Unit 8: Recursion

Dave Abel

April 6th, 2016

Takeaway

Repeated self reference, or “recursion”, is *everywhere*, in the world and in computation! It’s simple, beautiful, and incredibly powerful.

Outline

- Definition
- Examples & Intuition
- Recursive Algorithms
- Recursive Searching and Sorting
- Recursion and Theory

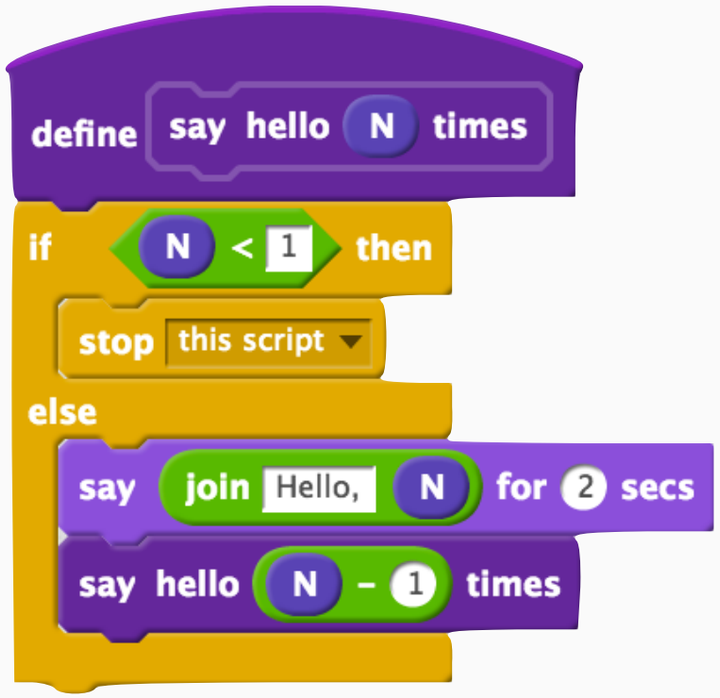
Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example one: **A Scratch block** is recursive if it *calls itself:*



Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example two: a tree!

Recursion

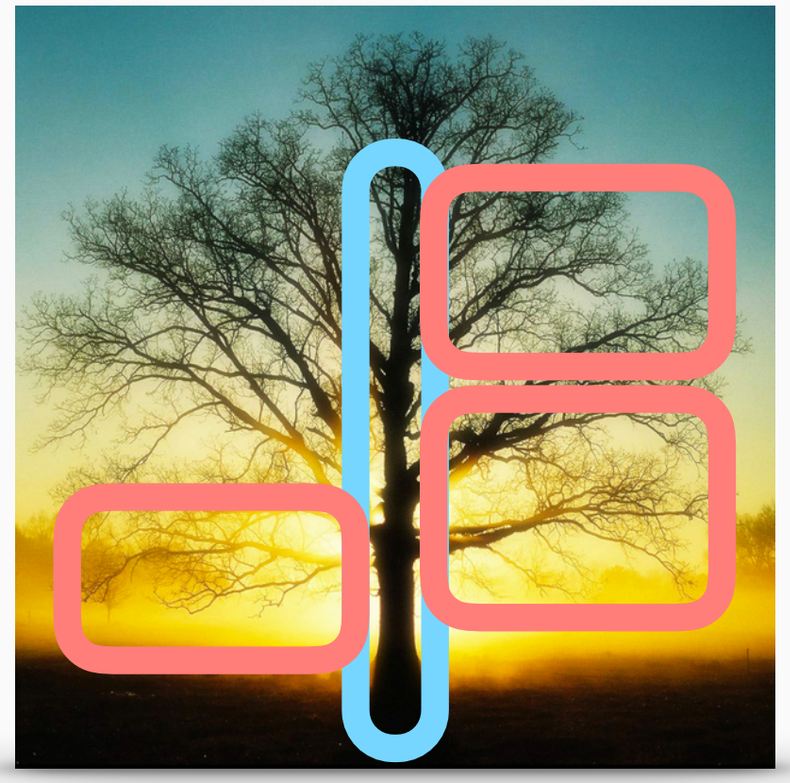
- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference
- Example two: a tree!

A tree is: a stick, with some number of trees coming off of it.

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example two: a tree!

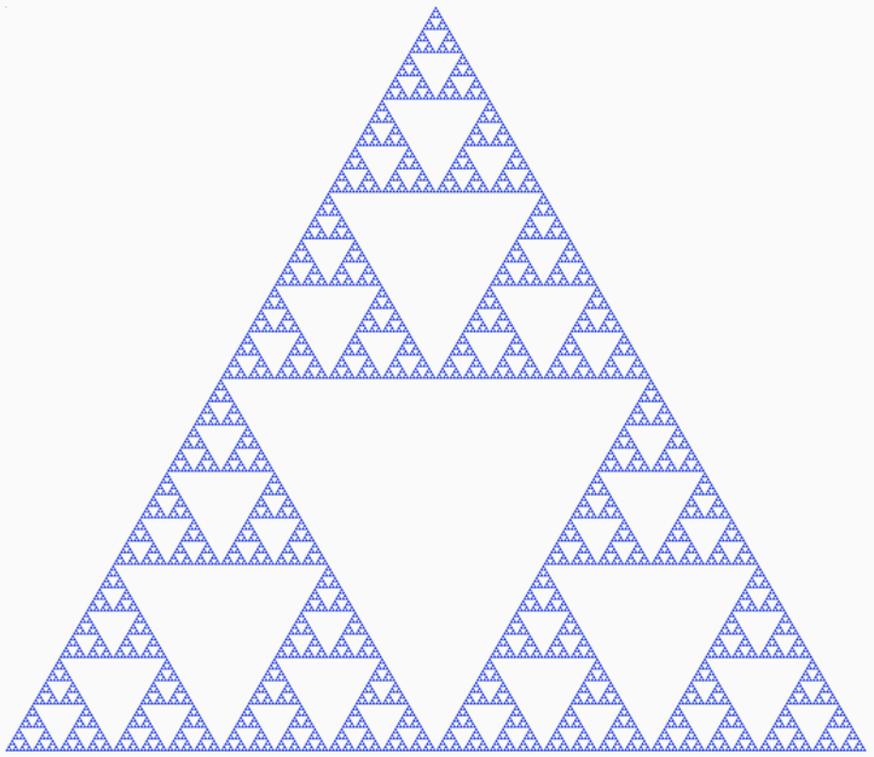


A tree is: a stick, with some number of trees coming off of it.

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

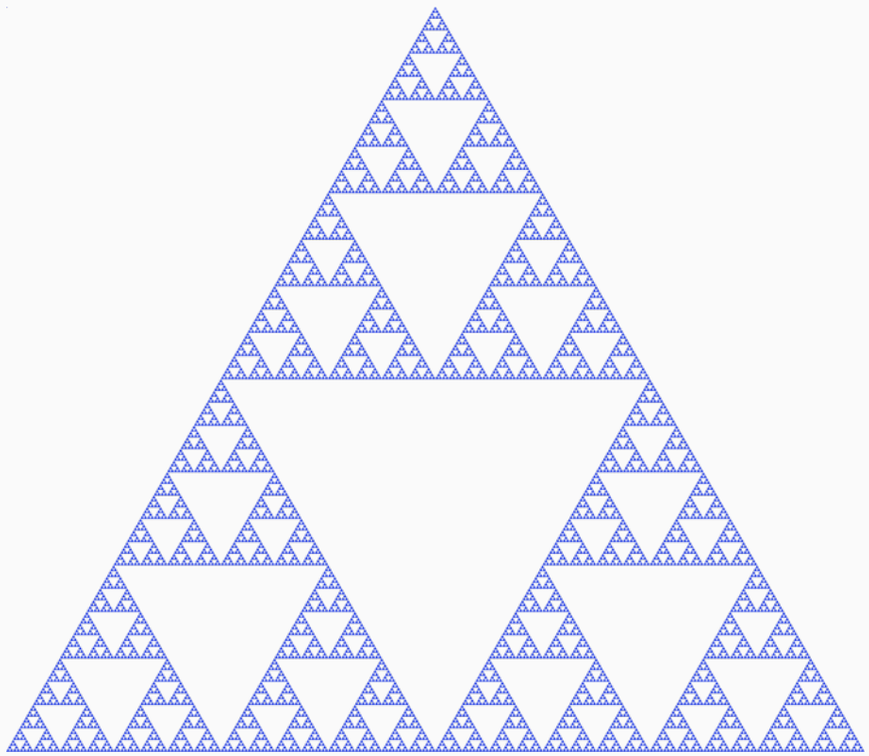
- Example three: **Recursive Shapes!**



Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example three: **Recursive Shapes!**



A recursive triangle is: a triangle, with a recursive triangle inside of it

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- In general, *recursive entities can be described as:*
  - A simple step
  - A recursive step

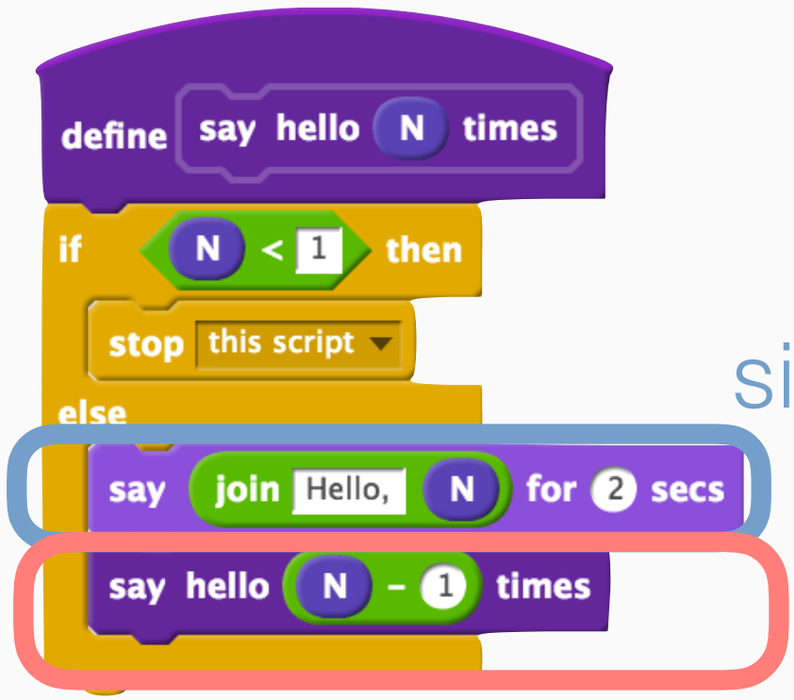
Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- In general, *recursive entities can be described as:*
  - A simple step
  - A recursive step

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference
- Example one: **A Scratch block** is recursive if it *calls itself*



simple step

recursive step

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference

- Example two: a tree!

A tree is: a stick, with some number of trees coming off of it.

Recursion

- **Definition:** a process, program, or object is said to be *recursive* if it involves repeated self-reference
- Many algorithms are *recursive!*
- Let’s look at a few.

Recursive Algorithms

- Problem: Is a word a palindrome?
  - *INPUT:* a word
  - *OUTPUT:* True if the word is a palindrome, False otherwise.
- Recursive solution:
  - A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.

Recursive Algorithms

- Problem: Is a word a palindrome?
  - *INPUT:* a word
  - *OUTPUT:* True if the word is a palindrome, False otherwise.
- Recursive solution:

Q: What's the simple step?
What's the recursive step?

  - A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.

Recursive Algorithms

- Problem: Is a word a palindrome?
  - *INPUT:* a word
  - *OUTPUT:* True if the word is a palindrome, False otherwise.
- Recursive solution:
  - A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.

Recursive Palindrome

- A word is a palindrome if: the outermost two letters are the same AND the remaining word is a palindrome.
  - This basically tells us a solution for solving the problem

Recursive Algorithms

- Problem: compute the length of a word
  - *INPUT:* A word
  - *OUTPUT:* The length of the word

Recursive Algorithms

- Problem: compute the length of a word
  - *INPUT:* A word
  - *OUTPUT:* The length of the word

Brainstorm a recursive solution with your neighbors!

Recursive Algorithms

- Problem: compute the length of a word
  - *INPUT:* A word
  - *OUTPUT:* The length of the word
- Here's my solution:
  - The length of a word is just 1, plus the length the word you get if you remove one character.

Recursive Algorithms

- Problem: Factorial
  - *INPUT:* A number
  - *OUTPUT:* The *factorial* of that number
  - Example: factorial(4) is 4*3*2*1, factorial(6) is 6*5*4*3*2*1

Recursive Algorithms

- Problem: Factorial
  - *INPUT:* A number
  - *OUTPUT:* The *factorial* of that number
  - Example: factorial(4) is 4*3*2*1, factorial(6) is 6*5*4*3*2*1

Brainstorm a recursive solution with your neighbors!

Recursive Algorithms

- Problem: Factorial
  - *INPUT:* A number
  - *OUTPUT:* The *factorial* of that number
- Observation: 4! = 4*3!, 3! = 3*2!, 2! = 2*1!, 1! = 1

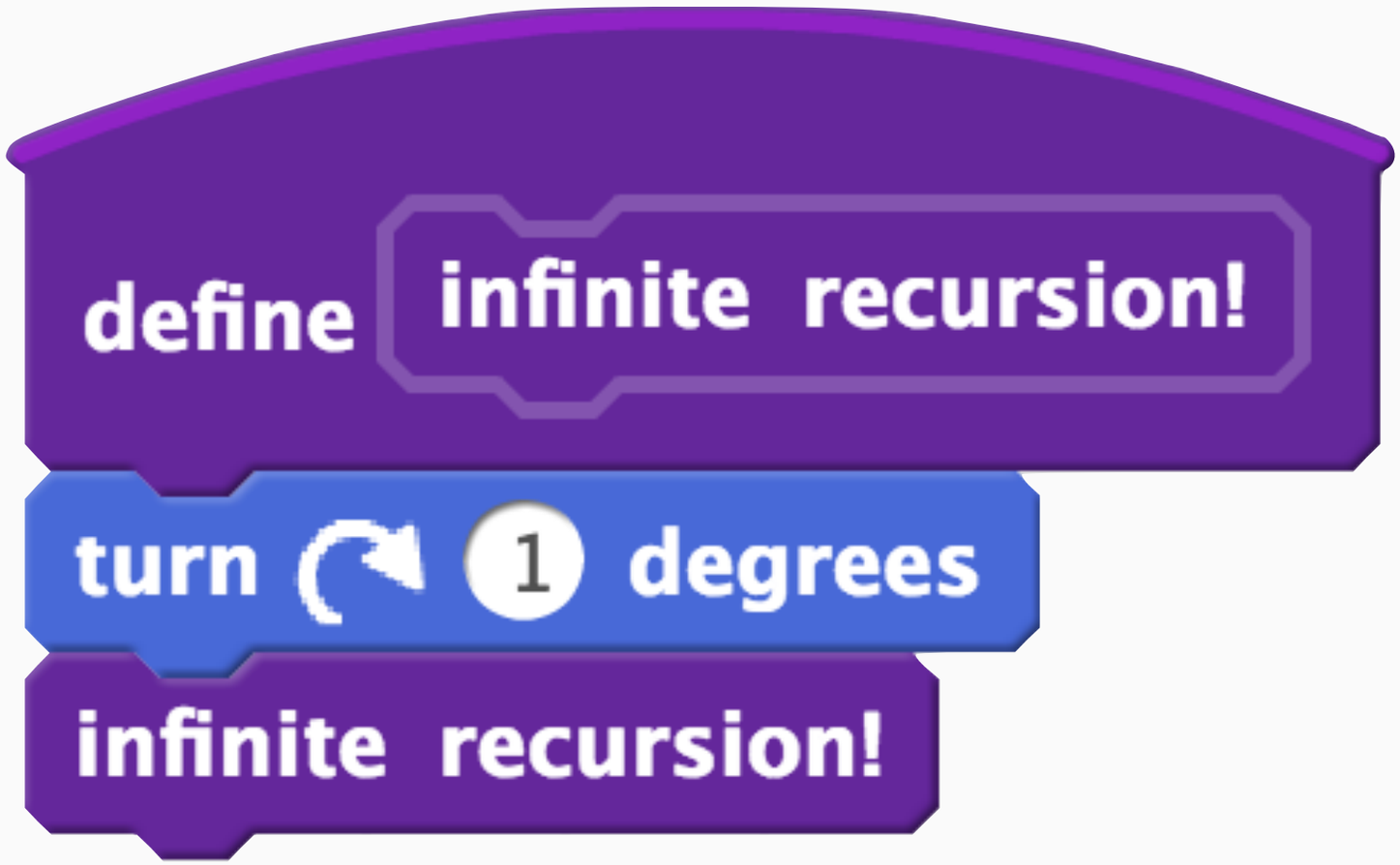
Recursive Algorithms

- Problem: Factorial
  - *INPUT:* A number
  - *OUTPUT:* The *factorial* of that number
- Observation: 4! = 4*3!, 3! = 3*2!, 2! = 2*1!, 1! = 1
- Here's my solution:
  - The factorial of a number is just that number times the factorial of one minus that number.

Simple step

Recursive step

Infinite Recursion



Recursive Algorithms

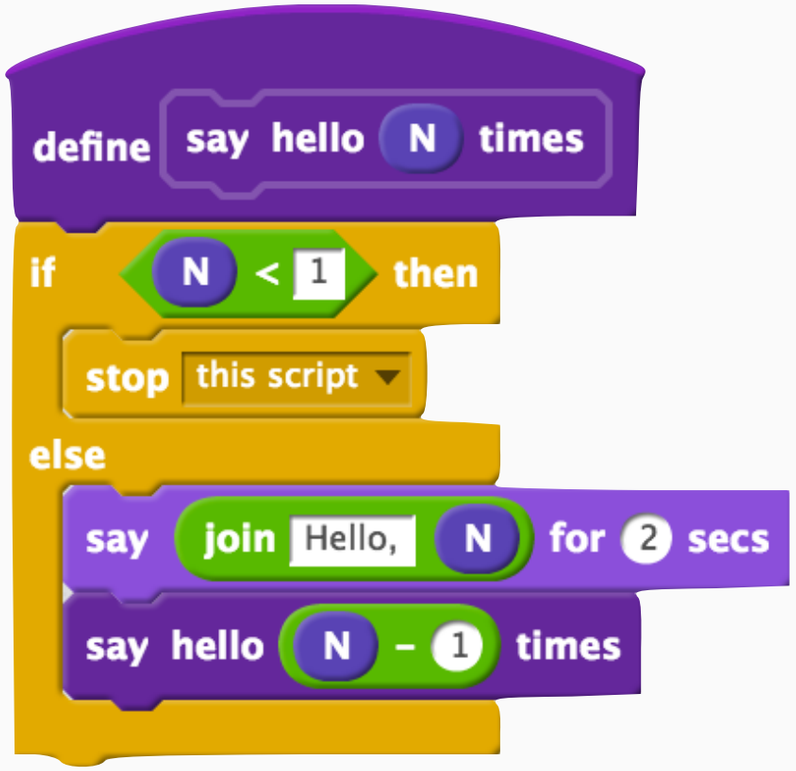
- Problem: compute the length of a word
  - *INPUT:* A word
  - *OUTPUT:* The length of the word
- Here's my solution:
  - The length of a word is just 1, plus the length the word you get if you remove one character.
  - **Critically we *need* tell the program how to stop.**

Recursion: Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.

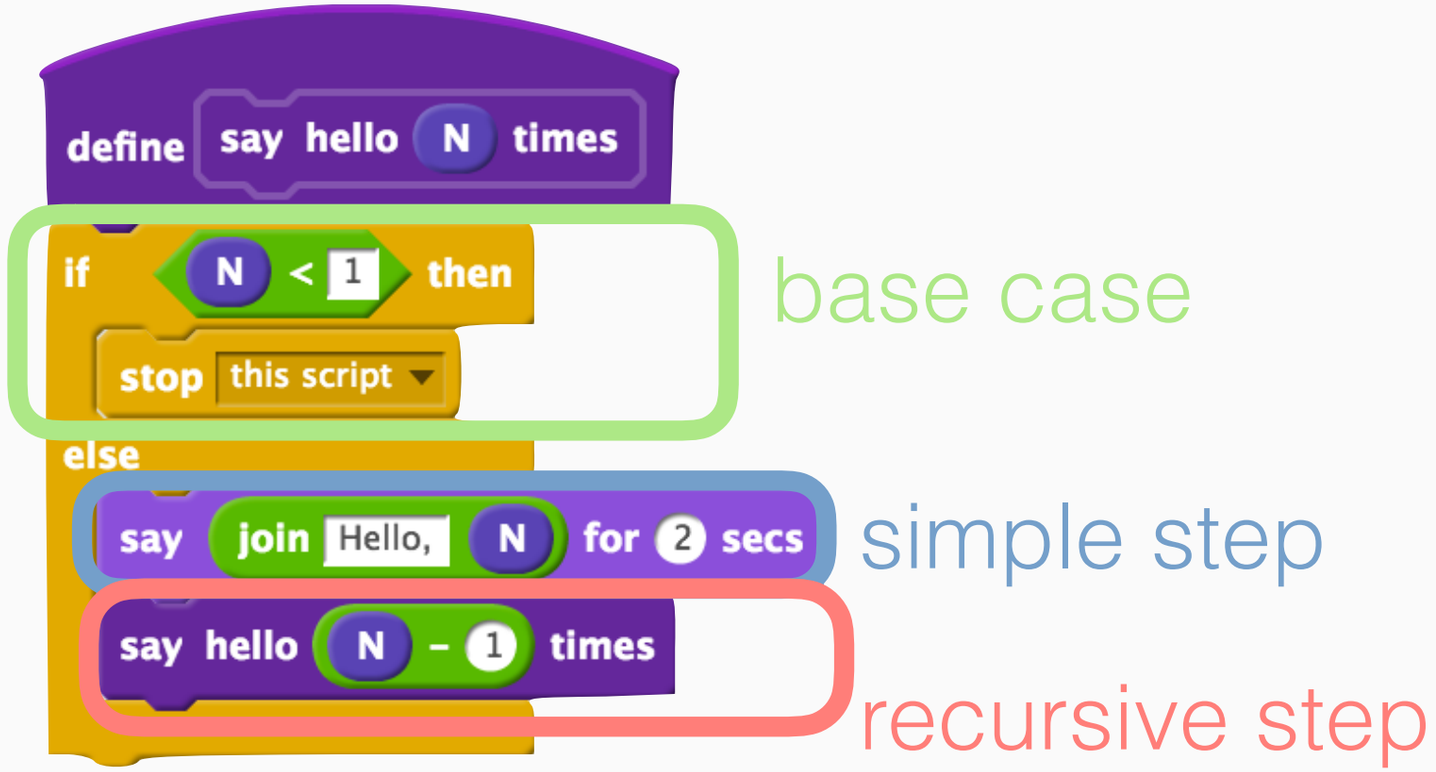
Recursion: Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.



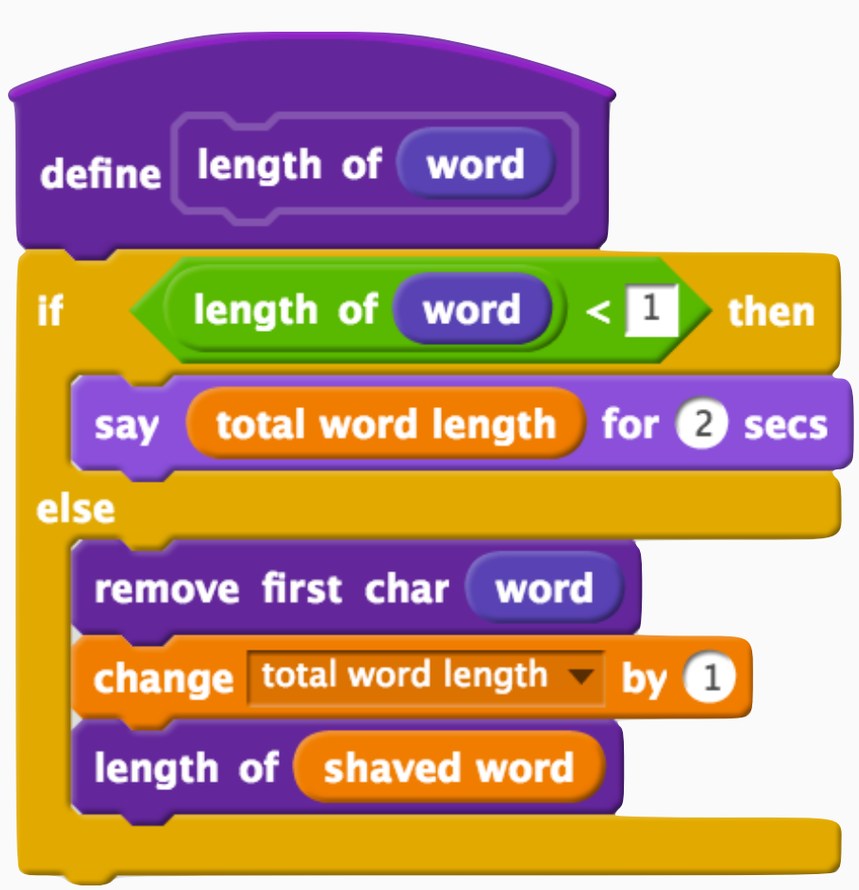
Recursion: Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.



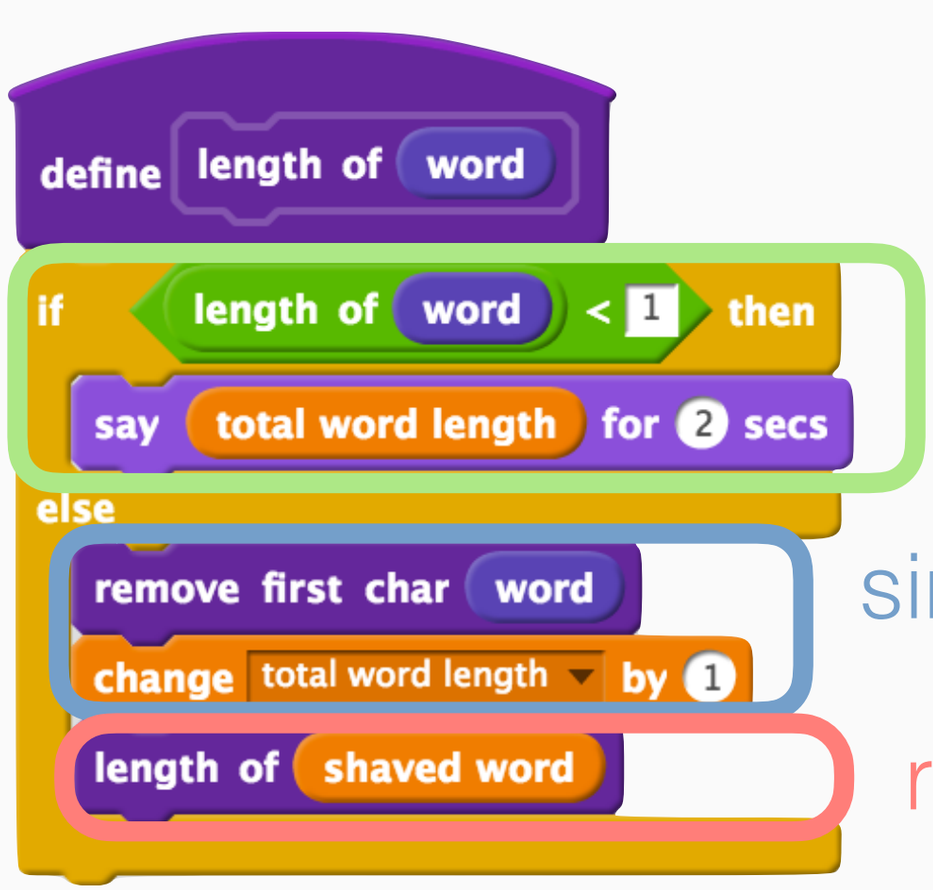
Recursion: Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.



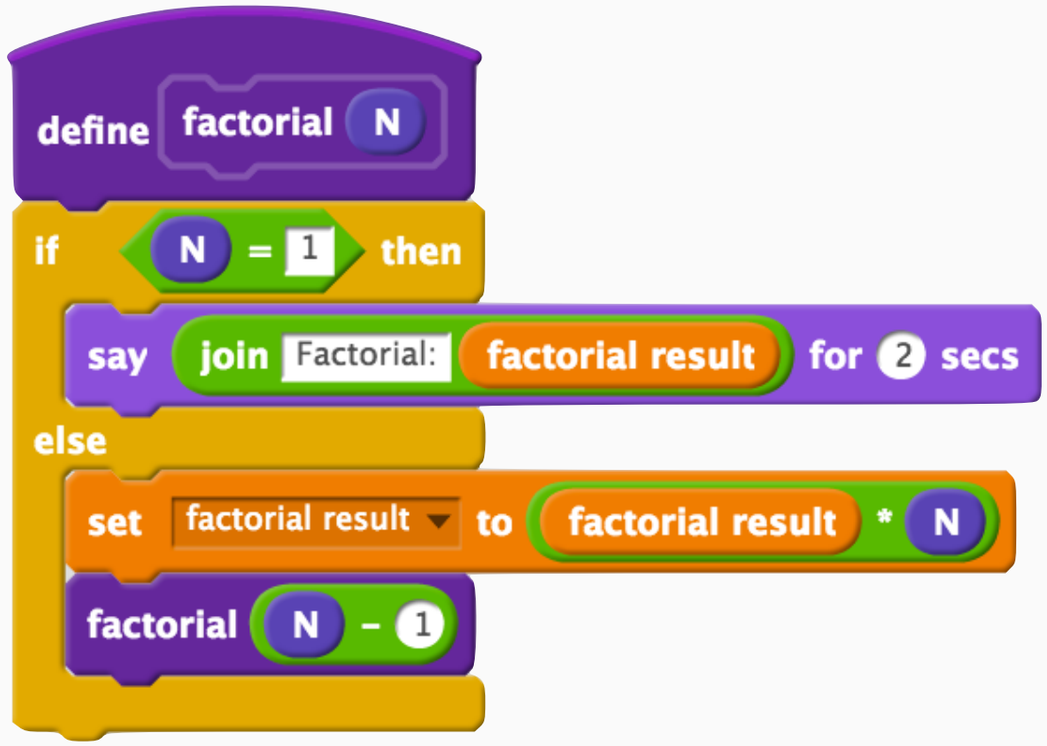
Recursion: Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.



Recursion: Base Case

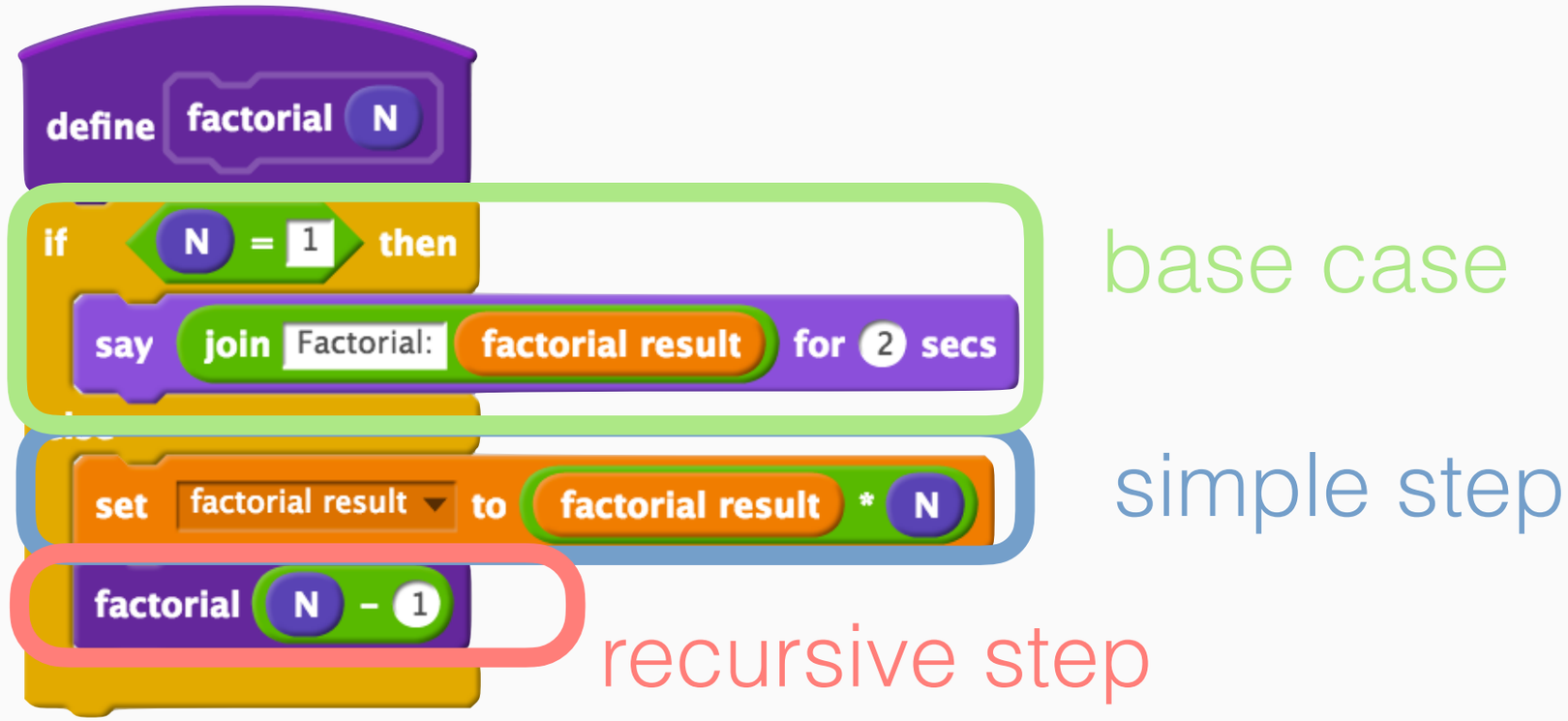
- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.



Discuss with your neighbor(s): what is the simple step? what is the recursive step? what is the base case?

Recursion: Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.

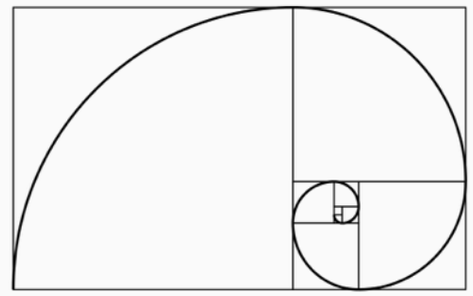


Discuss with your neighbor(s): what is the simple step? what is the recursive step? what is the base case?

Double Base Case

- Recursive Algorithms have a **base case**, which specifies when the algorithm should stop.

- Remember the **fibonacci** sequence?



  - Start with the sequence 1,1

  - To generate the next number in the sequence, add the two previous numbers!

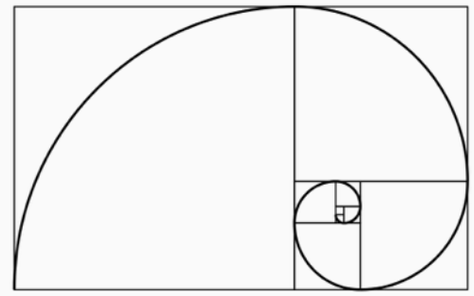
- So the next numbers are 2, then 3, then 5, etc.

Double Base Case

- **Consider the problem of writing the first N items of the fibonacci sequence.**

- **Q: What is the base case? Simple step? Recursive step?**

- Remember the **fibonacci** sequence?



  - Start with the sequence 1,1

  - To generate the next number in the sequence, add the two previous numbers!

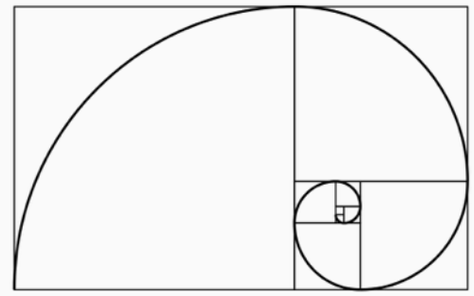
  - Generate the next N-1 numbers.

Double Base Case

- **Consider the problem of writing the first N items of the fibonacci sequence.**

- **Q: What is the base case? Simple step? Recursive step?**

- Remember the **fibonacci** sequence?
  - Start with the sequence 1,1
  - To generate the next number in the sequence, add the two previous numbers!
  - Generate the next N-1 Numbers



base cases

simple step

recursive step

Problem Spec: Fibonacci

- INPUT: A number, N
- OUTPUT: The first N numbers of the Fibonacci Sequence.

Problem Spec: Fibonacci

- INPUT: A number, N
- OUTPUT: The first N numbers of the Fibonacci Sequence.
- Math form: f(n) = f(n-1) + f(n-2), plus our base cases. Otherwise n goes off to negative infinity!

Problem Spec: Fibonacci

- INPUT: A number, N
- OUTPUT: The first N numbers of the Fibonacci Sequence.
- Math form: f(n) = f(n-1) + f(n-2), plus our base cases. Otherwise n goes off to negative infinity!
- In Scratch

Recursion: Recap

- **Definition**: a process, program, or object is said to be *recursive* if it involves repeated self-referenceSub bullet one

- In general, *recursive entities can be described as:*
  - **A simple step**
  - **A recursive step**
- Recursion can be *infinite*
- For recursion to be *finite*, we need a **base case.**