Computer Science — Activity—Introduction to Graphics

ASSIGNMENT OVERVIEW

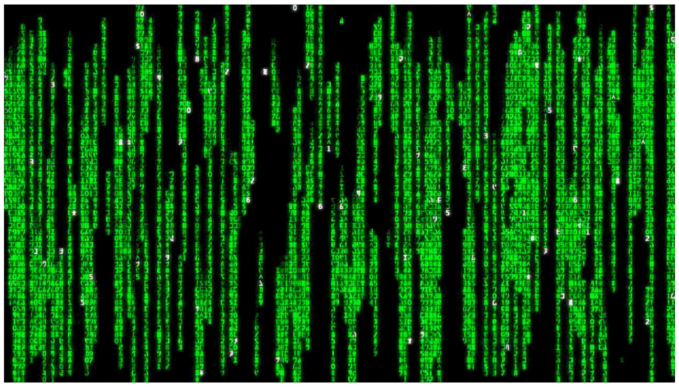
In this assignment you'll be writing a series of small programs that present information graphically on the computer screen, and learning about the coordinate systems used by the computer.

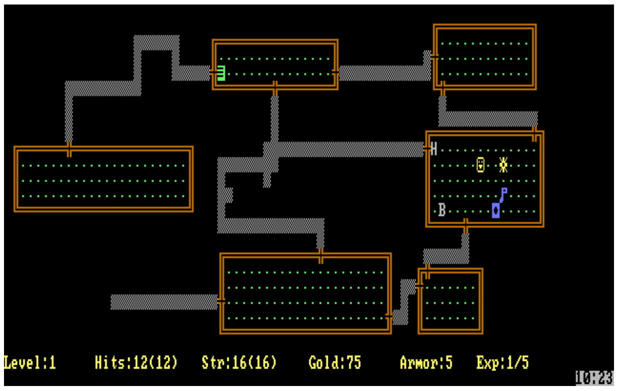
This assignment is worth 30 points and should be uploaded by the date given in class.

BACKGROUND

There are two ways that a computer can display information—data, a diagram, or an image —"graphically" on the computer screen: using text, and using pixels. Although text-based graphical display of information has been in use from the very beginning of computing, the first personal computer with a pixel-based graphical user interface (GUI) was the original Macintosh Computer, released in 1984.

Although pixel-based graphical displays are currently the most dominant graphical presentation mode in most people′s experience, you may have occasion to interact with a text-based display as well. Here are two examples:





ASSIGNMENT SPECIFICATION

Write three programs that will display take a series of values and display them in graphical form:

1. `horz_bar`
   This program begins by defining a list/array of 10 values, and then displays them as a series of horizontal bars represented by text characters. The bars are proportional to the magnitude of the value.
2. `vert_bar`
   Takes a list/array of 10 values and displays them as a series of vertical bars represented by text characters.
3. `vert_graph`
   Takes a list/array of 10 values and displays them as a series of vertical bars represented by rectangles.

DELIVERABLES

`horz_bar.py`, **`vert_bar.py`**, and **`vert_graph.pyde`**

These three files will be developed by you based on details in this document. To submit your assignment for grading, copy your files to your directory in `/home/studentID/forInstructor/` at *crashwhite.polytechnic.org* before the deadline given in class.

ASSIGNMENT NOTES

- `horz_bar` is a relatively straightforward program to write: given the values in a list, print a number of horizontal characters (use a `*` ?) that is proportional to the magnitude of each value. Begin by using nested loops to go through the values and printing a number of asterisks that corresponds *exactly* to each value in the list (see sample output). Then...

- Devise a formula that will print out a number of asterisks that is *proportional* to the relative magnitude of the values. You'll need to assume a certain screen width (80 characters is a standard value), and you'll need know the largest number in your list of values so that you can calculate the appropriate scale. Store those scaled values in a new array and print the asterisk bar graph based on those values.

- Scaling example: if we have a list of values
  ```
  values = [30, 60, 1, 120, 150, 180, 210, 270, 300, 240]
  ```
  and the width of the screen is just 80 characters, we need to calculate a new value where the maximum value in our list has the maximum width (80 characters). Consider this as a proportion, where the scaled value can be calculated as a fraction of the width of the screen:
  ```
  scaled_values[i] = values[i] / max(values) * width
  ```

- `vert_bar` uses the same strategies to determine the scaled output, but... how do we tilt our graph sideways so that the bars become columns? Let's say that we've assumed a certain screen height for the display (I'm assuming 12 here), and using that, I've scaled output to produce the list
  `scaled_values = [1, 2, 0, 4, 6, 7, 8, 10, 12, 9]`
  We know that the vertical bar graph we want to produce should look something like this. (I've included the numeric values for each column below the graph—you don't need to print those in your output.)

```
        *
        *
       **
       ***
      ****
     *****
    ******
    ******
   *******
   *******
 * *******
** *******
1204671198
      02
```

- `vert_bar` is a little bit tricky because we're trying to print a vertical graph from the top down. We'll need to set up a loop that runs from the highest value in our list (use Python's `max()` function to determine what that is) down to the lowest. Then we'll set up a loop to move across the columns (the values in our list). For each value, if a given column has a value that is equal to or greater than the row we're printing, we'll print an asterisk for it. If not, we'll print a blank space there so we can continue across this row, filling in the other columns as needed.)
  So, if the height is 12, we create one row-loop that sets `row` from 12 to 1 (backwards), and a second nested column-loop that goes through every item in my scaled values:

```
for i in range(len(scaled_list)):
    if scaled_list[i] >= row:
        print("*",end='')
    else:
        print(" ",end='')
```

- The graphical version of the vertical bar graph, `vert_graph`, relies on the same kind of logic as `vert_bar` used, although now we're working at the precision-level of a pixel on the screen. Using graphics with computers is necessarily more complex because different types of hardware—the monitor in this case—work differently. The drivers for your local computer manage most of the heavy lifting here, but it's for this reason that graphics programs have to run locally, on your own machine. The course server won't be able to display graphics output for you.
- We'll use the Processing system for displaying our graphics. This assignment assumes you have some facility with that.
- Let's begin by assuming the same list of values that we used above:
  `values = [30, 60, 1, 120, 150, 180, 210, 270, 300, 240]`
  We'll need to produce a *scaled* set of values so that the columns we draw fit nicely on the screen. Before when we did this we used a *height* based on the number of character lines I had available to me (12). Now, I've got *hundreds* of pixels available for drawing, so I'll need to change my scale calculation based on the height of the window. I'll say that my window is going to be 800 pixels wide-by-600 pixels tall. So, the height of my window is 600 pixels, or maybe a little less so I can leave room for axes and labels later on.

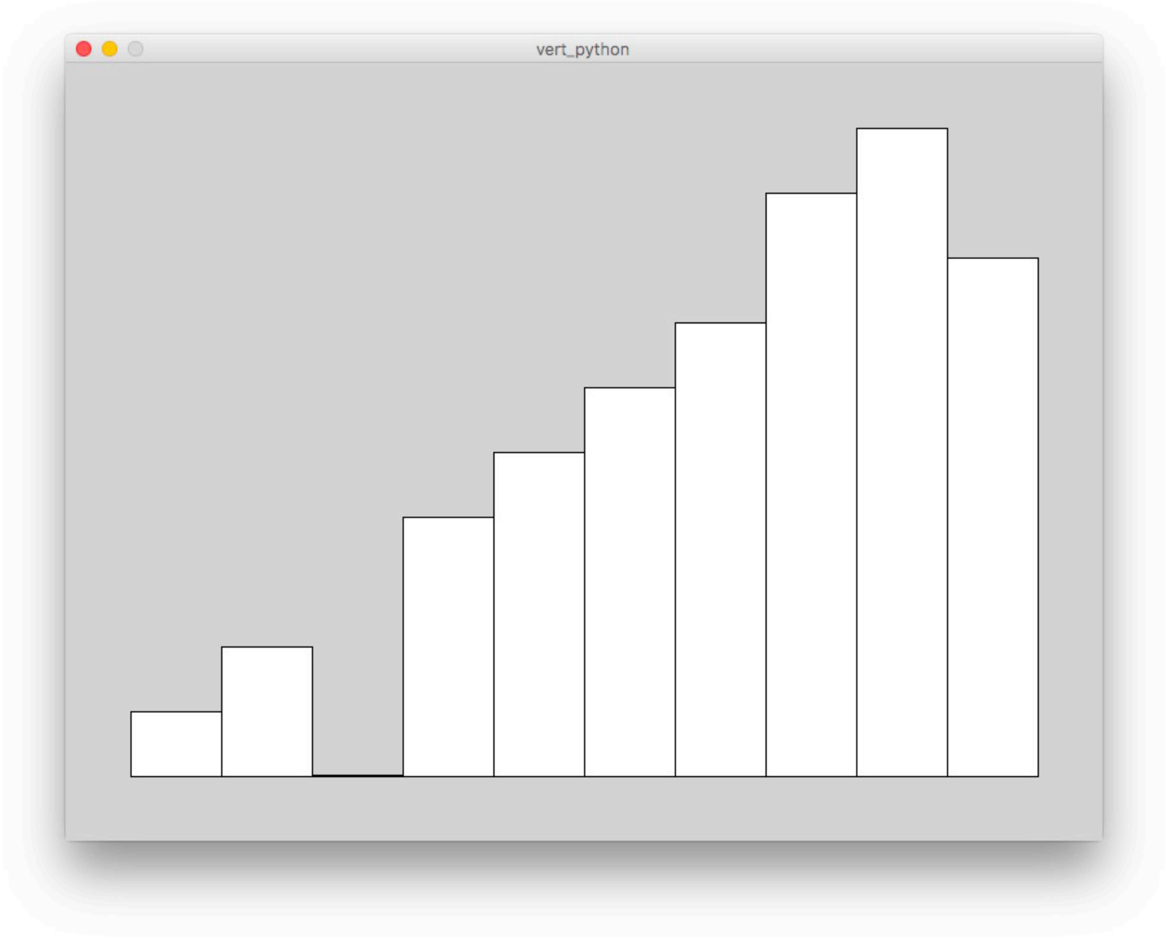
```
height = 500
values_scaled = []
max_value = max(values)
for i in range(len(values)):
    values_scaled.append(int(height * values[i] / max_value))
```

  With a height of 500 pixels, this produces the result:

```
scaled_values = [50, 100, 1, 200, 250, 300, 350, 450, 500, 400]
```

  I know how wide each bar should be too, right? There are eleven data points, and the width of the screen is 800. I'll take 100 off for the borders, so I've got 11 bars in 700 pixels, = 63.6 pixels width per bar.

- We know what that bar graph should look like, yes? It's a good idea to draw this out on graph paper, keeping in mind the size of the window and the fact that the *x-y* position (0, 0) is in the upper left corner of the window. Here's what I drew:



In Processing, each rectangle is defined in terms of the *x-y* coordinates of its upper left corner, and the width and height of that rectangle, and this makes the loop a little more tricky...

Take a look at this code and see if you can figure out how it works, based on the graph shown above:

```
bar_width = (width - horz_margin * 2)/len(values_scaled)
x = horz_margin
for i in range(len(values_scaled)):
    y = height - values_scaled[i]
    rect(x, y - vert_margin, bar_width, values_scaled[i])
    x += bar_width
```

GETTING STARTED

- Set up a directory that you can use to store the files that you'll be working with in this assignment.
- The three programs in this assignment are designed to be completed in order, so that you can start thinking about the spacial relationships of characters (and then pixels). Make sure you have a good understanding of each program before moving on to the next one.

EXTENSIONS

- The column-based bar graph is fine as far as it goes, but graphs are worth nothing without a scale. Add code to your `vert_graph` program so that there is an appropriate vertical scale that accompanies the graph.

QUESTIONS FOR YOU TO CONSIDER (NOT HAND IN)

1. How many pixels wide and high is the display on your computer? How many total pixels is that?
2. How man "pixels per inch," or "dots per inch" (dpi), does your computer display have? How does this compare with the display on your mobile phone?
3. Just about everybody appreciates the detail of pixel based images, but writing graphics-based programs is challenging. What are the benefits of writing graphics-based programs? What are some of the challenges?
4. Do a search for the keywords "curses" and "terminal." What does the curses programming library allow you to do that can't be done in a standard terminal?