CS 61A SU26

Lecture 05: Recursion

June 29, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

5 min

- Announcements
- Quizzes FAQ
- Computing in the News

Announcements

- **Syllabus quiz typo:** There are no assignment or quiz drops this summer
- **Quiz 1** happening today and tomorrow!
  - Ed partial closure
  - **Scheduling a quiz during your lab/discussion section does not count as an excused absence.** We'll only be lenient for Quiz 1.
- **Quiz 2** reservations are open!
  - From now on you'll need to show proof of quiz reservation during lab checkoffs
- **The time for Rebecca's instructor OH on Thu has changed from 6:30 pm to 7:00 pm** to take into account time to walk to Soda 347
  - Will only answer non-course questions during the walk (e.g. about career/professional development/grad school)
- FYI: I will mark slides as "Skipped" in Google Slides if we didn't have time to go over them, but they are great study resources and still in scope unless otherwise stated

Quizzes FAQ (1 of 2)

Q: When taking the quiz, will I see if I got the correct answer immediately?
A: No, with the exception of specific questions (e.g. coding questions)

Q: How many attempts do I get for each question?
A: Basically infinite. (For coding questions, 100 attempts and you can see all test results and use print debugging if you want.)

Q: When will quiz scores be released?
A: After everyone has taken the quiz, e.g. ideally on Wednesdays.

Q: What questions can I ask on quiz days in OH and Ed?
A: You can ask everything you normally would, except you cannot ask questions about the quiz content (you can, however, ask about quiz logistics or the concepts covered in the quiz).

Quizzes FAQ (2 of 2)

Q: After everyone has taken the quiz, will the questions and solutions be released?
A: No, because we may reuse questions in the future. However, you can go to OH and ask a staff member to view your quiz and go over it together. You just can't take notes/photos (academic integrity violation!)

Q: Will you release the distribution of exam scores?
A: No, generally for all exams (quizzes, midterm, and final) we will not release the distribution. However we as staff do look at it and if we determine that the exam was unfair or a lot of people did poorly, we reserve the right to make grading adjustments.

Q: When do quiz reservations open?
8 am the week before on PrairieTest

Q: Can I take the quiz during my lab/discussion time?
No, unless you want to forfeit your lab/discussion points. Taking a quiz does not count as an excused absence, see previous slide.

Q: Will I get my exam-related DSP accommodations?
Yes if you send us your letter with enough advance notice; see Lecture 3 slides or Ed for more info

Computing in the News

Today: TIDAL cracks down on AI music by cutting off monetization (Tech Crunch)

- TIDAL is a music streaming service (like Spotify or Apple Music)
- *"With the changes, fully AI-generated music on TIDAL will be identified and tagged as such, allowing listeners to see an "AI" badge next to any tracks deemed to be 100% AI. These tunes will not be able to be monetized or collect royalties, and will not be eligible for direct-to-fan sales, the company noted."*
- Yet another contentious topic about the role of AI in creative industries
- See also: Suno, Udio, Xania Monet, Controversy over fake artists on Spotify

Recursion: Conceptual

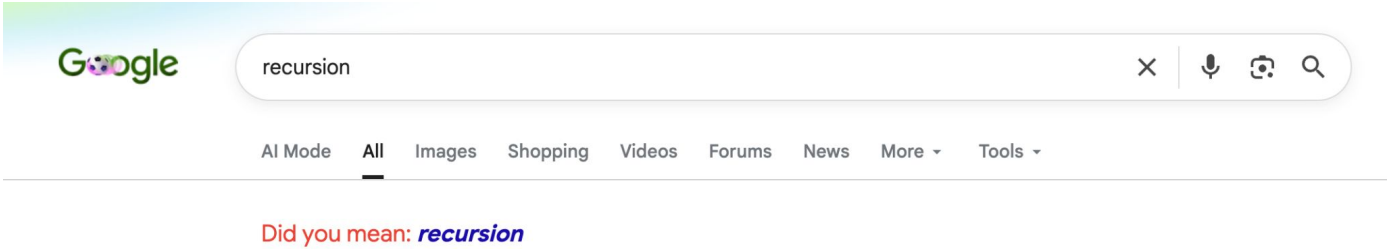
10 min

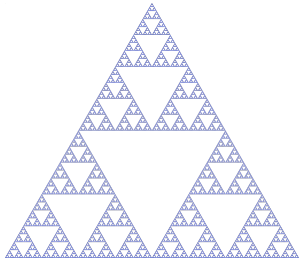
- Recursion definition
- Why recursion over iteration?
- Why *not* recursion?
- Example: Taco truck
- Anatomy of a recursive function

Recursion definition

A function is **recursive** if it calls itself (either directly or indirectly).

Each time we make a recursive call, we **break the larger problem into smaller sub-problems**, until we get to the simplest sub-problem (base case).





[Sierpiński triangle - Wikipedia]

Why recursion over iteration?

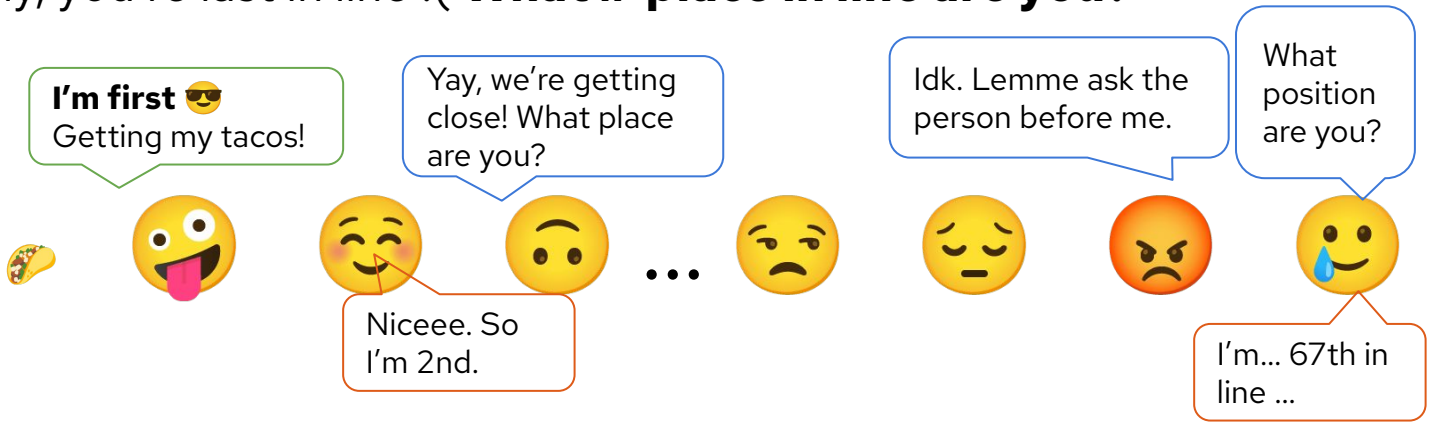
- So far, we've learned how to use while loops (iteration) to repeatedly do things and solve problems
  - However, iteration is actually a special case of recursion
- **Sometimes, it's easier to express the solution to a problem recursively rather than iteratively**
  - Ex: Processing recursive data structures like trees (e.g. your file system) or linked lists (more on that later!)
- **Some programming languages don't have loops**
  - Ex: Scheme (more on that later!)

Why *not* recursion?

- **Takes up more stack frames (computer memory) than iteration**
  → can lead to stack overflow (yes, that's where StackOverflow comes from)
  - In Python, it's called a `RecursionError`
- **Sometimes it's easier to express the solution to a problem iteratively**

You're in a long line...

Sadly, you're last in line :(**What # place in line are you?**



Rephrase: how many people are in front of you?

Ask the person in front of you what position they are in.

→ repeat (they ask the next person)

→ reach the person in front: 1st in line!

→ person behind them calculates position

Parts to a recursive problem

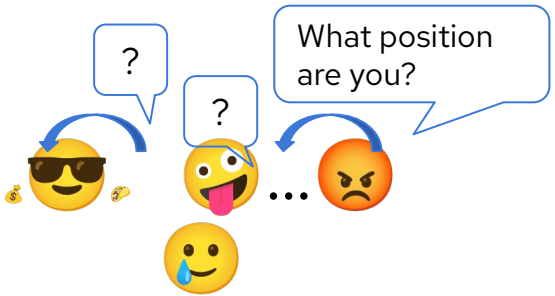
1. Base Case

Simplest possible version of the problem

Person in front of the line knows they're 1st.

2. Recursive Call

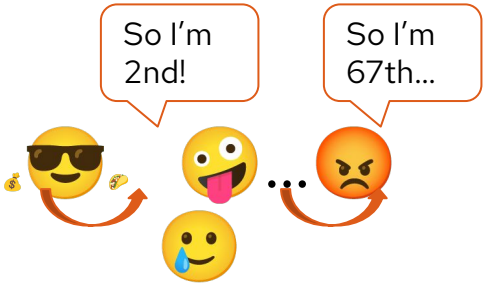
Recursive call on smaller subproblems



Asking the person in front of you.

3. Solve the bigger problem

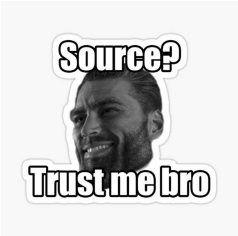
Solve the larger problem with the subproblems



Add 1 to the person before you.

Anatomy of a recursive function

1. **<u>One or more</u> base case(s)**
   a. What is the simplest possible input I could receive and what should I do in that case?
   b. When does our recursion stop?
2. **<u>One or more</u> recursive case(s)**
   a. How can I break the problem into smaller sub-problems?
3. **Solve the larger problem / recursive leap of faith**
   a. How do I use solutions from subproblem(s) to solve the larger problem?



Recursion: Coding

20 min

- WWPD: Countdown
- Factorial
- Self-Reference
- Mutual Recursion

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Which implementation of the countdown function results in "5, 4, 3, 2, 1, Blastoff!" being printed (one on each line) if countdown(5) is called?

Countdown Environment Diagrams

- Implementation A
- Implementation B

Factorial (1 of 3)

The **factorial** of a non-negative integer n is defined as the product of all integers from n to 1.

Ex: 5! = 5 * 4 * 3 * 2 * 1 = 120

Edge case: 0! = 1

We can define this recursively: n! = n * (n - 1)!

Notice: 5! = 5 * 4!

(4 * 3 * 2 * 1)

Factorial (2 of 3)

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

Let's examine the environment diagram for `fact(3)` to see how recursion works!

Factorial (3 of 3)

Takeaways:

- It's important to track the inputs to the function in different frames
- In recursive functions, we have to go all the way down the stack and then back up
- The base case is our stopping point (we don't make a recursive call there)

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

What would happen if you executed fact(-1)?

Demo: Large factorial

What happens if we call `fact` on a really large number?

Why does this happen?

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

True or false: All recursive functions must have some kind of if-else statement.

Self-Reference

Self-reference is when a function refers to its own name in the body.

Ex: `print_sums` (Python Tutor)

Mutual Recursion

Mutual recursion is a special type of recursion where 2 or more functions call each other.

Ex: `is_even` and `is_odd` (Python Tutor)

5 min break

Practice

(reminder to self to use high contrast theme)

40 min

- Sum Digits (Recursive)
- Luhn Algorithm (Mutual Recursion)

Practice: Sum Digits

- Implement `sum_digits` recursively
- Download starter code `05.py` from the course website under Lecture 5
- Run doctests with `python3 -m doctest 05.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Converting from iterative to recursive (1 of 2)

```
def sum_digits(n: int) -> int:
    if n < 10:
        return n
    else:
        last = n % 10
        all_but_last = n // 10
        return sum_digits(all_but_last) + last
```

```
def sum_digits(n: int) -> int:
    total = 0
    while n > 0:
        last = n % 10
        total += last
        n = n // 10
    return total
```

Converting from iterative to recursive (2 of 2)

```
def sum_digits(n: int, curr_sum: int) -> int:
    if n == 0:
        return curr_sum
    else:
        last = n % 10
        all_but_last = n // 10
        return sum_digits(all_but_last, curr_sum + last)
```

Takeaway: You can use arguments to store intermediate state

Luhn Algorithm (1 of 3)

Used to verify credit card numbers!

1. From the rightmost digit moving left
   a. Double the value of every second digit. If this results in a sum > 9, then sum the digits of the product
2. Take the sum of all the digits

See more: Luhn algorithm - Wikipedia

1	3	8	7	4	3

Luhn Algorithm (2 of 3)

Used to verify credit card numbers!

1. From the rightmost digit moving left
   a. Double the value of every second digit. If this results in a sum > 9, then sum the digits of the product
2. Take the sum of all the digits

See more: Luhn algorithm - Wikipedia

1	3	8	7	4	3
2*1 = 2	3	2*8 = 16	7	2*4 = 8	3

Luhn Algorithm (3 of 3)

Used to verify credit card numbers!

1. From the rightmost digit moving left
   a. Double the value of every second digit. If this results in a sum > 9, then sum the digits of the product
2. Take the sum of all the digits

See more: Luhn algorithm - Wikipedia

1	3	8	7	4	3
2*1 = 2	3	2*8 = 16	7	2*4 = 8	3
2		1+6 = 7		8	

2+3+7+7+8+3 = 30
30 is divisible by 10 → Valid number!

Practice: Luhn Algorithm

- Implement `luhn_sum` (and `luhn_sum_double`) using mutual recursion
- Download starter code `05.py` from the course website under Lecture 5
- Run doctests with `python3 -m doctest 05.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?
- Recursive call visualizer (created by Pamela Fox, former UC Berkeley lecturer)