SCHEME

COMPUTER SCIENCE MENTORS 61A

March 14 to March 19, 2016

1 What Would Scheme Print?

Solution: Solutions begin on the following page.

1. What will Scheme output? Draw box-and-pointer diagrams to help determine this.

(a) `(cons (cons 1 nil) (cons 2 (cons (cons 3 (cons 4 5)) (cons 6 nil))))`

Solution:
`((1) 2 (3 4 . 5) 6)`

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

Solution:
`4`

(c) `((lambda (x y z) (y x)) 2 / 2)`

Solution:
`0.5`

(d) `((lambda (x) (x x)) (lambda (y) 4))`

Solution: 4

(e) `(define boom1 (/ 1 0))`

Solution: Error: Zero Division

(f) `boom1`

Solution: Error: boom1 not defined

(g) `(define boom2 (lambda () (/ 1 0)))`

Solution: boom2

(h) `(boom2)`

Solution: Error: Zero Division

(i) Why/How are the two "boom" definitions above different?

Solution: The first line is setting boom1 to be equal to the value `(/ 1 0)`, which turns out to be an error. On the other hand, boom2 is defined as a lambda that takes in no arguments that, when called, will evaluate `(/ 1 0)`.

(j) How can we rewrite boom2 without using the lambda operator?

Solution:
```
(define (boom2) (/ 1 0))
```

2. What will Scheme output?

(a) `(if (/ 1 0) 1 0)`

Solution:
```
Error: Zero Division
```

(b) `(if 1 1 (/ 1 0))`

Solution:
```
1
```

(c) `(if 0 (/ 1 0) 1)`

Solution:
```
Error: Zero Division
```

(d) `(and 1 #f (/ 1 0))`

Solution:
```
#f
```

(e) `(and 1 2 3)`

Solution:
```
3
```

(f) `(or #f #f 0 #f (/ 1 0))`

Solution:
`0`

(g) `(or #f #f (/ 1 0) 3 4)`

Solution:
`Error: Zero Division`

(h) `(and (and) (or))`

Solution:
`#f`

(i) Given the lines above, what can we say about interpreting `if` expressions and booleans in Scheme?

Solution: `if` functions and boolean expressions will short-circuit, just like in Python. All values have a boolean value of `#t` unless they are specifically `#f`. This means that unlike in Python, 0 and 1 are both considered `#t`!

3. The following line of code does not work. Why? Write the lambda equivalent of the `let` expressions.

```
(let ((foo 2)
      (bar (+ foo 2)))
    (+ foo bar))
```

Solution: The above function will error because it is equivalent to:
`((lambda (foo bar) (+ foo bar)) 2 (+ foo 2))`

In other words, foo has not been defined in the global frame. When bar is being assigned to `(+ foo 2)`, it will error. The assignment of foo to 2 happens in the lambdas frame when it's called, not the global frame.

If we had the line `(define foo 3)` before the call to let, then it would return 7, because within let, foo would be 2 and bar would be `(+ 3 2)`, since it would use the foo in the Global frame.

2 Scoping

4. What is the difference between dynamic and lexical scoping?

Solution:

- **Lexical:** The parent of a frame is the frame in which a procedure was defined (used in Python).
- **Dynamic:** The parent of a frame is the frame in which a procedure is called.

5. What would this print using lexical scoping? What would it print using dynamic scoping?

```
a = 2
def foo():
    a = 10
    return lambda x: x + a
bar = foo()
bar(10)
```

Solution:

- **Lexical:** 20
- **Dynamic:** 12

6. How would you modify and environment diagram to represent dynamic scoping?

Solution: Assign parents when you create a frame (do not set parents when defining functions!). The parent in this case is the frame in which you called this function.

7. Implement `waldo`. `waldo` returns `#t` if the symbol waldo is in a list. You may assume that the list passed in is well-formed.

```
scm> (waldo '(1 4 waldo))
#t
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f
```

Extra challenge: Define `waldo` so that it returns the index of the list where the symbol waldo was found (if waldo is not in the list, return `#f`).

```
scm> (waldo '(1 4 waldo))
2
scm> (waldo '())
#f
scm> (waldo '(1 4 9))
#f
```

Solution:

```
(define (waldo lst)
    (cond ((null? lst) #f)
          ((eq? (car lst) 'waldo) #t)
          (else (waldo (cdr lst)))
      )
  )
```

Challenge solution:

```
(define (waldo lst)
    (define (helper lst index)
        (cond ((null? lst) #f)
              ((eq? (car lst) 'waldo) index)
              (else (helper (cdr lst) (+ index 1)))
          )
      )
    (helper lst 0)
  )
```