

LEC 08

CSE 123

Recursion

Questions during Class?

Raise hand or send here

sli.do #cse123A

BEFORE WE START

Talk to your neighbors:

Hot weather or cold weather?

Instructors: Nathan Brunelle

	Arohan	Ashar	Neha	Rohini	Rushil
TAs:	Ido	Zachary	Sebastian	Joshua	Sean
	Hayden	Caleb	Justin	Heon	Rashad
	Srihari	Benoit	Derek	Chris	Bhaumik
	Kuhu	Kavya	Cynthia	Shreya	Ashley
	Ziao	Kieran	Marcus	Crystal	Eeshani
	Prakshi	Packard	Cora	Dixon	Nichole
	Niyati	Trien	Lawrence	Evan	Cady

Lecture Outline

- **Announcements** 
- Recursion review
- Recursive Tracing Practice

Announcements

- Quiz 1 Tuesday (May 6)!
 - Topics: ListNodes, LinkedIntList, Runtime analysis
 - Topics not on Quiz: Recursion
 - Practice quiz and reference sheet for Quiz 1 will be posted soon!
 - Quiz 0 feedback will be released before Quiz 1
- Programming Assignment 1 due in one week (May 7) @ 11:59pm
- Creative Project 1 and Resubmission Cycle 1 grades released today
- Resubmission Period 3 closes this Friday (May 2) @ 11:59pm
 - Available assignments: **C0**, P0, C1
 - ***Last opportunity to resubmit C0***

Lecture Outline

- Announcements
- **Recursion review** ◀
- Recursive Tracing Practice

Recursion (1)

“The repeated application of a recursive procedure or definition”

- Oxford Languages

- Real-world definition: defining a problem in terms of itself
 - Case in point: above definition
 - Further natural examples:



Recursion (2)

“The repeated application of a recursive procedure or definition”

- Oxford Languages

- Real-world definition: defining a problem in terms of itself
 - Case in point: above definition
 - Further natural examples:



Recursion (3)

“The repeated application of a recursive procedure or definition”

- Oxford Languages

- Real-world definition: defining a problem in terms of itself
 - Case in point: above definition
- Computer science definition: when a method calls itself
 - “Alternative” to iteration (can combine for powerful results)
- 🤔 Wouldn't that just lead to an infinite loop?
 - Yes! If you do things wrong...
 - Let's review *how* method calls work

Recursion (4)

“The repeated application of a recursive procedure or definition”

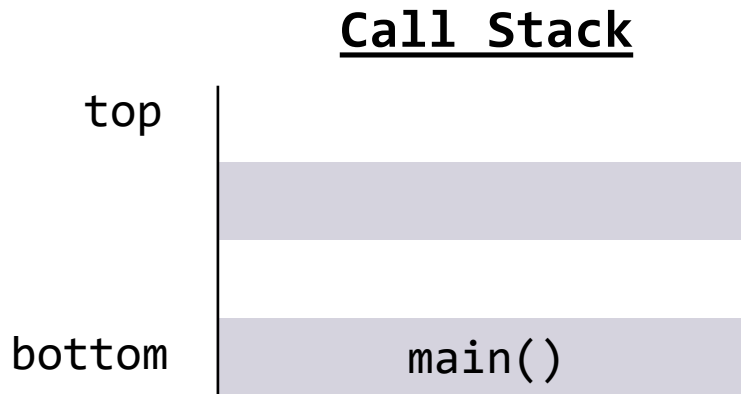
- Oxford Languages

- Why do we care?
 - It's a fundamental Computer Science topic
 - Some things are easier to describe with recursion than with loops
 - There are some problems that lend themselves more easily to a recursive solution than an iterative (loop-y) solution
 - Though all problems that can be solved with recursion can be solved with loops, and vice versa

Method Calls

- Regardless how you use them, methods work the same way!
 - Pause execution, finish method, return where you left off
- How does Java keep track of the prior method?
 - Sort of like the "history" of the method calls it has paused... (LIFO)
 - Something called the **Call Stack**

Call Stack



Console:

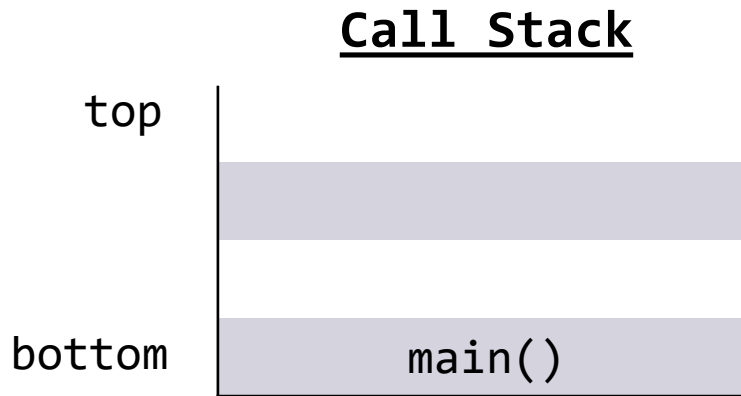
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

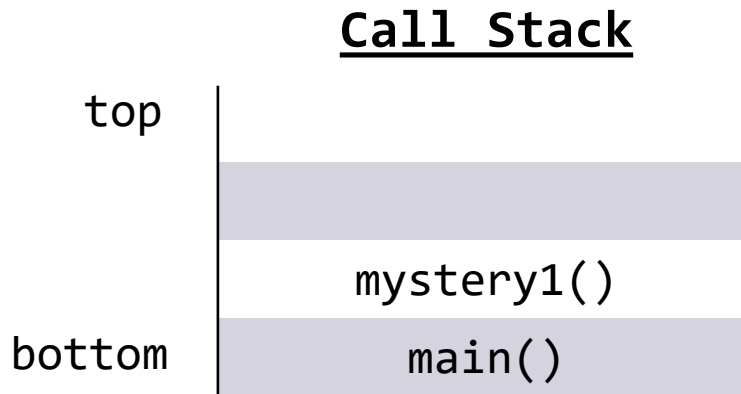
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

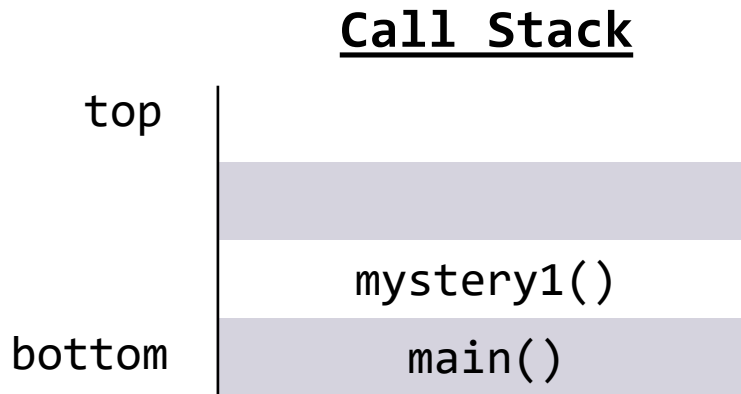
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One

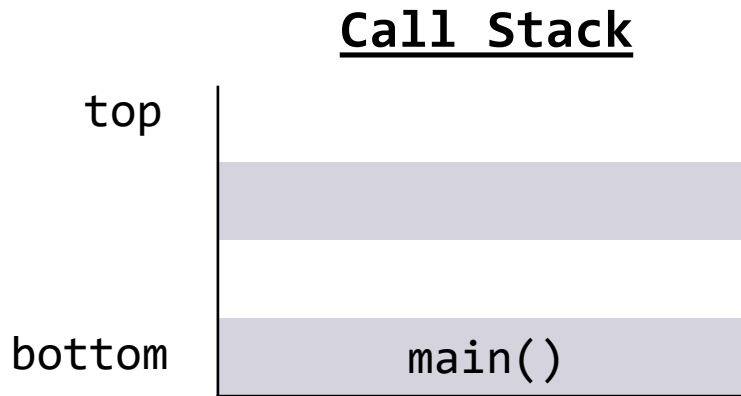
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One

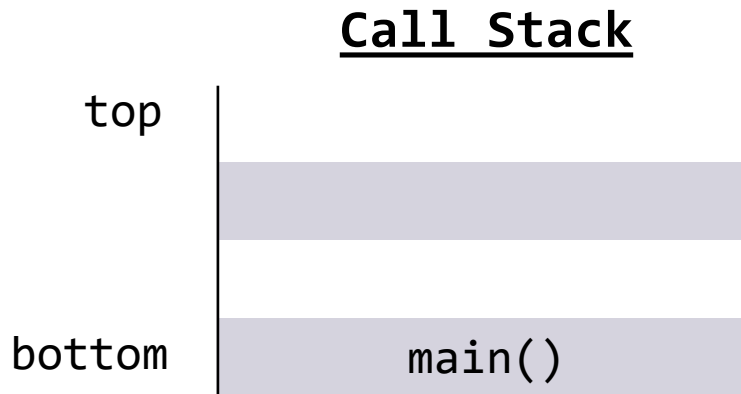
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One

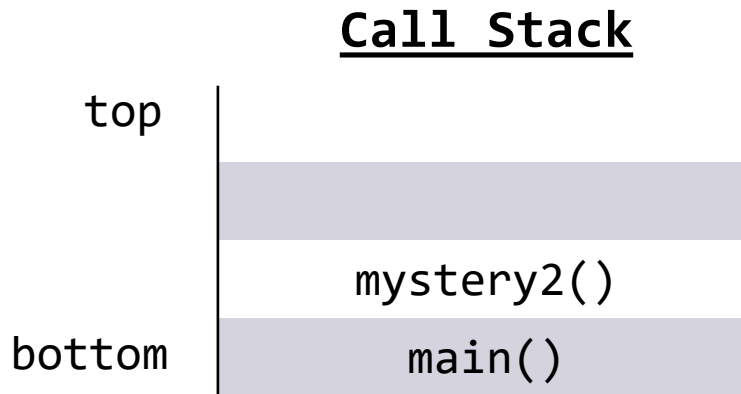
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```


Call Stack



Console:

One

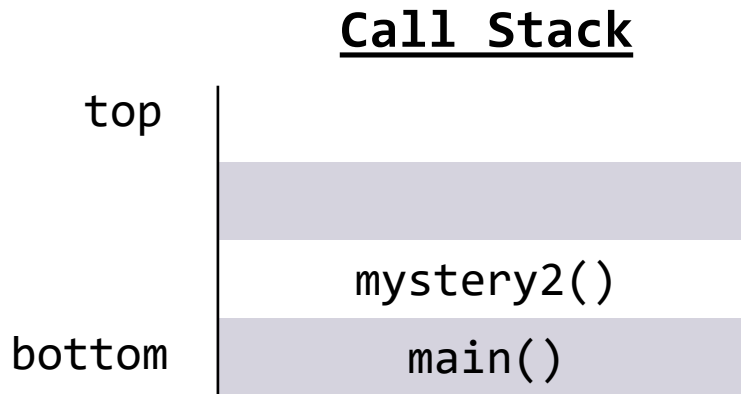
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two

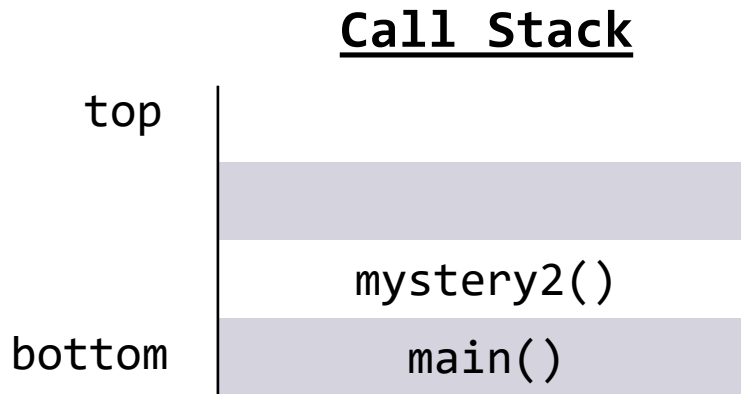
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two

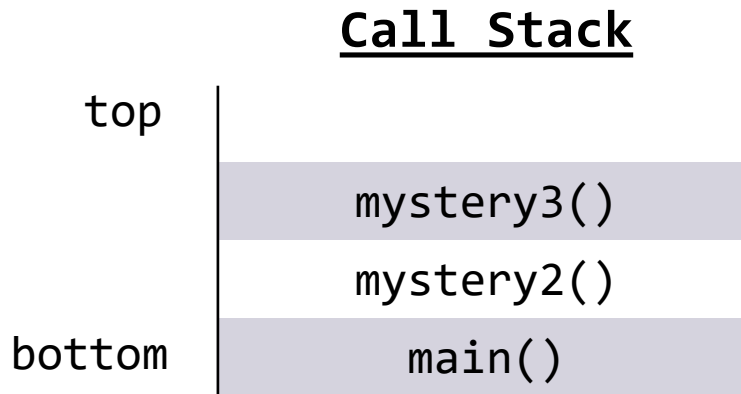
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two

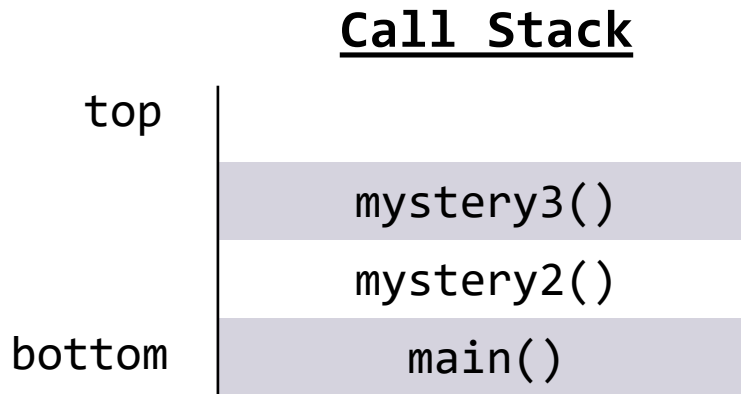
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two
Three

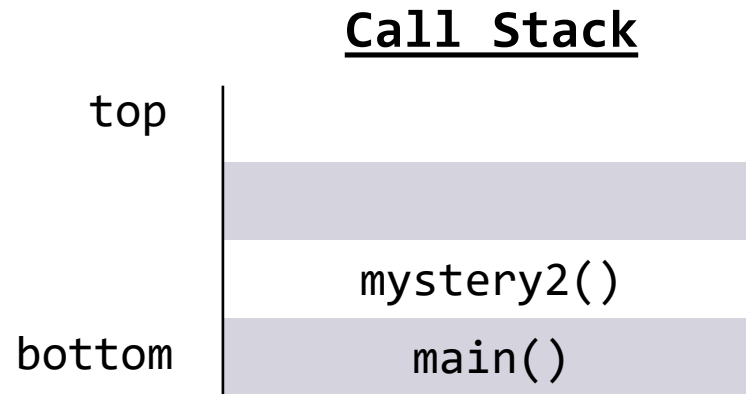
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two
Three

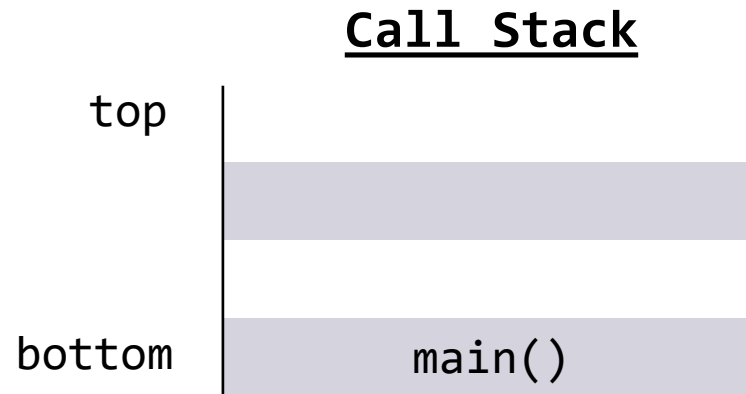
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two
Three

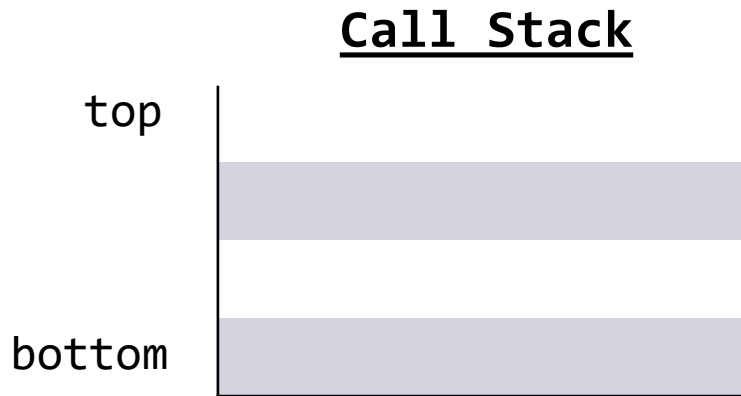
```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```

Call Stack



Console:

One
Two
Three

```
public static void main(String[] args) {  
    mystery1();  
    mystery2();  
}
```

```
public static void mystery1() {  
    System.out.println("One");  
}
```

```
public static void mystery2() {  
    System.out.println("Two");  
    mystery3();  
}
```

```
public static void mystery3() {  
    System.out.println("Three");  
}
```


Method Calls



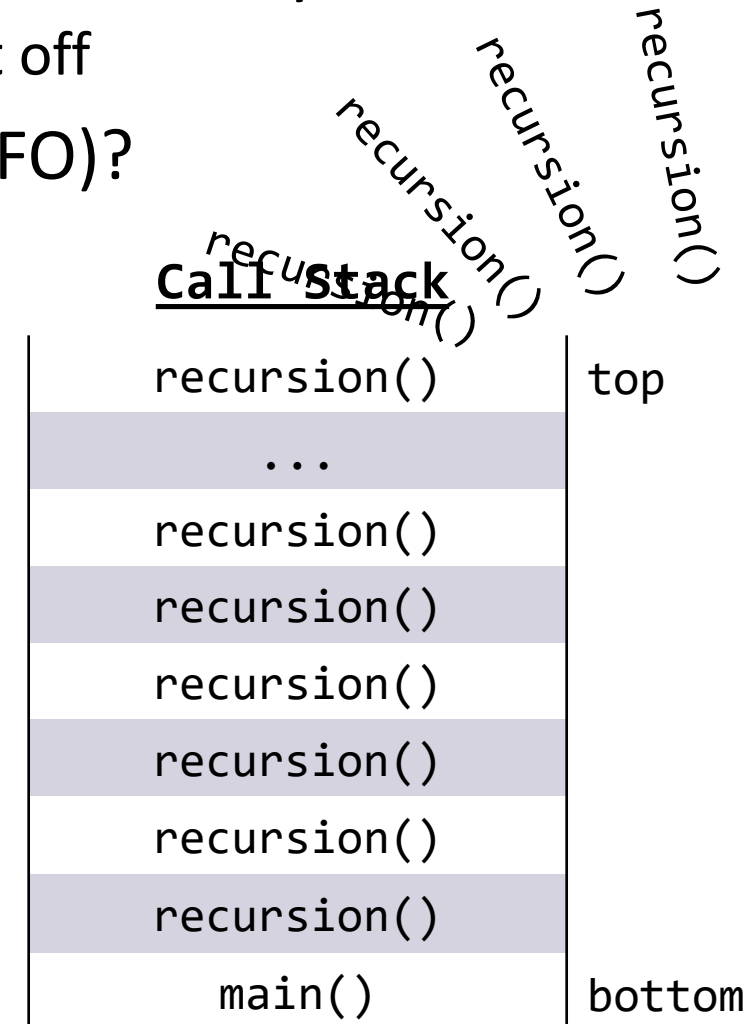
- Regardless how you use them, methods work the same way!
 - Pause execution, finish method, return where you left off
- How does Java keep track of the prior method (LIFO)?
 - Something called the **Call Stack**



Wouldn't that just lead to an infinite loop?

```
public static void recursion() {  
    System.out.println("Woah");  
    recursion();  
}
```

StackOverflowException!



Method Calls

- Regardless how you use them, methods work the same way!
 - Pause execution, finish method, return where you left off
- How does Java keep track of the prior method (LIFO)?
 - Something called the **Call Stack**



Wouldn't that just lead to an infinite loop?

- Yes! We get something called a **StackOverflowException**
- How do we avoid infinite recursion?

Recursive Methods

- 2 components of every recursive method:
- Recursive case
 - Decompose problem into subproblem
 - Make the actual recursive call
 - Combine results meaningfully
- Base case
 - Simplest version of the problem
 - No subproblems to break into
 - Return known answer



Recursive Methods

- 2 components of every recursive method:
- Recursive case
 - Decompose problem into subproblem
 - Make the actual recursive call
 - Combine results meaningfully
- Base case
 - Simplest version of the problem
 - No subproblems to break into
 - Return known answer



If decomposing moves you closer to the base, no infinite recursion!

Math Examples

$n!$ or $\text{factorial}(n)$ = product of all positive integers $\leq n$

- Two ways of viewing this idea:
- $n! = n * (n - 1) * (n - 2) * \dots * 2 * 1$
 - Iterative approach - loop through all values and multiply together
- $n! = n * (n - 1)!$
 - Recursive approach – decompose into subproblem and combine
 - What would our base case / simplest input (n) be?

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2!$$



$$2! = 2 * 1!$$



$$1! = 1 * 0!$$

$$0! = 1$$

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2!$$



$$2! = 2 * 1!$$



$$1! = 1 * 0!$$



$$0! = 1$$

Math Examples

$n! / \text{factorial}(n) = \text{product of all positive integers } \leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2!$$



$$2! = 2 * 1!$$



$$1! = 1 * 1$$

Math Examples

$n! / \text{factorial}(n) = \text{product of all positive integers } \leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2!$$



$$2! = 2 * 1!$$



$$1! = 1$$

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2!$$



$$2! = 2 * 1$$

Math Examples

$n! / \text{factorial}(n) = \text{product of all positive integers } \leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2!$$



$$2! = 2$$

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 3 * 2$$

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 3!$$



$$3! = 6$$

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 4 * 6$$

Math Examples

$n!$ / factorial(n) = product of all positive integers $\leq n$

- Reminder, recursive approach: $n! = n * (n - 1)!$
 - Let's trace through with a simple example 3!



$$4! = 24$$

