Exercise 1: Getting started with Racket

The goal for this exercise is to write some simple code in Racket. You should first complete the software installations in Lab 1 before working on this exercise.

Deadline: September 17, 2019 before 10:00pm

Starter code

- ex1.rkt

Task 1: Working with Racket

Complete the following functions in `ex1.rkt`, as specified by their comments:

- `celsius-to-fahrenheit`
- `n-copies`
- `num-evens`
- `num-many-evens`
- `double-list`

A reminder that you may **not** use any mutating functions (e.g., `set!`), nor may you use any loop constructs (e.g., `for/list`). The purpose of this exercise is to get you writing pure recursive functions, and so you'll need to make good use of recursion to do so.

You may also **not** use the built-in `count` function (this defeats the purpose of the exercise).

You may, here and for the rest of the course, define your own helper functions, and use functions you're asked to implement in the implementation of other functions.

Tips:

- Use the menu command `Racket -> Reindent All` (or keyboard shortcut) to fix code formatting frequently. This is a great way to keep track of the structure of your code, and also quickly identify missing parentheses.
- Remember that Racket, at least for now, is *expression-based*. The *body* of a function should consist of a single expression, and the value returned by the function is simply the result of evaluating the body. Because of this, you don't need a `return` keyword like you'll be familiar with from other languages.

Task 2: Calculator I

Over the course of the next few weeks, we will build a calculator that will evaluate **binary arithmetic expressions**. The features supported by our language will grow over the next exercises. For now, our expressions will be very simple, and will have the following grammar:

```
Binary Arithmetic Expression Grammar

<expr> = NUM | (<op> <expr> <expr>)
<op>   = + | - | * | /
```

where `NUM` represents any integer literal in base 10, and the arithmetic operations `+`, `-`, `*`, `/` have the standard mathematical meanings.

Arithmetic expressions will be represented as a list of elements in Racket, and operators will be represented using symbols. For example, `(list '+ 2 3)` and `(list '/ 8 2)` are examples of valid expressions.

Note that with the way quoting distributes in Racket, `'(+ 2 3)` represents the same data structure as `(list '+ 2 3)`. Likewise, `'(+ 2 (+ 4 5))` represents `(list '+ 2 (list '+ 4 5))`. In other words, you should *not* nest the quote operator `'`.

Your tasks are as follows:

- Write a function `calculate` that returns the value of an expression.
  - **Hint**: You might find the Racket built-in functions `number?` and `list?` helpful.
- Write a function `calculate-many` that takes a list of expressions, and returns a list containing the values of each the expressions.
  - **Hint**: If you want to look ahead, try looking up the Racket built-in function `map`.

Update (Sep 12): You may not use the `eval` function. Your function should be recursive!

Submission instructions

Submit the file `ex1.rkt` MarkUs.