Some DrRacket Tidbits

You can download DrRacket from:

http://www.racket-lang.org/, click the **Download Racket** button near the top right;

How to set the Racket Language Level:

- go to the **Language** menu at the top of the window, and select the **Choose Language...** command;
- in the window that opens up, first look for **Teaching Languages**. Click on that radio button.
- Now look for **How to Design Programs**, and under that select the **Beginning Student** language. Then click the **OK** button in the lower-right corner
  - Back in DrRacket, click the **Run** button on the top-right corner of the window to finish setting the desired language level
- You'll know you've done it correctly if you see the following at the top of the Interactions window:

```
Welcome to DrRacket, version 5.3.2 [3m].
Language: Beginning Student; memory limit: 256 MB.
```

To use the image and universe teachpacks:

- To be able to use functions and other items from the latest versions of the `image` and `universe` teachpacks, put these two expressions at the beginning of your Definitions window:

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

  - Note that you do need to include these lines at the beginning of your Definitions window **every** time you want to use operations from those teachpacks.

A selection of functions from the image teachpack:

```
; signature: circle: number string string -> image
; purpose: expects a radius in pixels, either "solid" or
;    "outline", and a color written as a string, and
;    produces a circle image with that radius, style, and color

(circle 30 "solid" "red")

; signature: image-width: image -> number
; purpose: expects an image, and produces the width of that image in
;    pixels

(image-width (circle 30 "solid" "red"))   should produce: 60

; signature: image-height: image -> number
; purpose: expects an image, and produces the height of that image in
```

```
;    pixels

(image-height (circle 30 "solid" "red")) should produce: 60

; signature: rectangle: number number string string -> image
; purpose: expects a width and height in pixels, either "solid"
;    or "outline", and a color expressed as a string, and produces
;    a rectangular image of that width, height, style, and color

(rectangle 50 20 "outline" "blue")

; signature: regular-polygon: number number string string -> image
; purpose: expects the length of each side, the number of sides,
;    either "solid" or "outline", and a color expressed as a string,
;    and produces an image that is a regular polygon with that many
;    sides, each of that length, in that mode and color

(regular-polygon 15 6 "solid" "purple")

; signature: overlay: image image ... -> image
; purpose: expects any number of images, and produces a new image
;    that is the first image on top of the second image on top of
;    the third image and so on, all lined up by their pinholes

(overlay (circle 15 "solid" "red")
         (rectangle 40 60 "outline" "blue")
         (regular-polygon 70 5 "solid" "green"))

; signature: beside: image image ... -> image
; purpose: expects any number of images, and produces a new image
;    that is the first image to the left of the second image to the
;    left of the third image and so on, lined up by their pinholes

(beside (rectangle 10 60 "solid" "blue")
        (rectangle 10 40 "solid" "green")
        (rectangle 10 60 "solid" "blue")
        (rectangle 10 40 "solid" "green"))

; signature: above: image image ... -> image
; purpose: expects any number of images, and produces a new image
;    that is the first image above the second image above the third
;    image and so on, lined up by their pinholes

(above (rectangle 40 20 "solid" "blue")
       (rectangle 30 20 "solid" "green")
       (rectangle 40 20 "solid" "blue")
       (rectangle 30 20 "solid" "green"))

; signature: text: string number string -> image
; purpose: expects a string, a desired font-size, and a color
;    expressed as a string, and produces an image of text in that
;    font-size and color
```

```
(text "Hi!" 15 "black")
```

```
(overlay (text "Hi!" 40 "black")
         (rectangle 60 50 "solid" "yellow")
         (rectangle 80 55 "solid" "brown"))
```

To give a name a value

You can give a name to a value -- you can define an identifier and give it a value -- by using the `define` operation. The syntax is:

`(define` ***name-you-choose expression*****`)`**

This expression doesn't have a value, but it does have an important side-effect -- after this, that name you have chosen can be used as a simple expression whose value is that expression's value.

For example, after the expression:

```
(define BLUE-DOT (circle 5 "solid" "blue"))
```

...you can now use `BLUE-DOT` anywhere an image expression can be used, and it will be an image of a little solid blue circle of radius 5 pixels.

To write a test using check-expect

The `check-expect` operation expects two expressions: the one you want to test, and one giving the expected value for that expression:

`(check-expect` *expression-to-test expected-value-expression*`)`

For example, to test if the expression `(number->string (+ 1 6))` is really `"7"`, you'd use:

```
(check-expect (number->string (+ 1 6)) "7")
```

Saving your Definitions window

Use the **File** menu's **Save Definitions** or **Save Definitions as...** commands, or the **Save** button.

Getting help in DrRacket

- DrRacket has a help system (although it has more than just Beginning Student information in it...!)
- If you click on the **Help** menu, and select **Help Desk**, you can reach some useful Racket resources;
- Or, if your mouse cursor is on the name of an operation or function, and you type the F1 key (or the fn key and the F1 key at the same time on Mac OS X), a browser window will try to open up with available help about that operation;
  - LAB GOTCHA, though: depending on what browser you are using, you may have to give this web-based help permission to be shown...!
  - ALSO: if there are multiple versions of an operation, remember that you are in the **HtDP2 - Beginning Student level** at this point.