COSC 101 Homework 4 Fall 2014

Due date: **Wednesday, October 1, 11:55pm**

This homework will improve your skills with **for** loops and **if** statements by having you develop programs that use loops and conditionals to repeat code patterns that are similar but not exactly the same. The first part is a continuation of this week's lab; the remainder consists of a couple of exercises that iterate over a string, one of which requiring you to use string indexing.

Retro Font

1. In this question you practice breaking down a problem into manageable parts, using for loops to map numbers to patterns and using a conditional to alter the paint color.

   As in the lab, use the Python module `block_paint` to implement two letters of a *scalable* retro font, similar to the fonts created by illuminating selected light bulbs on marquee displays. The `block_paint` module is included in the homework zip file.

   Review the description of lab 3 for information about `paint(x, y, color)` and `end()` methods of the `BlockPaint` object.

   In `hw4_letters.py`, you first ask for the size, in pixels, of the letter. You can assume the user will enter an integer, but you should check it is odd and greater than or equal to 5. If the integer is invalid, print a message saying so, and exit.

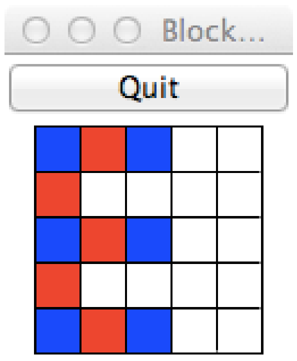
   You should then write code to draw two letters: E and X.

   Improving on the lab two colors are used to paint each letter. When painting blocks for each letter, the color used should *always* be `'red'` if the sum of the coordinates is odd. Otherwise, any other color may be used.

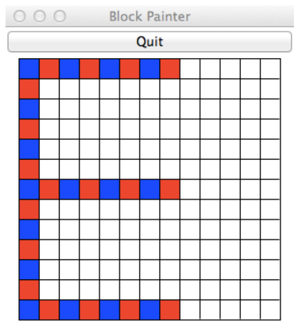
   E The letter E should be anchored at the left side of the window, with the width of the letter extending to 2/3 the width of the window. For example, if the size is 5, the width of the E should be $5 * 2/3 = 3$; if the size is 13, the width of the E should be $13 * 2/3 = 8$.

   You should have **no more than 2 for** loops in your code to draw the E. Note that the colors alternate between red and your other color depending on the coordinates.

Letter E at scale 5

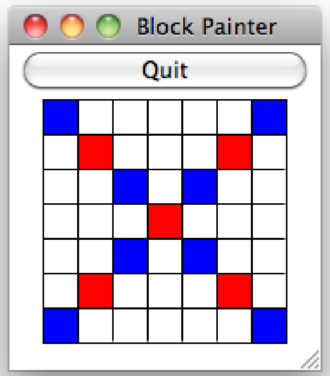


Letter E at scale 13

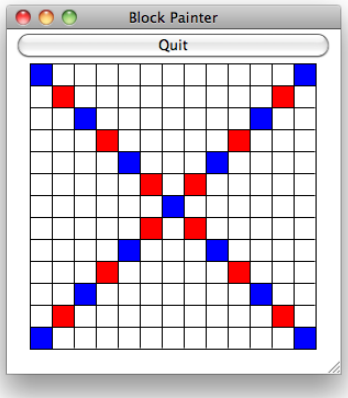


X The letter X should extend to the corners of the window.
You should use **only 1 `for`** loop in your code to draw the X.

Letter X at scale 7



Letter X at scale 13



String Manipulations

2. One suggestion to build a satisfactory password is to use the initial letter of a memorable sentence, the more personal the better, making sure that the result is not a dictionary word[1].

   In the file `hw4_pass.py`, implement a program that asks the user for a sentence, then build the password. The program then prints the string formed by the first letter of each word. The first letter is defined by any character that follows a space, i.e. the character `' '`.

   Examples:

```
Enter a phrase: Why should you care what other people think?
The first letters from each word form the word Wsycwopt


Enter a phrase: I have no  ideas ...
The first letters from each word form the word Ihn i.
```

   You are required to use string indexing to solve this question.

[1]Using a combination of upper-case and lower-case letters and replacing some of the letters with symbols should be done to strengthen the password.

3. Valid internet email addresses must have a particular format, their sequences of characters must follow a specific pattern. Forms on the web often check if email addresses are valid. This exercise imitates such validation, but is a simplification of the reality. In the file `hw4_email.py` you will write a program that takes apart a string to determine the validity of an email address.

   A valid email address consists of a *local* part, followed by `'@'`, followed by a *domain* part. We define an email address to be valid if the string has the following three properties.

   1. One `@` is present (zero or more than one `@` are invalid email addresses).
   2. At least one character forms the local part.
   3. At least one character forms the domain part.

   For example:

String	Valid?	local	domain	top-level
`user@localserver`	Yes	`user`	`localserver`	`''`
`niceandsimple@example.com`	Yes	`niceandsimple`	`example.com`	`com`
`Abc.example.com`	No	–	–	–
`A@b@c@example.com`	No	–	–	–
`@host`	No	–	–	–

   Your program first asks the user to enter an email address and then returns if the string entered is a valid email address or not.

   If it is valid the program should also print the following information:

   - the local part,
   - the domain part,
   - if it is from the colgate domain, i.e., the domain part is equal to `colgate.edu` and
   - if it is not from the colgate domain the top-level domain. The top-level is part of the domain and is
     - the characters after the last occurring dot, if a dot exists or
     - the empty string if there is no dot.

   Think about how to break down the problem into manageable parts so as to implement and test one at the time. You should build your program incrementally.

   You can use two **for** loops (the second can be useful to determine the top-level string to print). You are not required to use string indexing but we recommend using string indexing as it makes the program easier to write.

   Your program's output should look **exactly** like the examples provided below.

```
Enter an email: efourque@colgate.edu
Valid email
Local part: efourquet
Domain part colgate.edu
From COLGATE!


Enter an email: ef@col@gate.edu
Invalid email

Enter an email: @colgate.edu
Invalid email

Enter an email: joe@
Invalid email
```

```
Enter an email: joe
Invalid email


Enter an email: e@colgate.com
Valid email
Local part: e
Domain part colgate.com
Not a colgate email...
From top-level domain: com


Enter an email: efourquet@colgate.ca
Valid email
Local part: efourquet
Domain part colgate.ca
Not a colgate email...
From top-level domain: ca


Enter an email: e@colgate
Valid email
Local part: e
Domain part colgate
Not a colgate email...
From top-level domain:
```