Discussion 3

Recursion and Tree Recursion

Antonio Kam

`anto [at] berkeley [dot] edu`

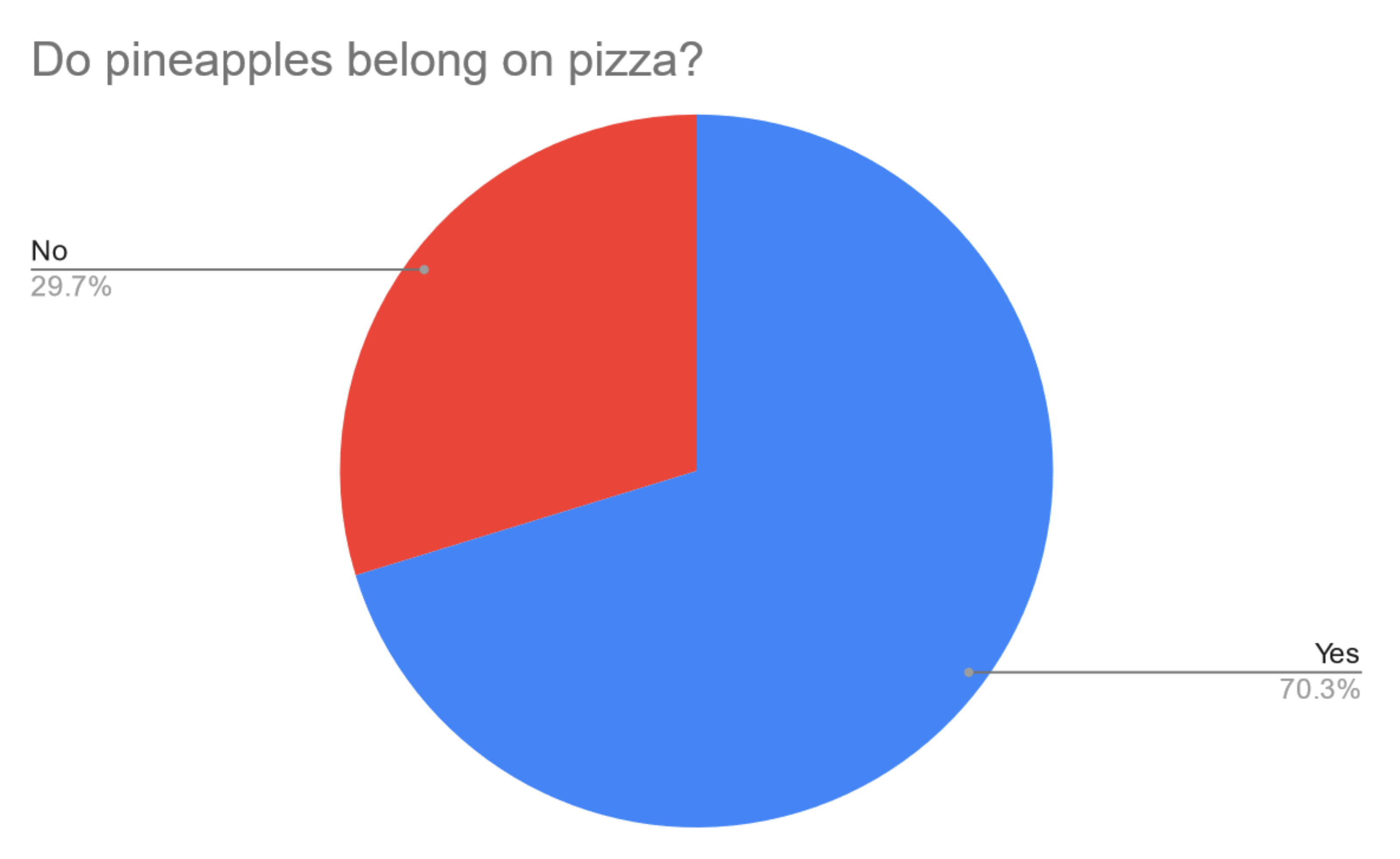
Announcements

- Lab 2 due today (2022/06/30)
- HW 1 due today (2022/06/30)
- Hog Checkpoint due tomorrow (2022/07/01)
  - Finish all of Phase 1 (all autograder tests passing) by then to get checkpoint credit
- My office hours are 1-2 PM Tuesdays and 3-5 PM Wednesdays
- I won't be in Berkeley from July 6th to July 11th (there will still be section at this time; you'll just have someone covering for me!)
  - Attendance will still work even if you don't use the same form

Temperature Check 🌡️

- Recursion
- Tree Recursion

Results from last section



Questions and Comments from last section

- Mini-lectures in the middle of labs are good!
  - Will continue to do this for future labs 😎
- I think the consensus is that a hybrid of whiteboarding and using slides is a pretty good option
  - I'll do a mix with more focus on whiteboarding from here on

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**

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

```python
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
```

Question 1 (Walkthrough)

Write a function that takes two numbers `m` and `n` and returns their product. Assume `m` and `n` are positive integers. **Use recursion!**

*Hint: 5 * 3 = 5 + (5 * 2) = 5 + 5 + (5 * 1).*

```python
def multiply(m, n):
    """ Takes two positive integers and returns their product using recursion.
    >>> multiply(5, 3)
    15
    """
    "*** YOUR CODE HERE ***"
```

Worksheet!

Attendance

`links.rouxl.es/disc`

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

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

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!