Discussion 12

Interpreters and SQL 💀

Antonio Kam

`anto [at] berkeley [dot] edu`

All slides can be found on

`teaching.rouxl.es`

Announcements

- Scheme Project
  - Checkpoint 1 on Friday
  - Checkpoint 2 on Tuesday
  - Due next Friday (EC for day before, as usual)
- No HW Released for this week, but SQL HW will be released on Monday

Notes from last section

- Rank 61a, 61b, 61c, and 70 in terms of stress :'D
  - 70 > 61A/61B/61C
- When should we start studying for finals
  - I think starting now is not a bad idea
- Seoul dog or top dog
  - Never been to seoul dog, so top dog
- favorite pokemon?
  - mega altaria ☁️ 🥺

Temperature Check 🌡️

- Interpreters
- SQL

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

```
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))
```

`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)
    [...]
```

SQL 💀

Select Statements and Queries

Select Statements

- You can experiment with all of this on [sql.cs61a.org](sql.cs61a.org)
- If we have a pre-existing table (for example, the `records` table), we can grab values from that table using a `FROM` clause
- Using `*` will select all columns from a table

```
SELECT [columns] FROM [tables] WHERE [condition] ORDER BY [criteria] LIMIT [number];
```

Demo:

```
SELECT * FROM records WHERE title = "Programmer";
SELECT name, salary FROM records WHERE division = "Accounting"
  ORDER BY salary DESC LIMIT 5;
```

Worksheet!

Joins

Joins

- Sometimes, people might store data in multiple tables
- It's hard to access data from both these tables
- That's where *joins* come in!

```sql
SELECT * FROM records, meetings; -- can select multiple tables
```

Ambiguous Joins

- Tables might have overlapping column names
- We need a way to distinguish between these columns
  - Especially if you need to join a table with itself (useful if you want to compare 2 people with each other)
- Use the `as` keyword

Ambiguous Joins

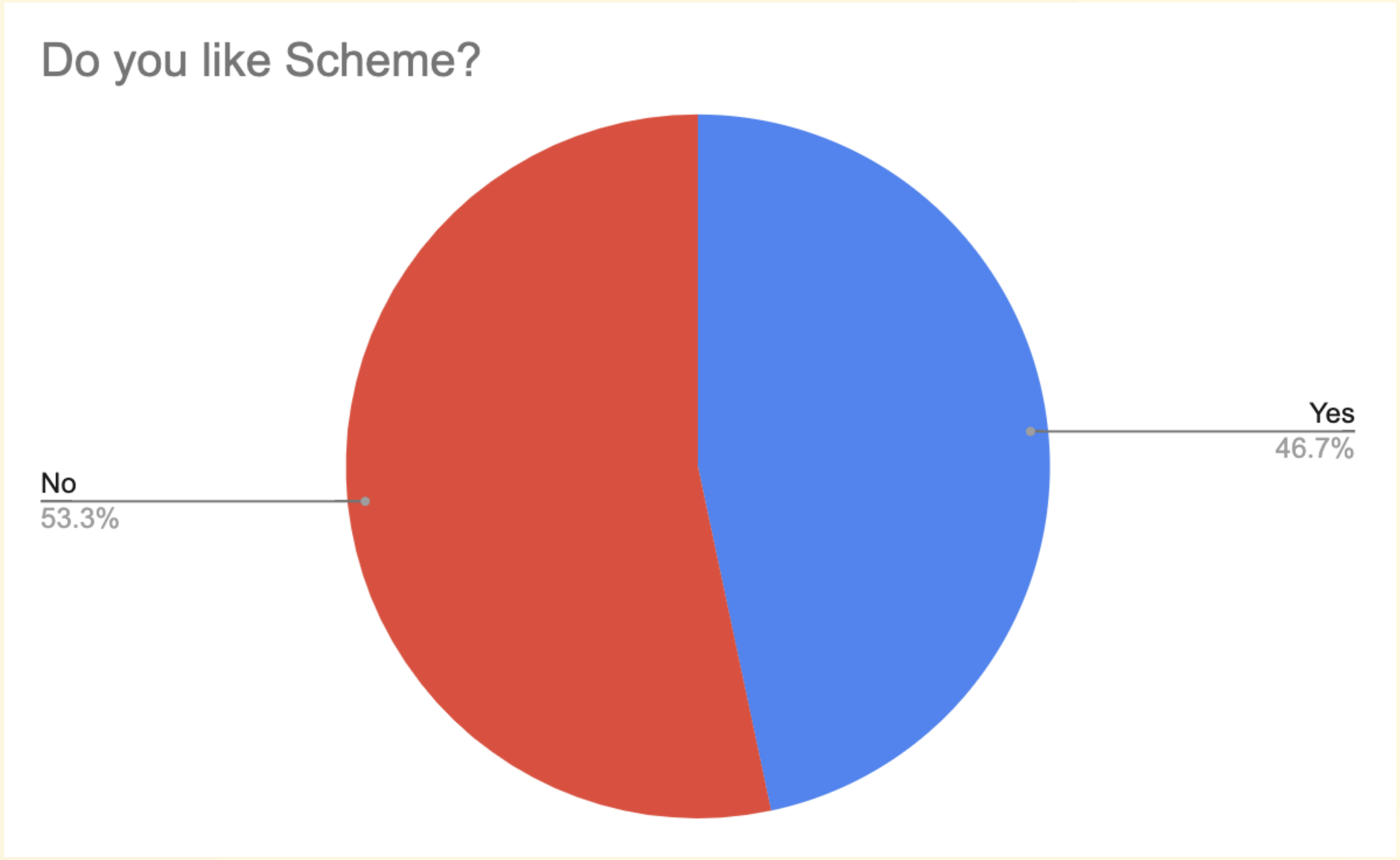
```
SELECT a.name, a.title FROM records AS a, records AS b
  WHERE a.name = "Louis Reasoner" AND a.supervisor = b.name;
```

Aggregation 😡

Aggregation

- Aggregation tends to be useful when you have multiple groups, and you want to group by certain pieces of data.
- You can also combine multiple rows into 1 with aggregation
  - `SELECT COUNT(*) FROM RECORDS;`
  - `SELECT name, MAX(salary) FROM RECORDS;`
- `GROUP BY` will allow you to perform these aggregation functions on specific groups
  - `SELECT division, MIN(salary) FROM records GROUP BY division;`
- `WHERE` statements for `GROUP BY`s uses the `HAVING` clause
  - `HAVING` filters out entire groups
  - You can have both `WHERE` and `HAVING` in the same statement

Results from last section (`links.rouxl.es/disc`)



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!