CS 105: Introduction to Computer Science

Prof. Thao Nguyen

Spring 2025

Materials adapted from Dave Wonnacott

Recap/References for Basic Recursive Design

While working with Basic Recursive Design, remember:

- Five things to consider, from "Basic Recursive Design" section of FVtE book
  - most significant, usually: "find a simpler instance of the same problem"
- Steps for converting those five things to consider into Python
  - discussed in book, also done via example
- "Helpful, Competent Friend" metaphor
  - If you're asked for the earliest letter in an 8-letter word ...
  - what *smaller* earliestLetter problem could you ask a *helpful, competent friend*
  - so that you could make your answer most easily?

Group discussion of "earliest_letter" problem.

Also, discussion of power function, if requested.

Programming Idioms & Notional Machine

Now that we've covered the fundamentals of programming and design

- Will explore some choices and how we express our design (idioms)
  - Often there are several ways to program one idea
  - Later, we will discuss reasons to choose one option versus another
- A programming idiom often reflects a way of thinking about computation
  - This abstract understanding of computation is referred to as a *notional machine*
  - A collection of related idioms that work together are referred to as a *programming paradigm*

The "Dictionary Notional Machine" for 1 function

- We can think of the computer as running a single function as follows:
  - make an arrow from each parameter name to the value of the argument passed
  - move through the body of the function, a line at a time (based on "if", of course)
  - each time we see an "=", adjust the set of name/value associations
- This can be animated by using:
  - the PyCharm "debug" view
    - Doesn't interact well with doctest, so use AFileForDebugging.py
  - pythontutor.com
    - Doesn't work well with doctest and types, just omit those

The "Dictionary Notional Machine" for function calls

- Each *call* to a function gets its own dictionary
- Sometimes we only need to think about one thing at a time
  - this is what the debugger shows us
  - sometimes this is enough to understand a problem
  - technical term: the "function-*call stack*" of "*frames*" (dictionaries) of current calculations
- Sometimes we need the "big picture", interactions between steps
  - The pycharm debugger helps us see all functions currently running (the "*call stack*")
  - A "function-*call tree*" shows all calculations from the whole run of the program
  - Neither tool shows the whole call tree

Tools and abstraction

Sometimes the "big picture" is too much, but there's a lot going on

- As with programming, **abstraction** is the answer:
  - In pycharm debugger, use "step over" to see the entire result of a function call
  - If the result of that function isn't right, debug it first
  - Once a function is debugged, just use "step over"
  - For a recursive function, step over simpler *instances*
    - E.g., debugging power(3, 5) without watching details of power(3, 4)
- This approach works best if we can limit interactions among functions
  - So far, functions interact mostly with parameters and return values
    - These are known as "*pure* functions"

The "Substitution Notional Machine"

- Within a function, just substitute variable values for names, in the text
  - E.g., for power(5, 3), replace *base* with 5, and *exp* with 3, in the body
  - Only straightforward for *pure* functions with unchanging variables
- As with the Dictionary Notional Machine
  - Simple things are simple
  - Looking at all the detail is often too confusing
  - **Abstraction** is the answer, skip over sub-function details
- Important differences from the Dictionary Notional Machine
  - We can chose the order, or even substitute *expressions* without filling in all the values
  - For trusted code, we can substitute the *postcondition expression* (providing abstraction)
  - Together, these can help us reason about general properties, check for consistency