CS 111 - Homework 1

Deadline

11:59 pm on **Friday, September 2, 2016**

How to submit

Each time you would like to submit your work:

- save your current DrRacket Definitions window contents in a file with the name `111hw1.rkt`
  - Note: please use that *exact* name -- do not change the case, add blanks, etc. If I search for a file with that name in your submission, I want it to show up! 8-)
- transfer/save that file to a directory on `nrs-labs.humboldt.edu`
  - If you are in a campus lab, you can do so by copying them to the `U:` drive
  - If you are not in a campus lab, you can do so by using `sftp` (secure file transfer) and connecting to `nrs-labs.humboldt.edu`, and then transferring them.

    Graphical versions of `sftp` include WinSCP and Secure Shell Transfer Client for Windows, Cyberduck and Fugu for Mac OS X, and FileZilla that has versions for both Windows and Mac OS X. (Mac OS X and Linux also come with a command-line `sftp` -- an intro to this is included in the posted handout, "useful details - `sftp`, `ssh`, and `~st10/111submit`", available on the public course web site in the References section.)
  - Whichever version of `sftp` you are using, use a **host name** of `nrs-labs.humboldt.edu` if you are not in a campus lab (you can get away with just `nrs-labs` for the host name in a campus lab), your campus username, and when you are prompted for it, your campus password. IF you need to give a port number, give a port number of **22**.
- Now that your file is ON nrs-labs.humboldt.edu, you need to log onto nrs-labs.humboldt.edu using `ssh`, so you can submit your file to me.
  - In BSS 313, the Windows computers have a graphical version of `ssh` called PuTTY. There is a graphical version of `ssh` for Mac OS X named Jelly FiSSH, (although Mac OS X and Linux also come with command-line `ssh`, and an intro to this is also included in the "useful details - sftp, ssh, and ~st10/111submit" handout.)
  - Again, whichever version of `ssh` you are using, use a **host name** of `nrs-labs.humboldt.edu` if you are not in a campus lab (you can get away with just `nrs-labs` for the host name in a campus lab), your campus username, and when you are prompted for it, your campus password. IF you need to give a port number, give a port number of **22**.
- WHILE you are logged onto nrs-labs:
  - IF you saved your file in a folder, use `cd` to change to the folder/directory where you saved it -- for example, if you saved it in the folder `111hw1`, then you would go to that directory by saying:

    ```
    cd 111hw1
    ```
  - use the `ls` command to make sure your `111hw1.rkt` file is really there:

    ```
    ls
    ```
  - type the command:

```
~st10/111submit
```

...and when asked, enter a homework number of `1`

...and when asked, enter `y`, you do want to submit all files with an appropriate suffix (I don't mind getting some extra files, as long as I also get `111hw1.rkt`; I'd rather receive too many files than too few due to typos.)

...make sure to carefully check the list of files submitted, and make SURE it lists `111hw1.rkt` as having been submitted! (The most common error is to try to run `~st10/111submit` while in a different directory than where your files are...)

- (we practiced the above during the Week 1 Lab, on Friday, August 26 -- ASK ME if this is still not clear, or if you have any problems with submission! If you get stuck, E-MAIL me your `111hw1.rkt` file as an e-mail attachment before the deadline, and then submit it as soon after that as you are able.)
  - I am happy to walk through submitting files with you -- just come by my office during office hours, or e-mail me to set up an appointment.

Purpose

To practice with simple and compound expressions of various data types in Racket.

Important notes

- Along with this homework handout is another handout, "Some DrRacket Tidbits". This includes helpful reminders, a few additional operations, and additional features we'll be talking about and using in the nearish future.
- Remember that, within a compound expression, the expressions after the operation are called **arguments**. So, in the compound expression:

  ```
  (* 4 5 7 2 15)
  ```

  ... the operation `*` happens to have five arguments, each of type number, and in:

  ```
  (circle 45 "solid" "blue")
  ```

  ... the operation `circle` has three arguments, the first of type number, the second of type string, and the third of type string.
- You'll see the term "**signature**" in the descriptions of image operations in the "Some DrRacket Tidbits" handout. A signature shows the **number** and **type** of arguments expected by an operation, and the **type** of expression that results from that operation. For example,

  ```
  ; signature: circle: number string string -> image
  ```

  ...is a short-hand way of saying that the `circle` operation expects three arguments, of type number, then string, and then string, and it results in an image.

  The purpose statement underneath each signature further **describes** what the operation expects and what it results in.
- We said that a Racket compound expression's syntax is an opening parenthesis, an operation and one or more expressions separated by white space, and a closing parenthesis:

  ```
  (operation expression ...)
  ```

  Notice that we never said that the argument expressions had to be simple expressions! They may be simple

OR compound! So, each of the following is a syntactically-correct Racket compound expression:

```
(+ 1 3)
(/ 7 12)
(* (+ 1 3) (/ 7 12))     ; multiply (the sum of 1 and 3) and
                         ;          (the quotient of 7 divided by 12)

(+ (* (+ 1 3) 27) 3.4)   ; add (the product of
                         ;             (the sum of 1 and 3) and 27)
                         ;     to 3.4
```

Problem 1

You will complete all of Homework 1's problems in a file named `111hw1.rkt` (that is, save your DrRacket Definitions window to a file named `111hw1.rkt`).

Start up DrRacket, setting the language to How To Design Programs - **Beginning Student** level and adding the HTDP/2e versions of the image and universe teachpacks by putting these lines at the beginning of your Definitions window:

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

Put a blank line, and then type in:

- a comment-line containing your name,
- followed by a comment-line containing `CS 111 - HW 1`,
- followed by a comment-line giving the date you last modified this homework,
- followed by a comment-line with no other text in it --- for example:

```
; type in YOUR name
; CS 111 - HW 1
; last modified: 2016-08-29
;
```

Below this, after a blank line, now type the comment lines:

```
;
; Problem 1
;
```

Now type the comments and expressions specified below in the Racket Definitions window. Run (push the Run button) and look in the Interactions window (in the lower window) to see if you get the expected results for each.

1 part a

Type the comment:

```
; 1 part a
```

...and after a blank line type in a **simple** expression of type **number** representing the number 643.25.

1 part b

Type the comment:

```
; 1 part b
```

...and after a blank line type in a **compound** expression of type **number** representing the product of 47 and 0.038.

1 part c

Type the comment:

```
; 1 part c
```

...and after a blank line type in a **simple** expression of type **string** representing a color.

1 part d

Type the comment:

```
; 1 part d
```

...and after a blank line type in a **compound** expression of type **boolean** representing whether one string expression of your choice has the same value as another string expression of your choice.

1 part e

Type the comment:

```
; 1 part e
```

...and after a blank line type in or insert a **simple** or **compound** expression -- your choice! -- of type **image**.

Problem 2

Next, in your definitions window, after a blank line, type the comment lines:

```
;
; Problem 2
;
```

Racket happens to have a built-in operation, `max`, for finding the maximum value within a given set of numbers.

After a blank line, for each of the following, use `max` as the operation in a compound expression with the specified quantity of arguments that are each of type number.

WARNING: They will not all work! For EACH that does NOT work, COMMENT OUT that expression in your definitions window (put a ; in FRONT of it) and re-run (so that subsequent expressions can execute!)

- Write a compound expression with `max` as the operation and NO arguments.
- Write a compound expression with `max` as the operation and exactly ONE number argument.
- Write a compound expression with `max` as the operation and exactly TWO number arguments.
- Write a compound expression with `max` as the operation and exactly THREE number arguments.
- Write a compound expression with `max` as the operation and MORE THAN THREE number arguments.

Problem 3

Next, in your definitions window, type the comment lines:

```
;
```

```
;  Problem 3
;
```

The following are currently **NOT** "correct" Racket expressions -- they do NOT follow Racket's syntax rules.

After a blank line, give the specified comment and then correct each given "bad" expression such that the result DOES follow Racket's syntax rules, typing the result in your definitions window.

(Note: there is more than one reasonable way to correct each of these!)

3 part a

Type the comment:

```
; 3 part a
```

...and after a blank line correct the expression:

```
rectangle ("solid" 44 "blue")
```

3 part b

Type the comment:

```
; 3 part b
```

...and after a blank line correct the expression:

```
(#true)
```

3 part c

Type the comment:

```
; 3 part c
```

...and after a blank line correct the expression:

```
(+ (4 * 8) (- 15 2)) (55))
```

Problem 4

Next, in your definitions window, type the comment lines:

```
;
;  Problem 4
;
```

Look over pages 1 and 2 of the handout, "Some DrRacket Tidbits", posted along with this homework handout. Choose at least three different image or image-related functions from this handout that you will try out.

Then, after a blank line, write at least three compound expressions, each using at least one different image or image-related function from this handout.

Problem 5

Next, in your definitions window, type the comment lines:

```
;
;  Problem 5
;
```

Remember that an argument within a compound expression may itself be a compound expression.

After a blank line, write a **SINGLE compound expression** in DrRacket that meets **all** of the following requirements:

- It should result in an image.
- It should use **at least THREE different** image or image-related operations from the 2htdp/image teachpack. (More is fine!)
  - They can be image operations from the "Some DrRacket Tidbits" handout, OR they can be image operations (from the `2htdp/image` teachpack) that you find by exploring DrRacket's Help feature, described a bit at the end of the "Some DrRacket Tidbits" handout.

Problem 6

Next, in your definitions window, type the comment lines:

```
;
;  Problem 6
;
```

We are going to make a class "quilt" of 200-pixels-wide, 200-pixels-high images, with each class member designing one of the 200-pixel-by-200-pixel images.

After a blank line, write one or more expressions that meet the following requirements:

- the final resulting image somehow should be the result of **at least THREE different** image or image-related operations from the 2htdp/image teachpack. (More is fine!)
  - They can be image operations from the "Some DrRacket Tidbits" handout, OR they can be image operations (from the `2htdp/image` teachpack) that you find by exploring DrRacket's Help feature, described a bit at the end of the "Some DrRacket Tidbits" handout.
- the final resulting image should be an expression of type image that is **precisely** 200 pixels wide and 200 pixels high
  - You might ask -- how can I be SURE my final image is the right size? The function `image-width` expects an image arguments, and produces its width in pixels -- and the function `image-height` expects an image argument, and produces its height in pixels. So, you can use these functions to check the width and height of your final image, if you would like.
- ACCEPTABLE "extended" version of this problem: IF you would like to use `define` to give names to some expressions to make your task easier, that is allowed and encouraged (but not required) for this problem.
- As long as you meet the above requirements, your image may be as simple or as complex as you would like.

NOTE that all of the class's images from this problem are (eventually!) going to be combined into a single class "quilt" image. (That's why it is important that your "quilt square" be precisely 200 pixels wide and 200 pixels high.)