`pQuadratic` — Using REPL

Learning Outcomes

After completing this program, students will be able to

- Start and stop the **Racket Scheme** REPL.
- Start and stop the **Java** `jshell` REPL.
- Write and run a simple function in both Scheme and Java using their respective REPL.

This assignment is an introdution to using a *read-execute-print* loop (REPL). Students will start with a Java REPL, using a language they know in a new, interpreted, way. Then they will explore the MIT Scheme REPL, the version of Scheme that we will be using this semester, doing the same work in the new language.

Procedure

1. Read the **whole** assignment. This is important for *every* assignment: it puts the task into your brain so that it can begin working on answering the questions. Of particular interest when you read are the SLO (first section above); gives you the links between the assignment to the big picture (learning computer science).

2. Configure your Racket by installing the `simply-scheme` language definition.

   - This need only be done once on the lab computers (or any other Racket/Dr. Racket installation).
   - Using the `raco` program, install the `simply-scheme` language definition into your local directory:

     ```
     $ raco pkg install simply-scheme
     ```

     It should then download and compile some amount of code. You can test that it works by starting `racket` with the new language definition:

     ```
     $ racket -I simply-scheme
       Welcome to Racket v8.14 [cs].
       >
     ```

     As shown, it should start the REPL w/o any error messages. `Ctrl-D` (for "end-of-file") or the command `(exit)` terminates the Racket REPL.

3. Review how to solve the *quadratic equation*:

   - What are the *input* value(s)?
   - What are the *output* value(s)?
   - How is it calculated?

4. Figure out how you will capture (to a text file) a session inside a terminal. You will be turning in one text file containing a session with the Scheme REPL and a session with the Java REPL.

   Probably the easiest way to do this is to cut and paste out of the terminal window. The terminal is using special characters to move the cursor around and those are captured by character capture programs.

The Java REPL

5. Since Java version 9, the package has included the `jshell` program in the SDK distribution. If the Java binaries (think `javac`, `java`) are in the path in your terminal, then `jshell` should run:

```
$ jshell
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=lcd
  Welcome to JShell -- Version 21.0.4
  For an introduction type: /help intro

jshell>
```

[1]

Use `help` to get interactive help inside the program.

At the prompt you are in an interactive Java environment. To define a function, say `square`, just write what would go inside the class definition:

```
jshell> int square(int n) {
   ...>     return n * n;
   ...> }
  created method square(int)

jshell> square(7)
$2 ==> 49
jshell>
```

`jshell` (like many shell-like programs) ends when it encounters the `^D` (Ctrl-D) or End-Of-File character.

The Scheme REPL

6. Racket (Dr. Racket is the GUI version; we will use the CLI) is a Scheme implementation that is designed for teaching and for defining other languages (variants of Scheme, mostly). We will use the `simply-scheme` variant so that we can use the exercises in that book.

   Start the Racket REPL with the command `racket`. Quitting, as described above, uses `Ctrl-D` or the `(exit)` command.

   You can submit properly formatted Scheme expressions to the REPL and it will **R**ead, **E**valuate, and **P**rint the results in a **L**oop. You can do math or define functions, like `square`:

```
$ racket -I simply-scheme
Welcome to Racket v8.14 [cs].
> (+ 1 2 3)
6
> (list (* 1 5) (* 2 5) (* 3 5))
'(5 10 15)
> (define (square n)
    (* n n)
```

[1] `_JAVA_OPTIONS` is set on lab machines to improve GUI appearance.

```
  )
> (square 7)
49
>
```

Error Handling `racket` shows an error message when something is not right:

```
> (* a b)
a: undefined;
 cannot reference an identifier before its definition
  in module: top-level
 [,bt for context]
> user break [,bt for context]
```

Your Quadratic Equation

7. Each student will generate their own quadratic equation. Take each of your three (3) initials (if you have fewer, select enough letters to have three; if you have more, use the first three). Translate each into a number by mapping the alphabet to the range 3..28: a=3, b=4, c=5, ..., y=27, z=28.

   Call the three numbers i_1, i_2, and i_3 (initials one through three).

   Your quadratic equation is then

   $(i_1 + i_2)x^2 + (i_1 + i_3)x - (i_2 + i_3)$

   For example:

   Brian C. Ladd
   $i_1 = 4, i_2 = 5, i_3 = 14$.
   $q(x) = 9x^2 + 18x - 19$

The Questions

8. For each of the two REPLs: run the REPL and capture the whole session into a text file. Use the programming language in the interpreter to answer the following questions, once in each language.
   1. What version of the REPL are you running? (Very Easy)
   2. Find the *cube* of $i_1 + i_2 + i_3$.
   3. Using the *quadratic formula*, find **both** real roots (zeroes) of *your* quadratic equation.
   4. Write a function, `q` that takes a floating point number and evaluates your quadratic equation at that point. Use it to verify your zeros and determine whether the quadratic is *positive* or *negative* between the two zeroes.

Submit through Classroom Management System

9. Submit *both* interactive sessions, one for Java and one for Scheme, in one text file.