Dynamic Scope

The way in which names are looked up in Scheme and Python is called lexical scope. You can see what names are in scope by inspecting the definitions:

Lexical scope: The parent of a frame is the environment in which a procedure was defined.

Dynamic scope: The parent of a frame is the environment in which a procedure was defined.

### Example: Length of a List

```python
def length(x):
    if null?(x):
        return 0
    else:
        return 1 + length(cdr(x))
```

### Tail Recursion

From the Revised Report on the Algorithmic Language Scheme:

“Implementations of Scheme are required to be properly tail-recursive. This allows the execution of an iterative computation in constant space, even if the iterative computation is described by a syntactically recursive procedure.”

```scheme
(define (factorial n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1))))
)
```

In Python, recursive calls always create new active frames. The last sub-expression in a tail context

- All non-predicate sub-expressions in a tail context are evaluated
- The last body sub-expression is in a tail context
- All non-predicate sub-expressions in a tail context are evaluated
- The last body sub-expression is in a tail context

### Tail Calls

A procedure call that has not yet returned is called a tail call. Scheme interpreters should support an extended number of active tail calls using only a constant amount of space.

A tail call is a call expression in a tail context:

- The last body sub-expression is a `lambda` expression
- Tail-expression 2 is a tail context if expression
- All non-predicate sub-expressions in a tail context are evaluated
- The last non-expression is a tail context and, or, begin, or let

```scheme
(define factorial (lambda (n)
  (if (<= n 1)
      1
      (* n (factorial (- n 1))))
))
```

Tail Call

Tail calls are not tail calls if more computation is still required on the calling procedure.

Linear recursive procedures can often be rewritten to use tail calls.

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```
Eval with Tail Call Optimization
The return value of the tail call is the return value of the current procedure call. Therefore, tail calls shouldn’t increase the environment size.

Tail Recursion Examples

Which Procedures are Tail Recursive?
Which of the following procedures run in constant space?

(1) 
;; Compute the length of s.
(define (length s)
  (if (null? s)
      -1
      (add1 (length (cdr s)))))

;; Return the nth Fibonacci number.
(define (fib n)
  (define (fib-iter current k)
    (if (= k n)
        current
        (fib-iter (+ current (fib (- k 1))) k))))
  (if (= 1 n)
      0
      (fib-iter 1 2)))

;; Return whether s contains v.
(define (contains s v)
  (if (null? s)
      false
      (if (= v (car s))
        true
        (contains (cdr s) v))))

;; Return whether s has any repeated elements.
(define (has-repeat s)
  (if (null? s)
      false
      (if (contains? (cdr s) (car s))
          true
          (has-repeat (cdr s)))))

Map
and Reduce

Example: Reduce
(define reduce-procedure s start)
(if (null? s)
    start
    (reduce-procedure (cdr s) procedure start (car s))))

Recursive call is a tail call
Space depends on what procedure requires
(reduce * '(3 4 5) 2) 120
(reduce (lambda (x y) (cons y x)) '(3 4 5) '(2)) (5 4 3 2)

Example: Map with Only a Constant Number of Frames
(define map-procedure s)
(if (null? s)
    nil
    cons (procedure (car s)) (map-procedure (cdr s))))

An Analogy: Programs Define Machines
Programs specify the logic of a computational device

Interpreters are General Computing Machine
An interpreter can be parameterized to simulate any machine

General Computing Machines

An Analogy: Programs Define Machines
Programs specify the logic of a computational device

Interpreters are General Computing Machine
An interpreter can be parameterized to simulate any machine

Our Scheme interpreter is a universal machine
A bridge between the data objects that are manipulated by our programming language and the programming language itself
Internally, it is just a set of evaluation rules