Assignment: 01
Due: Tuesday, September 22 at noon (Waterloo time)
Language level: Beginning Student
Files to submit: sequences.rkt, translations.rkt, crl-points.rkt,
valid-date.rkt, valid-date-cond.rkt,
valid-date-bool.rkt
Practice exercises: HtDP 3.3.2, 3.3.3, 3.3.4, 4.1.1, 4.3.2, 4.4.1, 4.4.3, and 5.1.4

- Become familiar with the CS 135 assignment policies on the course website.
- Read 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 web 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 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. For this assignment, you should read and apply sections 1-4 of the style guide.
- Submit early and often. Submissions spike near the deadline and submitting early will avoid problems due to server load. There is typically a grace period of ten minutes or so after the submission deadline to allow assignments already in the queue at the noon deadline to be processed and fully recorded. However, you should not depend on this. Once the submission system closes we do not accept further submissions for marks, so submitting early is always a good idea.
- This assignment covers concepts up to the end of Module 04. Unless otherwise specified, you may only use Racket language features we have covered up to that point.
- It is very important that your function names, strings, and symbols match ours. Basic tests results will catch many, but not necessarily all of these types of errors.
- A total of 70% of the marks are allocated as indicated at the beginning of each question. The remaining 30% are allocated in the following manner: 10% for Understandability (including formatting naming, organization, helper functions, and similar aspects of good coding style); 10% for Purpose and Contract; and 10% for Testing (including examples).
- Racket has an if construct but you must never ever ever use it in CS 135. It is forbidden! Similarly we use true and false, not #t and #f.
1. [10% Correctness]

Make sure you learn the material in Module 02 before trying this question. The module makes a big deal about how there may be many possible substitutions when evaluating an expression but that we all need to agree on exactly one in any given situation. This will continue to be a big deal in the course as we introduce more and more programming constructs.

To help reinforce the substitution rules we have an online tool, known as the “Stepper” at https://www.student.cs.uwaterloo.ca/~cs135/assign/stepping/

The use of https is important; that is, the system will not work if you omit the s.

You will need to authenticate yourself using your Quest ID and password. Once you are logged in, click on the first question under “Module 2a: Expressions”. Note the “Show instructions” link at the bottom of each problem. Read the instructions!

Complete the required module 02 stepper problems on expressions, functions, and constants. You start with the practice problems, which don’t count for marks, but should help you understand what to do. For these practice problems a “Hint” button is enabled. Use it if you get stuck.

If you’re having trouble understanding what to do, it may help to watch this short video: https://www.student.cs.uwaterloo.ca/~cs135/videos/a01/a01.10_stepping.mp4

You can re-enter a step as many times as necessary until you get it right, so keep trying until you completely finish every question. All you have to do is complete the questions online—we will be recording your answers as you go, and there is no file to submit. The basic tests for this assignment will tell you whether or not we have a record of your completion of the stepper problems. Note that you are not done with a question until you see the message Question complete! You should see this once you have arrived at a final value and clicked on “simplest form” (or “Error,” depending on the question).

You are stepping through the given expressions assuming that constant definitions that appear above the expression exist, and have been fully processed as though you have pressed Run in DrRacket. This means that you are not required to do any simplification of the constant definitions as part of the stepping.

DrRacket has a builtin stepper, but you should be aware that DrRacket’s evaluation rules are different from the ones presented the course materials and implemented in the online stepper. These differences will be more obvious and important in later assignments, so we suggest that you avoid the DrRacket stepper when solving these problems.

If you get stuck on a stepping question, please do not post publicly to Piazza requesting the next step. This is a violation of the academic integrity policy. Review the substitution rules carefully to try to solve the problem yourself. If you’re stuck on one problem, try another and come back. As a last resort, you may make a private post to Piazza describing where you are stuck. Course staff will provide guidance directing you to the next step, but they will not give you the answer.
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 \((\ast 5 \left(\frac{x}{2}\right))\) not \((\div \left(\ast 5 x\right) 2)\). However, fractional numeric literals such as \(\frac{1}{3}\) should be written as \(1/3\).

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

\[
\text{mean}(x1, x2) = \frac{x1 + x2}{2}
\]

you would submit:

\[
\text{(define \(\text{mean}\) \(x1\) \(x2\))}
\text{(\(/ \left(\ast x1 x2\right) 2\))}
\]

Unlike most of the other functions you write in this course, the functions in this question produce inexact numbers. Therefore in your examples and tests you will want to use check-within, not check-expect. Use a tolerance of 0.01 for your examples and tests. Note that you will not need a lot of examples and tests for each function. Three or four well-chosen test cases per function should be adequate to test them thoroughly (but use your judgement).

Place your solutions for the following functions in the file translations.rkt.

(a) [5% Correctness] Write a function, \(\text{volume}\), to compute the volume of a sphere, as given by

\[
\text{volume}(r) = \frac{4}{3}\pi r^3
\]

where \(r\) is the sphere’s radius.

(b) [5% Correctness] The Fibonacci numbers are one of the most famous number sequences in all of mathematics. They are defined by letting \(F_1 = F_2 = 1\), and calculating all subsequent values via \(F_{n+2} = F_n + F_{n+1}\). The first few elements are \(1, 1, 2, 3, 5, 8, 13, 21, 34\ldots\)

There are many different ways to compute the Fibonacci numbers, which all serve as good demonstrations of elementary computer programming principles. For now, we will simply use the following closed-form solution:

\[
F_n = \frac{\varphi^n - (-\varphi)^{-n}}{2\varphi - 1}.
\]

Here, \(\varphi\) is the golden ratio \(\frac{1 + \sqrt{5}}{2}\). Name it \(\text{phi}\) (the name of the Greek character).

Read the Racket documentation carefully to understand the difference between the built-in functions \text{exp} and \text{expn}.

Write a function \(\text{fib}\) that consumes an integer \(n > 0\) and computes \(F_n\) using the formula above. Your answer should be an inexact approximation and should not attempt to round the result.
3. [10% Correctness] A sequence of numbers $a_1, a_2, \ldots$ is said to be *arithmetic* if there is a constant $d$ such that, for each $i$, $a_{i+1} = a_i + d$. It is called *geometric* if there is a constant $r$ such that, for each $i$, $a_{i+1} = a_i \cdot r$.

Write the function `sequence-type` that consumes four numbers, $a, b, c,$ and $d$, and produces the symbol `arithmetic` if the sequence $a, b, c, d$ is arithmetic, `geometric` if the sequence $a, b, c, d$ is geometric, `both` if it is both, and `neither` if it is neither. (Hint: Be very very careful when dealing with zero.)

**A note about testing:** Unfortunately, our testing scripts are imperfect. Part of your mark will be for test coverage, but since we do not know what helper functions you come up with, our scripts only tally tests for the main functions we ask you to write. You may – and should! – test your helper functions, but you should also test the main function thoroughly to get these coverage marks. (This note applies for all other assignment questions as well.)

Place your answers in the file `sequences.rkt`.

4. [10% Correctness] The fictional Competitive Racket League (CRL) holds a series of tournaments each year. At each tournament, competitors earn points for finishing in the top three – first, second, or third place. Currently, points are allocated as follows:

- A first-place result earns 50 points
- A second-place result earns 20 points
- A third-place result earns 10 points

In addition, participants who have five or more top three results in the tournament season get a consistency bonus of 15 points.

As with any competition, the Competitive Racket League has rules, and sometimes competitors break those rules. Depending on the level of their violation, they are subject to the following penalties:

- A competitor with a status of `minor-violation` has 5% of their point total deducted.
- A competitor with a status of `major-violation` has 25% of their point total deducted.
- A competitor with a status of `disqualified` has all of their points deducted.

Competitors who are not subject to penalties are in `good-standing`, and keep all their points.

Write a function `crl-points` which computes the number of points for a given competitor in the tour. The function consumes three numbers corresponding to the number of first, second, and third place finishes respectively, and a symbol indicating whether the competitor has a `minor-violation`, `major-violation`, is `disqualified` or is in `good-standing`.

Note that points are always whole numbers, so if the computation results in a decimal it should be rounded down.

For example, `(crl-points 0 2 4 'minor-violation)` produces 90, because this competitor received 2 second place finishes and 4 third place finishes, plus the consistency bonus, but had a `minor-violation`. A 5% reduction of 95 is 90.25, which is rounded down.

Place your solution in the file `crl-points.rkt`. 
5. We can use Nat values to represent dates. For example, 20200922 represents the date September 22, 2020. The final two digits of the number represent the day, and the two digits to the left of that represent the month. The remaining digits represent the year.

This means that not every Nat represents a date. For example, 123456789 is invalid, because there are not 89 days in any month, and because there is no month 67. Note that 12345 would be a valid year, however.

Some years are leap years. On those years February 29 is a valid date. Otherwise February 29 is not a valid date. Other months have either 30 or 31 days (this remains stable year to year, and you should consult a calendar to determine which months have 30 days and which have 31).

(a) [10% Correctness] Write a function leap-year? which consumes a Nat that represents a year (not a full date). The function should produce true if the given Nat represents a leap year, and false otherwise.

Here are the rules for determining whether a year is a leap year:

- A year divisible by 400 is a leap year.
- Otherwise, a year divisible by 100 is not a leap year.
- Otherwise, a year divisible by 4 is a leap year.
- Otherwise, the year is not a leap year.

For example, (leap-year? 1900) produces false because 1900 is divisible by 100 (but not 400), but (leap-year? 1904) produces true because 1904 is divisible by 4, but not 100 or 400.

Place your solution in the file valid-date.rkt. (You may find this function helpful for future parts of this question.)

(b) [10% Correctness]

Write a function valid-date? which consumes a Nat representing a date (not just a year). The function should produce true if the Nat represents a valid date and false otherwise. For example, (valid-date? 123456789) produces false, (valid-date? 20200922) produces true, and (valid-date? 30313) produces true.

The Julian calendar used in Britain (which Canada adopted as a British colony) has a strange quirk where the dates September 3 1752 through September 13 1752 (inclusive) do not exist! ¹ So (valid-date? 17520912) produces false.

Your solution may use both cond and boolean operators.

Place your solution in the file valid-date.rkt.

(c) [5% Correctness] Re-implement valid-date? as specified in part (b), but instead of the constraint that your solution may use both cond and boolean operators, your solution should not use cond.

This reimplementation is not as mean as it sounds. Think carefully about the design recipe. What parts can you reuse from the other implementation?

In addition to `valid-date?`, all helper functions (including `leap-year?`) should avoid the use of `cond`.
Place your solution in the file `valid-date-bool.rkt`.

(d) [5% Correctness] Re-implement `valid-date?` as specified in part (b), but instead of the constraint that your solution may use both `cond` and boolean operators, your solution should not use any boolean operators or `false?`.
In addition to `valid-date?`, all helper functions (including `leap-year?`) should avoid the use of boolean operators and `false?`.
Place your solution in the file `valid-date-cond.rkt`.

A note about constants: Months are best understood as words (“March” or “Mar” as opposed to 3) and so should be named constants. Unusual dates in your code (such as breakpoint days in 1752) should also be documented as constants. Other constant values are up to your judgement. If a stranger (say a graduate TA marker) was to read your code and see the numbers you used, would they understand where those numbers came from? Which numbers in your solution would be familiar to strangers? Which would not?

A note about filenames: This question requires you to submit three files (`valid-date.rkt`, `valid-date-bool.rkt`, and `valid-date-cond.rkt`) that have similar functionality. You are allowed to copy and paste some information between these files, but you must be very very careful to include the right function implementations in the right files.

Challenges and Enhancements: Each assignment in CS 135 will continue with challenges and enhancements. We will sometimes have questions 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.

`check-expect` has two features that make it unusual:

1. It can appear before the definition of a function it calls (this involves a lot of sophistication to pull off).
2. It displays a window listing tests that failed.

However, otherwise it is a conceptually simple function. It consumes two values and indicates whether they are the same or not. Try writing your own version named `my-check-expect` that consumes two values and produces `'Passed` if the values are equal and `'Failed` otherwise. Test
your function with combinations of values you know about so far: numbers (except for inexact numbers; see below), Booleans, symbols, and strings.

Expecting two inexact numbers to be exactly the same isn’t a good idea. For inexact numbers we use a function such as check-within. It consumes the value we want to test, the expected answer, and a tolerance. The test passes if the difference between the value and the expected answer is less than or equal to the tolerance and fails otherwise. Write my-check-within with this behaviour.

The third check function provided by DrRacket, check-error, verifies that a function gives the expected error message. For example, (check-error (/ 1 0) "/: division by zero")

Writing an equivalent to this is well beyond CS135 content. It requires defining a special form because (/ 1 0) can’t be executed before calling check-error; it must be evaluated by check-error itself. Furthermore, an understanding of exceptions and how to handle them is required. You might take a look at exceptions in DrRacket’s help desk.