Rules for Environment Diagrams

- **Variables:**
  Look up the variable name in the specified environment frame and find the value it is bound to. If the variable is not in the current frame, look in enclosing frames by following the frame arrows. If you reach the Global Environment and the variable is still not bound, then the variable is unbound.

- **Lambda Special Form:**
  Create a procedure object (two bubbles). The left bubble records the formal parameter list and the body of the procedure. The right bubble points to the environment in which the procedure was created.

- **Define Special Form:** (define <name> <expression>)
  Evaluate the expression, then bind the name to this value in the current frame.

- **Procedure Application:**
  1. Draw a new frame.
  2. Draw a dotted line from the procedure object to this new frame.
  3. Figure out where the frame points to (its parent frame): find the frame that the right bubble of the procedure you are applying points to, and make the new frame point to the same place. An alternative (or additional) idea is to “link” the environment pointer in the procedure and the parent frame pointer together to make obvious that these are, in fact, the same pointer.
  4. Bind the formal parameters of the procedure to their respective actual argument values in this new frame.
  5. Evaluate the body of the procedure with respect to this new environment.

- **Let Special Form:**
  One could desugar the let into a lambda application. A short cut is to “hang” a new frame (that is, create a new frame that points to the current frame) with the variables of the let statement bound to their appropriate values (be careful when finding these values – they are evaluated in the current frame, not the new frame). Then evaluate the body of the let in the new environment.

- **Set! Special Form:**
  Find the variable name (use the instructions above), and change its value.
Block Structure, Lexical Scoping and the Environment Model

\[
\begin{align*}
\text{params: } & a \ b \ c \\
\text{body: } & (\text{define } G \ldots) \\
& \ldots \\
& (+ (G \ a \ b) \ldots)
\end{align*}
\]

\[
\begin{align*}
\text{params: } & a \ x \\
\text{body: } & (\ast a \ b)
\end{align*}
\]

\[
(\text{define } F \ (\text{lambda} \ (a \ b \ c)
& \ (\text{define } G
& \ (\text{lambda} \ (b \ a)
& & (- a \ b))
& \ (\text{define } H
& \ (\text{lambda} \ (a \ x)
& & (* a \ b))
& (+ (G \ a \ b)
& (H \ b \ a)))
& (F \ 2 \ 3 \ 4)
& \ldots)
\end{align*}
\]

\[
\begin{align*}
\text{params: } & a \ b \\
\text{body: } & (+ (G \ a \ b)
& (H \ b \ a)) \ w.r.t. \ E1
\end{align*}
\]

\[
\begin{align*}
\text{params: } & b \ a \\
\text{body: } & (- a \ b) \ w.r.t. \ E2
\end{align*}
\]

\[
\begin{align*}
\text{params: } & a \ x \\
\text{body: } & (* a \ b) \ w.r.t. \ E3
\end{align*}
\]
Draw the changes that occur in the empty environment as the following code is evaluated in the GE:

\[
\text{(begin (define x 10) (define y 20))}
\]

Desugar the following let statement, and show its evaluation (with respect to GE) in the environment diagram above:

\[
\text{(let ((y (* 5 y)))}
\text{(* x y))}
\]
Draw the changes that occur in the empty environment as the following code is evaluated in the GE:

```
(define prev?
  (let ((last 'undef))
    (lambda (x) (cond ((eq? x last) #t)
          (else (set! last x)
               #f))))
```

Add to the environment diagram as the following expressions are evaluated in the global environment:

```
(prev? 10)
```

```
(prev? 10)
```