LET IS LAMBDA

Here is the structure of a let expression:\(^1\):

\[
\begin{align*}
\text{(let ((<var_1> <exp_1>)} \\
&\quad (\text{<var_2> <exp_2>}) \\
&\quad \vdots \quad (\text{<var_n> <exp_n>}) \\
&\text{<body>)}
\end{align*}
\]

This creates a frame in which each of the let variables are bound to their assigned values, and then the body is evaluated in the context of that frame.

In fact, that is exactly what procedure application is, and indeed, the Scheme interpreter dynamically re-writes let expressions as lambda constructions and applications.

So... in a profound sense... let is lambda.

**Question 1.** Rewrite the following procedure using lambda instead of let. Keep symbol names as they are in your rewrite.

\[
\begin{align*}
\text{(define (quadratic x y)} \\
&\quad \text{(let ((a (+ x y))} \\
&\quad &\quad (b (- x y))} \\
&\quad &\quad (+ (* a a) (* b b))))
\end{align*}
\]

Here are the rules for environment diagrams and procedure application again:

Frames contain bindings from symbols to values.

define creates a binding from a symbol to a value.

set! rebinds a symbol (which must already exist) to a new value.

Procedures are double-circle diagrams, where
the left circle points to the procedures parameters and body, and
the right circle points to the frame context from where the procedure was created

To apply a procedure:
create a new frame,
in it bind the parameters' symbols to their associated values,
and link that frame back to the frame from which the procedure was created, then
evaluate the body of the procedure from this new frame.”

Question 2. Draw the environment structure generated by the following code. Also indicate the ultimate value of the procedure application.

```
(define (quadratic x y)
  (let ((a (+ x y))
       (b (- x y)))
    (+ (* a a) (* b b))))

(quadratic 5 2)
```