# CS 105: Introduction to Computer Science

Prof. Thao Nguyen

Spring 2025

Materials adapted from Dave Wonnacott

# Recursion

A function can not only call *other* functions, it can (sometimes) call *itself*!

- If you're not careful, this is *circular reasoning*
  - like if circle_overlap and circle_overlap_two_figures *both* just call the other
    - then, no function is actually doing any of the work!
- Normally, a function should call on some simpler function, e.g.,
  - cube_x could call on square_x
  - fourth_power_x could call on cube_x, fifth_power_x could call on fourth_power_x, etc.
  - ... no problem, right? That's top-down design ... each function does a bit of the work!
- A function can call on *itself* for a *simpler instance of the problem*, e.g.,
  - power(x, 3) could call on power(x, 2)
  - power(x, 4) could call on power(x, 3)
  - ... consider this as an example of design-by-cases, with each exponent at case!
  - group exercise: write this code, try it out, or pythontutor, for exponents 2 through 5 (or more)

# Recursion (group exercise)

Since a function can not only call *other* functions, it can (sometimes) call *itself*...

- Edit the power function you wrote before, with design-by-cases for
  - power(x, 3) could call on power(x, 2)
  - power(x, 4) could call on power(x, 3)
- Since almost all the cases are basically similar,
  - edit all but the simplest so they are identical
  - blend them all into one case, by using <= or >= or something, rather than lots of ==
- Try running the function, in pythontutor ... does it still work?

# Thinking about recursion

How can we avoid circular reasoning?

Discussion: thinking *abstractly* about function calls, both recursive and otherwise.

# Basic Recursive Design checklist

If you're not sure whether your function works, or why it doesn't, ask yourself:

1. Have I identified one (or more) *valid simpler* instance(s) of the *same problem*?
2. Have I built *my answer* from the answer to the simpler instance(s)?
3. Have I identified a *base case*?
4. Is my *answer* in the base case correct?
5. Does Step 1 always get *simpler* and always *reach the base*?

(See "From Vision to Execution", Section 4.4 "Basic Recursive Design")

Good way to build instinct: Draw a diagram, circle sub-problem.

# Function/Algorithm Design examples

Which of these should be use, to square or cube something? Done!

What about to find the alphabetically-earliest letter in a word?

- Relate to a solved problem/library function
- Design by cases
- Top-down design
- Or, now, "basic recursive design"

Group exercise: start writing "earliest_letter" function