Homework Assignment 1

Part I

1. For this exercise, each student has their own mathematical expression. To obtain your expression, upload an incomplete submission (*e.g.,* file `hw1.rkt`) to our grading server and follow the URL given.

   (a) Implement the given expression in Racket and bind it to variable `ex1`. Note that we are interested in **syntactically** equivalent expressions, not just semantically equivalent, *e.g.* $2 + 3$ is syntactically *different* than $3 + 2$, although semantically equivalent.

   (b) Implement the sequence of evaluations of the given expression down to a value and bind that list to variable `ex2`, as we learned in the course. For instance, for expression $3.14159 \times (10 \times 10)$ you would write the following term.

```
(define ex2
  (list
    (* 3.14159 (* 10 10))
    (* 3.14159 100)
    314.159))
```

2. For this exercise, each student has their own Python code. To obtain your expression, upload an incomplete submission to our grading server and follow the instructions. Implement the given Python-like `ex3` function in Racket. Please use a **function definition** and not a basic definition. Additionally, note that the solution must be **syntactically** equivalent, not just semantically equivalent, that is, the body of `ex3` should be syntactically equivalent to the Python code. **Important:** If your expression contains `==`, then use Racket's `=`.

Part II

3. *Your goal is to implement the code in Listing 1 in Racket, as we learned in class: by using lists to define a user data-structure.*

   To this end, you will need to implement the constructor and selectors of each field, as well as the operation to *insert* a node in the BST. The code in Listing 1 is a Python implementation of binary tree taken from the Wikipedia page on BST's[1].

   - This exercise is about transferring your knowledge, from Python into Racket. You are being asked to "translate" an algorithm, **not** to rethink the algorithm.
   - The equivalent of `None` in Racket is `null`.
   - The equivalent of testing if a value `is None` in Racket is to call function `null?`.
   - Please use the function names declared in the homework assignment template, as otherwise you will get 0 points in this assignment.

4. *Your goal is to check if a datum is syntactically valid, with respect to the specification we introduced in class.*

   Recall function `quote` we learned in class. This function produces a logical representation of the code given as parameter. The serialized code that results from `quote` is known as a *datum*, or a *quoted* term. In the following exercises, the quoted term shall **not** include boolean expressions and conditionals. A quoted expression will include numbers, `define`, `lambda`, and function application.

   - For the sake of simplicity, there is no need to recursively check the syntactic validity (eg, you do not need to check the if the body of a `lambda` is syntactically valid). For instance, given a `lambda` are the parameters symbols? Does the body of a `lambda` has expected number datums as we discussed in class?

[1] `https://en.wikipedia.org/wiki/Binary_search_tree`

Listing 1: A binary search tree written in Python.

```python
class Tree:
  def __init__(self, left, value, right):
    self.left = left;
    self.value = value;
    self.right = right;

  def set_left(self, left):
    return Tree(left, self.value, self.right)

  def set_value(self, value):
    return Tree(self.left, value, self.right)

  def set_right(self, right):
    return Tree(self.left, self.value, right)


def insert(node, value):
  if node is None:
      return Tree(None, value, None)
  if value == node.value:
      return node.set_value(value)
  if value < node.value:
      return node.set_left(insert(node.left, value))
  return node.set_right(insert(node.right, value))
```

- You do *not* need to check the semantic validity of the datum (eg, check if a variable is defined).

(a) Function `lambda?` takes a datum and returns a boolean whether or not the quoted term is a `lambda`. You can check if a datum is a list of symbols with a combination of functions `symbol?`[2] and `andmap`:[3]

(b) Function `lambda-params` takes a quoted lambda and returns the list of parameters (symbols) of the given function declaration. *Hint:* Your solution can safely assume that the input is valid, no error checking required.

(c) Function `lambda-body` takes a quoted lambda and returns a list of terms of the given lambda. *Hint:* Your solution can safely assume that the input is valid, no error checking required.

(d) Function `apply?` takes a datum and returns a boolean whether or not the quoted term is a function application.

(e) Function `apply-func` takes a quoted function application expression and returns the function being called. *Hint:* Your solution can safely assume that the input is valid, no error checking required.

(f) Function `apply-args` takes a quoted function application expression and should return the arguments (expressions) of the function being called. *Hint:* Your solution can safely assume that the input is valid, no error checking required.

(g) Function `define?` takes a datum and returns a boolean whether or not the quoted term is a `define`. *Hint:* Solve this exercise *after* you solve `define-basic?` and `define-func?`.

(h) Function `define-basic?` takes a datum and returns a boolean whether or not the quoted term is a *basic* definition, according the specification we learned in class.

(i) Function `define-func?` takes a datum and returns a boolean whether or not the quoted term is a *function* definition, according to the specification we learned in class.

[2] `https://tinyurl.com/yblyxmoz`
[3] `https://tinyurl.com/y7kv2mzt`