The Style

Style is an extremely important part of writing good code. Familiarly styled code makes reading it much easier to understand. Most organizations follow a style guide, a set of conventions to write well structured code. Similar to styling conventions we follow when writing papers in English, style guidelines allow the reader to focus on the thoughts expressed in your code rather than the appearance of your code. Along with others reading your code, styling your code properly will also help you understand code you have already written.

It is an expectation of this course that all code written must adhere to these style guidelines. These conventions will help you write code that is easy to read, debug, and edit.

Document Structure

Top Down

Your programs should be organized top-down. This means that in a program that contains multiple functions, the primary function should come first, followed by helpers. Your helper functions should be in the order that they appear in the primary function. For example, the following code is organized top-down

```
; my-function : Number String -> Number
; Add double the string-length to twice the number cubed
(check-expect (my-function 2 "hi") 20)
(check-expect (my-function 3 "hello") 64)
(define (my-function n s)
  (+ (double (cube-num n)) (double-length s)))

; double : Number -> Number
; Computes 2n
(check-expect (double 4) 8)
(define (double n) (* n 2))

; cube-num : Number -> Number
; Produces the cube of this number
(check-expect (cube-num 5) 125)
(define (cube-num n) (expt n 3))

; double-length : String -> Number
; Produces twice the length of this string
(check-expect (double-length "goodbye") 14)
(define (double-length s)
  (double (string-length s)))
```

Please note that many of the above helper funcitons were written solely to illustrate the top-down organization we expect.

Exercises

For most homework assignments, you will submit a single file containing many exercises. Keep all your code for each exercise together and label each exercise with a comment. This will help avoid confusion when grading your assignments.

Separate Data Definitions

Don't mix your data definitions with the rest of the program. Your data definitions should appear at the beginning of each relevant exercise. If you are reusing a data definition from a previous exercise, you do not need to repeat it in each exercise. If you are reusing a data definition from a previous assignment, however, you must include it in your submission unless we explicitly state otherwise.

Tests

Tests belong in between the purpose statement and the relevant function definition. Here's an example:

```
;; plus-one: Number -> Number
;; Adds one to the given number.
(check-expect (plus-one 1) 2)
(check-expect (plus-one 0) 1)
(define (plus-one n)
        (add1 n))
```

Naming

Use descriptive but concise names for all your functions and constants. Your names should make sense with respect to the problem you are solving.

Capitalization

Function, argument, struct, and field names should not contain capital letters. Constants, however, should be in all caps. Names with multiple words should have dashes to separate each word. Here are some examples:

```
(define WIDTH 100)
(define HEIGHT 200)

;; area: Number Number -> Number
;; Computes the area of a rectangle from the width and height.
(check-expect (area 5 10) 50)
(define (area w h)
        (* w h))

;; cube-num: Number -> Number
;; Computes the given number raised to the third power.
```

```
(define (cube-num n)
        (expt n 3))
```

Predicates

Any function that returns a boolean should end with a `?` (pronounced "huh"). Built in functions that return booleans follow this convention. For example, `boolean?`, `number?`, and `string?` pronounced "booleanhuh", "numberhuh", and "stringhuh" respectively.

Spacing

Indentation

DrRacket handles this one for you. Click on "Racket" > "Reindent All" to reindent your entire file properly. Press tab to reindent the current line or the selected area. To save time while submitting your homework, you should be sure to reindent your file before submitting.

Line Length

Do not exceed 102 columns/characters per line for code *or* comments. DrRacket will show you the length of the current line in the bottom right corner of the window. You can also click "Edit" > "Find Longest Line" to find the longest line of the current file. To help avoid long lines while writing your code, you can turn on a max-length guide by clicking on "Edit" > "Preferences" > "Editing" > "General Editing", and checking the "Maximum character width guide" box and setting the value to 102.

Dangling Parenthesis

The closing right-parenthesis should be on the same line as the last expression of your code.

```
;; ---------------- GOOD
(define (f l)
        (cond [(empty? l) 0]
                  [else (f (rest l))])) ;; All closing parens should go HERE

;; ---------------- BAD
(define (f l)
        (cond [(empty? l) 0]
                  [else (f (rest l))]
        ) ;; Not HERE
) ;; Or HERE
```

New Lines

New lines in DrRacket are free! Use them to break up logically distinct tasks. Don't use them unnecessarily as it can make your code look messy.

Here are some examples of good and bad spacing:

```
;; ----------------- GOOD
(define (foo x y z)
  (max (* x y)      ;; Break after each argument to max,
       (* y z)      ;; and align all arguments in a column
       (* x z)      ;; (This works best with short-named functions)
       (* x y z)))

;; ----------------- OK
(define (foo x y z)
  (max          ;; Break after max itself
   (* x y)      ;; Then indent each argument 1 space
   (* y z)      ;; (This works better when function names are long)
   (* x z)
   (* x y z)))

;; ----------------- BAD
(define (foo x y z)
  (max (* x y)
   (* y z)     ;; This indentation is an inconsistent
   (* x z)     ;; mix of the previous two styles
   (* x y z)))

;; ----------------- BAD
(define (foo x y z)
  (max (* x y) (* y    ;; This linebreak is just weird.
                  z)
       (* x z) (* x    ;; This is ugly. And avoidable!
                  y
                  z)
                  )
                  )
```