45. a.

$$\frac{\begin{array}{c}\ell \notin \operatorname{dom} \sigma \\ \langle e, \rho\{x \mapsto \ell\}, \sigma\{\ell \mapsto \text{unspecified}\}\rangle \Downarrow \langle v, \sigma'\rangle\end{array}}{\langle \text{VAL}(x, e), \rho, \sigma\rangle \rightarrow \langle \rho\{x \mapsto \ell\}, \sigma'\{\ell \mapsto v\}\rangle}$$

b.
```
1 (val x 2)
2 (define f() x)
3 (val x 3)
4 (f)
```

If val uses the Scheme semantics, (f) will evaluate to 3 because the third line is equivalent to (set x 3), thereby changing what x binds to. So, when f is called, x is bound to 3. If the new semantics are used, the fourth line (f) will evaluate to 2, because it was defined in an environment where x is bound to 2, and the third line simply creates a new binding. Since the binding of x to 2 still exists, (f) will use that binding to believe that x binds to 2.

c. I prefer the original Scheme semantics because it is much less confusing when writing code. It makes sense that if you use 'val', then the name you defined should <u>only</u> bind to one value in one location. This makes it easier to keep track of the state of your environment, and allows you to change the value of a variable, then call a previously defined function which uses that variable, and it will have the newly assigned value.