Discussion 3

Recursion

Antonio Kam

anto [at] berkeley [dot] edu

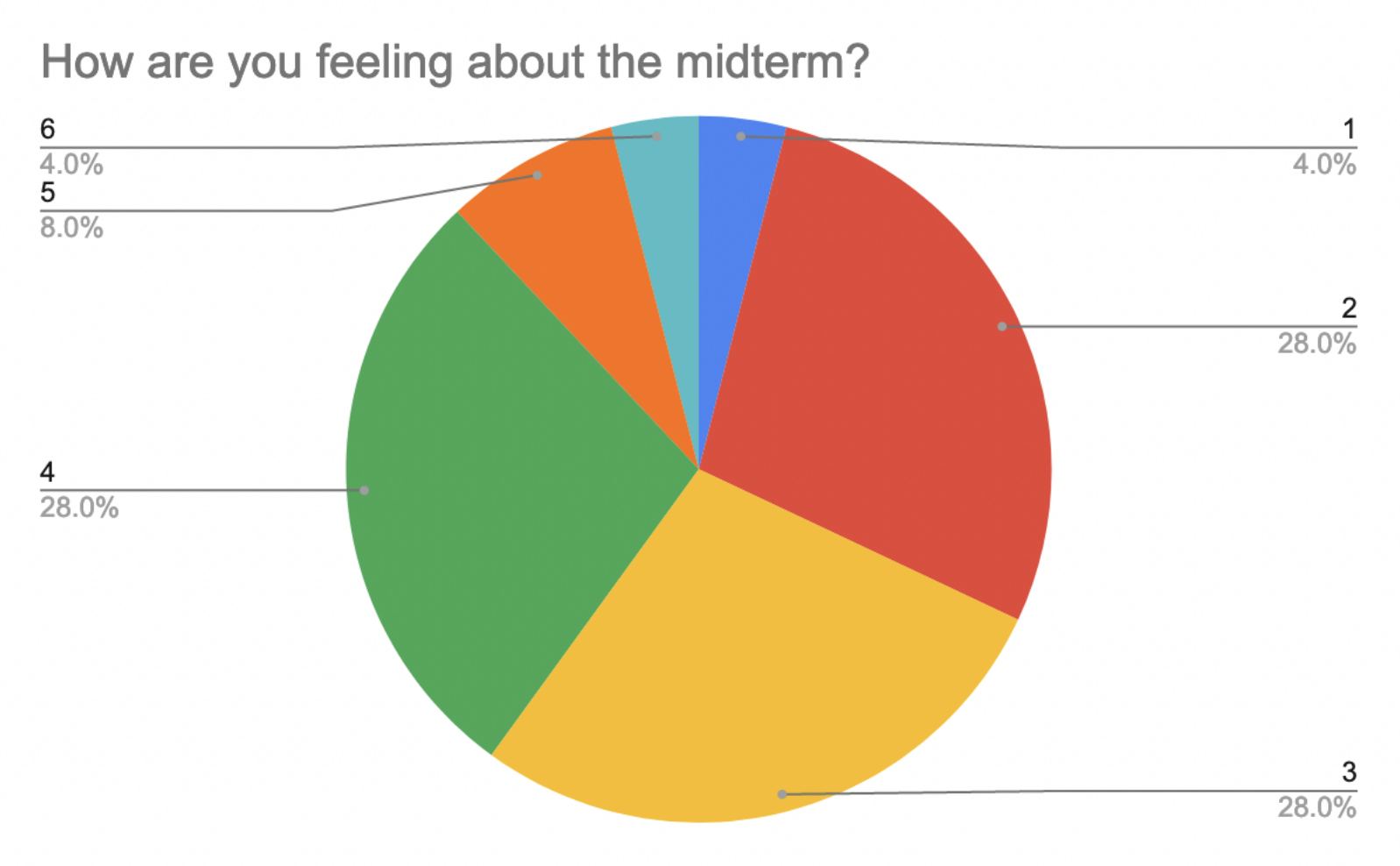
Announcements

- Exams have been released 📜
  - There is a video walkthrough for all questions, please watch them if you want to better understand a question 📹
- HW 03 released today 🖋️
- New CSM sections will open soon 👀
  - Not sure when, probably next week? 🙄
- Midterm clobber policy
  - I believe it was mentioned in lecture, but exact logistics for this have yet to be figured out

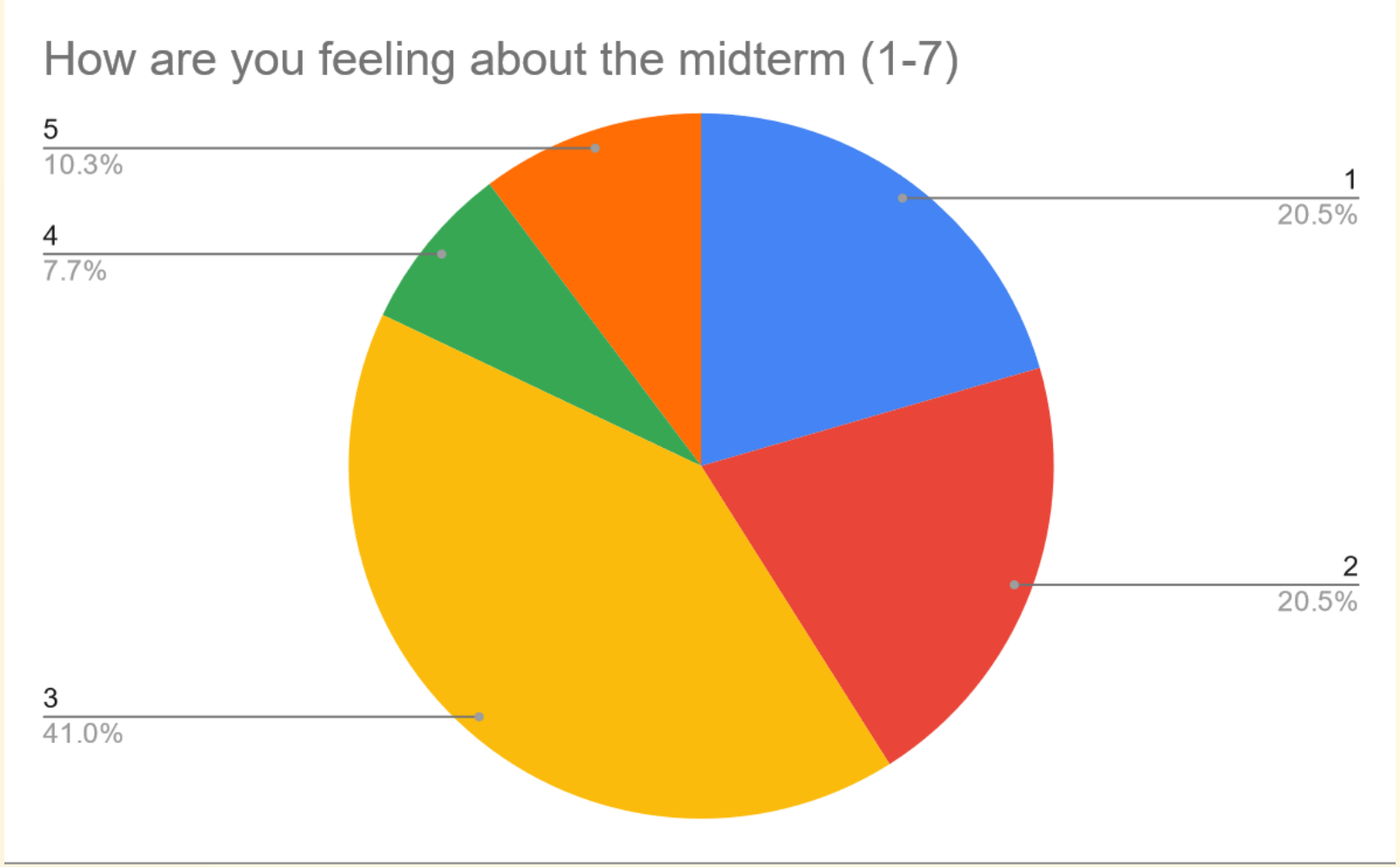
Temperature Check 

- Environment Diagrams
- Higher-order Functions
- Recursion

Results from last section



Results from SU22



Questions and Comments from last section

- Making mistakes then trying to find where they are is confusing!
  - Honestly very interesting - this video might be interesting to watch; while it's not perfectly sound educational research, results have found that students tended to improve more with this method, but also found it more confusing 🤔
  - I was only planning on doing that for environment diagrams, because it's pretty difficult to make it work with other topics, but if you're interested, watch the video! I found this comment super interesting, so thank you to whoever left this 🥳
- HOF Double Inputs:
  - `(lambda x: lambda y: x + y)(3)(4)`

Questions and Comments from last section

- Nested `while` loops
  - Think of which block gets executed on the repeat; which indent is handling what situation
  - Reminder bullet point for me to write up an example
- Environment Diagram review
  - Would highly recommend trying out FA19's `pumbaa` question and really understanding it - it's a truly good question for learning the rules behind environment diagrams
- "My friend loves Hollow Knight and he has introduced me to the game's music; the soundtracks are truly amazing :D"
  - I tend to listen to soundtracks to decide which games to play (which is quite backwards, whoops)

Midterm

- Important to mention that the midterm is worth `30` points out of the total `300` (which goes to *around* `310` including extra credit)
- Exams are not the only component of your grade in this class - you have discussion/lab attendance, and homeworks/projects where you're given unlimited attempts, and there are also no hidden tests
  - This means that quite a large portion of the points in this course are not based on exam performance.
- Many people struggle on exams; it's completely normal to not feel too good about your own performance (in fact, quite a lot of exams in higher education will have averages lower than what you may have been used to in HS)
- Please feel free to reach out to me if you have any questions.
- Other resources: Advising OH, Instructor OH, etc.

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)
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
```

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).*

```
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`

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!