Discussion 3

Recursion

Antonio Kam

`anto [at] berkeley [dot] edu`

Announcements

- Hog checkpoint due tomorrow! Please do this it's very important
- There's a project party instead of OH tomorrow from 3-6PM

Temperature Check 🌡️

- Environment Diagrams
- Higher-order Functions
- Recursion
- Tree Recursion

Questions and Comments from last section

- You're doing a good job
  - 🥳 🥳 🥳
- If we can do more practice problems, that’ll be great.
  - Depends on pacing - some concepts are a bit more abstract (e.g. environment diagrams), while some other might be covered here in section before the lecture itself (like tree recursion today 💀)
- If there's not enough time to complete an entire problem, could you still show the answers please?
  - good point, will do
- SHOW VID OF u SPEEDCUBing
  - 💀

Questions and Comments from last section

- more environment diagrams
  - we'll see them a lot more in the future, so don't worry about this 😉
- How hard would you rate the difficulty of exams for 61A based on your experiences and others' feedback?
  - I think the exams are definitely hard, but they're doable with enough thinking
  - The amount of thinking time needed does vary though
- Some things specifically applicable towards the hog project.
- I prefer Dr Pepper over Coca-Cola
  - I've never actually tried Dr Pepper so I have no clue whether I can back this claim or not lol

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

What is recursion?

- A *recursive* function is one where a function is defined in terms of itself.
- Similar to higher-order functions except it returns a *call* to a function rather than the function itself
- Will be hearing me talk about this a lot: **recursive leap of faith**

Analogy

- Imagine you're in a line waiting for boba, but you don't know how many people there are in front of you (and you want to count how many people there are in front of you)
- In this case, you can ask the person in front of you about how many people they have in front of them, and then they repeat this same process until...
- The person at the front now tells the person behind them that there's nobody in front of them
- Then everyone just needs to add 1 to their answer, and bring it backwards from there.

3 Steps of Recursion

1. Base Case
   - What is the smallest version of the problem we know the answer to?
   - I tend to think of this as the simplest input
2. Recursive Case (recursive call on a smaller version of the problem)
   - What can I do to reduce my input to something simpler?
   - Similar to `while` loops
3. Connecting it all together
   - Assuming your recursive call is correct (**recursive leap of faith!**), how do you solve the real problem

Example with analogy

1. Base Case
    - I'm at the front of the line
2. Recursive Case (recursive call on a smaller version of the problem)
    - I ask the person in front of me to tell me how many people they have in front of them (assume that the answer that they give is correct (**recursive leap of faith**))
3. Connecting it all together
    - Add 1 to their answer

Example

```
def factorial(n):
    if n == 0 or n == 1: # Base Case
        return 1
    else: # Recursive Case
        return n * factorial(n - 1)
```

Example

- To calculate a factorial of an integer, what you do is multiply the integer itself with the factorial of one less than itself
  - `factorial(5)` = `5 * factorial(4)`
- Notice the recursive pattern - `factorial(4)` will call `factorial(3)`, and so on and so forth, until our *base case* is reached.
- We know the result of `factorial(1)`, so calling `factorial(1)` will just return 1 (*base case*)

Example (Another Perspective)

- What's the smallest input? What's the simplest problem I know the answer to?
  - `0` is the smallest input - `factorial(0)` also returns `1`.
- How can I reduce my problem?
  - If you have `factorial(n)`, you can reduce your problem down by calling `factorial(n - 1)`.
  - In this step, you also assume your reduced problem gives you the correct answer (so `factorial(n - 1)` gives you the correct result - which is the recursive leap of faith)
- How do I use that result to solve my problem?
  - Multiply by `n`
  - `n * factorial(n - 1)`

Recursion vs Iteration

Recursion	Iteration
Base case is needed for a recursive problem	A condition for a `while` loop is needed
Need to reduce down to the base case	Need to reduce down to the `while` condition
Can't use variables to keep track of values because they reset (need a helper function for that)	Can have variables to keep track of values.
Needs lots of frames - takes up memory	Loops happen in 1 frame

Recursion vs Iteration

```python
# Recursion
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Iteration
def factorial(n):
    result = 1
    while n > 0:
        result = result * n
        n -= 1
    return result
```

Worksheet!

Tree Recursion 🎄

Tree Recursion

- Tree recursion is recursion but with two (or more!) recursive calls
- Useful when you need to break down a problem in more than 1 way
- Useful when there are multiple choices to deal with at one function call
- The recursive call diagram will expand similar to the roots of a tree

Example 1: Recursive Fibonacci

```python
def fib(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return fib(n - 1) + fib(n - 2)
```

- Notice how this still follows the rules of recursion
  - We have base case(s)
  - We reduce our problem (`fib(n - 1)` and `fib(n - 2)`)
  - We connect it together (with `+`)
- Often you combine things with `+`, `-`, `*`, `/` or some other function (`max`, `min`, etc).

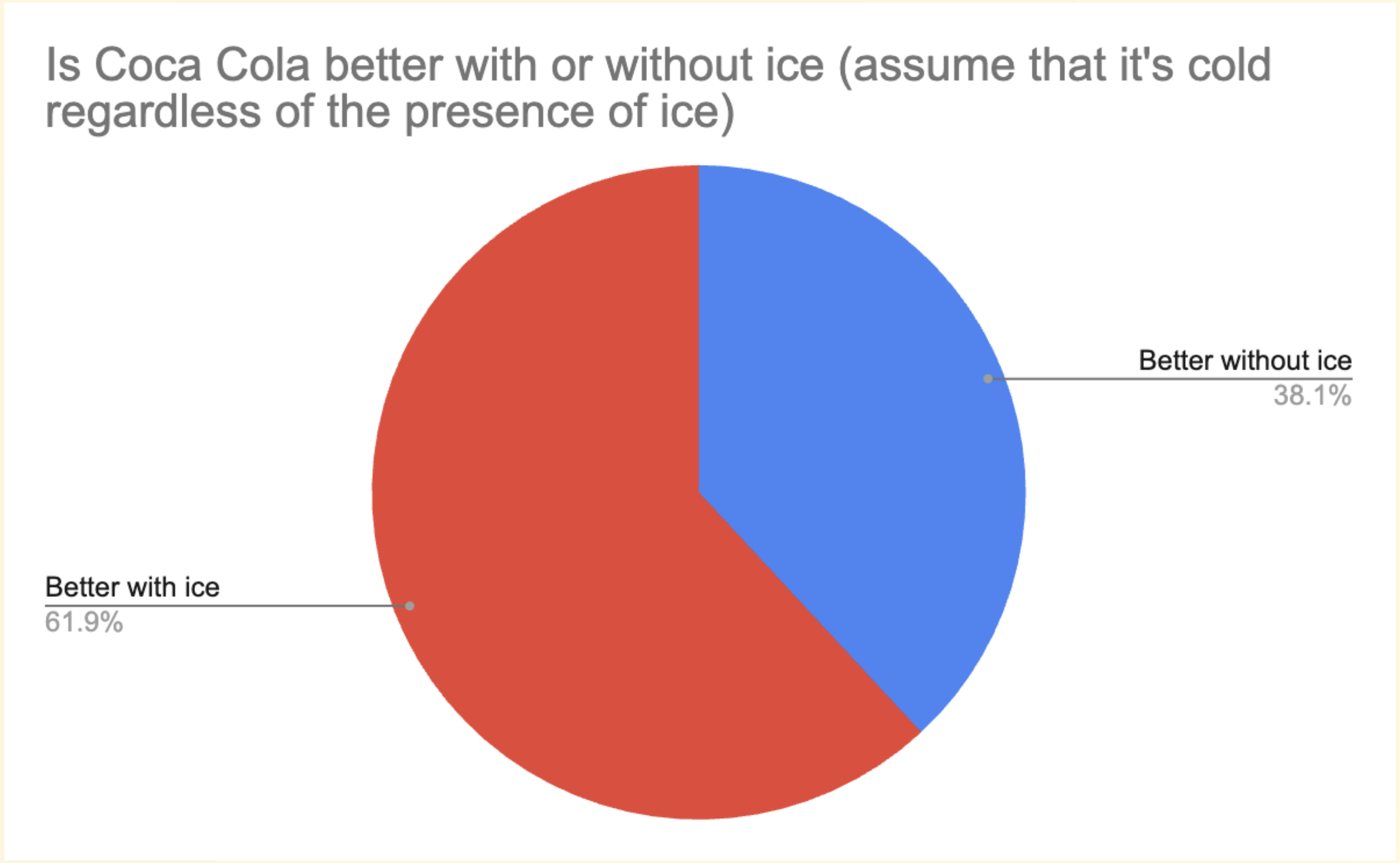
Example 1: Recursive Fibonacci

You can also write down

```
def fib(n):
  if n == 0 or n == 1:
    return n
  else:
    return fib(n - 1) + fib(n - 2)
```

Worksheet!

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!