3.4 Local Binding

- Recall Scheme's **`let`**:

```
> (let ((x 5)
        (y 6))
    (+ x y))
11
```

- Make **`let`** look different in our defined language:

```
--> let x = 5
        y = 6
    in +(x, y)
11
```

3.4 Local Binding: Step 1

- Syntax:

<expression> ::= **let** {<identifier> **=** <expression>}*
in <expression>

- Add some code to grammar:

```
(define grammar-3-4
  '((program (expression) a-program)
    (expression (number) lit-exp)
    (expression (identifier) var-exp)
    . . .
    (expression
      ("let" (arbno identifier "=" expression)
       "in" expression)
      let-exp)
```

3.4 Local Binding: Step 2

- Add some code to datatype definition:

```
(define-datatype expression expression?
        ...
      (let-exp
        (ids (list-of symbol?))
          (rands (list-of expression?))
          (body  expression?)))
```

3.4 Local Binding: Step 3

- Add some code to **`eval-expression`**:

```
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      ...
      (let-exp (ids rands body)
        (let ((args (eval-rands rands env)))
          (eval-expression body
                           (extend-env ids args env))))
```