

DISCUSSION 11





Programs as Data

Mingxiao Wei

mingxiaowei@berkeley.edu

Nov 10, 2022

LOGISTICS

- Scheme Interpreter 
 - Checkpoint 2 due Sun 11/13
 - The whole project due Tue 11/22
 - Everyone gets 2 EC points for free 
 - One EC problem, worth 1 EC point 
 - Submit the whole project by Mon 11/21 for 1 EC point 
- Homework 08 due next Thu 11/17

ABOUT THE STRIKE

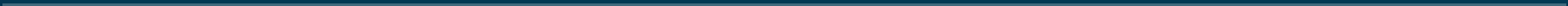
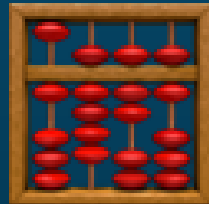
- [Why Strike](#)
- [Unfair Labor Practice Charged Filed Against UC](#)
- [EECS Staffing Bargaining Proposal](#)
- [How Can Y'all Support](#)
- [61A-wise Logistics](#)
- As a result, unless the strike gets called off before next Monday (in which case I'll try to email the whole section ASAP), we won't have lab next Monday. Discussion TBD.

FROM LAST TIME... 🙄

What's the best emoji (and reasoning, if you'd like to provide)?

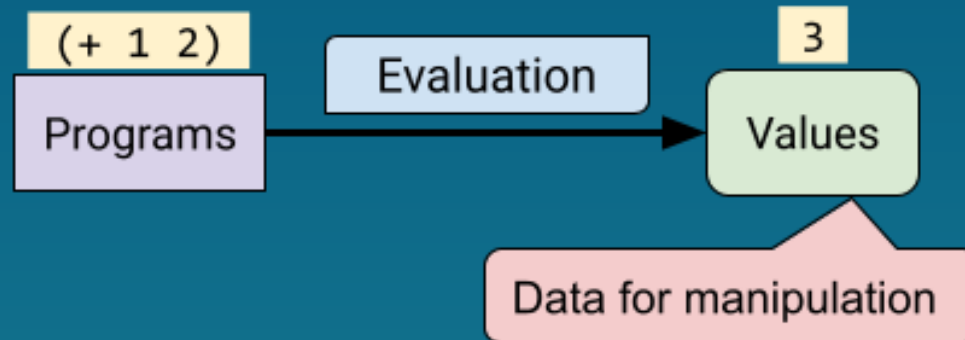
👊 🐾	💪
😂	Doge
N/A	🖌️
😂	frowning face emoji
happy emoji	1
😭	👉 yeaaaah
Laughing face	Smiley
potato	laugh
Eat	pink bow
white heart	lol
😂	the cry laugh 😭
🌿	😓
🐵 3 🐵	monkey covering eyes
😂	

PROGRAM AS DATA

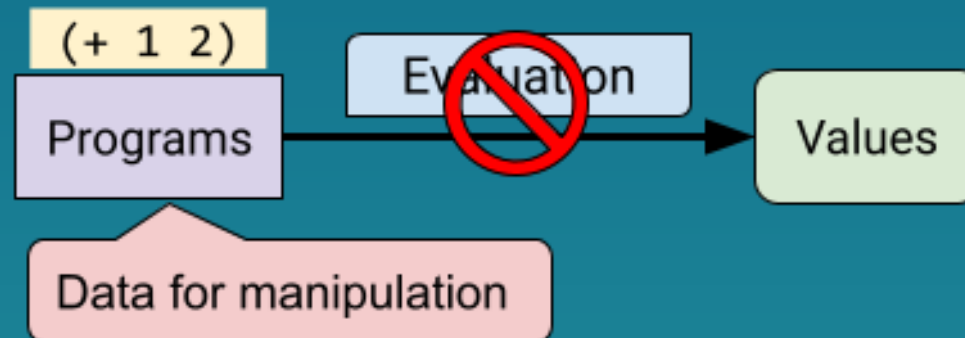


PROGRAM AS DATA - INTRO

- "Data: information in digital form that can be transmitted or processed" - Merriam-Webster Dictionary
- Up until now, we've treated values as data: numbers, booleans, strings, functions, objects, etc.



- Now, we'd like to treat programs (code) as data - "halt" the evaluation process and manipulate the program directly.



PROGRAM AS DATA - EXAMPLE

; ----- values as data -----

```
scm> (define x (+ 30 4))
```

x

```
scm> (+ 1000 (* 2 100) x)
```

1234

; ----- program as data -----

```
scm> (define y '(+ 1 2))
```

y

```
scm> (append y '(3 4))
```

(+ 1 2 3 4)

PROGRAM AS DATA - PYTHON

- In Python, we can store one-line expressions as strings, and use the built-in `eval()` to evaluate a string as an expression in the current frame

```
>>> expr = "'QwQ' if x > 1 else 'QAQ'"
>>> x = 2
>>> eval(expr)
'QwQ'
>>> x = 0
>>> eval(expr)
'QAQ'
>>> new_expr = expr[:-5] + "':3'"
>>> eval(new_expr)
':3'
```


PROGRAM AS DATA - PYTHON

- Now, what if we want to write a function that would create and evaluate this line of code `[<expr> for i in range(3)]`
 - where we could pass in any arbitrary expression `<expr>`, that would then be evaluated 3 times and have the results listed.
- `<expr>` could be any piece of code, such as `print('hello'), i >= 1`, which should not be evaluated until after they are inserted into the list comprehension
- Therefore, we need to manipulate the code directly

PROGRAM AS DATA - PYTHON

Trial 1: Solving with a traditional function

```
def list_3(expr):  
    return [expr for i in range(3)]
```

```
>>> lst = list_3(print(10))  
10  
>>> lst  
[None, None, None]
```

- Hmm... This isn't quite what we want. Because of Python evaluation rules, instead of evaluating a list of 3 `print(10)` statements, our `expr` was evaluated before the function was ever called, meaning 10 was only printed once.
- The issue here is order of evaluation - we don't want our expression parameter to be evaluated until after it is inserted into our list comprehension.

PROGRAM AS DATA - PYTHON

Trial 2: Returning the expression as a string and evaluating it afterwards

```
def list_3(expr):  
    return f"[{expr} for i in range(3)]"
```

```
>>> list_5("print(10)")  
"[print(10) for i in range(3)]"  
>>> lst = eval(list_3("print(10)"))  
10  
10  
10  
>>> lst  
[None, None, None]
```

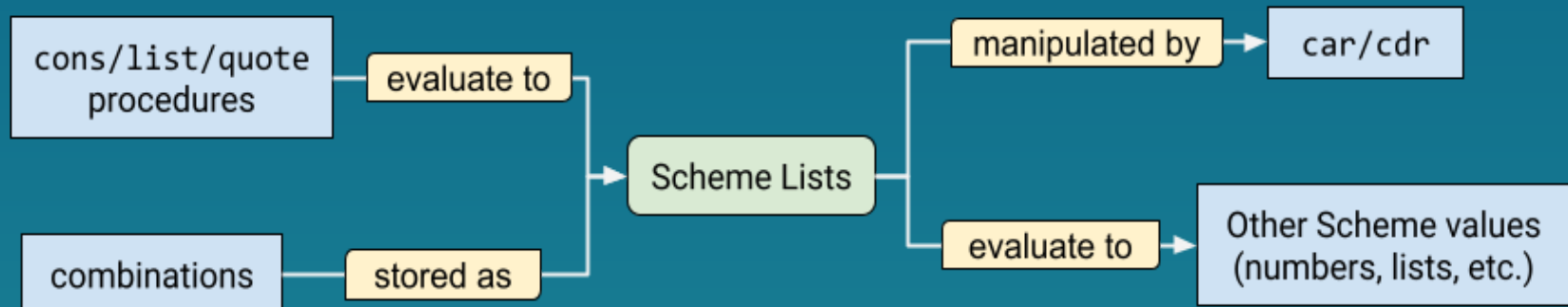
- Now we see the number 10 printed 3 times as a side effect of our function, just like we want!
- We circumvented Python's evaluate-operands-before-evaluating-function body rule by passing in our desired expression as a string.

WORKSHEET Q1

SCHEME PROGRAM AS DATA

SCHEME PROGRAM AS DATA

- Recall there are 2 type of expressions in Scheme:
 - Primitive/Atomic Expressions: numbers, symbols, booleans
 - Combinations: call expressions, special forms, etc.
- While in Python, we can store expressions as strings, in Scheme, all combinations (non-primitive expressions) are stored as Scheme lists



SCHEME PROGRAM AS DATA

```
(eval <expr>)
```

- `eval` takes in one operand
- The operand is first evaluated to a value - when evaluating a call expression, we evaluate all operands before applying the operator. `eval` then evaluates this value as another expression in the current environment
- `<expr>` is evaluated twice

```
scm> (define x '(+ 2 3))  
x  
scm> x  
(+ 2 3)  
scm> (eval x)  
5
```

PROGRAM AS DATA - EXAMPLE

; ----- values as data -----

```
scm> (define x (+ 30 4))
```

x

```
scm> (+ 1000 (* 2 100) x)
```

1234

; ----- program as data -----

```
scm> (define y '(+ 1 2))
```

y

```
scm> (append y '(3 4))
```

(+ 1 2 3 4)

```
scm> (eval (append y '(3 4)))
```

10

QUASIQUOTATION



QUASIQUOTE & UNQUOTE

`(quasiquote <expr>)` or ``<expr>`

`(unquote <expr>)` or `,<expr>`

- Without `unquote`, `quasiquote` behaves the same as `quote`
- `unquote` allows an expression to be evaluated before the entire list is returned
- Together, they are analogous to f-strings in Python and are useful for creating lists that represent expressions

QUASIQUOTE & UNQUOTE - EXAMPLE

```
scm> (define a 6)
a
scm> (define b 1)
b
scm> `( ,a ,b a)
(6 1 a)
scm> '(a b ,a)
(a b (unquote a))

scm> `(+ ,a ,b)
(+ 6 1)
scm> (eval `(+ ,a ,b))
7
scm> (eval `( ,a ,b a))
Error: int is not callable: 6
scm> (eval (cons 'list '(a b 'a)))
(6 1 a)
```

WORKSHEET Q2-7

ATTENDANCE! 🤠

go.cs61a.org/mingxiao-att

- The attendance form and slides are both linked on our [section website](#)!
- Once again, please do remember to fill out the form by midnight today!!