Today we will learn...

- Introduce mutable environments, composed of frames
- Implement frames

Section 3.2 of the SICP book. The interactive version of Section 3.2.
Visualizing the environment
Environment visualization

**Figure 3.1:** A simple environment structure.
Source: SICP book Section 3.2

```
; E0 = (handle 0)
E0: [  
  (x . 3)  
  (y . 5)  
]

; E1 = (handle 1)
E1: [  E0  
  (z . 6)  
  (x . 7); shadows E0.x  
  ; (y . 5)  
]

; E2 = (handle 2)
E2: [  E0  
  (m . 1)  
  (y . 2); shadows E0.y  
  ; (x . 3)  
]```
The heap at runtime

- arrows are *references*, or heap handles:
- boxes are *frames*: labelled by their handles
- each frame has local variable bindings (e.g., m:1, and y:2)

**Figure 3.1:** A simple environment structure.

Source: SICP book Section 3.2
Source: SICP book Section 3.2

The heap at runtime

- arrows are *references*, or heap handles:
- boxes are *frames*: labelled by their handles
- each frame has local variable bindings (eg, m:1, and y:2)
- an *environment* represents a *sequence of frames*, connected via references. For instance, the environment that consists of frame 3 linked to frame 1.
- variable lookup follows the reference order. For instance, lookup a variable in frame 3 and then in frame 1.

*Figure 3.1: A simple environment structure.*
Source: SICP book Section 3.2
List all variable bindings in environment (handle 1)

**Figure 3.1**: A simple environment structure.
Source: SICP book Section 3.2
Implementing mutable environments
Implementing mutable environments

Heap

- A heap contains **frames**

Frame

- a reference to its parent frame (except for the root frame which does not refer any other frame)
- a map of local bindings

Example of a frame: \[ E0 (y . 1) \]
Example of a root frame: \[ (a . 20) (b . (\text{closure } E0 (\lambda y a))) \]
Let us implement frames...
(demo time)
Usage examples

```
; (closure E0 (lambda (y) a)
  (define c (d:closure (handle 0) (d:lambda (list (d:variable 'y)) (d:variable 'a))))
;E0: [
;   (a . 20)
;   (b . (closure E0 (lambda (y) a)))]

(define f1
  (frame-put
    (frame-put root-frame (d:variable 'a) (d:number 10))
    (d:variable 'b) c))
(check-equal? f1 (frame #f (hash (d:variable 'a) (d:number 10) (d:variable 'b) c)))

; Lookup a
(check-equal? (d:number 10) (frame-get f1 (d:variable 'a)))

; Lookup b
(check-equal? c (frame-get f1 (d:variable 'b)))

; Lookup c that does not exist
(check-equal? #f (frame-get f1 (d:variable 'c)))
```
More usage examples

```
; E1: [ E0
    ; (y . 1)
]
(define f2 (frame-push (handle 0) (d:variable 'y) (d:number 1)))
(check-equal? f2 (frame (handle 0) (hash (d:variable 'y) (d:number 1))))
(check-equal? (d:number 1) (frame-get f2 (d:variable 'y)))
(check-equal? #f (frame-get f2 (d:variable 'a)))
;; We can use frame-parse to build frames
(check-equal? (parse-frame '[[ a . 10) (b . (closure E0 (lambda (y) a))]]]) f1)
(check-equal? (parse-frame '[[ E0 (y . 1) ]]) f2))
```
Frames

\( (\text{struct frame (parent locals))} \)

- parent is either \#f or is a reference to the parent frame
- locals is a hash-table with the local variables of this frame

Constructors

\[
(\text{struct frame (parent locals) #:transparent})
(\text{define root-frame (frame \#f (hash)))}
(\text{define (frame-push parent var val)}
  (frame parent (hash var val)))
(\text{define (frame-put frm var val)}
  (frame (frame-parent frm))
  (hash-set (frame-locals frm) var val)))
(\text{define (frame-get frm var)}
  (hash-ref (frame-locals frm) var \#f))
\]

Description

- root-frame creates an orphan empty frame (hence \#f). This function is needed to represent the top-level environment.
- frame-push takes a reference that points to the parent frame, and initializes a hash-table with one entry (var, val). This function is needed for \( E \leftarrow E' + [x := v] \)
- frame-put updates the current frame with a new binding. This function is needed for \( E \leftarrow [x := v] \)