Homework 3: Yet More Pictures

CIS 252 ✯ Introduction to Computer Science

Administrivia

Notes This homework is based on the first four chapters of Haskell: The Craft of Functional Programming (HCFP).

You may use any code you want from the lectures. However, you should include a note in your comments indicating that you are doing so and specifying which functions you are reusing.

You will again need the file http://www.cis.syr.edu/courses/cis252/code/Pictures.lhs to use the Pictures module. Do not add your code to this file, but (as before) include an import Pictures directive at the top of your file.

You should follow the design recipe and documentation requirements with one exception: You don’t need to include examples or test cases in your program (they’d be way too big). Instead, simply write “See the Homework 3 write-up for examples” in the EXAMPLES part of your comments. Also see http://www.cis.syr.edu/courses/cis252/info/hw-policy.html#crit:hw for the grading criteria for homeworks.

You may work singly or in pairs on this assignment.

What to turn in You should turn in a hard copy of your source code and a transcript demonstrating convincingly that your code is correct. Be sure to send in your electronic version of lines.hs per http://www.cis.syr.edu/courses/cis252/info/hw-policy.html.

Due This is due in the bin in CST 3-212 by noon on Friday, February 8.

Exercises

Exercise 1. Write a Haskell function

altThree :: Int -> Picture -> Picture -> Picture -> Picture

such that altThree n pic1 pic2 pic3 returns a row of n pictures in which pic1 appears first, followed by pic2, followed by pic3, followed by pic1, followed by pic2, followed by pic3, followed by pic1, and so on.

Exercise 2. Write a Haskell function

indexedRow :: Int -> Int -> Picture -> Picture -> Picture -> Picture

such that indexedRow c n pic1 pic2 returns a row of n pictures: pic1 is at index c, and pic2 occurs everywhere else. Your code should raise an error if either c < 1 or n < 2. Note: You may assume that c is never greater than n.

Hint: Think about the cases of (i) c = 1, (ii) c = n, and (iii) 1 < c < n. The functions row, column, and side3 from class may be very helpful.

Exercise 3. Write a Haskell function

diagonal :: Int -> Picture -> Picture

such that diagonal n pic returns an n × n grid of copies of the picture pic: all occurrences of pic along the main diagonal (i.e., from the top left to the bottom right) should have their color inverted. Your code should raise an error if n is less than 1.

Your code should raise an error if n is less than 1.

Hint: Think about the cases of (i) n = 1, (ii) n = 2, and (iii) n > 2. Draw the pictures!

Exercise 4. Write a Haskell function

steps :: Int -> Picture -> Picture

such that steps n pic returns an n × n grid of copies of the picture pic: all copies of pic along the secondary diagonal (as well all copies to the right of and below that diagonal) should be inverted. Your code should raise an error if n is less than 1.

(Note: The secondary diagonal goes from the bottom left to the top right.)

Final plea: Before sitting down to bang out code, first think out your solutions with paper and pencil — especially for the last problem. Also, if any of your functions is longer than about five lines (not counting helper functions), then you’re likely making things more complicated than need to be. Simple and direct generally works.
### Definitions of blank, block, blob, and exx

<table>
<thead>
<tr>
<th>blank, blob, block, exx :: Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank = [&quot;.....&quot;, &quot;.....&quot;, &quot;.....&quot;,&quot;.....&quot;, &quot;.....&quot;]</td>
</tr>
<tr>
<td>block = invertColour blank</td>
</tr>
<tr>
<td>blob = [&quot;.....&quot;, &quot;.....&quot;, &quot;.....&quot;, &quot;.....&quot;, &quot;.....&quot;]</td>
</tr>
<tr>
<td>exx = [&quot;#...#&quot;, &quot;.#.#.&quot;, &quot;..#..&quot;, &quot;.#.#.&quot;, &quot;.#...#&quot;]</td>
</tr>
</tbody>
</table>