Discussion 9

Tail Calls, Interpreters

Antonio Kam

`anto [at] berkeley [dot] edu`

All slides can be found on

`teaching.rouxl.es`

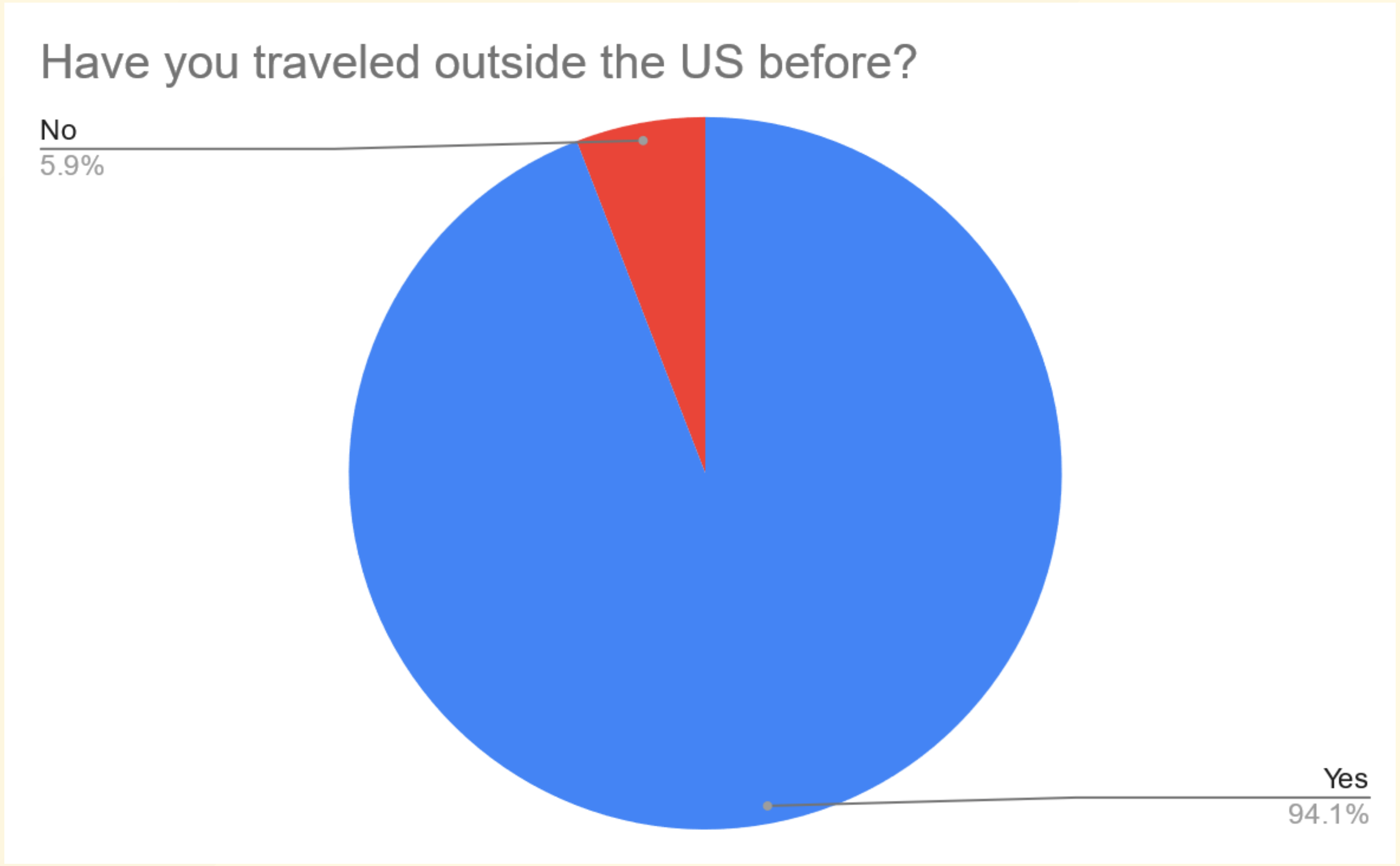
Announcements

- Small Group Tutoring Section
  - On Piazza
- Midterm Walkthrough Video
  - Will eventually come out
- Recommend reviewing pre-midterm content - build that foundation
  - Recursion!
- Study groups should have been sent out just now (sorry for not sending these earlier, I kinda forgor 💀)
  - For those that have one, guide for how I used my study group can be found on my teaching website
- Ethan's Scheme Guide (CS 61A website)

Announcements

- Ants due Thursday (hand in on Wednesday for 1 extra EC point)
- HW 5 due Thursday
- Scheme project begins on Thursday 👀
  - There are 2 checkpoints for this rather than the usual 1 checkpoint

Results from last section



Notes from last section

- Basically nothing lol

Temperature Check 🌡️

- Scheme Lists
- Dynamic Scope
- Recursion
- Tail Recursion
- Interpreters

Dynamic Scope

Dynamic Scope vs Lexical Scope

- Lexical Scope
  - What you normally see with Python functions/ `lambda` functions in scheme
  - The parent of the function is always where the function was *defined*
- Dynamic Scope
  - `mu` procedures in scheme use dynamic scope
  - Parent is the frame in which the function was *called*
  - You will implement this in the scheme project

Example where it doesn't matter

```scheme
(define lamb2 (lambda (x) (+ x y)))
(define cow2 (mu (x) (+ x y)))
(define y 5)
(lamb2 1)
(cow2 1)
```

Worksheet

```scheme
(define (goat x y) (lambda (x) (+ x y)))
(define (horse x y) (mu (x) (+ x y)))
(define horns (goat 1 2))
(define saddle (horse 1 2))
(define x 10)
(define y 20)
(horns 5)
(saddle 5)
```

Tail Recursion

Tail Recursion

- Sometimes with recursive functions it's possible to write it in a **tail recursive** way.
- A **tail call** occurs when a function calls another function as the *last action* of the current frame.

Example of non-tail recursive function

```
(define (factorial n)
  (cond
    ((= n 0) 1)
    (else (* n (factorial (- n 1))))
  )
)
```

- The recursive call might be in the last line, but it isn't the last action here
- Another way to see this is through environment diagrams (I'll draw them up in a bit)

Example of tail recursive function

```
(define (factorial n)
  (define (helper n result)
    (cond
      ((= n 0) result)
      (else (helper (- n 1) (* result n)))
    )
  )
  (helper n 1)
)
```

- In this case, the last recursive call here is our helper function itself `(helper (- n 1) (* result n))`
- Tail recursive functions very often need helper functions (for additional variables)

Why use tail recursion

- You won't get a recursion depth error with a tail recursive function because only a constant number of frames are needed at any given point
- More efficient - brings recursion closer to iteration in efficiency
- Scheme will discard the frames already used - doesn't need as much space to run

Tail Context

- When trying to identify whether a function call is a tail call, we look to see whether the call expression is a tail context
    - If there are any recursive calls before the tail context, the function is not tail recursive
- Essentially, the idea is to look at the *last* expression of the body: this may come in different forms
    - The second/third 'operand' inside `if`
    - Any non-predicate body inside `cond`
    - Last 'operand' in `and` or `or`
    - Last 'operand' in `begin`
    - Last 'operand' in `let`
    - (Will be useful for implementing EC! Reference the discussion worksheet later)

Question 2 - Is Tail Call

```
(define (question-a x)
  (if (= x 0) 0
      (+ x (question-a (- x 1)))))
```

Question 2 - Is Tail Call

```
(define (question-b x y)
  (if (= x 0) y
      (question-b (- x 1) (+ y x))))
```

Question 2 - Is Tail Call

```
(define (question-c x y)
  (if (> x y)
      (question-c (- y 1) x)
      (question-c (+ x 10) y)))
```

Question 2 - Is Tail Call

```
(define (question-d n)
  (if (question-d n)
      (question-d (- n 1))
      (question-d (+ n 10))))
```

Question 2 - Is Tail Call

```
(define (question-e n)
  (cond ((<= n 1) 1)
        ((question-e (- n 1)) (question-e (- n 2)))
        (else (begin (print 2) (question-e (- n 3))))))
```

Worksheet!

Attendance

`links.rouxl.es/disc`

Interpreters

What is an interpreter

- An interpreter is a program used to understand other programs
- We will start off with the Calculator language to look at Scheme Syntax (and how to interpret it)
- You will be writing an interpreter for the final project (this is a super cool project)

How does it work

- Essentially is the code version of what we've studied during week 1!
- Evaluate operators/operands, apply operands to operators; this whole process is done through our own interpreter

`Pair` Class

- We will be using the `Pair` class (very similar to `Link`, but with a few added bonuses/slight differences)
    - Has a `first` and `rest` attribute
    - Also has a `map` method where it applies a function to every argument
    - Must provide `nil`, does not default to `Link.empty` like with our `Link` class
- If `p` is a `Pair` containing a proper call expression, we get the operator by doing `p.first`, and get the operands with `p.rest`. To get the first operand, we need to do `p.rest.first`

`Pair` class

```python
class Pair:
    """Represents the built-in pair data structure in Scheme."""
    def __init__(self, first, rest):
        self.first = first
        if not scheme_valid_cdrp(rest):
            raise SchemeError("cdr can only be a pair, nil, or a promise but was {}".format(rest))
        self.rest = rest

    def map(self, fn):
        """Maps fn to every element in a list, returning a new
        Pair.

        >>> Pair(1, Pair(2, Pair(3, nil))).map(lambda x: x * x)
        Pair(1, Pair(4, Pair(9, nil)))
        """
        assert isinstance(self.rest, Pair) or self.rest is nil, \
            "rest element in pair must be another pair or nil"
        return Pair(fn(self.first), self.rest.map(fn))
```

Worksheet!

`Q9`, `Q10`

`calc_eval` and `calc_apply`

```python
def calc_eval(exp):
    if isinstance(exp, Pair): # Call expressions
        return calc_apply(calc_eval(exp.first), exp.rest.map(calc_eval))
    elif exp in OPERATORS:      # Names
        return OPERATORS[exp]
    else:                       # Numbers
        return exp
```

```python
def calc_apply(operator, args):
    """Apply the named operator to a list of args.

    >>> calc_apply('+', as_scheme_list(1, 2, 3))
    6
    >>> calc_apply('-', as_scheme_list(10, 1, 2, 3))
    4
    >>> calc_apply('*', nil)
    1
    >>> calc_apply('*', as_scheme_list(1, 2, 3, 4, 5))
    120
    >>> calc_apply('/', as_scheme_list(40, 5))
    8.0
    """
    if not isinstance(operator, str):
        raise TypeError(str(operator) + ' is not a symbol')
    if operator == '+':
        return reduce(add, args, 0)
    [...]
```

Worksheet!

Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!