Problem Solving for ICS

February 10, 2016

Language: Intermediate Student with Lambda

In the first tutorial, you saw how to gain access to a library containing functions to manipulate images, with the expression:

```
(require 2htdp/image)
```

The documentation for this library is at Racket Documentation -> How To Design Programs Teachpacks -> 2.3 "image.rkt".

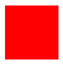
The file `checkerboard.rkt`, emailed to you, had examples of the use of some functions in this library, including `square` and `circle` for drawing simple geometric figures, and `beside` and `above` for combining images.

In this problem-solving session, you will use these and any other functions from the image library to create images that are beautiful, interesting, and/or unusual. Your code must satisfy the following requirements:

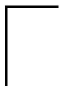
- **At least one function that produces an image must use recursion (preferably structural) on unbounded natural numbers, or unbounded lists, or both.**
- Your code must contain purpose and contract for every function. The type of an image is `Image`.
- For functions that produce images: instead of tests, your code should contain expressions that demonstrate the use of your functions by displaying example images in the Interactions window when the program is run. Each image-valued expression should follow a string that describes what created the image. For example:
  ```
  "Here is the result of (circle 20 'solid 'red)."
  (circle 20 'solid 'red)
  ```
- For functions that produce numbers, strings, symbols, Booleans, or lists or structures containing these, you should provide examples and tests as in the design recipe.

You can look up functions from the image library in the Racket documentation, but to save you reading it all, here are some that you may find useful (there are many others), described by example:
Examples:

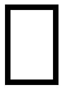
```
> (square 20 'solid 'red)
```



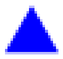
```
> (rectangle 20 30 'outline 'black)
```



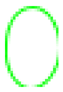
```
> (rectangle 20 30 'outline (pen 'black 4 'solid 'round 'round))
```



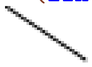
```
> (triangle 20 'solid (color 0 0 255))
```



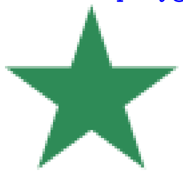
```
> (ellipse 20 30 'outline 'green)
```



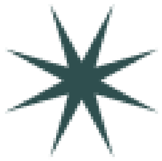
```
> (line 30 20 'black)
```



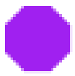
```
> (star-polygon 40 5 2 'solid 'seagreen)
```



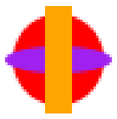
```
> (radial-star 8 8 32 "solid" "darkslategray")
```



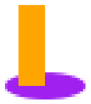
```
> (regular-polygon 10 8 'solid 'purple)
```



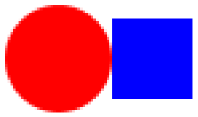
```
> (overlay (rectangle 10 40 'solid 'orange)
           (ellipse 40 10 'solid 'purple)
           (circle 17 'solid 'red))
```



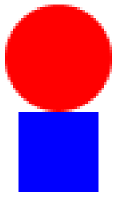
```
> (overlay/offset (rectangle 10 30 'solid 'orange)
                  5 15
                  (ellipse 30 10 'solid 'purple))
```



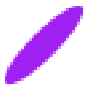
```
> (beside (circle 20 'solid 'red) (square 30 'solid 'blue))
```



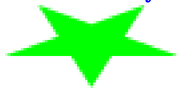
```
> (above (circle 20 'solid 'red) (square 30 'solid 'blue))
```



```
> (rotate 45 (ellipse 40 10 'solid 'purple))
```



```
> (scale/xy 1 1/2 (flip-vertical (star 40 'solid 'green)))
```



The names of predefined colours can be found at Racket Documentation -> The Racket Drawing Toolkit -> 7 Color Database.

Finally, if you find that your images take a long time to draw, the `freeze` function might help you. Images are represented during computation as a sequence of actions which are only done when the image must be displayed. If you use a subimage many times, it will be redrawn each time, in which case you might want to use `freeze` to draw it once and then have the drawing (bitmap) copied. This is similar to using `local` to avoid recomputing expressions whose values are used many times.

To submit: `ICS.rkt`