Discussion 8

Scheme 🥶

Antonio Kam

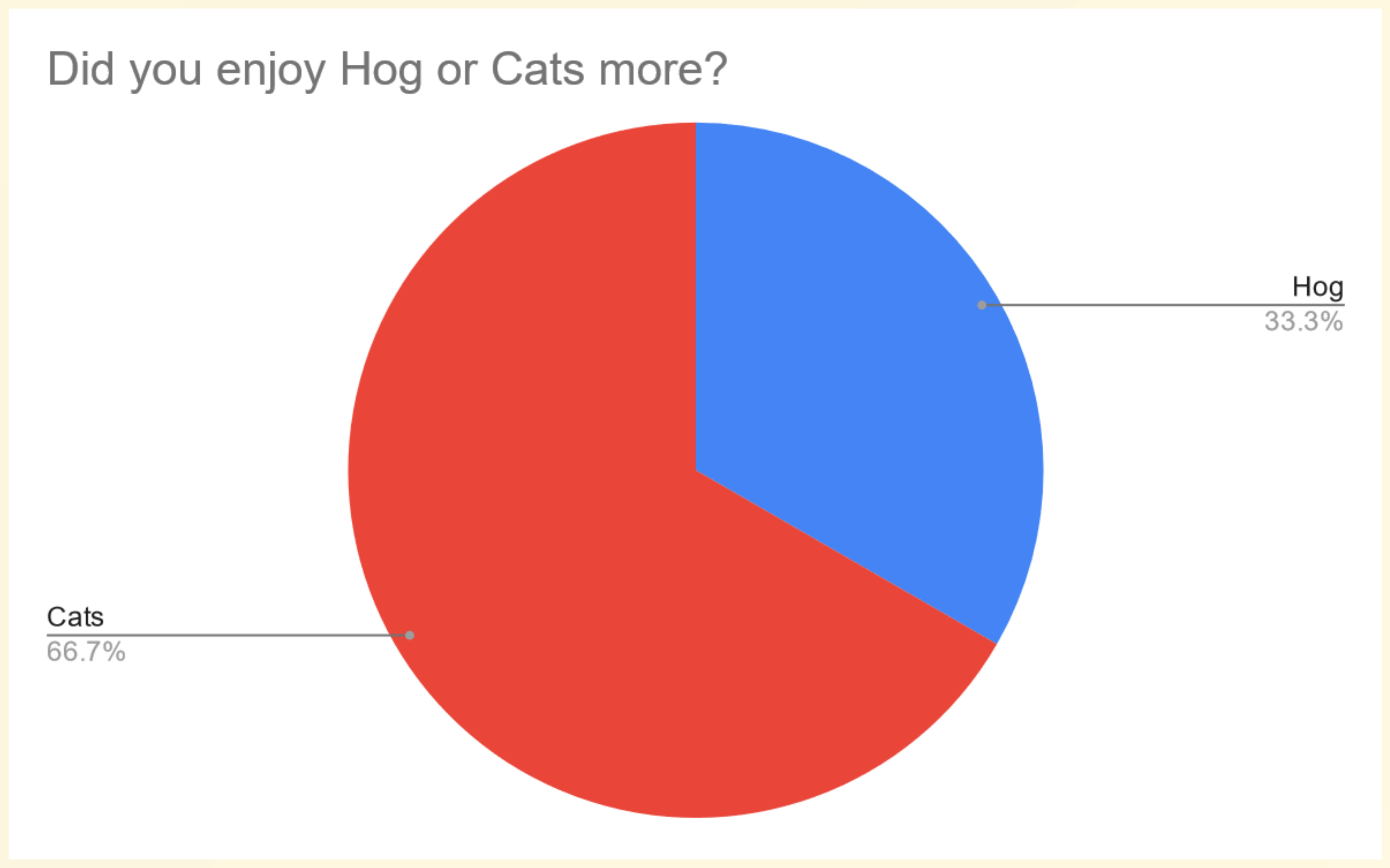
`anto [at] berkeley [dot] edu`

All slides can be found on `teaching.rouxl.es`

Announcements

- Check Piazza
- [Homework Recovery Sections]
- Ants has been released!
  - Checkpoint due Friday 🙀
- Magic: the Lambda-ing has been released
  - Smaller project; worth 2 points of extra credit
  - OOP based
- I will email study groups before tonight - keep an eye out for that!
  - Will also provide tips and tricks for how to utilize them
    - I'll put this on my website (if I remember)
  - They're fully randomized

Results from last section



Notes from last section

- For attendance (especially for discussion), fill in the attendance form the moment the word gets released
  - I close the form after section for discussion because I also conveniently have something to do on my laptop right after
- You can check your discussion/lab attendance on howamidoing.cs61a.org
- Keep in mind I read everything that people put in all feedback forms

Temperature Check 🌡️

- Linked Lists
- Scheme 💀
- Scheme Lists 💀 💀
- Would like to mention that I will be talking a lot this discussion at the start cause a lot of this is syntax

What is Scheme

- Scheme is another language that you need to learn 👀
- It's a dialect of Lisp (List Processor)
- Everything is done with recursion 🥳
  - No `while` / `for` loops
  - Good thing about this is that you get a **lot** of recursion/tree recursion practice with scheme
- No mutation in scheme
- IMO Scheme is a very good way to demonstrate that once you learn the logic for one programming language, learning a second one is way easier!
- There are a lot of parentheses 😭

Primitives

- Scheme has a set of **atomic** primitive expressions.
  - Similar to the primitives in Python
  - These expressions cannot be divided up further

```
scm> 1
1
scm> 2
2
scm> #t
True
scm> #f
False
```

Booleans (Python)

Remember this table?

Falsey	Truthy
`False`	`True`
`None`	Everything else
`0`	
`[]`, `""`, `()`, `{}`	

Booleans (Scheme)

Falsey	Truthy
`#f`	Everything else

🥳 This is something you need to remember 😭

`define`

- In scheme, everything that isn't a primitive is done with **prefix notation**
  - `(<keyword> [<arguments> ...])`
- In scheme, we use the `define` keyword in order to bind values to symbols, which work the same way as variables.
  - This is also used to define functions - more on this later
  - This keyword returns the symbol:

- In scheme, everything that isn't a primitive is done with **prefix notation**
  - `(<keyword> [<arguments> ...])`
- In scheme, we use the `define` keyword in order to bind values to symbols, which work the same way as variables.
  - This is also used to define functions - more on this later
  - This keyword returns the symbol:

```
>>> x = 3
```

```
scm> (define x 3)
x
```

Intro WWSD

Call Expressions

- Call expressions apply a procedure to some arguments
  - `(<operator> [<operands> ...])`
- Exactly the same process as Python
- Evaluate the operator (make sure it's a procedure/function)
- Evaluate each of the operands (from left to right)
- Apply the operands to the operator

Call Expression WWSD

Call Expressions

```
>>> add(1, 2)
3
```

```
scm> (+ 1 2)
3
```

- Important to note that all the operands are evaluated!

Special Forms

- They still look like call expressions (syntax-wise), but instead of evaluating all the operators, there are certain rules for evaluation.

Special Forms (`if`)

- `(if <predicate> <if-true> [<if-false>])`
- `<predicate>` and `<if-true>` are required, `<if-false>` is optional
- Rule for evaluation:
  - Evaluate `<predicate>`
  - If `<predicate>` is truthy (don't forget what Scheme does!), evaluate `<if-true>`
  - Else evaluate `<if-false>` (if it exists)
- This means that not all of its operands will be evaluated!

Special Forms (`if`)

```
scm> (if (> 4 3) 3 2)
3
scm> (if 0 3 2)
3
scm> (if #f 3 2)
2
scm> (if (= 3 2) (/ 1 0) 3)
3
scm> (if (= 3 3) (/ 1 0) 3)
Error
```

Special Forms (Boolean Operators)

- `and`, `or`, `not`
  - `(and 1 2 3)` → `3`
  - `(or 1 2 3)` → `1`
  - `(not 0)` → `#f`
- Equivalence
  - `=` - used for numbers
  - `eq?` - `is` in Python
  - `equal?` - `==` in Python

Defining Functions

- All functions are `lambda` functions in scheme.

```
(lambda ([<params> ...]) <body>)

scm> (lambda (x) (+ x 2))
(lambda (x) (+ x 2))
scm> (define f (lambda (x) (+ x 2)))
f
scm> f
(lambda (x) (+ x 2))
```

Defining Functions

- There is a bit of a shorthand to write functions:

```
(define (<name> [<params> ...]) <body>)

scm> (define (f x) (+ x 2))
f
scm> f
(lambda (x) (+ x 2))
```

Worksheet!

Attendance

`links.rouxl.es/disc`

Pairs and Lists

What are Scheme Lists?

- All Scheme lists are *very* similar to the Python linked lists that we've been dealing with.
- Python:
  - `lnk.first` - gets the first element
  - `lnk.rest` - gets the rest of your linked list
- Scheme:
  - `(car lnk)` - gets the first element
  - `(cdr lnk)` - gets the rest of your scheme list
- Weird names!

Creating Scheme Lists

```
>>> Link(1, Link(2), Link(3))
Link(1, Link(2), Link(3))
```

```
scm> (cons 1 (cons 2 (cons 3 nil)))
(1 2 3)
scm> (list 1 2 3)
(1 2 3)
scm> '(1 2 3)
(1 2 3)
```

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!