Fall 2025
Clarke, Garbary, Morland, Petrick, Roh, Watson

Assignment: 02

Due:	Tuesday, September 23, 2025 9:00 pm
Coverage:	L03
Language level:	Beginning Student
Allowed recursion:	None
Files to submit:	`speed.rkt`, `median.rkt`, `blood.rkt`, `weekday.rkt`

- Make sure you read the official assignment post on **Piazza**.
- **You will not receive marks for** *A*02 **unless you have earned full marks for** *A*00 **before the** *A*02 **due date.** Assignments may be submitted as often as desired up to the due date.
- For any assignment, you may not use functions or language constructs from lectures after the "coverage" lecture listed on it — L03 in the case of this assignment — unless specifically requested in the question.
- Functions and symbols must be named **exactly** as they are written in the assignment questions. You may define helper functions, if needed.
- You are expected to provide purpose, contracts, test cases for functions we ask you to write and submit, as defined in L03.
- Your test cases must provide full code coverage.

Here are the assignment questions that you need to submit.

1. **(15%):** Complete the required stepping problems for A02 at:

   `https://student.cs.uwaterloo.ca/~cs135/stepping/`

   You should refer to the instructions from A01 Question 1 for the stepper question instructions.

   For Boolean expressions and **`cond`**, DrRacket's stepper has slightly different behaviour from the examples in class: 1) It turns `true` and `false` into `#true` and `#false`. 2) It turns square brackets `[]` into round brackets `()`. For these problems, you should stick to the format of the examples in class. The practice steppers illustrate the expected format.

2. **(20%):** Speed

   Using appropriate units for quantities such as distance, mass, or velocity is often important for real-world measurements. In fact, NASA's Mars Climate Orbiter crashed into Mars in 1999 because some of the programmers were assuming metric units while others were assuming imperial units![1]

   In this question, you will write functions to convert between units. Place your solutions in the file `speed.rkt`. Do not perform any "rounding". You do not have to worry about "divide by zero" errors.

   (a) The unit of speed most often used in physics is metres per second (m/s). A common imperial unit of speed is miles per hour (mph). Write a function `m/s->mph` that consumes a speed in the units of m/s and produces the same speed in units of mph. You must use the fact that one mile is exactly 1609.344 metres. Define constants to help with the conversion. As a good test case, you should ensure that your function produces 3,600 (the number of seconds in an hour) when applied to 1609.344 (1 mile expressed in metres).

   (b) A more unusual unit of speed is *Smoots per millifortnight* (S/mfn). You must use the facts that one Smoot[2] is exactly 1.7018 metres and one millifortnight[3] is exactly 1209.6s. Define a constant, or constants, to help with the conversion. Write a function `mph->s/mfn` that consumes a speed in units of mph and produces the same speed in units of S/mfn. As a good test case, you should ensure that your function produces $317\frac{1249}{1675}$ (i.e. $\frac{532224}{1675}$) when it is called with 1.

3. **(20%)**: The following function produces the median of 3 numbers:

```
(define (median-of-3 a b c)
  (cond
    [(or (and (<= b a) (<= a c)) (and (<= c a) (<= a b))) a]
    [(or (and (<= a b) (<= b c)) (and (<= c b) (<= b a))) b]
    [(or (and (<= b c) (<= c a)) (and (<= a c) (<= c b))) c]))
```

   Simplify this function so that it reduces the maximum number of inequality comparisons that could be performed. Inequalities include `>`, `>=`, `<` and `<=`. Racket has versions of the inequality operators that consume more than 2 arguments. You may **not** use them for this question.

   In addition to the above inequality operators, you may only use **`define`**, **`cond`**, `=`, **`and`**, **`or`**, or **`not`**. Note: **`else`** is allowed whenever **`cond`** is allowed. Do not use any helper functions for this question.

   The original function performs a maximum of 12 comparisons. A trivial transformation gets it down to 8. How much lower can you go? There may be several solutions depending on

[1] https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf, p.16
[2] https://en.wikipedia.org/wiki/Smoot
[3] https://en.wikipedia.org/wiki/Fortnight

how you decide to write the function. Full marks will be granted if you can get the maximum number of possible comparisons down to at most 5 comparisons (and your function is still correct).

Some example test cases for `median-of-3` are given below.

```
(check-expect (median-of-3-simple 1 2 3) 2)
(check-expect (median-of-3-simple 2 1 3) 2)
```

It is permissible to include `median-of-3` in your code and use it to run `check-expect`s.

Name your function `median-of-3-simple`. Place your solution code in the file `median.rkt`.

4. **(20%)**: Everyone has a "blood type" that depends on many things, including the antigens they were exposed to early in life. There are many different ways to classify blood; one of the most common is by group: O, A, B, and AB. This is augmented by the "Rh factor" which is either "positive" or "negative". This yields a set of eight relevant types. We'll use the following symbols to represent them:

```
;; A Blood-Type is (anyof 'O- 'O+ 'A- 'A+ 'B- 'B+ 'AB- 'AB+)
```

If a person needs a blood transfusion, the type of the donor's blood is restricted to types which the recipient's body can accept. In the following chart, a checkmark indicates which types of blood are acceptable to donate to a recipient. For example, a person with type `'O+` can donate to a person with type `'A+`, but not to someone with a type `'B-`. You can observe that `'O-` is sometimes referred to as the *universal donor* and `'AB+` is sometimes referred to as the *universal recipient*.

Recipient \ Donor	O-	O+	A-	A+	B-	B+	AB-	AB+
O-	√							
O+	√	√						
A-	√		√					
A+	√	√	√	√				
B-	√				√			
B+	√	√			√	√		
AB-	√		√		√		√	
AB+	√	√	√	√	√	√	√	√

(a) Write the function `can-donate-to/cond?` which consumes a symbol denoting the donor's blood type as the first parameter (`Blood-Type`) and a symbol denoting the recipient's blood type (`Blood-Type`) as the second parameter. Produce `true` if the donor's blood type is acceptable for the recipient's blood type, according to the above chart, and `false` otherwise. `can-donate-to/cond?` **must** use **`cond`** expressions **without** using **`and`**, **`or`**, or **`not`**.

You should try not to hard-code every possible combination of blood types in your solutions. Try to find patterns in the data and use those patterns to make your function easier to read and understand.

Hint: Use `symbol=?` to compare two symbols for equality.

(b) Write the function `can-donate-to/bool?` which is identical to `can-donate-to/cond?` except that it uses **only** a Boolean expression. That is, it does *not* have a **`cond`** expression.

Make sure you type all symbols exactly as they are written in the question. Place your answers in the file `blood.rkt`.

5. (**25%**): Write a function `date->day-of-week` which consumes a natural number and produces a symbol corresponding to the day of the week, according to the Gregorian calendar.

   An integer encodes a date as follows:

   - The final two digits correspond to the day of the month
   - The two digits before that correspond to the month of the year
   - The digits before that correspond to the year

   The function should return one of seven symbols, according the the following data definition according to the day of the week the consumed date corresponds to.

```
;; A Week-Day is (anyof 'Monday 'Tuesday 'Wednesday 'Thursday
;;                      'Friday 'Saturday 'Sunday)
```

   For example:

```
(check-expect (date->day-of-week 20240924) 'Tuesday)
(check-expect (date->day-of-week 38781202) 'Monday)
```

   You can assume that all dates are correct by adding a requires clause to the contract indicating that the natural number corresponds to a valid date. By adding this clause, your function does not have to check for invalid input, such as, April 65, 1900 (`(date->day-of-week 19000465)`). You can also assume that no dates before January 1, 1753 CE, will be tested[4]. You may only use allowed constructs from lecture L03. You may not use Racket's date functions.

   It is acceptable to consult rules or algorithms for computing the day of the week given a date, but you must cite any resources you use in your Racket file.

   For this question only, you may use Racket's `floor` function, where `(floor x)` produces the largest integer that is no more than `x`. For this question only, you may use Racket's `modulo`

[4]The Gregorian calendar we use today was not fully established in the United Kingdom and her colonies before that date (`https://en.wikipedia.org/wiki/Adoption_of_the_Gregorian_calendar`, as retrieved on 2023-06-23).

function, which differs from `remainder` in how negative values are handled. It's easiest to see the difference by comparing some expressions. For example, compare

```
(remainder -13  4)
(modulo    -13  4)
(remainder  13 -4)
(modulo     13 -4)
```

Put your solution in `weekday.rkt`.