When we start DrRacket the screen is split in two as shown below: the upper panel is where you can write your program. The lower panel is where you can interact with Racket directly. A program is just a sequence of definitions or expressions. Pushing the Run button will run your program as if you had typed those definitions and expressions directly in the Interactions panel below. The picture is taken from Realm of Racket¹ and shows DrRacket ready to run a Racket program. In our class we will start (following our textbook²) with a simpler language, entitled BSL (Beginner Student Language, a subset of Racket). Please remember to select BSL before you start working with DrRacket so as to have no surprises.

Let’s start talking about expressions. Numbers are expressions, they are their own values: the value of 3 is 3. You can make more complicated (and useful) expressions if you use operators: + and * are some such operators for numbers. As an example 1 + 2 is an expression whose value is 3. Likewise 1 + 2 * 3 is another expression whose value is: ____________ (write your answer here). Because we have rules of precedence for operators (“Please Excuse My Dear Aunt Sally” kids are sometimes taught to say) we need to be sure we evaluate the expressions in accordance with these rules. Here’s an abstract syntax tree for 1 + 2 * 3 (picture above) where nodes could be thought of to hold intermediate results during evaluation. The expression above would be written in Racket/BSL as follows: (+ 1 (* 2 3))

Your first lab this week will ask you to work out some conversions. Let’s practice with abstract syntax trees (this will be an equivalent way of getting that lab right). Write here the expression that the tree to the right represents:

_____________________________________

Write it again, in BSL/Racket notation:

_____________________________________  

The expression evaluates to: _____________

Let’s run it in DrRacket and confirm its value.

¹ Available for free in .pdf format from http://it-ebooks.info/book/2628/
² http://www.ccs.neu.edu/home/matthias/HtDP2e/index.html (How to Design Programs 2nd ed.)
Now let’s discuss definitions.

What does this do? Write your answer here:
(define a 3)

What is now the value of (+ a 5)
What does this do?
(define (f x) (+ x 3))

What is now the value of (f 5)
What is the value of (f -1)
What is the value of (f (+ 3 4))
What does this do?
(define (g x) (+ 1 x))

What is the value of (g 9)

And why?
What is f? Where did we define it?
What is g? Where did we define it?
What is add1? Did we define it?
What is string-append? Did we define it?
Where can we read about it?

What is an example of using it?

Now let’s work out some problems:

1. Write a function that takes a number as an argument and returns a number three times bigger.
2. Write a function that receives a string as an argument and returns its length.
3. Write a function that receives an image as an argument and returns its area in pixels.

We will work the previous three functions together in class (and you can take notes). Then you will have to write a function triple that consumes a string, and produces a new string which contains exactly 3 copies of the original string. For example:

(check-expect (triple "hello") "hellohellohello")

The function string-append may be useful. It works like this:

(string-append "hello" "goodbye") ;; produces "hellogoodbye"

Write your code for triple below and turn in this page in when asked, in class.