Welcome to Lecture 9: Tree Recursion + Fractals

We will start at 10 AM

Announcements

- Project 3 will be released later today
- Victoria’s OH got rescheduled to today (Tuesday) 7 to 9 PM - online only
- *Andrew out this week - no OH
- No class on Thursday, July 4th.
- Limited OH on Friday, July 5h

Today’s Topics

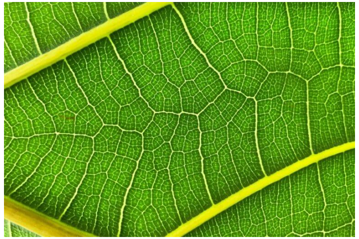
- Announcements
- Review
- Fractals
- Multiple recursive calls (tree)
- Practice Problems

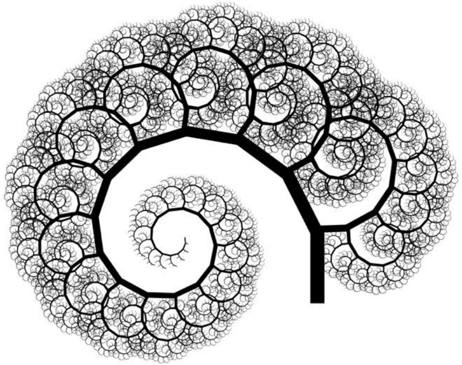
Review from Last Lecture

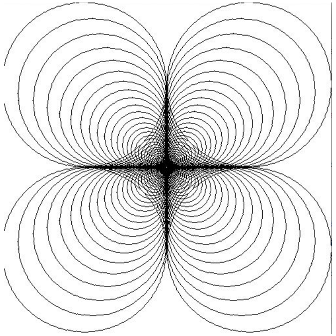
- Intro to Recursion (Linear only)
    - Two main components to recursion:
        - Base Case: The simplest, most reduced case
        - Recursive Case
            - Split the problem into smaller parts
            - Invoke the recursive function
            - Combine into a total output
- We can have multiple base cases and multiple recursive cases
- Data type of function will usually match the base case and recursive case
- When we call the recursive function, we open a new frame with an input that has been reduced in some way (reduced meaning closer to base case)

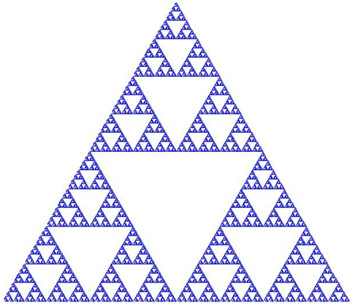
Tree Recursion: Fractals

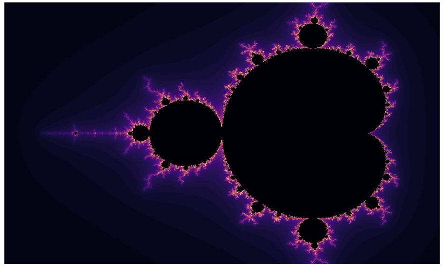
- "a complex geometric shape with a detailed structure that repeats itself at any scale"
- A series of patterns that are repeated recursively
- We will have a problem like this on the midterm!
- Fractals in nature! Ex: seashells, spiral galaxies, trees, leaves, the brain / neurons, veins, lightning bolts, rivers branching, etc.









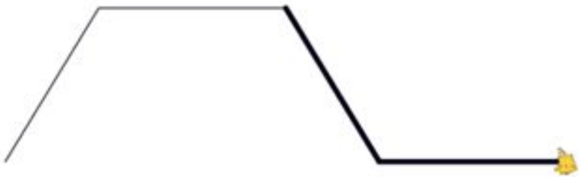


Fractals

- We can build awesome and fun images!
- Let's play around in Snap*!*

Fractals

n = 1

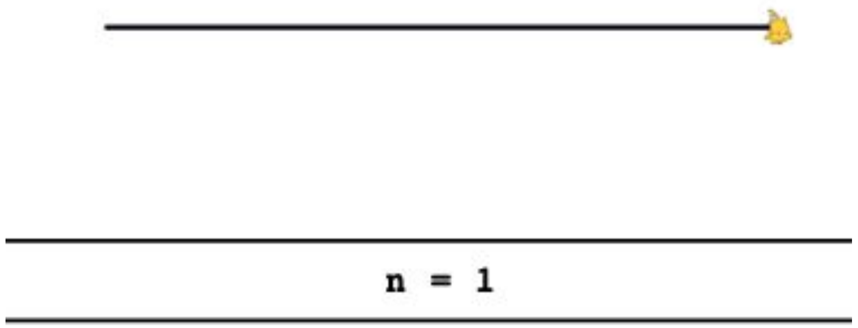


n = 2

n = 3

n = 4

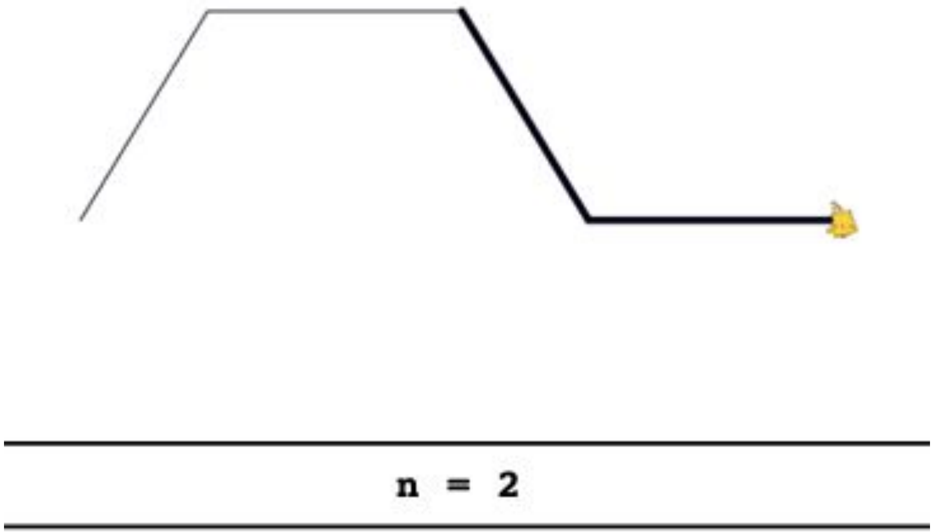
Fractals



What is the first thing that happens?

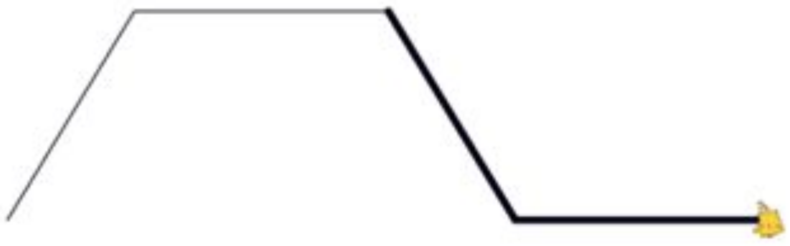
We move in a straight line!

Fractals



What is the first thing that happens?

Fractals

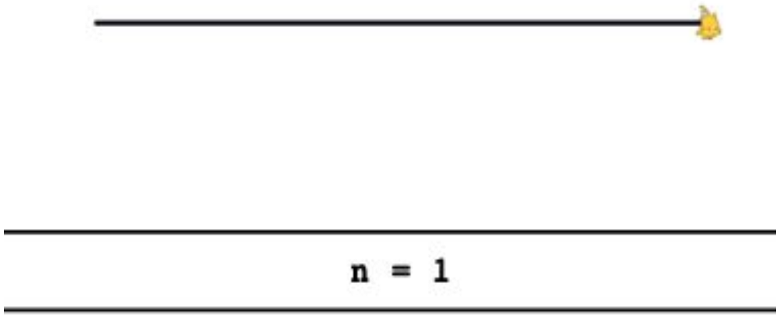


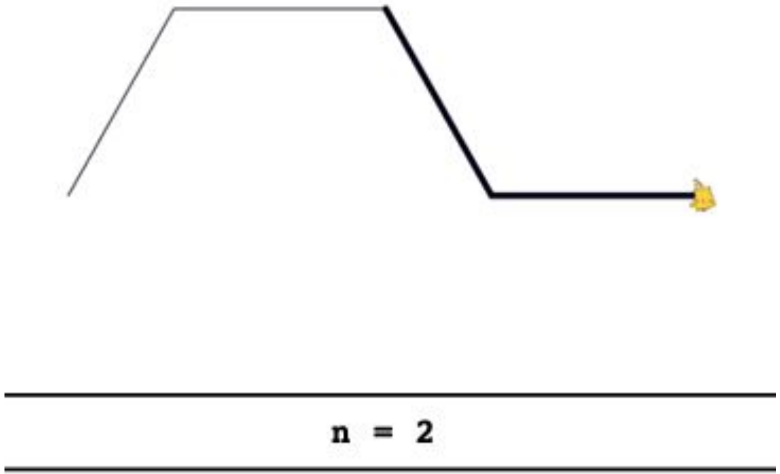
n = 2

What is the first thing that happens?

We turn! Then we move. Then turn again… etc.

Fractals

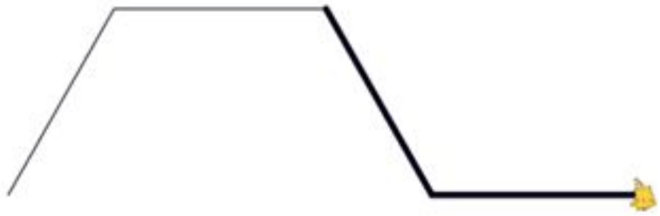




- How many times, do we see the bolded lines of the n=1 image in n=2 image?

Fractals

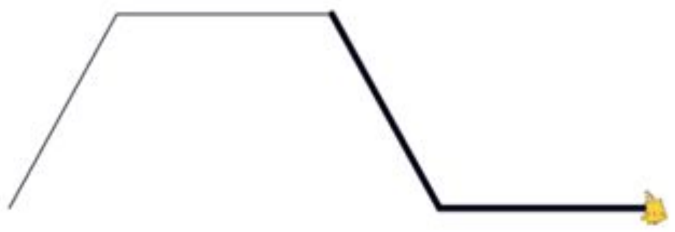
n = 1



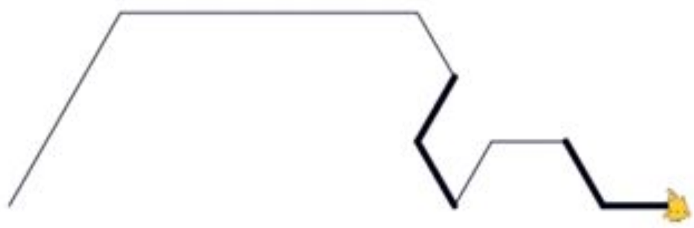
n = 2

- How many times, do we see the bolded lines of the n=1 image in n=2 image? **2**

Fractals



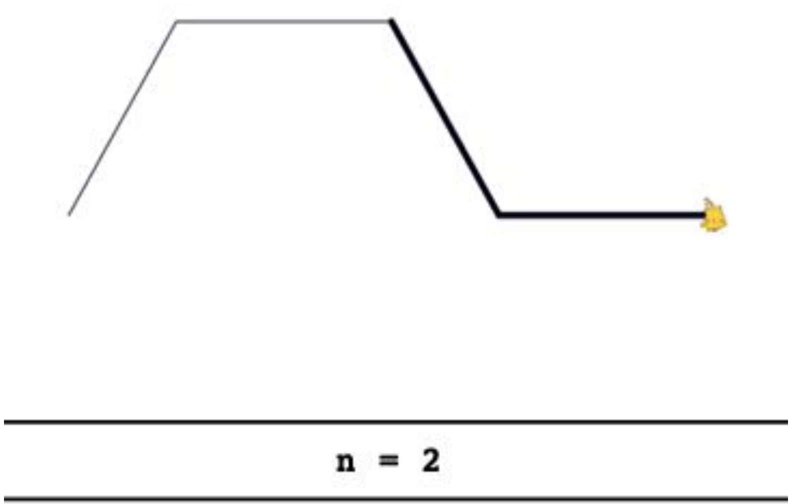
n = 2

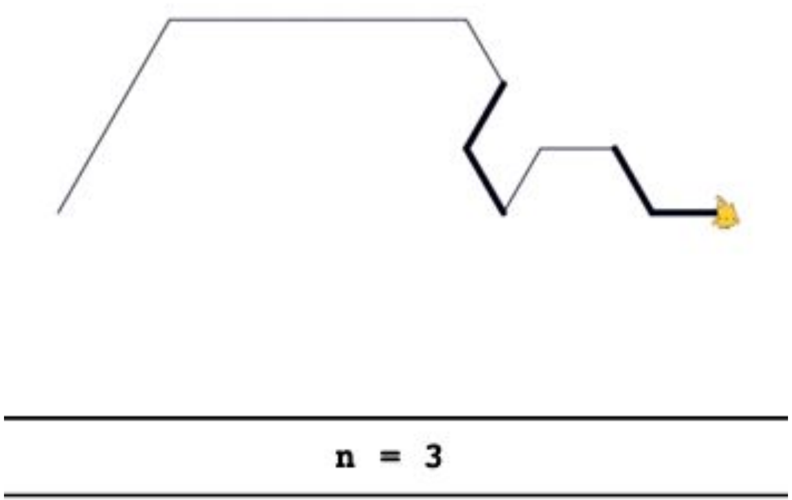


n = 3

- How many times, do we see the bolded lines of the n=2 image in n=3 image?

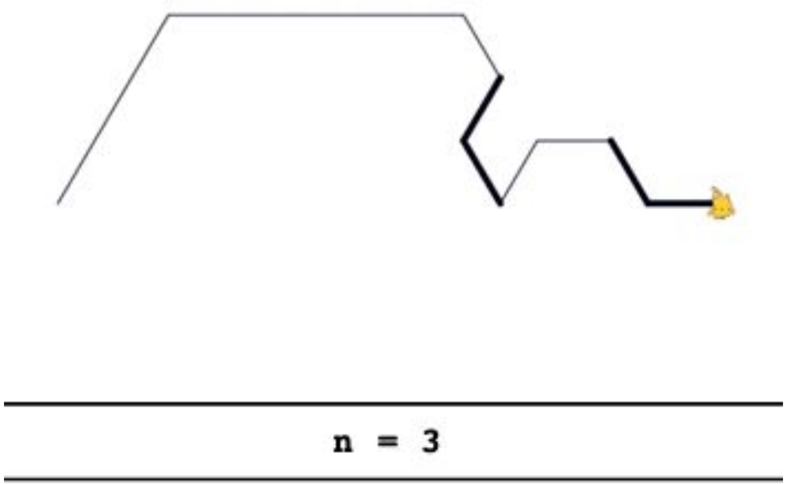
Fractals

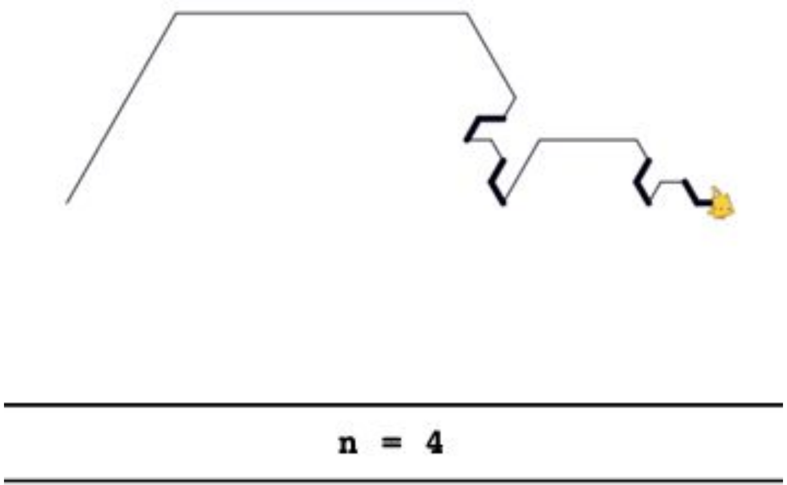




- How many times, do we see the bolded lines of the n=2 image in n=3 image?

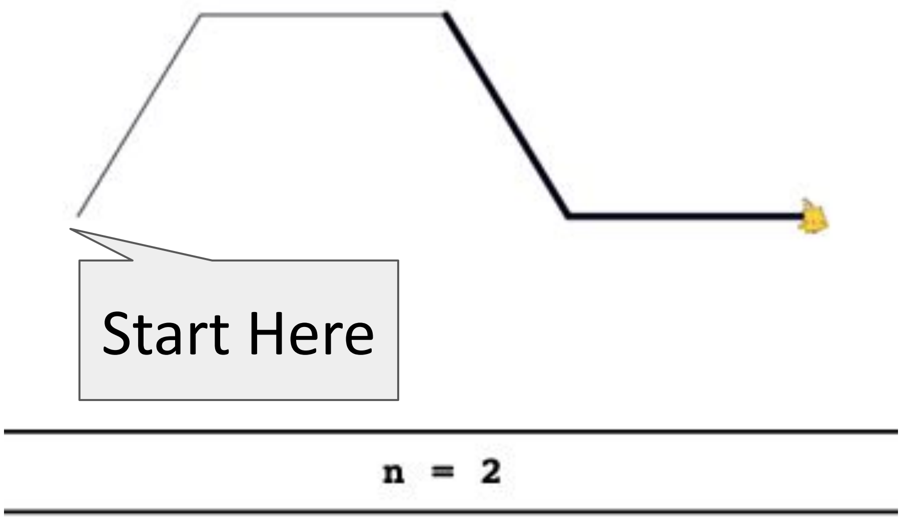
Fractals





- How many times, do we see n=3 image in n=4 image?

Let’s Build it!

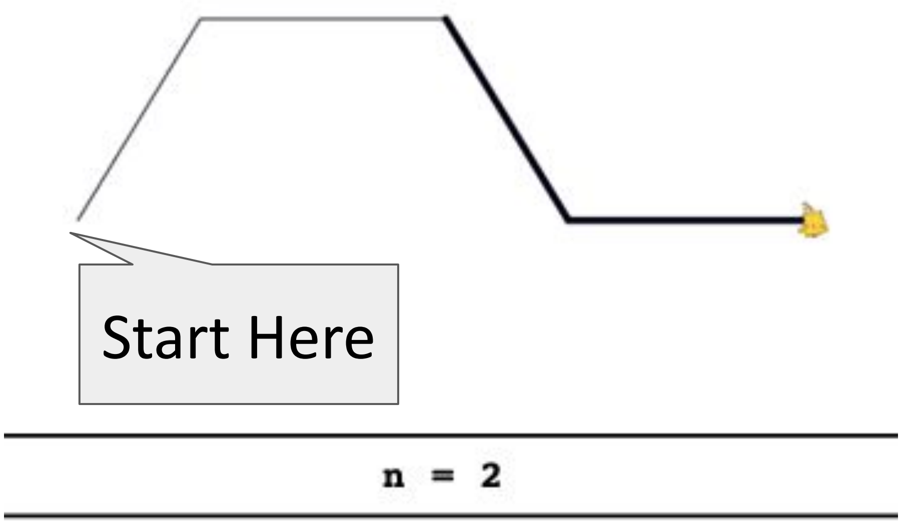


We can trace our recursive fractal based on n = 2.

We know the bolded lines are the recursive calls.

- What happens first?

Let's Build it!



We can trace our recursive fractal based on n = 2.

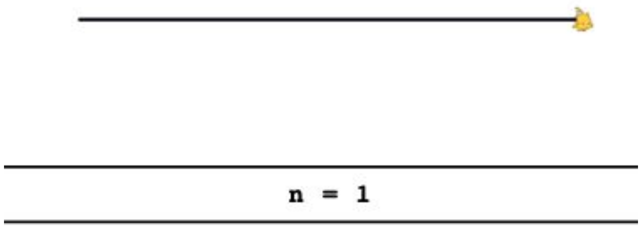
We know the bolded lines are the recursive calls.

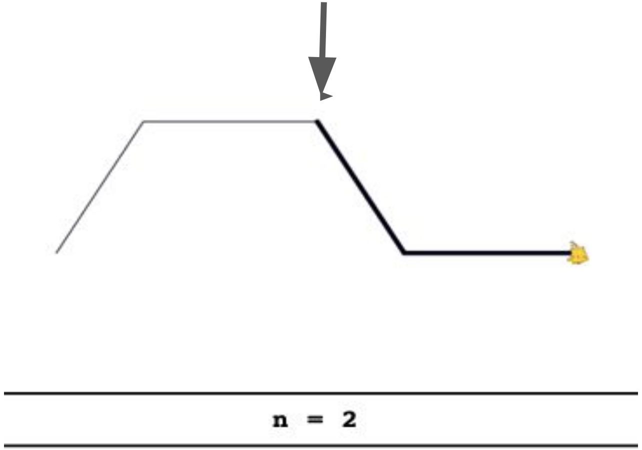
- What happens first?
  - Turn
  - Move
  - Turn
  - Move
  - Recursive Call
  - Turn
  - Recursive Call

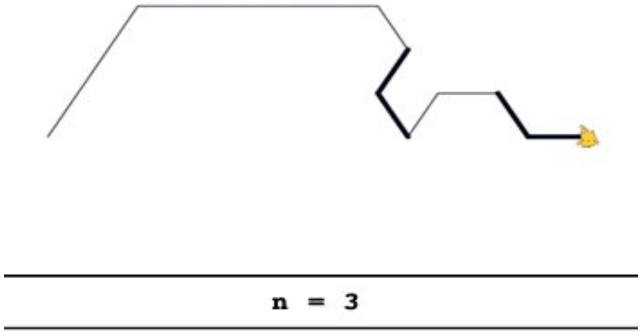
Let’s Build it!

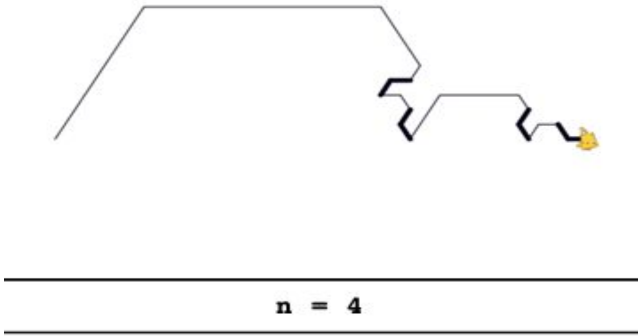
The degrees we turn are 60 and the recursive size is 1/3

Trace recursive fractal based on n = 2

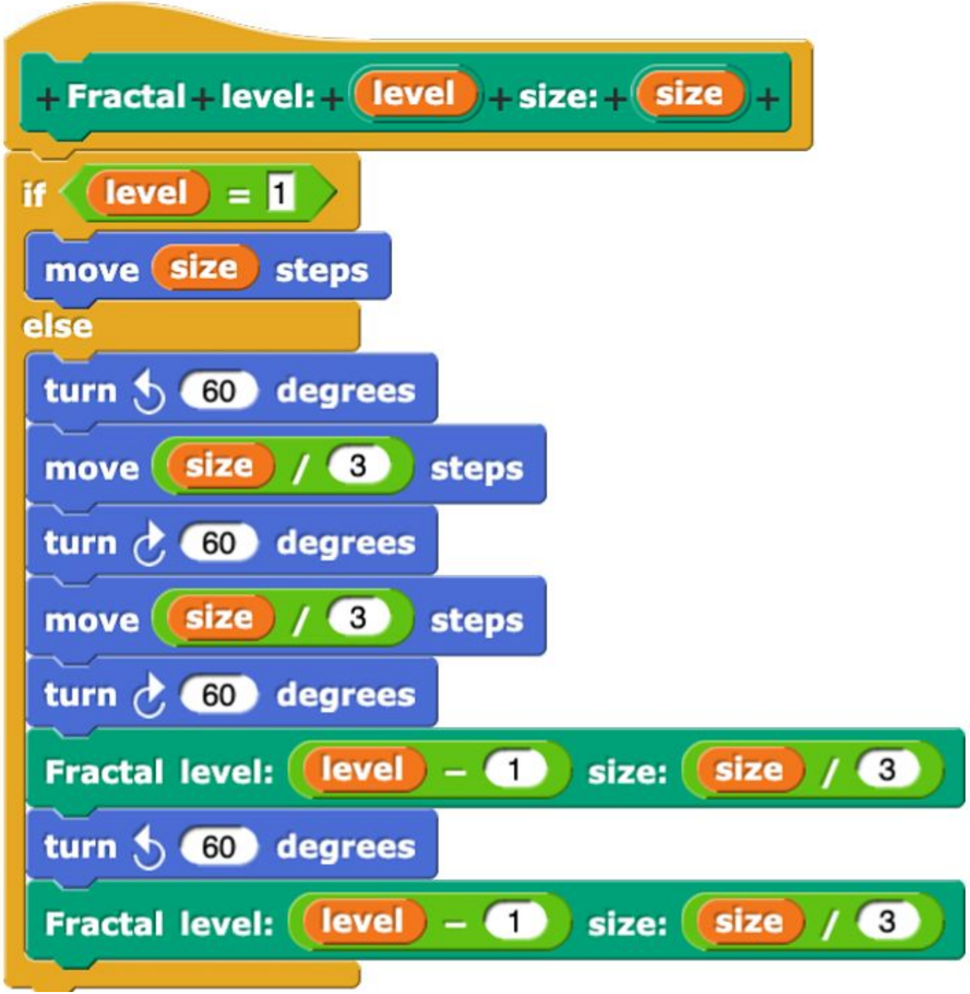








Let's Build it!

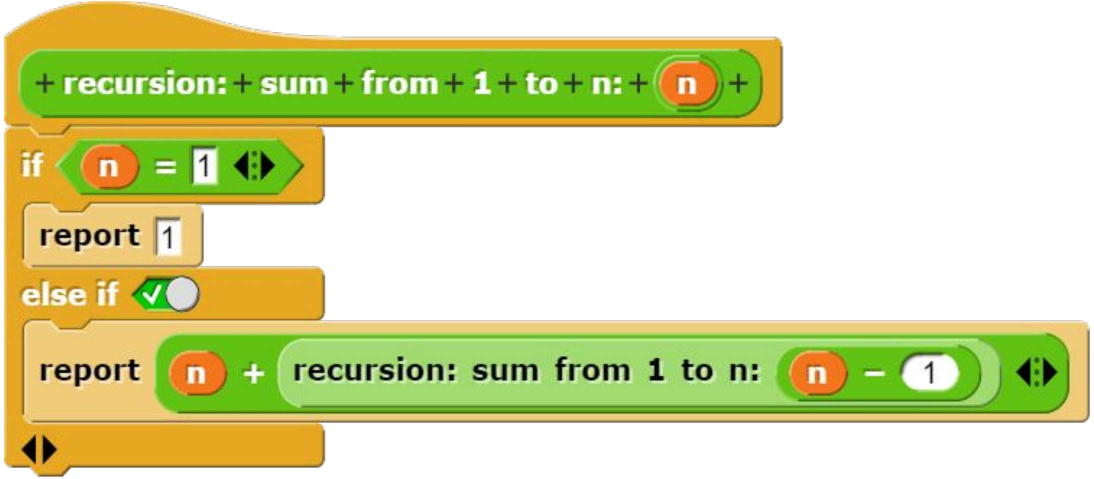


Fractals

- We can always trace the code, given the images at each level
- The sprite MUST end up in same direction it started in
- We MUST always have a base case!
- The number of times we see the image of n - 1 in n, tells us the number of recursive calls we need in the fractal.

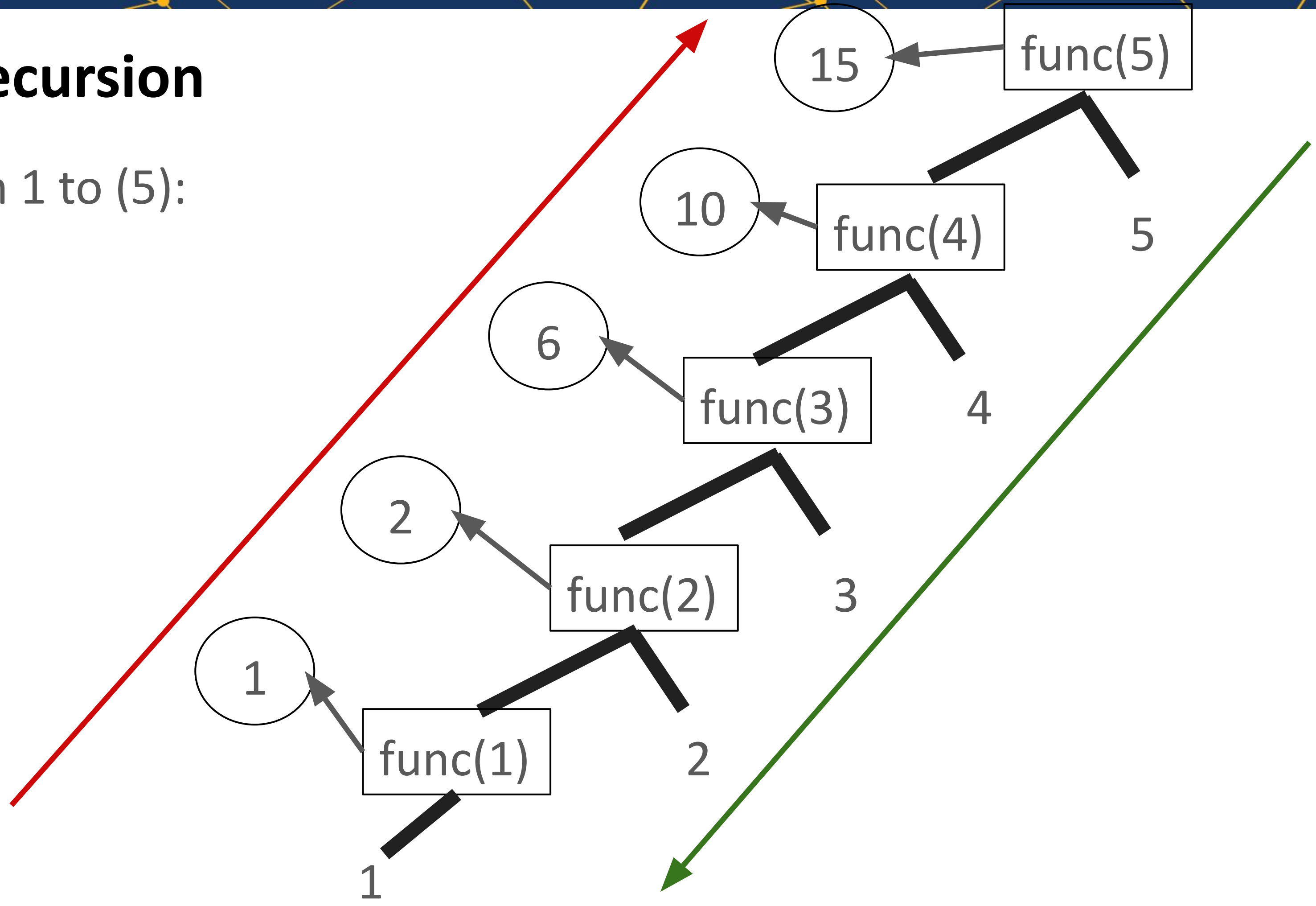
Definitions

- What is linear recursion?
  - A function that invokes itself as part of the definition
- What is tree recursion?
  - A recursive function with multiple recursive calls



Linear Recursion

Calling sum 1 to (5):



Tree Recursion Example

- Objective: Write a function that returns the fibonacci number:
- Fibonacci Numbers: A sequence of numbers where each number is the sum of two before it: 1, 1, 2, 3, 5, 8, 15….

0	1	2	3	4	5	6
1	1	2	3	5	8	15

1 + 1 = 2

1 + 2 = 3

2 + 3 = 5

3 + 5 = 8

5 + 8 = 15

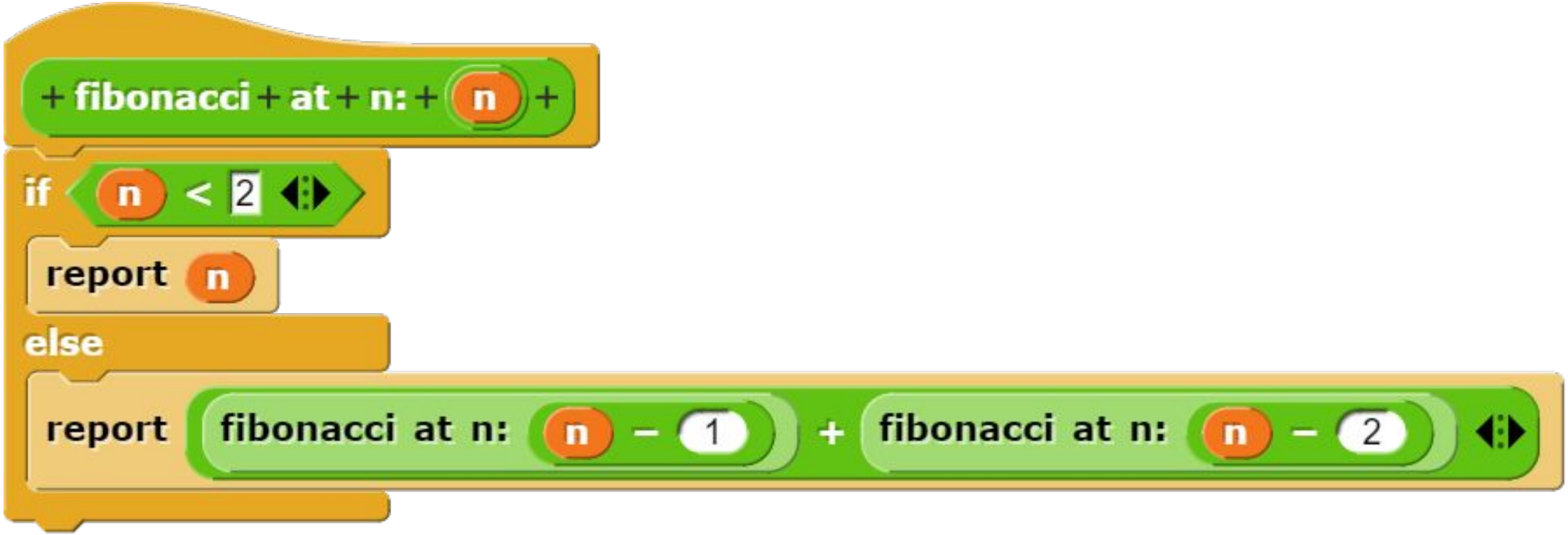
Tree Recursion Example

- What might be the base case?
- Why will we need to recursive calls?
- What do we put into the recursive calls / how are they different?

Tree Recursion Example

- What might be the base case? **n < 2**
- Why will we need to recursive calls? **Because it adds the sum of last two number**
- What do we put into the recursive calls / how are they different? **n - 1 and n - 2**

Tree Recursion Example

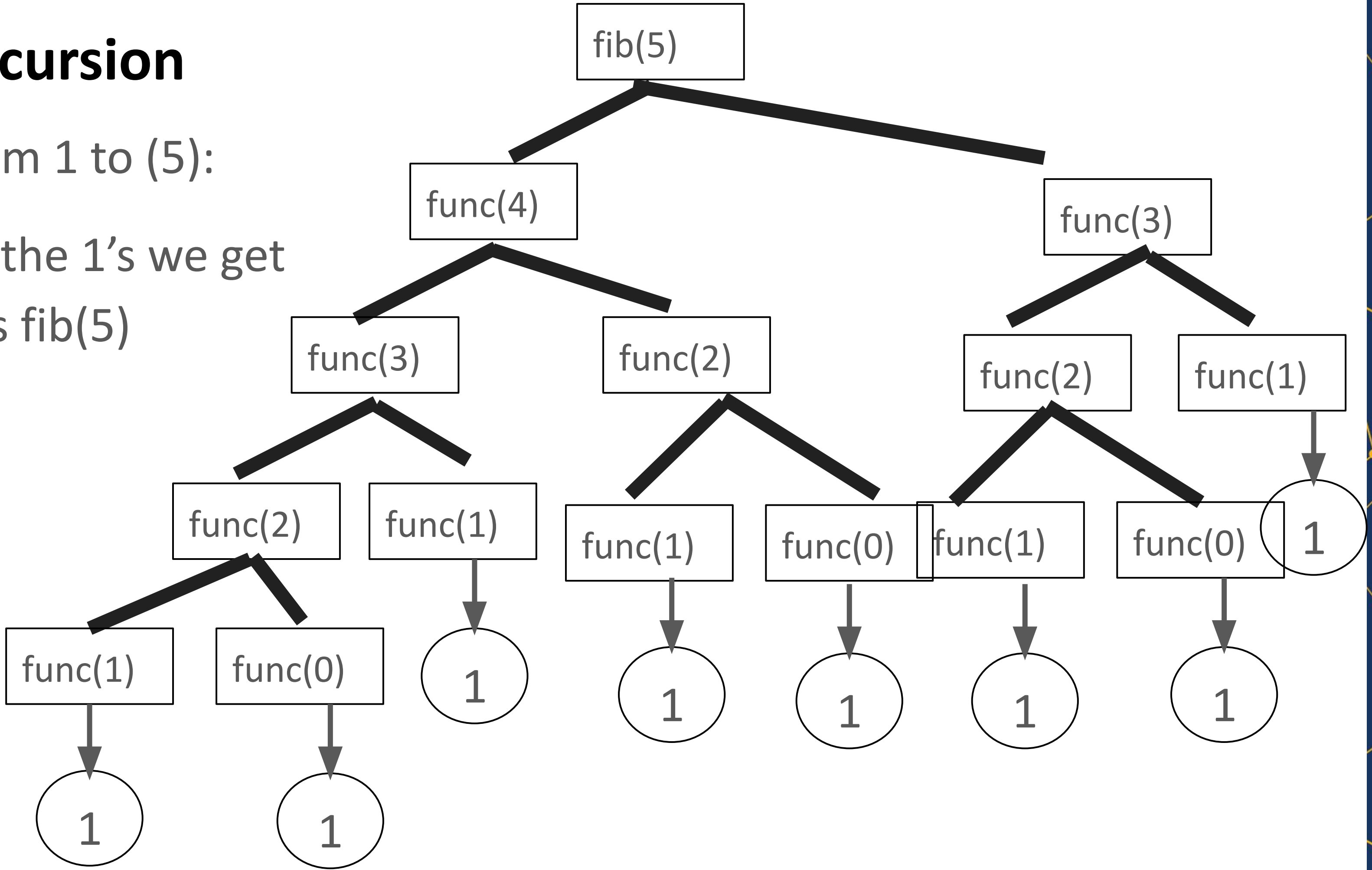


Tree Recursion

Calling sum 1 to (5):

Count all the 1's we get

8 which is fib(5)



Tree Recursion Example

- Objective: Write a function that takes in a value of money and a list of all the denominations. Your job is to make change / find all combinations of denominations we can make to create that value.
- Note: You can reuse numbers from the list infinitely!
- Note: If we create a combination that is larger than the value, we shouldn't count it.
- Output: A number - the number of combinations

- 4 ways to make change for 10:
    - 1 ten | 1 five and 5 ones | 2 fives | 10 ones

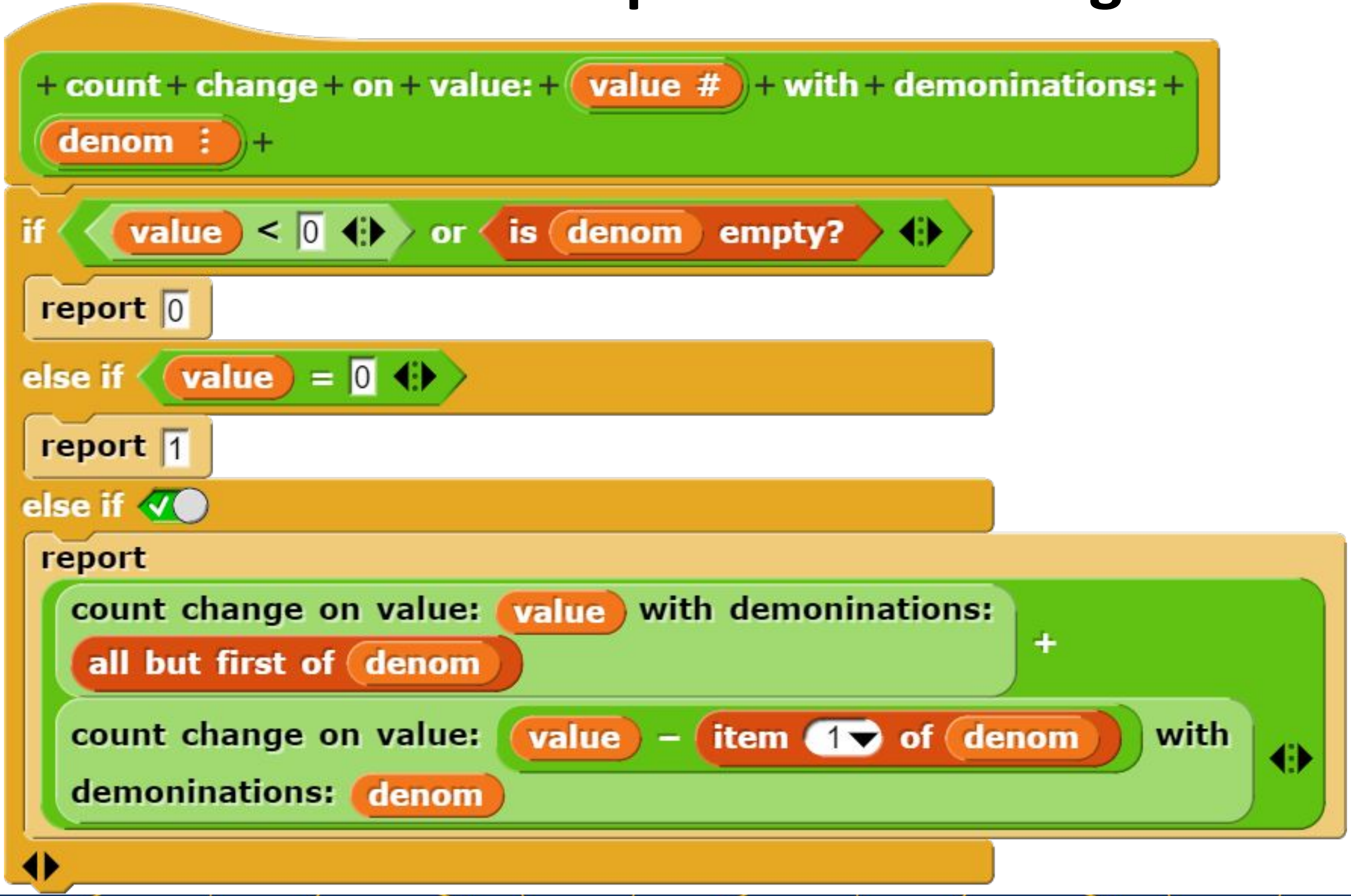
Tree Recursion Example: Count Change

- What are the base cases? Should we have more than one?
- When do we want to count the combination, and when do we not want to count it?
- What are the recursive cases?

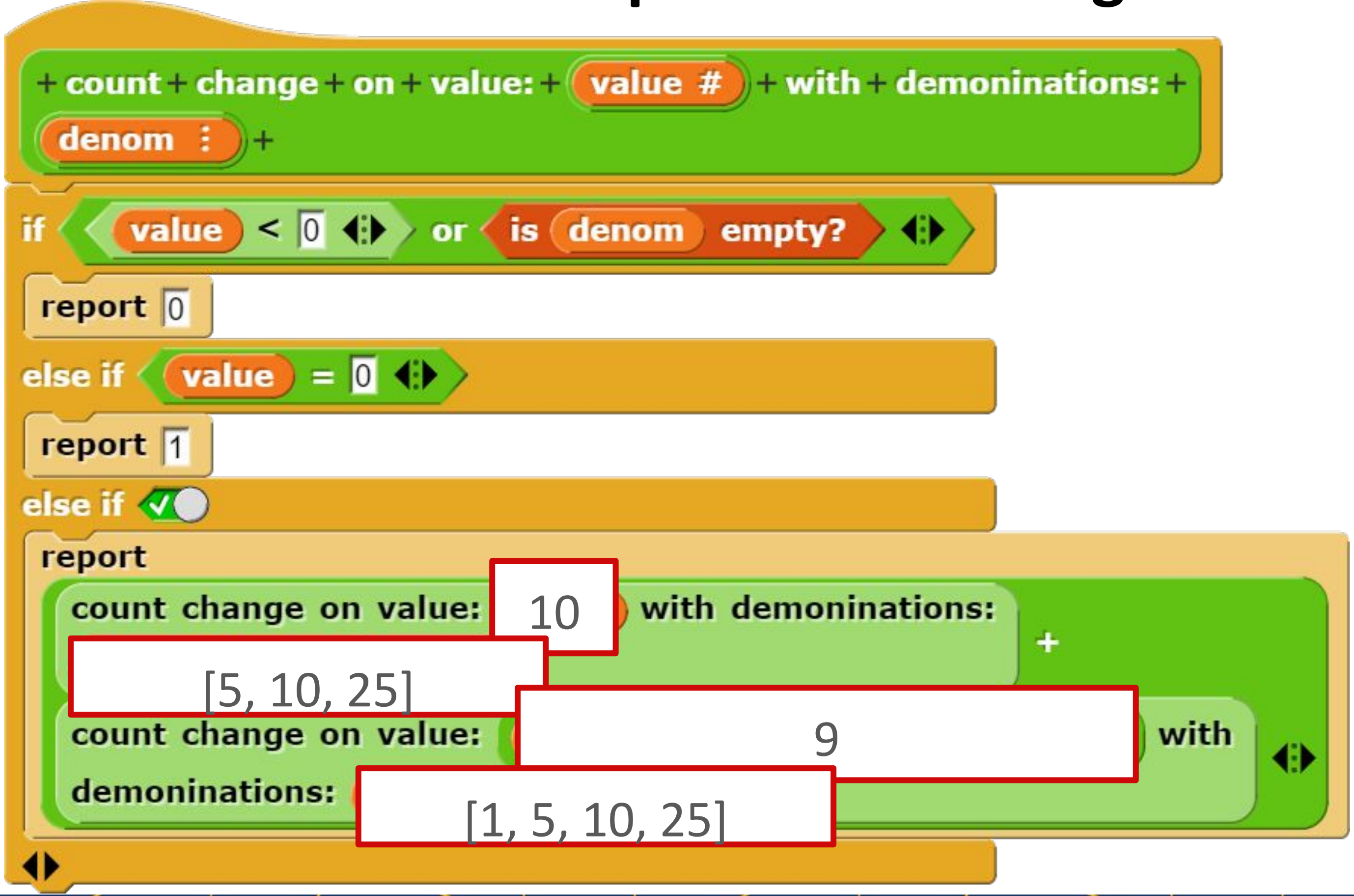
Tree Recursion Example: Count Change

- What are the base cases? Should we have more than one?
  - **There will be two cases. Value = 0 or our list of denominations is empty**
- When do we want to count the combination, and when do we not want to count it?
  - **Only if we successfully find a way to make change. So, we will keep subtracting a denomination from value. If value is = 0, then we successfully found a combo. If value < 0, we DON'T count it**
- What are the recursive cases?
  - **All but first of (denominations) and value - item 1 of denominations**

Tree Recursion Example: Count Change

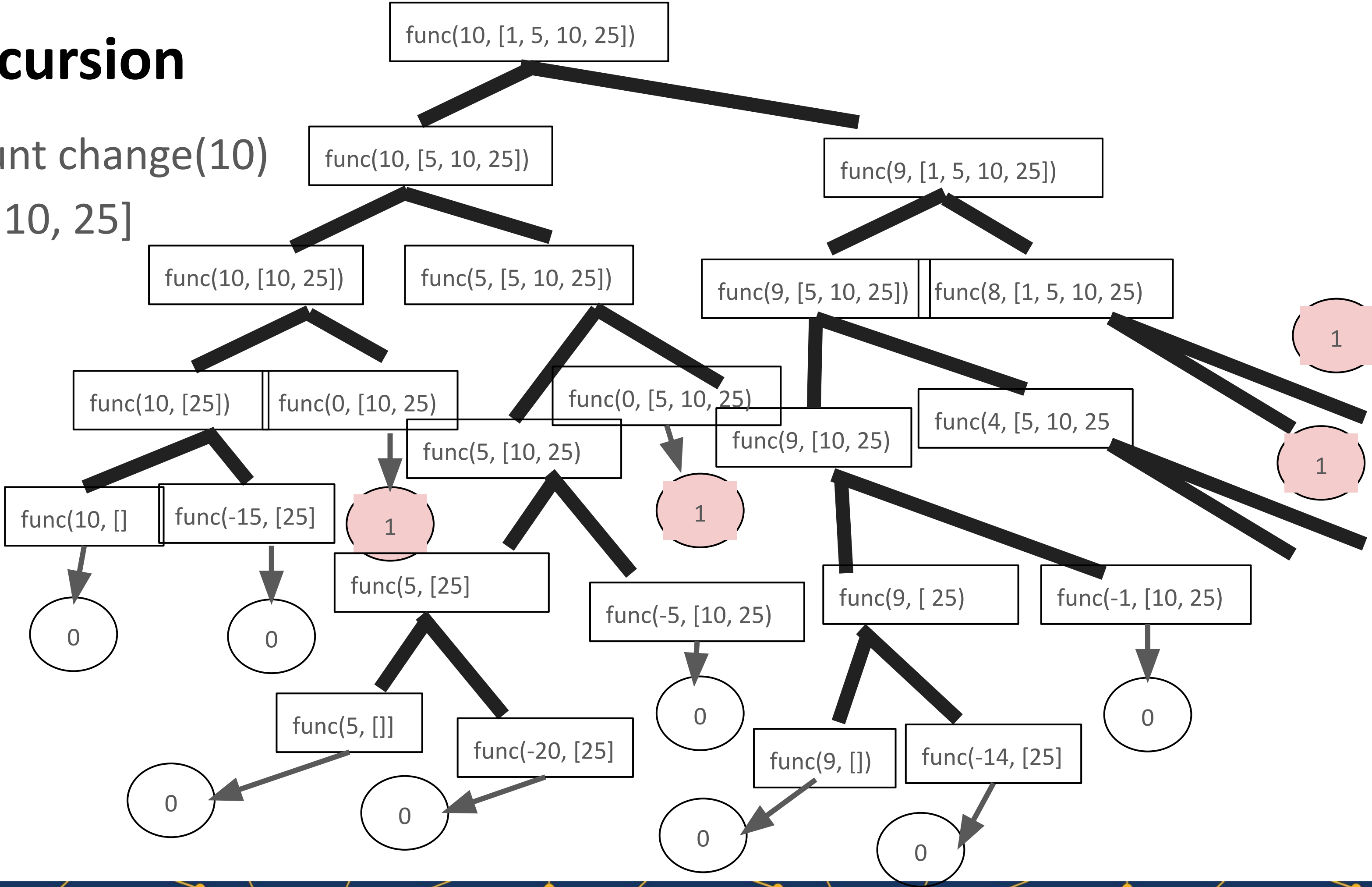


Tree Recursion Example: Count Change



Tree Recursion

Calling count change(10) with [1, 5, 10, 25]



Count Change

- Why do we need this part:

Count Change

- Why do we need this part:

- What would have happened if we had just called the same values on both recursive calls? Like this:

- We would only ever get 0 or 1 depending on the inputs!

Count Change with Incorrect Structure

- Count change (10) with [1, 5, 10, 25]
- Count change (9) with [5, 10, 25]
- Count change (4) with [10, 25]
- Count change (-6) with [25]
- Returns 0!

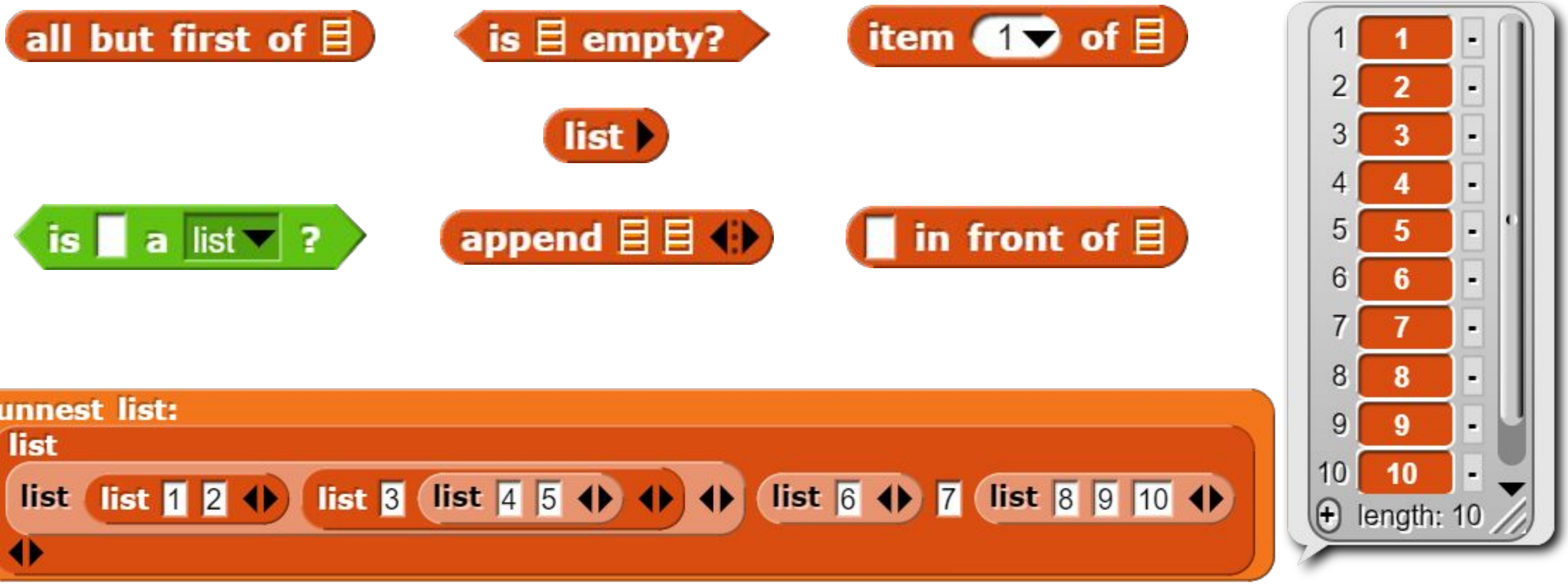
Count Change

Tree Recursion Example

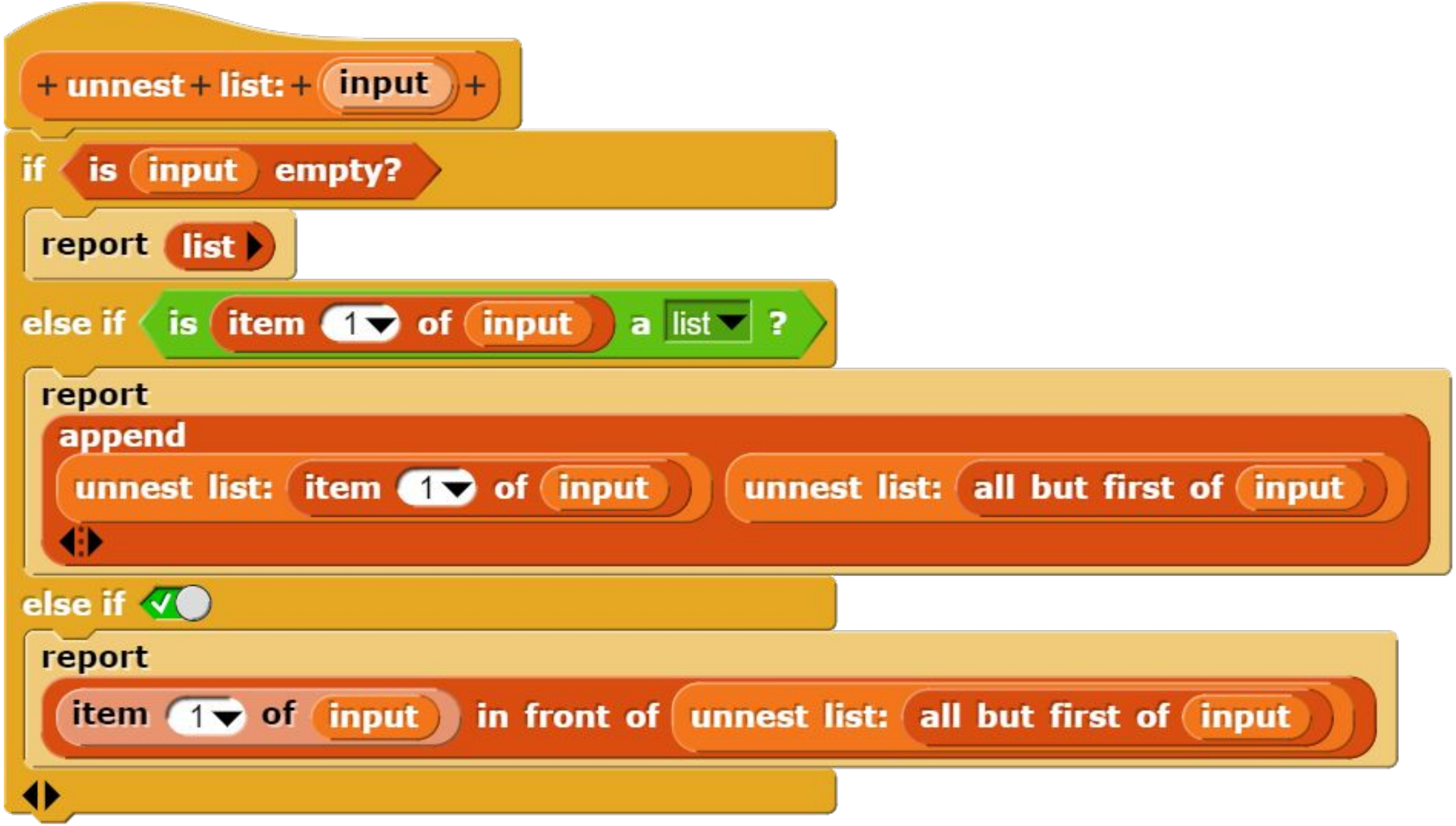
- Objective: Given a very nested list, write a function that unnests the list to become a 1D list.

Tree Recursion Example

- Objective: Given a very nested list, write a function that unnests the list to become a 1D list.

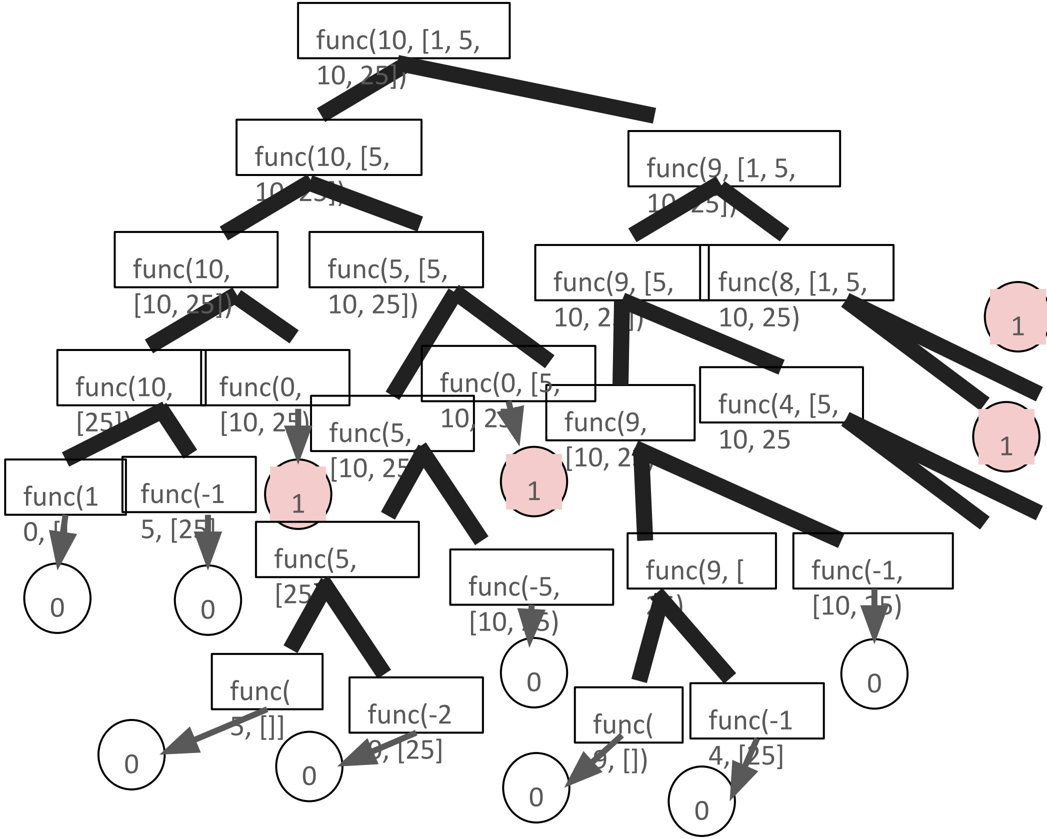


Tree Recursion Example



Trees

- Trees are a structures that allows us to represent our data.
- We can represent any data as a tree (which can be just deeply nested lists)
- When we need to grab a value of a tree, we will need to “traverse” the tree
  - We can do this more easily using recursion!



Trees

- They commonly have these attributes:
- Nodes:
  - Each element is a node
- Depths
  - How nested or long it is
- Parents:
  - Which nodes it stems from
- Children
  - Which nodes stem from it
- Root:
  - The first value on the top
- Leaves:
  - The bottom values that have no children