A201/A597 Introduction to Programming I

First Summer 2007

Lecture Twelve: Wednesday May 23, 2007 (ED1204)

We continue to review for the exam. Keep coming to class.

1. Solution to yesterday's minute paper.

The problem was to summarize the letters in a given string by indicating how many times each one occurs. Here's a solution that uses the tools we have seen thus far:

```
input = raw_input("Enter your string: ")

reported = ""

for letter in input:
    if letter not in reported:
        print "I see this letter:", letter
        reported = reported + letter
```

This, of course, doesn't solve all the problem, just reports the letters, each letter once.

You notice we keep track of what letters we have encountered so far just like in the hangman program we developed in class on Tuesday.

How could we also count how many times the letter occur?

How about this:

```
input = raw_input("Enter your string: ")

reported = ""

for letter in input:
    if letter not in reported:
        print "I see this letter:", letter, "it occurs",
        reported = reported + letter
        count = 0
        for character in input:
            if character == letter:
                count = count + 1
        print count, "times."
```

This does solve the problem, since for every letter we count the number of times it occurs in the string when we see it the first time, using a second loop (inside the first one).

This is a good solution, but it may have a drawback: it doesn't remember anything.

2. Let's talk about dictionaries.

A dictionary is a set of associations. Each association is a pair: (key, value). The keys in a dictionary are unique: there are no two entries with the same key.

A dictionary is sometimes called an associative array.

Like a real dictionary it resembles a table with two columns. Left column holds the key, right column the corresponding entry. Keys are unique, values don't have to.

You can use a dictionary to store usernames and associated passwords.

```
>>> dictionary = {}
>>> dictionary["dgerman"] = "n0thing"
>>> dictionary = {}
>>> dictionary["dgerman"] = "n0thing"
>>> dictionary
{'dgerman': 'n0thing'}
>>> dictionary["lbird"] = "dribl"
>>> dictionary
{'lbird': 'dribl', 'dgerman': 'n0thing'}
>>> dictionary["dgerman"]
'n0thing'
>>> dictionary["dgerman"] = "s0mething els3"
>>> dictionary
{'lbird': 'dribl', 'dgerman': 's0mething els3'}
>>>
```

In the example above I started with an empty dictionary, and added two entries. I then retrieved one of the entries and changed the other one.

Dictionaries look very much like lists, don't they?

The difference is that they store information not just by indexing with numbers[1].

Minute paper: same as yesterday but use dictionaries to store matrices.

We'll discuss this in class.

Using dictionaries we can solve the problem of yesterday as follows:

a) go through the letters in the input string one at a time
    a.1) if the letter is new create a dictionary entry with a value of 1 (one)
    a.2) otherwise increment the entry value by one each time you see the letter

[1] Try storing an association pair of the kind we stored above (username, password) with lists. You'd have a list of usernames and one of passwords. To find the password for a particular username would then be very cumbersome: given a username you have to find its location in the list of usernames, then use that location to find the corresponding password. The two arrays (or lists) would be parallel. With dictionaries we simply index using the username, which is much more convenient for the programmer.

```
input = raw_input("Enter your string: ")

counts = {}

for letter in input:
    if counts.has_key(letter):
        counts[letter] = counts[letter] + 1
    else:
        counts[letter] = 1

print counts
```

Of course, we could make this program report the outcome in a more expressive way, but printing the dictionary at the end is just as meaningful, since one can easily read the data.

3. Dictionaries as matrices.

Take a look at this, what do you think happens?

```
>>> m = {}
>>> m[0,0] = 2
>>> m[0,1] = 3
>>> m[1,0] = -1
>>> m[1, 1] = 10
>>> m
{(1, 1): 10, (1, 0): -1, (0, 0): 2, (0, 1): 3}
>>>
```

We could use anything as the key (or the value) in a dictionary entry. Here we use tuples for the keys and actual values for the entries. This is a simple way of encoding a 2 by 2 matrix that would be represented as `[[2, 3], [-1, 10]]` in (nested) list representation.

How do we print such a matrix? How do we generate a random size random matrix?

```
import random

rows = int(raw_input("Number of rows: "))
columns = int(raw_input("Number of columns: "))

matrix = {}

for row in range(rows):
    for column in range(columns):
        matrix[row, column] = random.randrange(-20, 20)

for row in range(rows):
    for column in range(columns):
        print str(matrix[row, column]).rjust(3),
    print
```

Both answers are included above: the code first generates a random matrix (whose values are randomly chosen between -20 and 20) then prints it. Note the alignment:

```
>>>
Number of rows: 3
Number of columns: 4
 -1  -5   4 -20
  6 -17  -3 -16
-10  -8  17 -10

>>>
```

What else do we want to do now?

I suppose we could try to get a jumpstart on the lab. Here's one of the extra problems:

```
import random

hei = int(raw_input("height: "))
wid = int(raw_input("width: "))

one = {}

for row in range(hei):
    for col in range(wid):
        one[row, col] = random.randrange(100)

for row in range(hei):
    for col in range(wid):
        print str(one[row, col]).rjust(3),
    print

print "---(that was the first matrix)----"

two = {}

for row in range(hei):
    for col in range(wid):
        two[row, col] = random.randrange(100)

for row in range(hei):
    for col in range(wid):
        print str(two[row, col]).rjust(3),
    print

print "---(that was the second matrix)----"

result = {}

for row in range(hei):
    for col in range(wid):
        result[row, col] = one[row, col] + two[row, col]

for row in range(hei):
    for col in range(wid):
        print str(result[row, col]).rjust(3),
    print
print "---(and this was their sum)----"
```

Lab Eight: Wednesday May 23, 2007 (ED2025)

Do the exact same thing only with dictionaries.

In addition, implement matrix addition using both nested lists and dictionaries.

So that's a total of three problems, more or less.

The matrix addition is implemented above with dictionaries.

I will let you work out the nested list implementation in lab.

Let's discuss the magic square problem instead, and work it out with dictionaries.

1. What's a magic square?

A magic square is a square matrix with n rows and n columns, whose n^2 elements are the integers from 1 to n^2 and arranged such that the sum of the numbers on each row and column, as well as on each of the two diagonals, is the same.

2. How do you check if a square is magic?

Calculate the 2 (n + 1) sums and verify they are the same.

3. What would the code for that look like? Is this it:

```
sum = 0
for row in range(size):
    sum += square[row, row]
print "first diagonal sum:", sum
```

No, this only calculates and reports the sum of the elements on the first diagonal.

We need the same for the second diagonal, then each of the n rows and columns.

```
for row in range(size):
    sum = 0
    for col in range(size):
        sum += square[row, col]
    print "row", row, "sum:", sum
```

The code above reports the sums on each of the n (here, size) rows.

4. How do we create a magic square?

You need to follow the instructions precisely.

Try to see what happens in this program trace:

```
Size: 3
We place  1 and the square becomes:
  0   0   0
  0   0   0
  0   1   0
---------------------
We place  2 and the square becomes:
  0   0   2
  0   0   0
  0   1   0
---------------------
We place  3 and the square becomes:
  0   0   2
  3   0   0
  0   1   0
---------------------
We place  4 and the square becomes:
  4   0   2
  3   0   0
  0   1   0
---------------------
We place  5 and the square becomes:
  4   0   2
  3   5   0
  0   1   0
---------------------
We place  6 and the square becomes:
  4   0   2
  3   5   0
  0   1   6
---------------------
We place  7 and the square becomes:
  4   0   2
  3   5   7
  0   1   6
---------------------
We place  8 and the square becomes:
  4   0   2
  3   5   7
  8   1   6
---------------------
We place  9 and the square becomes:
  4   9   2
  3   5   7
  8   1   6
---------------------
```

Follow the above using the instructions posted yesterday.

Homework Six.

Produce a concise and complete summary of chapters 1, 2, 3 in the text.

Just summarize all that you think is essential and that has helped us in problems so far.