CS116 Winter 2011 Assignment 3
Due: Tuesday, February 1st at 10:00AM

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

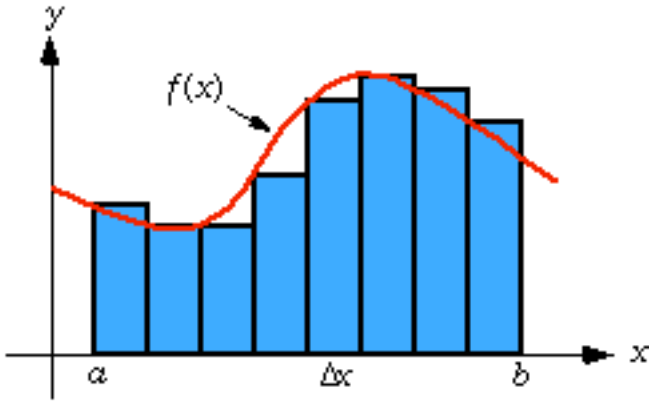
Important Information:

1. Read the course Style Guide for information on assignment policies and how to organize and submit your work. In particular, your solutions should be placed in files `a3qY.rkt`, where `Y` is a value from 1 to 4.
2. Be sure to download the interface file to get started.
3. Do not copy the purpose directly from the assignment description – it should be in your own words.
4. All helper functions should be contained within the main function definition using `local`.
5. When a ***short and simple**** function is used only once as a helper to an abstract list function, you must use lambda. For more complex helper functions, you may define them separately for readability.
6. For this assignment, use accumulative recursion as discussed in Module 3. **Solutions that do not use accumulative recursion where appropriate will not receive full marks.**
7. **You may not use abstract list functions for questions 1, 2 and 4. You may use them for question 3.**
8. **Hint: you may need several accumulators for some of the questions.**
9. You may assume that all consumed data satisfies any stated assumptions, unless indicated otherwise.

Language level: Intermediate Student with lambda.
Coverage: Module 3

1. The area under a curve, above the x-axis, and between two different points (x_1 x_2), where $x_1 < x_2$, can be approximated by the sum of the areas of the vertical rectangular bars of width 1 between the curve and x-axis, and between points x_1 and x_2. For example, the area under $y=x^2$, between 5 and 8 can be approximated as the sum: $5^2+6^2+7^2 = 25+36+49 =110$. Note that we do not use the square of the end point 8: 8^2 in our calculation.

   The following diagram shows that the area under f(x), between a and b is approximated by the sum of the eight shaded bars. Note that the value f(b) is not used for the total area calculation. In our method, we do not use the function value at the upper bound x_2, i.e. we do not use $f(x_2)$ to compute the total area. Another point is that in our calculation, $\Delta x=1$.



Complete the scheme function `area-under` which consumes a function named f and two different positive integers x_1 and x_2 ($x_1 < x_2$). It produces the approximation of the area (an integer) under the curve between x_1 and x_2 according to the rule given above.

If the curve is under the x-axis, the area is considered as a negative value. This is because when $f(x_i) <= 0$, the area equals $f(x_i)*1$ which is less than zero. Since we are producing an integer value, you should use the `round` function to round your result to an integer. Sometimes, the produced area is an inexact number. In this case, even after you use the `round` function to round it to an "integer", this "integer" is still an inexact number. In this case, you will need to use `check-within` to test your function. For example, `(check-within (area-under sin 22 50) -2 0.0001)` allows ±0.0001 range for your result -2.

To demonstrate that your program works for any function, you will also need to define two functions used in this assignment: $f_1=5x^3$ and $f_2=7\log x$. Since you will be using them for testing, f_1 and f_2 will have to be defined at the top level.

Here are some examples:

```
(area-under sqr 2 5) => 29
(area-under sin 22 50) => #i-2.0
(area-under f1 10 30) => 936000
(area-under f2 2 8) => #i60.0
(area-under (lambda (x) (* 1.5 x) 1 2) => 2
```

2. A Laputa crystal grows in the following manner:

   - at time 0, it is a solid of size (1 micrometre x 1 micrometre x 1micrometre) where a micrometre is one millionth of a metre.
   - during the first millisecond (ms), it grows only along x-axis to twice as long as its original length on x-axis (no growth along y-axis or z-axis) ,
   - during the second ms, it grows only along y-axis to three times as long its original length on y-axis (no growth along x-axis or z-axis) ,
   - during the third ms, it grows only along z-axis to four times as long as its original length on z-axis (no growth along x-axis or y-axis) ,

- during the 4th ms, it grows only along x-axis again, but this time, it grows to five times as long as its previous length along x-axis, i.e. the length at the end of the first ms,
- during the 5th ms, it grows only along y-axis again, but this time, it grows to six times as long as its previous length along y-axis, i.e. the length at the end of the second ms,
- … (this growth pattern continues: during each additional millisecond (ms), the crystal grows along the next axis and the growth rate is incremented by 1 from the previous growth rate. **Hint: the growth rate= time in ms +1**.)

Write a Scheme function `laputa-volume` which consumes an integer `ms` and produces another integer `volume`. Here, `ms` is the time in millisecond [ms>0] and `volume` is the volume of a Laputa crystal in micrometre cubed (micrometre^3) after `ms` milliseconds of growth.

For example:

```
(laputa-volume 2) => 6,
(laputa-volume 6) => 5040,
(laputa-volume 10) => 39916800.
```

3. A key or a note on a piano can be represented as a pair of integers (a, b) where a is the number of the octave between [0..8] and b is the sequence number within an octave between [1..12]. For example, the left most key of an 88-key piano is represented as (0,10) and the right most key is represented as (8,1). The Middle C is represented as (4,1). Recall that a pair is a list of two items.

   We use `bar` to represent a bar of notes in a piece of music. A `bar` is a `(listof note)[Non-Empty]`. The range of two different keys (key_1 and key_2) is the number of keys in between plus two, i.e. number of all the keys from key_1 to key_2 inclusive. For example, the range of middle C (4, 1) and middle E (4, 5) is 5. The range of key (4, 8) and (5, 3) is 8. The range of a bar of music is the range of the highest note and the lowest note. For example, the first bar of "Space Oddity" by David Bowie is `'((5 1) (5 1) (5 1) (5 1) (5 1) (5 1) (5 3) (5 1) (4 12))`, and its range is 4. If two keys are the same, the range is 1.

   Complete a Scheme function `largest-range`, which consumes a list of bars with at least two notes and produces a pair `(bar, range)` in which `bar` is the bar that has the largest range and `range` is the largest range. In case of tie, the first occurrence is returned. **Hint: you may use abstract list functions for your intermediate computations, but you must use accumulative recursion for your final results.**

   For example:

```
(largest-range '(((3 7) (3 4)  (4 1) (3 8))
                 ((4 5) (4 11) (5 4))
                 ((5 5) (5 3)  (5 11) (5 11) (6 2))))
=> '('('(4 5) '(4 11) '(5 4)) 12)
```

```
(largest-range  '(( 0 10) ( 2 3) (1 1))
                  ((4 1)(4 10 )(5 2))
                  ((3 5) (4 8) (2 5) (1 12))
                  ((1 2) (3 4) (1 12))))
=> '('((3 5) '(4 8) '(2 5) '(1 12)) 33)
```

4. Complete the Scheme function `longest-idle-period`, which consumes a pair `one-computer-usage`. This pair contains a string that is the name of a computer and a non-empty list of 24 zeros and ones that represents whether or not a computer is in use during the 24 hours of a day. `longest-idle-period` returns a list that contains the computer name, the starting hour of the day (in a 24-hour clock) that the computer has the longest stretch of idle period, and the number of hours in the longest stretch of idle period. If there is more than one period that has the same longest idle hours, the earliest occurrence is returned. **Hint: you will need several accumulators for this question. You will need to keep tract of the starting hour of the longest period, the longest period length so far, the current position in the usage list, and a couple of other values. Draw yourself a memory model to help you keep tract of how these values change.**

The 24-hour day starts from 12 am and ends at 12 am the next day. The starting hour on a 24-hour clock is represented as hour 0, and the last hour of the day is represented as hour 23. The first digit in the non-empty list of 24 zeros and ones represents the usage status from midnight to 1am. The second digit in the list represents the computer usage status between 1am and 2am, and so on. The value 0 in the list represents that the computer is never in use for a particular hour and 1 represents the computer is in use sometime during a particular hour.

For example, the pair (`"MC2037-1" '(0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 ))` represents that the computer named `"MC2037-1"` is in use from 1am to 7am, from 11am to 5pm, and from 11pm to 12pm.

Examples for `longest-idle-period`:

```
(longest-idle-period("MC2037-1" '(0 1 1 1 1 1 1 0 0 0 0 1
1 1 1 1 1 0 0 0 0 0 0 0))) => '("MC2037-1" 17 7)

(longest-idle-period("MC2037-9" '(1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1))) => '("MC2037-9" 0 0)

(longest-idle-period("MC2037-21" '(0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0))) => '("MC2037-21" 0 24)

(longest-idle-period("MC2026-3" '(0 1 0 0 1 1 1 0 0 0 0 1
1 1 1 1 1 0 0 0 0 1 0 1))) => '("MC2026-3" 7 4)
```