ENVIRONMENTS

Requirements for Handling Variable Assignments.

- Whenever we encounter a lambda-app expression with at east one parameter, we need to store variable assigment.
- Example:
- `((lambda (x) x) 5)` -> Need to store: `x := 5`

Requirements

- Consider:

```
((lambda (x y) (+ x y))) 6 7)
```

- We need to store the variable names and associated values:

```
'(x y) '(6 7)
```

Requirements

- Now consider a nested procedure:

```
((lambda (x y) ((lambda (a b) (+ x y a b)) 1 2) 3 4)
```

- We need to store:

```
'(x y) '(3 4)
'(a b) '(1 2)
```

Requirements

- Now consider a nested procedure with shadowing:

```
((lambda (x y) ((lambda (x y) (+ x y x y)) 1 2) 3 4)
```

- We need to store:

```
'(x y) '(3 4)
'(x y) '(1 2)
```

- What is the value of this expression?
- Are we going to get this value, if we start our search for a variable in the most recent assignment?

Requirements

- No consider two procedures in the body :

```
((lambda (x y) ((lambda (a b) (+ a b x y))) 1 2)
               ((lambda c d) (+ c d x y)) 3 4)) 5 6)
```

- We need to store: `'(x y) '(5 6)` `[e1]`
- As well as: `'(a b) '(1 2)` `[e2]`
- And: `'(c d) '(3 4)` `[e3]`
- Notice that `[e2]` needs to have access to `[e1]` .
- Euqally, `[e3]` need to have access to `[e1]`
- However, `[e2]` and `[e3]` should not have access to each other.

Interacting with Environments

- Environments are used to store variable assignments.
- We need to be able to:
  1. construct an empty environment
  2. extend an environment with variable assignments
  3. look up variables in our environment.

Use

- Variable assignments are created whenever we see a lambda-app expression (or something that can be converted to it.)
- Environments are passed along to the evaluation of the body.
- When encountering a variable expression, the environment is searched to find the appropriate assignment.
- We do not need to remove variable assignments.
- This is because of the scoping rules of Scheme.

Environment ADT

- (empty-env) = [ø]
- (apply-env [*f*] s) = *f*(s)
- (extend-env '(s_1 … s_k) '(v_1 … v_k) [*f*]) = [*g*]

where $g(s') = \begin{cases} v_i, \text{ if } s' = s_i \text{ for some } i,\ 1<=i<=k \\ f(s') \text{ otherwise} \end{cases}$

- In other words, s' either occurs in the current environment or we look it up in the enclosed environment *f*.