - What if we had 100 marbles or 1000?

What happens at each step?

- We eliminated 2/3rds of the marbles

Clicker Question: What is the complexity of this algorithm?

A: O(n)
B: **O(n^2)**
C: O(1)
D: O(log n)