Discussion 10

Scheme Data Abstractions

Antonio Kam

anto [at] berkeley [dot] edu

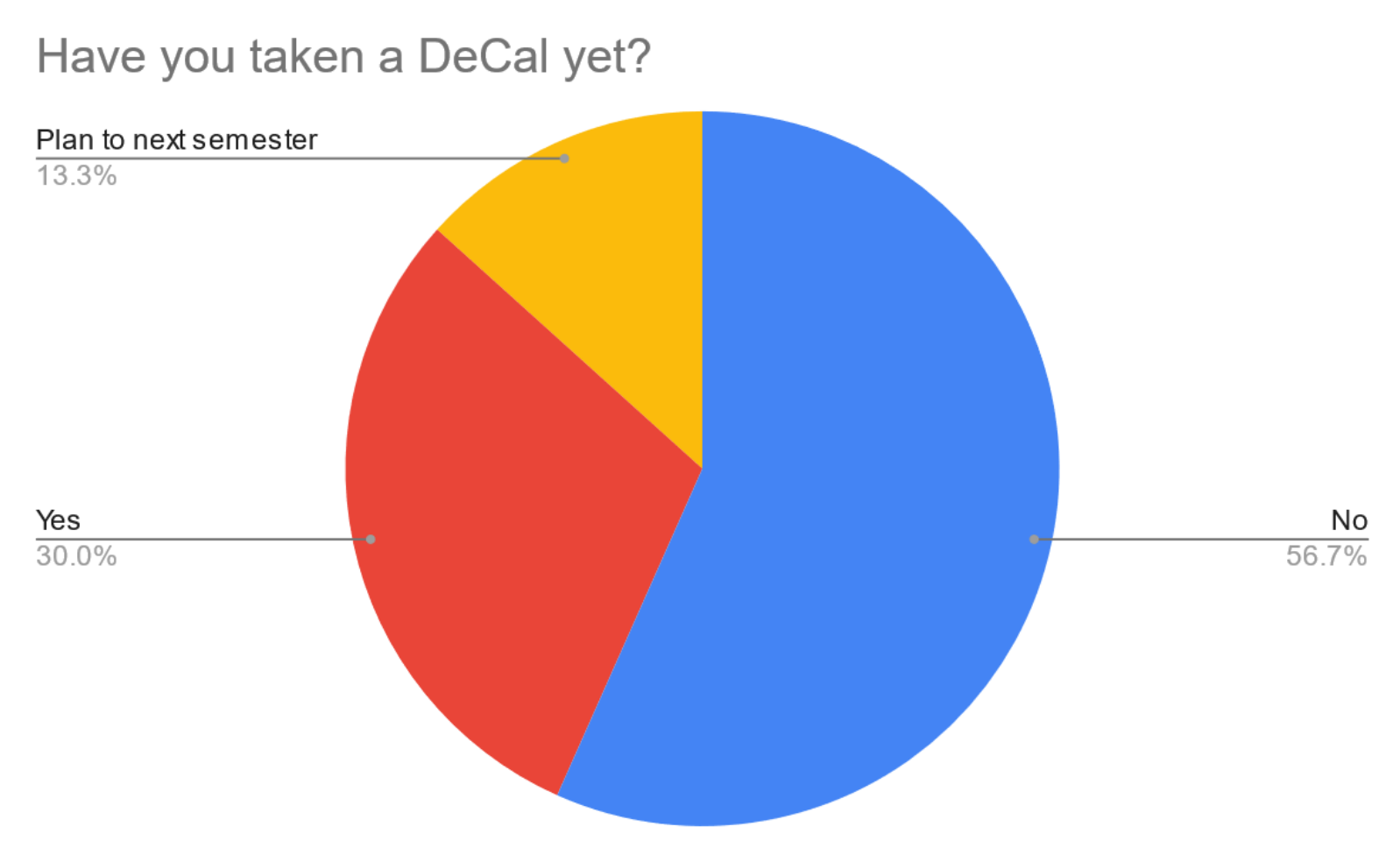
All slides can be found on

`teaching.rouxl.es`

Announcements

- Scheme Project has started 🥳 💀
  - I liked this project quite a bit, but it is fairly technical
  - Checkpoint 1 due on Tuesday, Checkpoint 2 due on Friday (August 5th)
  - Checkpoints are worth points
- Magic: the Lambda-ing due today!

Results from last section



Notes from last section

- Once again, pretty much nothing
- Please fill the forms in with something so I have something to say next time lol
- I'm gonna fill this slide by saying we're super close to being done with this semester 🤯 - you all got this!

Temperature Check 🌡️

- Scheme
- Scheme Lists
- Scheme Data Abstractions

What are data abstractions

- In Python, we used classes to describe certain objects
- Classes don't exist in scheme 🙀
- As a result, data abstractions help to fill in this gap
    - If we want something in scheme to describe objects, we use data abstractions!

Data abstractions

- Constructors
  - Similar to `__init__` in our classes
  - Function that builds our abstract data type
- Selectors
  - Functions that receive information from the data type
  - Similar to getting an instance variable value (`t.label` , `s.first`)

Example Data Abstraction (Python)

Constructor:

```
def couple(first, second):
    return [first, second]
```

Selectors:

```
def first(couple):
    return couple[0]

def second(couple):
    return couple[1]
```

```
c = couple(1, 2)
c1, c2 = first(c), second(c) # c1 = 1, c2 = 2
```

Example Data Abstraction (Scheme)

Constructor:

```
(define (couple first second) (list first second))
```

Selectors:

```
(define (first couple) (car couple))
(define (second couple) (car (cdr couple)))
```

```
(define c (couple 1 2))
(first c)
(second c)
```

Idea

- The idea of data abstractions here is that we don't think of the `couple` data abstraction as a list with two elements, but rather simply as the descriptors
- This is similar to real life where you don't think of driving a car as all its individual parts, but just as a large abstraction
- Data abstractions effectively hide the implementation from users - all people need to use are the constructors and selectors

Violating Data Abstraction Barriers

- One thing you cannot do when it comes to data abstractions is **violate the data abstraction barriers**.
- This basically means that you must use the constructors and selectors when dealing with data abstractions
  - This is because you do not necessarily know how the data abstractions are implemented

Example Data Abstraction (Python)

Constructor:

```
def couple(first, second):
    return {"first": first, "second": second}
```

Selectors:

```
def first(couple):
    return couple["first"]

def second(couple):
    return couple["second"]
```

```
c = couple(1, 2)
c1, c2 = first(c), second(c) # c1 = 1, c2 = 2
```

City

- Let's say we have a definition for a city data abstraction
- We just need the documentation for what the names mean, but we don't need to know how it's implemented:

```
(make-city name lat lon)

(get-name city)
(get-lat city)
(get-lon city)
```

- All we need to know is that we can use these definitions

Worksheet!

Attendance

`links.rouxl.es/disc`

Scheme Trees

- We've made the `Tree` class in Python before
- Scheme doesn't have classes!
- Let's make an Abstract Data Type instead 👀

- Constructor:
  - `(tree label branches)`
  - `branches` is a list
- Selectors:
  - `(label t)` - returns the label of the tree
  - `(branches t)` - returns a list of the branches of the tree

```
(define t
        (tree 5
            (list (tree 4 nil) (tree 7 nil))
        ))

(label t) -> 5
(label (car (branches t))) -> 4
```

`map` in Scheme

- `(map)` is Scheme's way of doing 'iteration'
- Scheme doesn't have for loops, so we can't do `for b in t.branches`
- We get around that with map
- Very similar to list comprehensions
- `(map <proc> <lst>)`
  - `proc` (procedure) must be a 1-argument function (important for later!)
  - Calls `proc` on each element in `lst` and returns a new list with that

`map` in Scheme

```
(define (double x) (* x 2))
(define lst '(1 2 3 4))
(define new-lst (map double lst)) -> (2 4 6 8)
```

What if you pass in a 2 argument function?

`map` in Scheme

```
(define (double x) (* x 2))
(define lst '(1 2 3 4))
(define new-lst (map double lst)) -> (2 4 6 8)
```

What if you pass in a 2 argument function?

Let's say I want to add 3 to every element

```
(define (add x y) (+ x y))
(define lst '(1 2 3 4))
(define new-lst (map (lambda (z) (add z 3)) lst)) -> (4 5 6 7)
```

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!