Assignment: 01
Due: Tuesday, January 21, 9:00 pm
Language level: Beginning Student
Files to submit: debug-a01.rkt, functions.rkt, jeopardy.rkt, grades.rkt, bonus-a01.rkt
Warmup exercises: HtDP 2.4.1, 2.4.2, 2.4.3, and 2.4.4
Practice exercises: HtDP 3.3.2, 3.3.3, and 3.3.4

Notes:

- Become familiar with the CS 135 assignment policies on the course website.
- Frequently scan the OFFICIAL A01 post on Piazza for answers to frequently asked questions, especially before you post to Piazza.
- The work you submit must be entirely your own. Do not look up either full or partial solutions on the Internet or in printed sources. Read about plagiarism on the course web site.
- If you haven’t completed Assignment 0 with full marks, you’ll receive 0 for this assignment.
- A significant proportion of your marks will be for good coding style. You should be familiar with the style guide and apply relevant parts to each assignment. Appropriate constant usage is particularly important for A01. You do not need to include the design recipe for A01.
- Submit early and often. If you don’t know what this means, talk to your prof or an ISA.

Here are the assignment questions you need to submit. All functions required to complete these questions can be found in section 1.5 of the documentation for the Beginning Student language.

1. [10% Correctness] Question 1
Consider the following poorly written Racket function that contains syntax errors and poor style. Correct the function.

```racket
(define (luminosity red-pixel\_value green-pixel\_value blue-pixel\_value)
  (+ (* 0.3 red-pixel\_value) + (* 0.59 green\_pixel-value) + (* 0.11 blue-value-pixel)))
```

Place your solution in the file debug-a01.rkt.

2. [20% Correctness] Question 2
Translate the function definitions into Racket, using the names given. Note that when you are asked to translate a function, it should be a direct translation. When asked to translate
\((a + b)\), the translation is \((+ a b)\), not \((+ b a)\). When translating \(x^2\), use the Racket function \((sqr x)\). When translating fractions, treat them as though they had brackets surrounding them. For example, \(5 \cdot \frac{x}{2}\) is translated to \((\ast 5 (/ x 2))\) not \((/ (\ast 5 x) 2)\).

For example, if we asked you to translate the function:

\[
\text{mean}(x_1,x_2) = \frac{x_1 + x_2}{2}
\]

you would submit:

\[
\text{(define (mean } x_1 x_2) \text{(/ (+ } x_1 x_2 \text{) 2) )}
\]

(a) A basketball player’s points per game is based on the number of field goals \(fg\), 3-point baskets \(3pt\), free throws \(ft\), and a positive number of games played \(gp\):

\[
\text{points-per-game}(fg,3pt,ft, gp) = \frac{2 \cdot fg + 3 \cdot 3pt + ft}{gp}
\]

(b) Newton’s law of universal gravitation describes the attraction between two masses.

\[
\text{force}(m_1,m_2,r) = G \cdot \frac{m_1m_2}{r^2}
\]

where \(G\) is the gravitational constant \(6.674 \times 10^{-11} \text{ N} \cdot \text{(m/kg)}^2\), and \(r\) is the distance between the two masses.

Notes:

- The units for the gravitational constant are provided for your information; they are not required as part of the function definition.
- The gravitational constant is always written as an uppercase \(G\) to distinguish it from \(g\) which is used to describe acceleration due to gravity.
- Since subscripts are not available when writing Racket code, use \(m_1\) and \(m_2\) as your function’s parameter names.
- To test your own function you can use this online calculator to generate independent results. This calculator uses a slightly different value for \(G\), but will give you results close to what your function should produce.


(c) When asked to choose a password for some computer application, you may have been given feedback that describes its strength. These applications may use password entropy, which describes the unpredictability of a password, as a way of indicating how strong your choice is. A basic formula for calculating password entropy is:

\[
\text{password-entropy}(s, pl) = pl \cdot \frac{\log s}{\log 2}
\]
where \( s \) is the number of possible symbols and \( pl \) is the number of symbols in the password, and you may use any base for the \( \log \) function. 
Note: the default base for the built-in Racket function \( \log \) is \( e \).

(d) A partition is a way of writing \( n \) as a sum of positive integers. For example \( 5 + 2 + 1 + 1 \) is a partition of \( 9 \). Mathematicians Hardy and Ramanujan developed the following formula to approximate the number of partitions for the positive integer \( n \):

\[
\text{partition-size-approximation}(n) = \frac{1}{4n\sqrt{3}} \cdot e^{\pi\sqrt{\frac{2}{3}n}}
\]

Your solution should use the built-in constant for the value of \( \pi \). Read the Racket documentation carefully to understand the difference between the built-in functions \( \exp \) and \( \text{expt} \).

Place your solutions in the file \texttt{functions.rkt}.

3. [20% Correctness] Question 3

The game show \textit{Jeopardy!} has three contestants who answer trivia questions. Each question has a monetary value. If the contestant answers the question correctly they gain the value of the question to their accumulated total, and if they answer incorrectly they lose the value of the question from their accumulated total. At the end of show, there is a “Final Jeopardy” question. For this question, each contestant must decide on a wager before the question is asked. An individual’s wager can be any integer between 0 and their accumulated total (inclusive). For example, if a contestant had accumulated $9475, he/she could wager any full dollar amount between 0 and 9475 (inclusive). For this question, you may assume that one of the three contestants has accumulated more than both of the other two contestants before “Final Jeopardy”, and all three contestants have accumulated integer totals that are more than $0. The built-in Racket function \texttt{max} may be helpful.

(a) Write a function \texttt{min-wager} that consumes three numbers (in the following order):
- the total of the contestant who has accumulated the most money before “Final Jeopardy”,
- the accumulated total of contestant X,
- the accumulated total of contestant Y.

This function should produce the minimum that the contestant who has accumulated the most money must wager, to guarantee that he/she will have more money than either of the other two contestants if he/she answers the “Final Jeopardy” question correctly. You may assume that one of the other two contestants has accumulated at least half as much money as the leading contestant. For example, \((\text{min-wager} \ 10000 \ 7000 \ 8000)\) would produce 6001, since if the contestant in the lead wagers $6001 and answers correctly, he/she would have a total of $16001. The most either of the other contestants could finish with is $16000. Note that contestant X may have less, more, or the same amount accumulated as contestant Y.
(b) Assume that the contestant with the most money accumulated uses the wager strategy described in part a). Write a function \texttt{missed-question} that consumes the three accumulated amounts as in part a) and produces the amount the leading contestant ends up with if he/she answers the question incorrectly. For example, \((\texttt{missed-question} 10000 \ 8000 \ 7000)\) produces 3999. In this case you may \textbf{not} assume that at least one of the other two contestants has at least half as much as the leading contestant. This means that the minimum wager may be 0.

You may use your function in part a) as part of your solution, however it is not required.

Place your solutions in the file \texttt{jeopardy.rkt}.

4. \textbf{[30\% Correctness] Question 4}

The end of the Winter term has come, and you decide to use Racket to calculate your grade going in to the CS 135 final. For this question, you do not need to worry about the course requirements of passing the exam and assignment components of the course separately.

(a) Write a function \texttt{cs135-grade-sofar} that consumes three numbers (in the following order):

- the midterm grade,
- the participation grade, and
- the overall assignments grade.

This function should produce the weighted grade in the course (as a percentage, but not necessarily an integer) going in to the final exam. For example, \((\texttt{cs135-grade-sofar} 80 \ 75 \ 90)\) produces 83.5. You may need to review the mark allocation in the course. You can assume that all grades are percentages and are given as numbers between 0 and 100, inclusive.

(b) The end of exams has now come, and you have received your final grade in the course. Write a function \texttt{cs135-final-exam} that consumes two numbers. The first number is the weighted grade going in to the final exam (as calculated above). The second number is the final grade received in the course. This function should produce the grade received on the final exam. For example, \((\texttt{cs135-final-exam} 73.2 \ 81)\) produces 88.8. When testing your function we will only use arguments that will produce a result between 0 and 100, inclusive.

Place your solutions in the file \texttt{grades.rkt}.

This concludes the list of questions for you to submit solutions (but see the following pages as well). Don’t forget to always check the basic test results after making a submission.

\textit{Assignments will sometimes have additional questions that you may submit for bonus marks. This is intended to be a personal challenge for students in the course if they wish to try. Course staff will not answer questions about the Bonus either in person or on Piazza.}
**Bonus Question [5%]:** In CS 135, your class participation grade (from clickers) is calculated in the following manner:

- Each question is worth two marks.
- You receive two marks for a correct answer, and one mark for an incorrect answer.
- You receive zero marks if you do not answer.
- To account for imperfect attendance, forgotten clickers, dead batteries etc., only your best 75% of the questions are used to calculate your grade.

Write a function `cs135-participation` that consumes three parameters (in order):

- the total number of clicker questions asked in the year,
- the number of questions you answered correctly, and
- the number of questions you answered incorrectly.

Your function must produce your class participation grade as a percentage (a number between 0 and 100 but not necessarily an integer). For convenience, you may assume the total number of questions is a positive Integer divisible by four.

**Note:** you may only use the features of Racket given up to the end of Module 1. You may use `define` and `mathematical` functions, but not `cond`, `if`, lists, recursion, Booleans, or other things we’ll get to later in the course.

Place your solution in the file `bonus-a01.rkt`.

Note that bonus questions are typically “all or nothing”. Incorrect or very poorly designed solutions may not be awarded any marks.

---

**Challenges and Enhancements**

Each assignment in CS 135 will continue with challenges and enhancements. We will sometimes have questions (such as the one above) that you can do for extra credit; other questions are not for credit, but for additional stimulation. Some of these will be fairly small, while others are more involved and open ended. One of our principles is that these challenges shouldn’t require material from later in the course; they represent a broadening, not an acceleration. As a result, we are somewhat constrained in early challenges, though soon we will have more opportunities than we can use. You are always welcome to read ahead if you find you want to make use of features and techniques we haven’t discussed yet, but don’t let the fun of doing the challenges distract you from the job of getting the for-credit work done first. On anything that is not to be handed in for credit, you are permitted to work with other people.
The teaching languages provide a restricted set of functions and special forms. There are times in these challenges when it would be nice to use built-in functions not provided by the teaching languages. We may be able to provide a teachpack with such functions. Or you can set the language level to “Pretty Big”, which provides all of standard Racket, plus the special teaching language definitions, plus a large number of extensions designed for very advanced work. What you lose in doing this are the features of the teaching languages that support beginners, namely easier-to-understand error messages and features such as the Stepper.

This enhancement will discuss exact and inexact numbers.

DrRacket will try its best to work exclusively with exact numbers. These are rational numbers; i.e. those that can be written as a fraction \( \frac{a}{b} \) with \( a \) and \( b \) integers. If a DrRacket function produces an exact number as an answer, then you know the answer is exactly right. (Hence the name.)

DrRacket has a number of different ways to express exact numbers. 152 is an exact number, of course, because it is an integer. Terminating decimals like 1609.344 are exact numbers. (How could you determine a rational form \( \frac{a}{b} \) of this number?) You can also type a fraction directly into DrRacket; \( \frac{152}{17} \) is an exact number. Scientific notation is another way to enter exact numbers; \( 2.43e7 \) means \( 2.43 \times 10^7 = 24300000 \) and is also an exact number.

It is important to note that adding, subtracting, multiplying, or dividing two exact numbers always gives an exact number as an answer. (Unless you’re dividing by 0, of course; what happens then?) Many students think that once they divide by a number like 1609.344, they no longer have an exact answer, perhaps because their calculators don’t treat it as exact.

But try it in DrRacket: \( \frac{2}{1609.344} \). DrRacket seems to output a number with lots of decimal places, and then a “...” to indicate that it goes on. But right-click on the number, and a menu will allow you to change how this (exact) number is displayed. Try out the different options, and you’ll see that the answer is actually the exact number \( \frac{125}{100584} \).

You should use exact numbers whenever possible. However, sometimes an answer cannot be expressed as an exact number, and then inexact numbers must be used. This often happens when a computation involves square roots, trigonometry, or logarithms. The results of those functions are often not rational numbers at all, and so exact numbers cannot be used to represent them. An inexact number is indicated by a \#i before the number. So \#i10.0 is an inexact number that says that the correct answer is probably somewhere around 10.0.

Try \( \sqrt{15} \). You would expect the answer to just be 15, but it’s not. Why? \( \sqrt{15} \) isn’t rational, so it has to be represented as an inexact number, and the answer is only approximately correct. When you square that approximate answer, you get a value that’s only approximately 15, but not exactly.

You might say, “but it’s close enough, right?” Not always. Try this:

\[
\begin{align*}
\text{(define (addsub x)}
\text{(- (+ 1 x) x))}
\end{align*}
\]
This function computes \((1 + x) - x\), so you would expect it to always produce 1, right? Try it on some exact numbers:

\[
\begin{align*}
\text{addsub 1} \\
\text{addsub 12/7} \\
\text{addsub 253.7e50}
\end{align*}
\]

With exact numbers, you always get 1, as expected. What about with inexact numbers?

\[
\begin{align*}
\text{addsub (sqrt 15)} & \Rightarrow \#i1.0, \text{ which is fine. } \\
\text{addsub (sqrt 2)} & \Rightarrow \#i0.9999999999999998, \text{ which is close to 1; that’s more or less what we expect from inexact numbers. But } \\
\text{addsub (exp 40)} & \Rightarrow \#i0.0. \text{ That answer is very different from 1! Can you find argument values that give different answers from these?}
\end{align*}
\]

If you go on to take further CS courses like CS 251 or CS 370, you’ll learn all about why inexact numbers can be tricky to use correctly. That’s why in this course, we’ll stick with exact numbers wherever possible.