A201/A597 Introduction to Programming I

First Summer 2007

Lecture Fourteen: Tuesday May 29, 2007 (ED1204)

Today and tomorrow we discuss the exam exclusively.

The problems you need to focus on have been listed on Sat May 26 on-line:

1. teach the computer to play the game of nim with the user
2. find/extract the prime factors of a positive integers
3. determine the greatest common divisor of two positive integers
4. report the letter profile of a string received from the user
5. generate random sized random matrix of integers and print it nicely
6. scalable patterns (4, A, C, E, Q, M, W, F, H, K, N, Z, P, R, V, X, Y)
7. teach the computer to play the game of hangman with the user
8. jumble the word (generate a permutation of the user input)
9. sorting a list of integers (bubble sort, selection sort)

You can find this list under What's New? For Sat May 26.

Today and tomorrow in lab and lecture we take these problems one by one.

1. Teach the computer to play the game of Nim with the user.

In this game we start with a pile of marbles and the players remove marbles one by one. The number of marbles removed in one move must be at least one but at most half the size of the pile. The player that brings the pile to a size of one wins the game. (In other words, the player that would have to pick up the last marble loses).

Below we discuss a solution. Make sure you understand there's always more than one way to solve a problem so your solution might differ from mine. No problem, as long as your solution is entirely correct.

Here's a run with my program:

```
What size for the pile: 20
The user moves first.
The height is: 20
How many do you take: 8
Computer's turn. The height is:  12
Computer has moved 6 pieces.
User's turn now.
The height is: 6
How many do you take: 3
Computer's turn. The height is:  3
```

```
Computer has moved 1 pieces.
User's turn now.
The height is: 2
How many do you take: 1
User has won the game because the height is now 1
```

In the interest of efficiency (shorter games when we test the program) I decided to make the size of the pile be determined by the user. So in the game above the pile was 20, the user always starts the game and the game above ends with the user's win.

The computer's moves are random but legal. The user's moves are up to the user. The program checks the user moves and if they are illegal the game is stopped, the user loses.

```
What size for the pile: 20
The user moves first.
The height is: 20
How many do you take: 12
Illegal move, the user has lost.
```

Here's another run, in which the computer wins the game.

```
What size for the pile: 10
The user moves first.
The height is: 10
How many do you take: 2
Computer's turn. The height is:  8
Computer has moved 3 pieces.
User's turn now.
The height is: 5
How many do you take: 1
Computer's turn. The height is:  4
Computer has moved 1 pieces.
User's turn now.
The height is: 3
How many do you take: 1
Computer's turn. The height is:  2
Computer has moved 1 pieces.
Computer has won the game because the height is now 1
```

Here's the code for the program:

```
import random

height = int(raw_input("What size for the pile: "))
print "The user moves first."
while height > 1:
    print "The height is:", height
    user = int(raw_input("How many do you take: "))
    if user <= 0 or user > height/2:
        print "Illegal move, the user has lost."
        break
    height = height - user
    if height == 1:
        print "User has won the game because the "height is now", height
```

```
        break
    print "Computer's turn. The height is: ", height
    computer = random.randrange(1, height/2 + 1)
    height = height - computer
    print "Computer has moved", computer,"pieces."
    if height == 1:
        print "Computer has won the game because the height is now", height
        break
    print "User's turn now. "
```

I have changed the fonts a bit in the two long lines to keep the original format.

Program synopsis: the program starts by determining the height. Then for as long as the height is meaningful the user moves and the computer moves. Each one of the two has the ability to break the loop and stop the game: the user either by a) winning or b) losing through an illegale move, the computer by winning. Notice that the condition at the top of the while loop really is significant only once, in the beginning. After that the check is superfluous, since the loop ends through one of three breaks and not through this check. Inside the loop the user's move is taken from the user. If it's illegal the game stops. If it's OK we check for user win, in which case the game ends. Otherwise the computer generates a random legal move and we check to see if it was a winning move. If so, the game ends. Otherwise the game continues.

Read the lab notes for additional details. In class we will review these step by step.

2. Find/extract the prime factors of a positive integer

```
Give me a number: 64
2 ** 6
Give me a number: 72
2 ** 3
3 ** 2
Give me a number: 342723
3 ** 1
4967 ** 1
23 ** 1
Give me a number: 4967
4967 ** 1
Give me a number: 13
13 ** 1
Give me a number: 322365
3 ** 1
5 ** 1
21491 ** 1
Give me a number: 1800
2 ** 3
3 ** 2
5 ** 2
Give me a number: bye.
```

As you can see this program reports the exponents as well and can serve as a prime number detector (see the cases of 4967, 21491 above).

Let's review the answers above just to be sure we understand the way the program works.

a) $64 = 2^6$
b) $72 = 2^3 * 3^2$
c) $342723 = 3^1 * 4967^1 * 23^1$
d) $4967 = 4967^1$ (prime number)
e) $13 = 13^1$ (prime number)
f) $322365 = 3^1 * 5^1 * 21491^1$
g) $1800 = 2^3 * 3^2 * 5^2$

The order of the factors is arbitrary (we will see in a minute why).

My code is as follows:

```
number = raw_input("Give me a number: ")
while not number == "bye":
    number = int(number)
    if number <= 0:
        print "The number is not good, I need a positive integer."
        number = raw_input("Give me a number: ")
        continue
    factors = {}
    check = 2
    while check <= number:
        if number % check == 0:
            if factors.has_key(check):
                factors[check] += 1
            else:
                factors[check] = 1
            number /= check
        else:
            check += 1
    for factor in factors.keys():
        print factor, "**", factors[factor]
    number = raw_input("Give me a number: ")
```

Program synopsis: ask the user for a number and if the user doesn't type "bye" we convert (parse) the entry to an integer and check it to make sure it's a positive integer. We then define a dictionary (factors) to store the factors as keys and their exponents as values. We start with 2 as the lowest possible prime factor and keep dividing the number by it until it doesn't divide the number any more. Each time a perfect division is noticed the number becomes smaller, the entry in the factors dictionary is checked and either a) initialized with 1 if this is the first time we encountered this factor, or b) increased by 1 if the entry already exists in the dictionary. When the number becomes smaller than the potential prime factor with which we are checking for divisibility, the process ends. As an example let's take 35. Check 35 against 2, 3, 4, 5 and the first that works is 5. Make an entry for 5, with a value of 1 and the number becomes 7. Note that what's left is always bigger than the number we're working with since we reveal the prime factors in order. Now 7 is not divisible by 5, but is still bigger. So we try with 6, it doesn't work, then with 7. This makes the number 1, adds a new entry in the dictionary and we need to stop since

the factor we're working on (7) is now less than the number itself (1). The number will always reach 1 when the program ends. Convince yourself by working out (see above) the prime factors of 342723. You know the answer from the run above. Once the process of factorization ends we print the dictionary in some fashion and the loop is restarted with a new input which can be "bye" or another number.

3. Determine the greatest common divisor of two positive integers.

```
>>>
Enter the first number: 2
Enter the first number: 3
gcd( 2 , 3 ) = 1
Enter the first number: 12
Enter the first number: 18
gcd( 12 , 18 ) = 6
Enter the first number: 28
Enter the first number: 35
gcd( 28 , 35 ) = 7
Enter the first number: 13
Enter the first number: 19
gcd( 13 , 19 ) = 1
Enter the first number: 56
Enter the first number: bye
Thank you for using this program.

>>>
Enter the first number: bye
Thank you for using this program.

>>>
```

Here's my code:

```
while True:
    one = raw_input("Enter the first number: ")
    if one == "bye":
        print "Thank you for using this program."
        break
    else:
        one = int(one)
        two = raw_input("Enter the first number: ")
        if two == "bye":
            print "Thank you for using this program."
            break
        else:
            two = int(two)
            for divisor in range(min(one, two), 0, -1):
                if one % divisor == 0 and two % divisor == 0:
                    print "gcd(", one, ",", two, ") =", divisor
                    break
```

Note: the core of the problem is shown in brown. We could easily make a function out of it, but functions are optional for this test so use them only if you need them.

Program synopsis: the program happens in an infinite loop that asks for the first number, then for the second and then computes the greatest common divisor of the two. If either one of the two numbers (meant to be positive integers) reads “bye” we break the infinite loop and the program stops. Otherwise a loop starting with the smaller of the two integers keeps checking numbers one by one, down to 1, until a number acts as a divisor. Worst case is when the two are relatively prime, in which case the last divisor checked is 1 and 1 will always match. Notice how the break statement always breaks the closest enclosing loop: the break in the for breaks it, which lets the enclosing while continue.

4. Report the letter profile of a string received from the user.

The problem is to summarize the letters in a given string by indicating how many times each one occurs. Here’s a solution that uses the tools presented up to and including ch. 4:

```
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
```

Let’s see this in action:

```
Enter your string: simple test
I see this letter: s
I see this letter: i
I see this letter: m
I see this letter: p
I see this letter: l
I see this letter: e
I see this letter:
I see this letter: t
```

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 Tue May 22.

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."
```

Let's see it in action:

```
Enter your string: simple test
I see this letter: s it occurs 2 times.
I see this letter: i it occurs 1 times.
I see this letter: m it occurs 1 times.
I see this letter: p it occurs 1 times.
I see this letter: l it occurs 1 times.
I see this letter: e it occurs 2 times.
I see this letter:   it occurs 1 times.
I see this letter: t it occurs 2 times.
```

This does solve the problem completely, 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 (as far as number of occurrences of letters is concerned). We could use dictionaries to keep track of occurrences as follows: go through the letters in the input string one at a time and a) if the letter is new create a dictionary entry with a value of 1 (one) otherwise b) increment the entry value by one each time you see the letter. In short:

```
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
```

In action:

```
Enter your string: simple test
{'p': 1, 's': 2, 'e': 2, 't': 2, ' ': 1, 'i': 1, 'm': 1, 'l': 1}
```

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.

5. Generate random sized random matrix of integers and print it nicely.

This was discussed in class on Wed May 23 and Thu May 24 respectively.

There are two options, both are acceptable:

a) store matrices as nested lists

```
import random

rows = int(raw_input("Height: "))
cols = int(raw_input("Width: "))

m = []

for row in range(rows):
    line = []
    for col in range(cols):
        line.append(random.randrange(100))
    m.append(line)

for row in range(rows):
    for col in range(cols):
        print str(m[row][col]).rjust(3),
    print
```

b) use dictionaries to store matrices

```
def show(matrix, height, width):
    for row in range(height):
        for col in range(width):
            print str(matrix[row, col]).rjust(3),
        print
    print "-----------"

import random

size = int(raw_input("Size: "))

m = {}

for row in range(size):
    for col in range(size):
        m[row, col] = 0

show(m, size, size)
```

Note: you don't need to use functions but you're welcome to.

Notice the two programs above look identical when you run them.

Minute paper: for each lecture program we will ask a starting question.

Labs will be a collection of extended minute papers (see lab notes below).

Lab Ten: Tuesday May 29, 2007 (ED2025)

The lab assignment for today is to attempt to develop on your own each of the problems discussed in the lecture notes. This makes today's lab a mock exam.

Use it to determine exactly where you are in your review of these problems. An exactly similar assignment (for the rest of the problems) is scheduled for tomorrow's lab. Details of how the labs will be run will be provided in lab. Likely we will ask you to split the two hours in units of 15-20 minutes during which you should be working on just one problem.

Both Adrian and Metal will be present, so the lab will be staffed such as to allow you to get help while you plan your test programs. You may check your notes but only when you are sure you can't produce anything on your own and even then you shouldn't type code straight from them: you should look up one thought or idea, then close your notes, then try to implement that idea by yourself.

Note that the exam will be closed book, two problems, randomly distributed. The labs are to be an exercise, a simulated exam to give you feedback for additional review. In class we will discuss the problems, and the discussion might overflow into the lab time. Perhaps we will distribute the problems randomly to you in lab as well.

For full credit for today and tomorrow you need only attempt all the problems and turn them in. The problems don't have to be complete, they only need to be attempted with honesty. How else can you find out what you're missing to do well on the exam, if not by trying to see how far you can go on your own, under our guidance.

There are 12 lab assignments per semester (summer session). Today's and tomorrow's are lab assignments 10 and 11; their purpose is to prepare you for the exam. Lab 12 (last) is the one on Thu. Its purpose is to ensure that you grade your own exam immediately after the class, giving you a chance to catch any mistakes and fix them.

Homework assignment(s).

No homework assignments will be issued this week.

There are 10 homework assignments per semester (summer session).

Seven have been assigned already. Three more will be assigned next week.

Thus, the last packet of homework assignments is coming to you next week.

It will be due Tue Jun 12, the week of the final.

There are no lab assignments next week, only a practical exam.

Details about the practical will be posted Friday June 1st (after the midterm).