

Discussion 2

Environment Diagrams and Higher-Order Functions

Antonio Kam

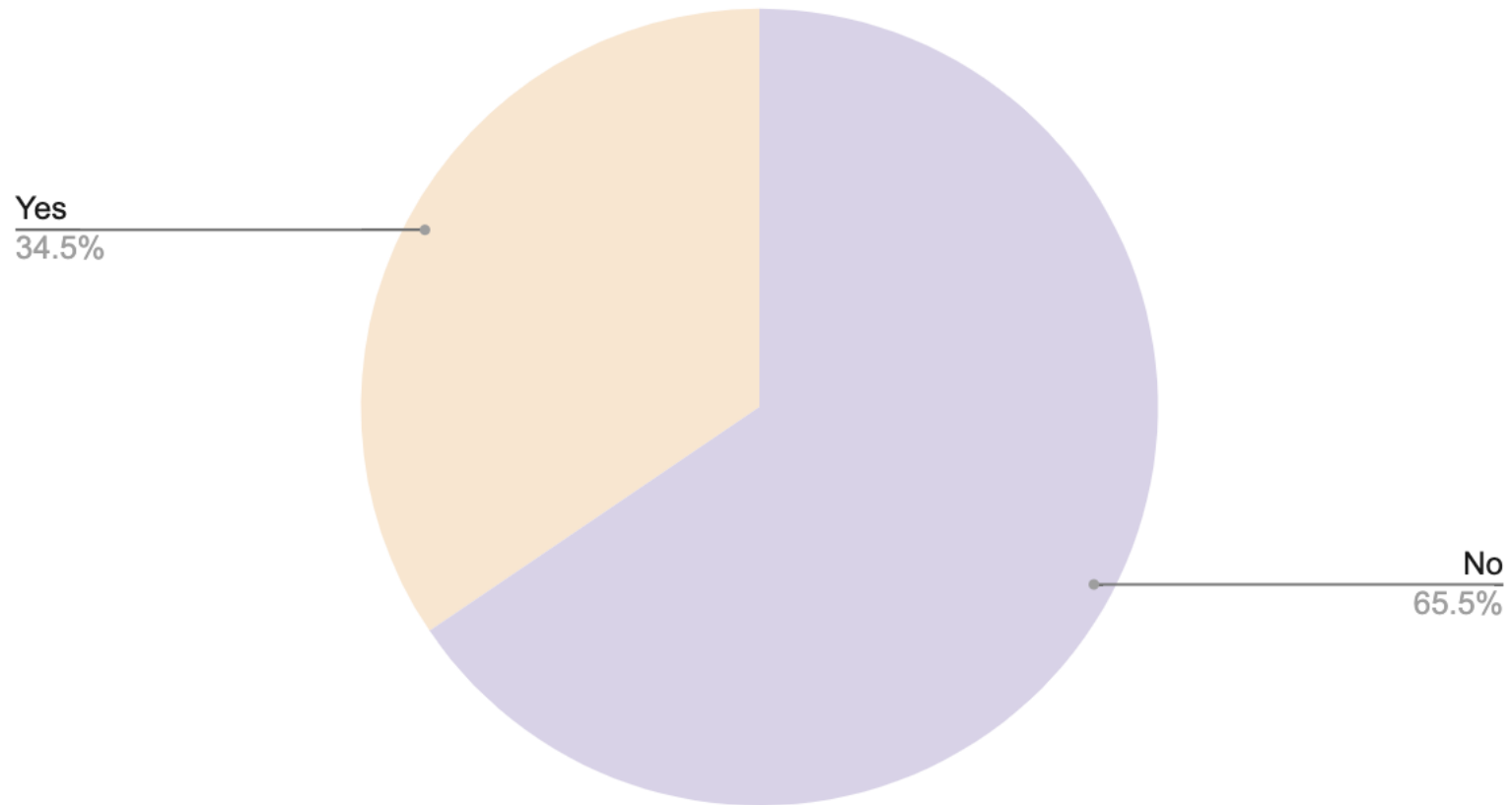
`anto [at] berkeley [dot] edu`

Announcements

- Hog
 - Released on Monday
 - Please start early! OH is usually far less busy at the start, so the more you wait to start, the more likely you'll have to wait longer in the queue
 - Getting started videos
 - Phase 1 due Feb 7 - worth 1 point
 - Submit full project 1 day early (Feb 9) for 1 bonus point (must have a full submission if you want full credit + the extra credit point)
- CSM Sections have opened
 - Small group tutoring sections (5-6 people), get to see more of the content
 - I joined a CSM section and found it incredibly useful! Would *highly* recommend

Results from last discussion

Are you (planning on) taking a DeCal this semester?



Questions and Comments from last section

- People like seeing stuff executed in the terminal
 - Neat, will do this more (often)
- Write bigger on the board!
 - whoops, ok!
- More examples 🙄
 - depends on whether i have the time to
- Write more specific on the slides so it's better for review
 - i prefer not having too much on slides just cause it makes me feel like i'm reading off them

Questions and Comments from last section

- BOTW
 - haven't played it, but i've played music from it in an orchestra lmao
- Marvel Snap
 - haven't gotten around to it, but it looks really interesting
 - card games/board games are fun
 - dominion

Questions and Comments from last section

- Midterm
 - don't have too much time to go over it, but i have resources you can look at
 - regardless, focus on environment diagrams
 - [cs61a.rouxl.es midterm advice \(links.rouxl.es/mt1-advice\)](https://cs61a.rouxl.es/midterm-advice)
 - there are video walkthroughs of past midterm questions that i like
 - i spent *minutes* of my life making it please use it
 - i'll go through the process of how i annotate a question after environment diagrams
- today is going to be me talking a lot

Temperature Check

- Environment Diagrams
- `lambda` functions
- Higher-order Functions

All slides can be found on

teaching.roux1.es

Environment Diagrams



Environment Diagrams

- Environment diagrams are a great way to learn how coding languages work under the hood
- Keeps track of all the variables that have been defined, and the values that they hold
 - Done with the use of *frames*
- Expressions evaluate to values:
 - `1 + 1` → `2`
- Statements do not evaluate to values:
 - `def` statements, assignments, etc.
- Statements change our environment

Frames

- The `Global Frame` exists by default
- Frames list bindings between variables and their values
- Frames also tell us how to look up values

Assignment

- Assignment statements bind a value to a name
 - The right side is evaluated before being bounded to the name on the left
 - `=` is not the same in Python and mathematics
- These are then put in the *correct frame* in the environment diagram

```
x = 2 * 2 # 2 * 2 is evaluated before bound to the name x
```

Assignment

$x = 2 * 2$ # $2 * 2$ is evaluated before bound to the name x

Global Frame

$x \mapsto 4$ ← result of evaluating
 $2 * 2$

def statements

- Creates function (objects), and binds them to a variable name
- The function is **not** executed until called!
- Name of the variable is the name of the function
- Parent of the function is the frame where the function is *defined*
- Keep track of:
 - Name
 - Parameters
 - Parent

Example

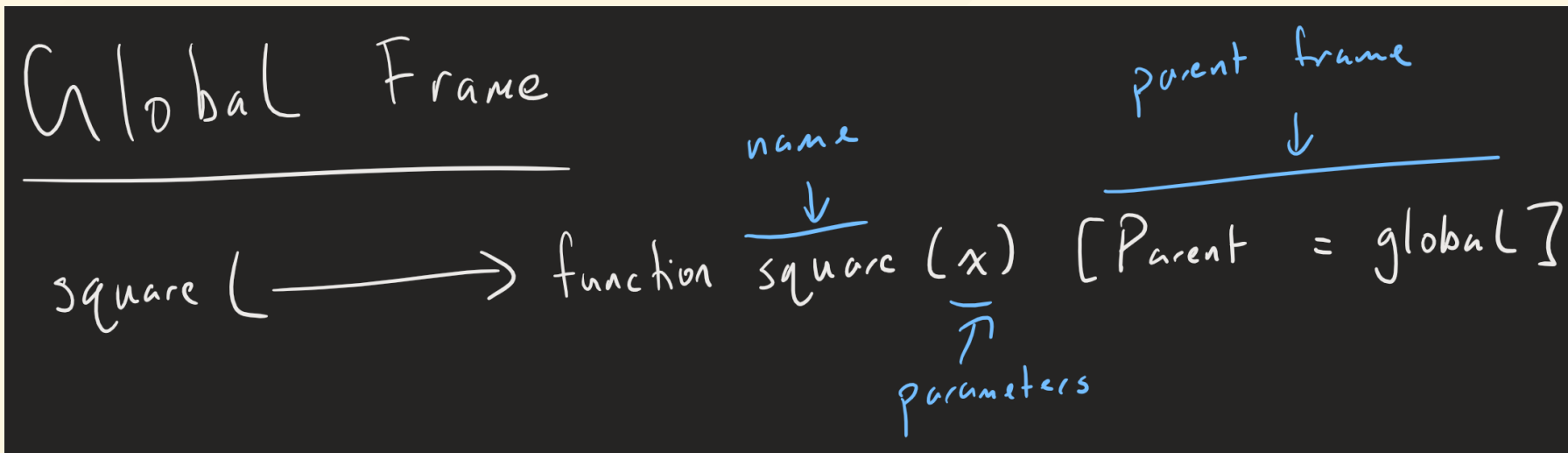
```
def square(x):  
    return x * x
```

- Keep track of the name, parameters, and parent!
- Uses *pointers* (unlike for primitive values)

Example

```
def square(x):  
    return x * x
```

- Keep track of the name, parameters, and parent!
- Uses *pointers* (unlike for primitive values)



Call Expressions

(Order of operations for nested call expressions)

Example 1

```
add(5, 9) # 14
```

Example 2

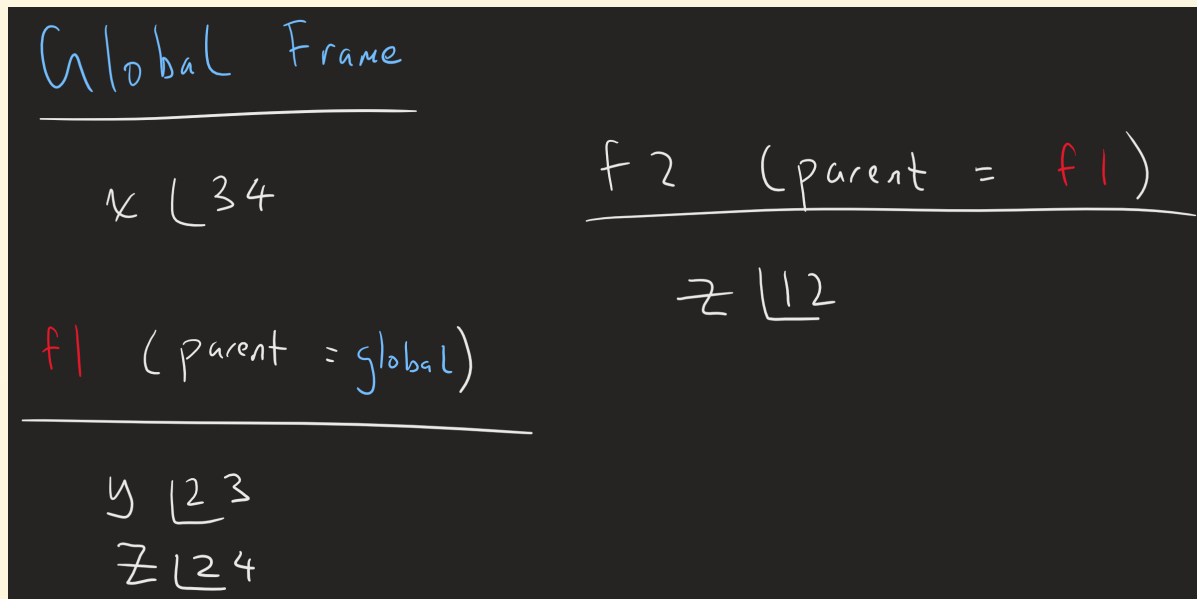
```
x = 3  
add(2, add(x, 4)) # 9
```

Variable Lookup 🙌

- Look in your current frame to find your variable
- If it doesn't exist, repeat the same process in the parent frame (including the lookup if you don't find anything)
- If you reach the global frame and still can't find anything, the program errors
 - This is because the variable doesn't exist 😭

Variable Lookup

Example



(Assume that we're looking for variables inside **f2**)

Variable Lookup

Example

Variable	Value
x	34
y	23
z	12

- If we start off in `f2`, we already see `z` in `f2`, so there is no need to look at the frame above.
- However, for the case of `y`, we do need to look up to its parent frame, and for `x`, we need to lookup 2 levels

New Frames

- New frames are made when a function is called
- Label your frame with a unique index (convention is `f1`, `f2`, etc.)
- Write down the name of the function object
 - Not necessarily the name of the variable!
- Write down the parent that the function you're calling has
- Separately, all frames (other than the global frame) have a return value
 - This can be `None` if nothing is specified

Example

```
def fun(x):  
    x = x * 2  
    return x
```

```
x = 30  
fun(x)
```

Example

```
def fun(x):  
    x = x * 2  
    return x
```

```
x = 30  
fun(x)
```

Global Frame

fun \rightarrow func fun(x) (p=g)

x 30

f1 fun(x) (p=g)

x 60

Return
value 60

Attendance

links.roux1.es/disc

Question 2

```
def f(x):  
    return x  
  
def g(x, y):  
    if x(y):  
        return not y  
    return y  
  
x = 3  
x = g(f, x)  
f = g(f, 0)
```

Annotating exam questions



lambda Syntax

- `lambda <args>: <body>`
- What goes in `<body>` must be a single expression

lambda Example

```
def func(x, y):  
    return x + y
```

```
func = lambda x, y: x + y
```

```
# Notice how I have to do the binding to a variable myself
```

```
def i(j, k, l):  
    return j * k * l
```

```
i = lambda j, k, l: j * k * l
```

lambda Example 2

lambda functions can also be used as the operator for a function!

```
(lambda x, y: x + y)(2, 3) # 5
```

Equivalent to

```
def add(x, y):  
    return x + y
```

```
add(2, 3) # 5
```

Higher Order Functions (HOF)

- HOFs are functions that can do the following things (can be both):
 1. Take in other functions as inputs
 2. Return a function as an output
- You can treat a function as just an object or a value (there's nothing special about them)
- `function` and `function()` mean different things!
 - `function` refers to the object itself (in the environment diagram, it refers to what the arrow is pointing to)
 - `function()` actually calls and executes the body of the function

HOF Example 1 (Functions as input)

```
def double(x):  
    return x * 2
```

```
def square(x):  
    return x ** 2
```

```
def double_adder(f, x):  
    return f(x) + f(x)
```

```
double_adder(double, 3) # 12
```

```
double_adder(square, 3) # 18
```

```
# Passed in two different functions
```

HOF Example 2 (Functions as output)

```
def f(x):  
    def g(y):  
        return x + y  
    return g
```

```
a = f(2)  
a(3) # 5
```

```
# Same thing as calling f(2)(3)
```


HOF Example 2

```
def f(x):  
    def g(y):  
        def h(z):  
            return x + y + z  
        return h  
    return g
```

```
lambda x: lambda y: lambda z: x + y + z
```

The two above are equivalent statements!

(Notice how the lambda one takes up far less space!)

lambda Functions and Higher-Order Functions

- A `lambda` expression evaluates to a `lambda` function
 - Can be used as the operator for a function!
- These functions work the same way as a normal function
 - Can be written in 1 line - faster way to make functions
 - Similar to `def` in usage, but different syntax
- `lambda`s are especially useful when you want to use a function once and then never use it again (will see examples of this)

Worksheet!

Currying

Currying is one application of the HOFs from earlier.

```
lambda x: lambda y: x + y
```

Instead of just any expression on the inside (for example `x + y`), we use a function!

```
def pow(x, y):  
    x ** y  
  
def curried_pow(x):  
    def f(y):  
        return pow(x, y)  
    return f  
  
curried_pow(3)(2)  
# is the same as  
pow(3, 2)  
# You will need as many inner functions as you have arguments
```

Currying

- Currying is the process of turning a function that takes in *multiple* arguments to one that takes in *one* argument.
- What's the point?
 - Sometimes functions with 1 argument are far easier to deal with
 - Can create a bunch of functions that have slightly different starting values which saves on repeating code
 - Useful for the `map` function (it requires functions that have only 1 argument)
- Kind of hard to see the benefits until you write production code

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.roux1.es/feedback

Thanks for coming! 🎉

Please give me feedback on what to improve!