CS 61A
Fall 2025

# Interpreters

Discussion 11: November 12, 2025

Switch to Pensieve:

- **Everyone**: Go to pensieve.co, log in with your @berkeley.edu email, and **enter your group number** as the room number (which was in the email that assigned you to this discussion). As long as you all enter the same number (any number), you'll all be using a shared document.

Once you're on Pensieve, you don't need to return to this page; Pensieve has all the same content (but more features). If for some reason Penseive doesn't work, return to this page and continue with the discussion.

## Attendance

Your TA will come around during discussion to check you in. You can start on the worksheet before being checked in; you don't need to wait for your TA to get started.

If you didn't attend for a good reason (such as being sick), fill out this form (within 2 weeks of your discussion): attendance form

## Getting Started

Everybody say your name, and then figure out who is planning to travel outside of the Bay Area the soonest. Feel free to discuss your travel plans.

## Representing Lists

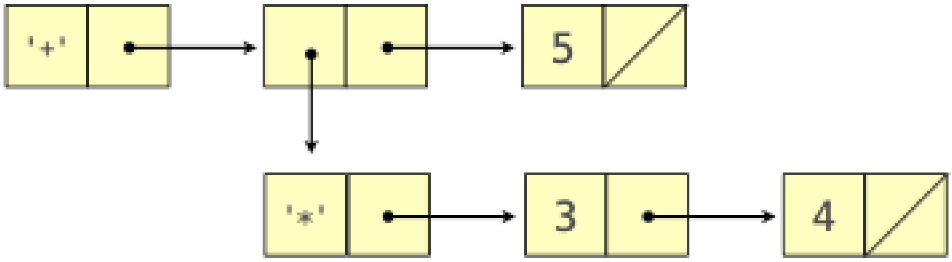
A Scheme call expression is a Scheme list that is represented using a `Link` instance in Python.

For example, the call expression `(+ (* 3 4) 5)` is represented as:

```
Link('+', Link(Link('*', Link(3, Link(4, nil))), Link(5, nil)))
```

Those `nil`'s are optional because `nil` is `Link.empty`, which is the default second argument to the `__init__` method of the `Link` class.

```
Link('+', Link(Link('*', Link(3, Link(4))), Link(5)))
```



**(+ (* 3 4) 5)**

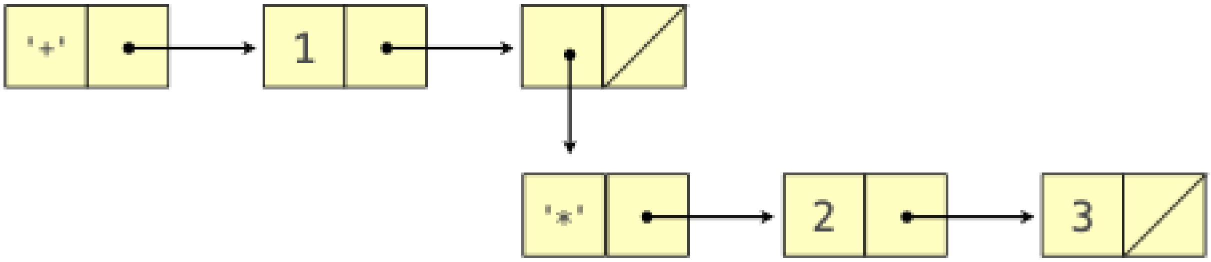
The `Link` class and `nil` object are defined in link.py of the Scheme project.

```
class Link:
    "A Scheme list is a Link in which rest is a Link or nil."
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest

    ... # There are also __str__, __repr__, and map methods, omitted here.

nil = Link.empty
```

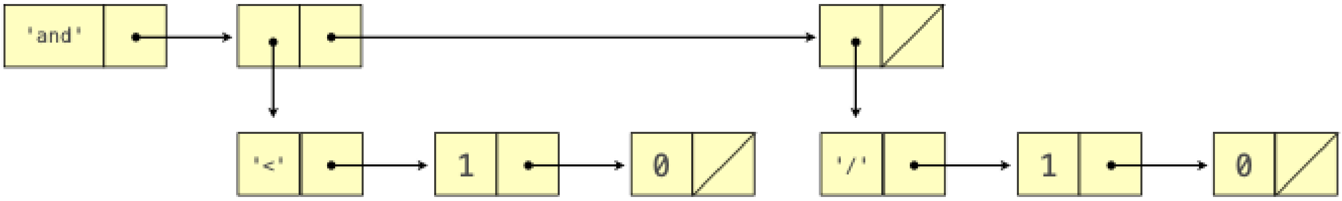
### Q1: Representing Expressions

Write the Scheme expression in Scheme syntax represented by each Link below. Try drawing the linked list diagram too.

```
>>> Link('+', Link(1, Link(Link('*', Link(2, Link(3, nil))), nil)))
```



```
>>> Link('and', Link(Link('<', Link(1, Link(0, nil))), Link(Link('/', Link(1, Link(0, nil
    ))), nil)))
```



**Discussion Time:** What does (and (< 1 0) (/ 1 0)) evaluate to? Discuss among your group until you all agree.

#f because the subexpression (/ 1 0) is never evaluated.

# Evaluation

To evaluate the expression (+ (* 3 4) 5) using the Project 4 interpreter, scheme_eval is called on the following expressions (in this order):

1. (+ (* 3 4) 5)
2. +
3. (* 3 4)
4. *
5. 3

6. 4
7. 5

**Discussion time:** Describe to each other why `*` is evaluated and what it evaluates to.

The `*` is evaluated because it is the operator sub-expression of `(* 3 4)`, which is an operand sub-expression of `(+ (* 3 4) 5)`.

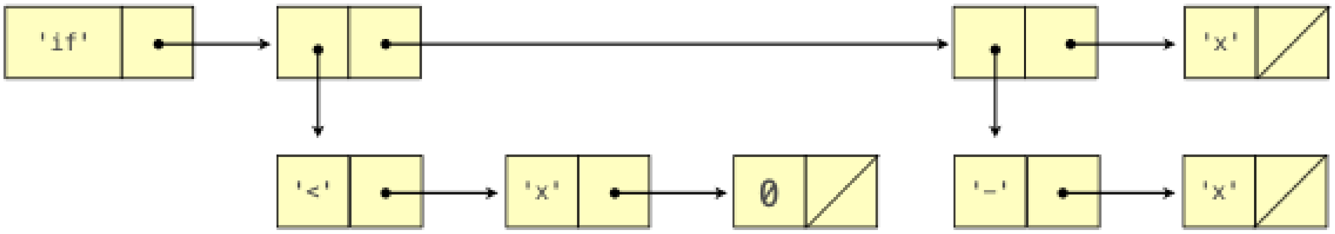
By default, `*` evaluates to a procedure that multiplies its arguments together. But `*` could be redefined at any time, and so the symbol `*` must be evaluated each time it is used in order to look up its current value.

```
scm> (* 2 3)  ; Now it multiplies
6
scm> (define * +)
*
scm> (* 2 3)  ; Now it adds
5
```

An `if` expression is also a Scheme list represented using a `Link` instance.

For example, `(if (< x 0) (- x) x)` is represented as:

`Link('if', Link(Link('<', Link('x', Link(0, nil))), Link(Link('-', Link('x', nil)), Link('x', nil))))`



To evaluate this expression in an environment in which `x` is bound to 2 (and `<` and `-` have their default values), `scheme_eval` is called on the following expressions (in this order): 1. `(if (< x 0) (- x) x)` 1. `(< x 0)` 1. `<` 1. `x` 1. `0` 1. `x`

**Discussion time:** Come up with a short explanation of why neither `if` nor `-` are evaluated even though they both appear in `(if (< x 0) (- x) x)`.

### Q2: Evaluation

(*Note*: Some past exams have had a question in exactly this format.) Which of the following are evaluated when `scheme_eval` is called on `(if (< x 0) (- x) (if (= x -2) 100 y))` in an environment in which `x` is bound to -2? (Assume `<`, `-`, and `=` have their default values.)

- `if`
- `<`
- `=`
- `x`
- `y`
- `0`
- `-2`
- `100`

*Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.*

- `-`
- `(`
- `)`

With `x` bound to -2, `(< x 0)` evaluates to `#t`, and so `(- x)` will be evaluated, but `(if (= x -2) 100 y)` will not. The operator and operands of a call expression are evaluated for every call expression that is evaluated. `(< x 0)` and `(- x)` are both call expressions.

### Q3: Print Evaluated Expressions

Define `print_evals`, which takes a Scheme expression `expr` that contains only numbers, `+`, `*`, and parentheses. It prints all of the expressions that are evaluated during the evaluation of `expr`. They are printed in the order that they are passed to `scheme_eval`.

**Note:** Calling `print` on a `Link` instance will print the Scheme expression it represents.

```
>>> print(Link('+', Link(Link('*', Link(3, Link(4, nil))), Link(5, nil))))
(+ (* 3 4) 5)
```

```
def print_evals(expr):
    """Print the expressions that are evaluated while evaluating expr.

    expr: a Scheme expression containing only (, ), +, *, and numbers.

    >>> nested_expr = Link('+', Link(Link('*', Link(3, Link(4, nil))), Link(5, nil)))
    >>> print_evals(nested_expr)
    (+ (* 3 4) 5)
    +
    (* 3 4)
    *
    3
    4
    5
    >>> print_evals(Link('*', Link(6, Link(7, Link(nested_expr, Link(8, nil))))))
    (* 6 7 (+ (* 3 4) 5) 8)
    *
    6
    7
    (+ (* 3 4) 5)
    +
    (* 3 4)
    *
    3
    4
    5
    8
    """
    if not isinstance(expr, Link):
        "*** YOUR CODE HERE ***"


    else:
        "*** YOUR CODE HERE ***"
```

If `expr` is not a link, then it is a number or `'+'` or `'*'`. In all of these cases, the `expr` should be printed to indicate that it would be evaluated.

If `expr` is a link, then it is a call expression. Print it. Then, the operator and operands are evaluated. These are the elements in the list `expr`. So, iterate through `expr` (using either a `while` statement or `expr.map(...)`) and call `print_evals` on each element.