



# Methods

Chris Piech

CS106A, Stanford University

This is Method Man. He is part of the Wu Tang Clan. ☺

*Civilization advances by extending the number of operations we can perform without thinking about them.*

-Alfred North Whitehead



# Learn How To:

1. Write a method that takes in input
2. Write a method that gives back output
3. Trace method calls using stacks



# Calling Methods

```
turnRight();  
  
move();      readInt("Int please! ");  
  
println("hello world");  
  
rect.setFilled(true);  
  
drawRobotFace();  
  
add(rect);  
  
preventGlobalWarming();
```



# Defining a Method

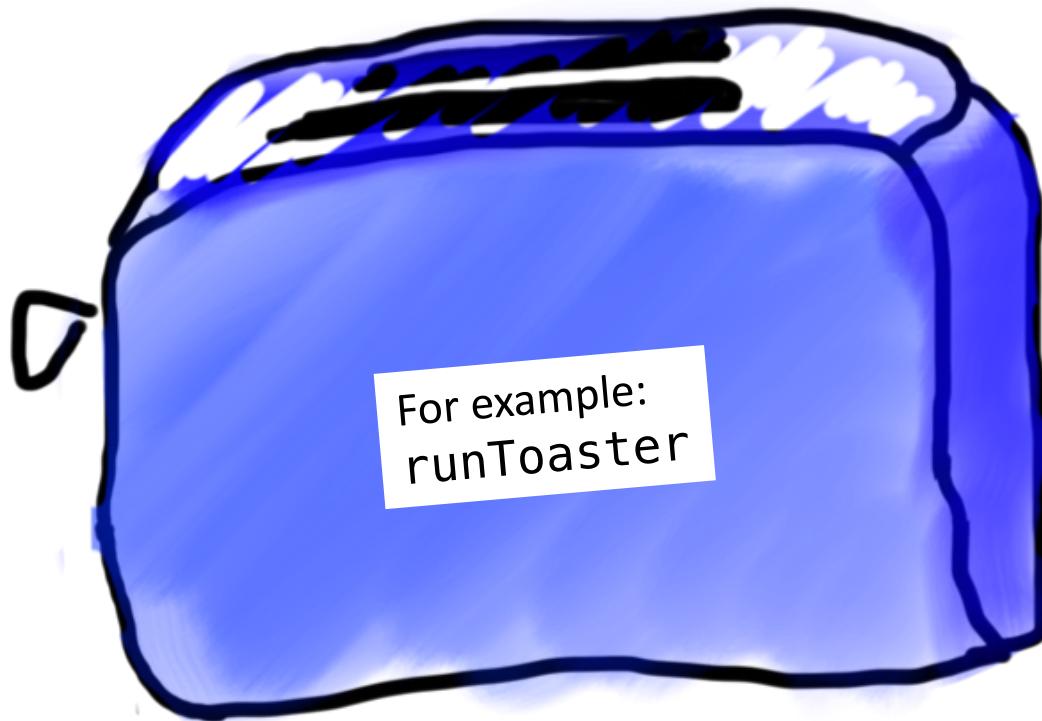
```
private void turnRight() {  
    turnLeft();  
    turnLeft();  
    turnLeft();  
}
```



Big difference with Java methods:  
Java methods can **take in data**, and can **return data!**



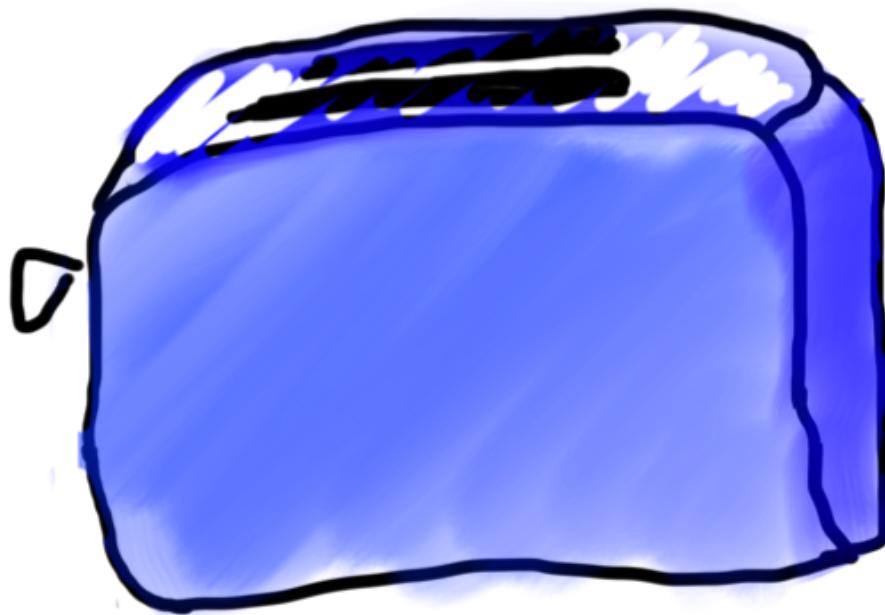
# Toasters are Methods



# Toasters are Methods



parameter



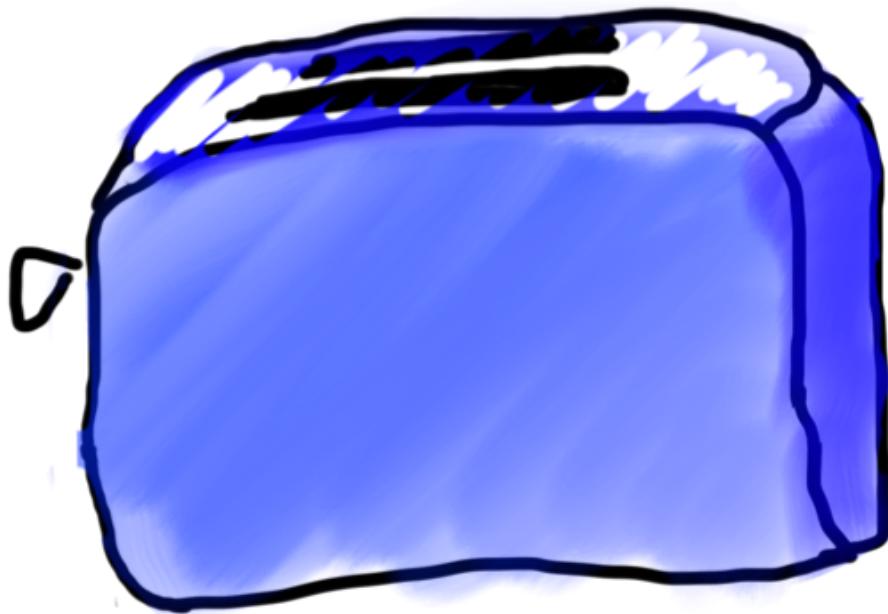
# Toasters are Methods



parameter



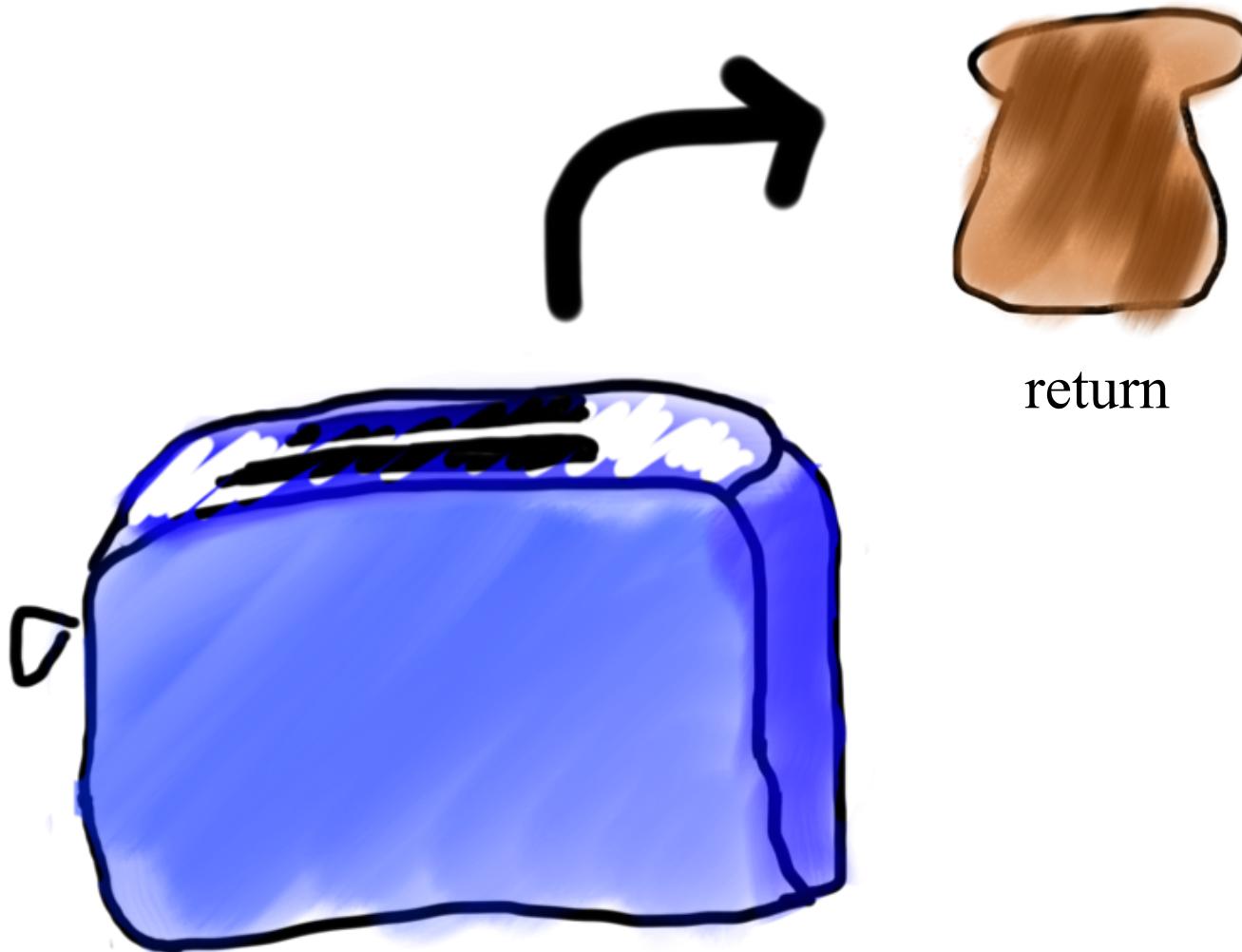
# Toasters are Methods



# Toasters are Methods



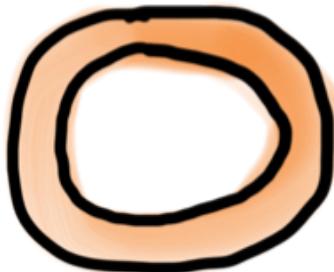
# Toasters are Methods



# Toasters are Methods



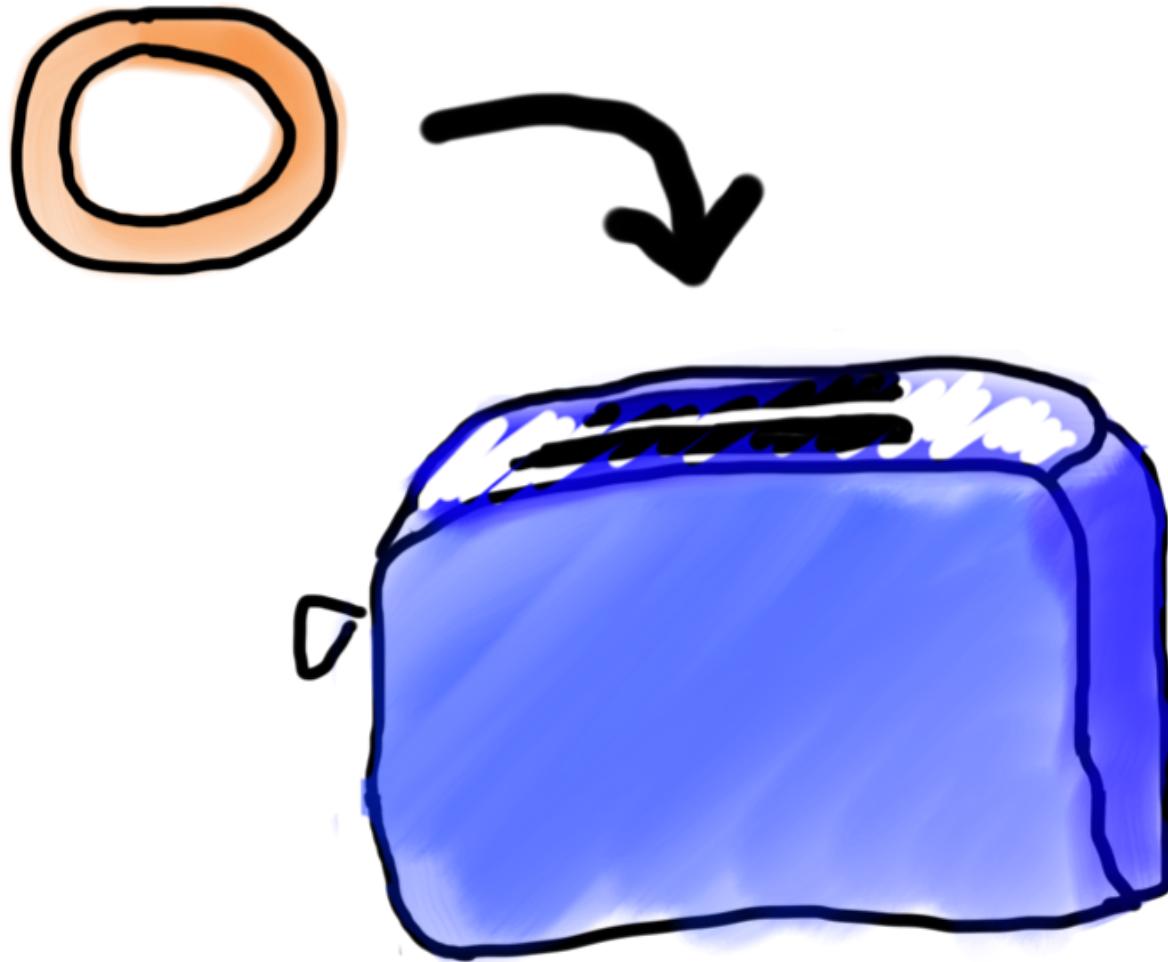
# Toasters are Methods



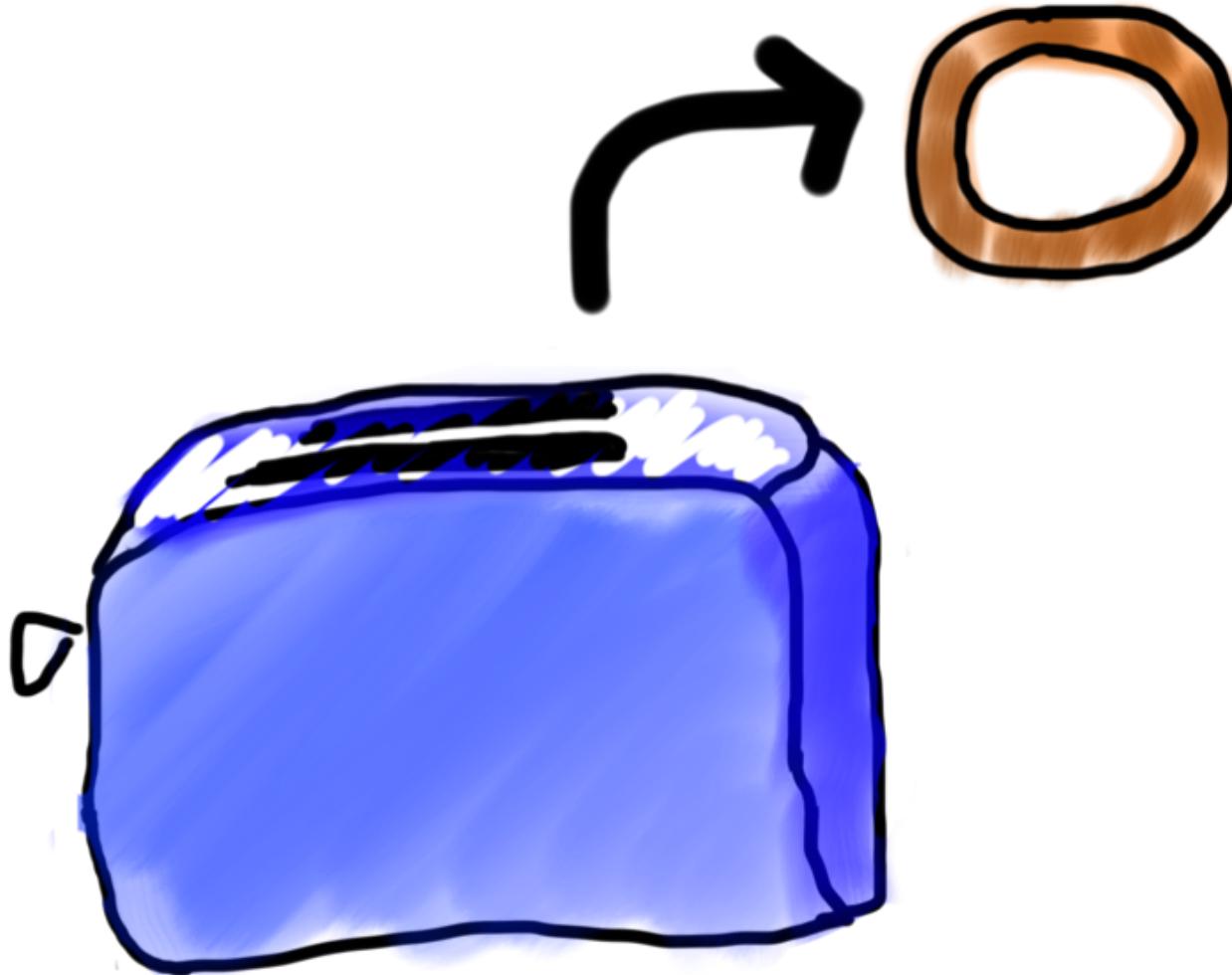
\* You don't need a second toaster if you want to toast bagels. Use the same one.



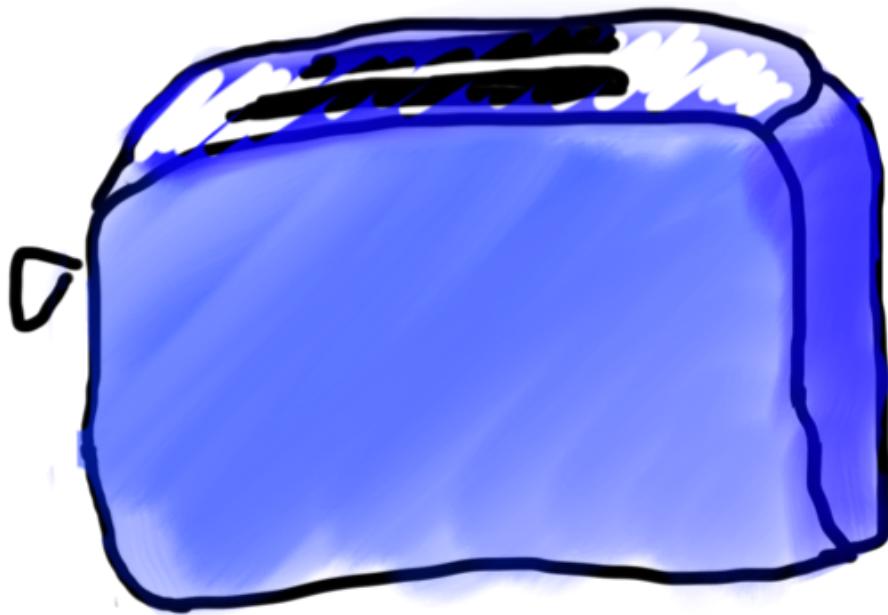
# Toasters are Methods



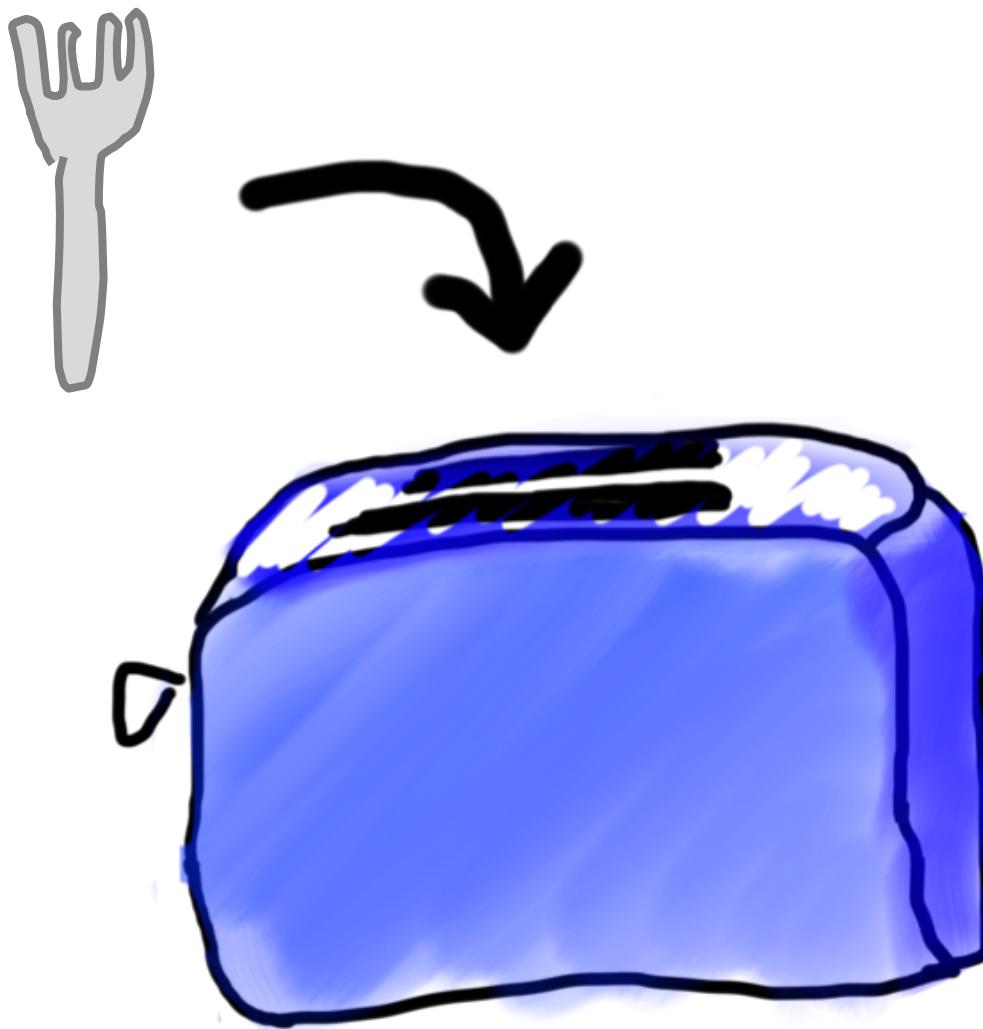
# Toasters are Methods



# Toasters are Methods



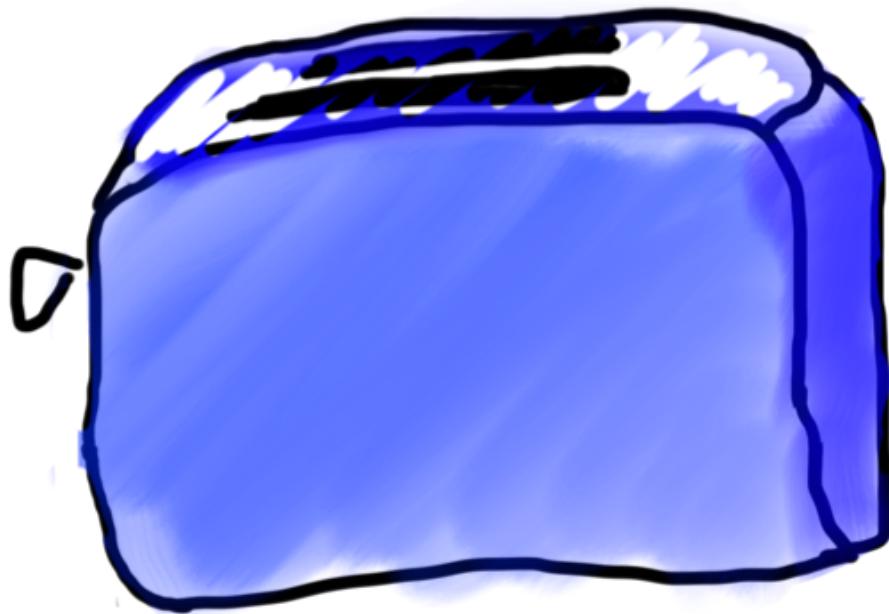
# Toasters are Methods



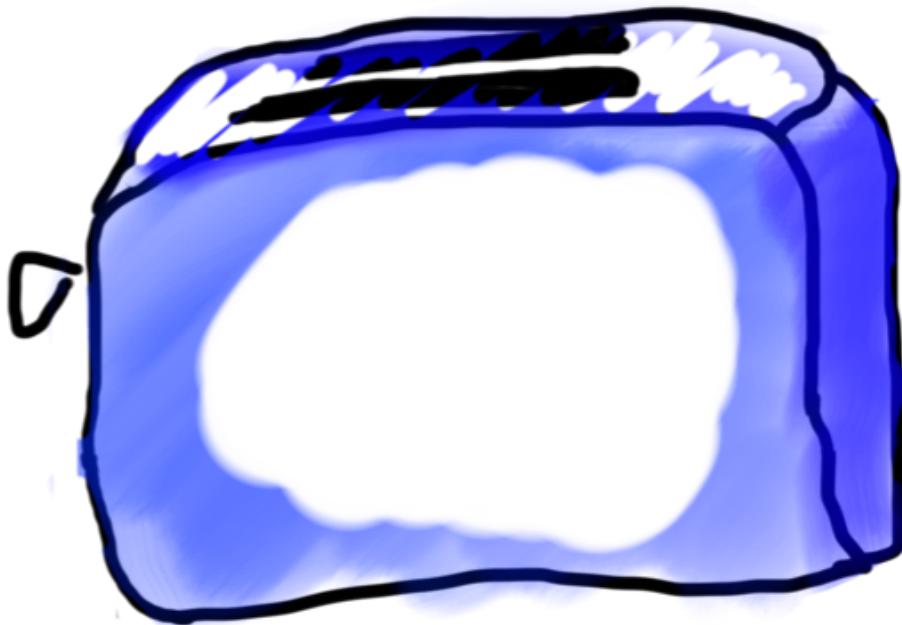
# Toasters are Methods



# Methods are Like Toasters



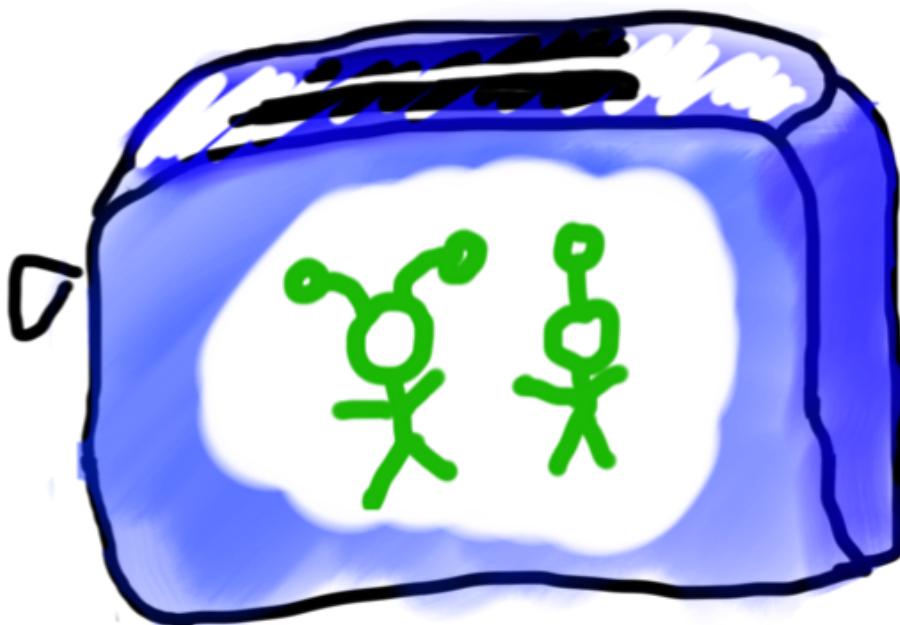
# Methods are Like Toasters



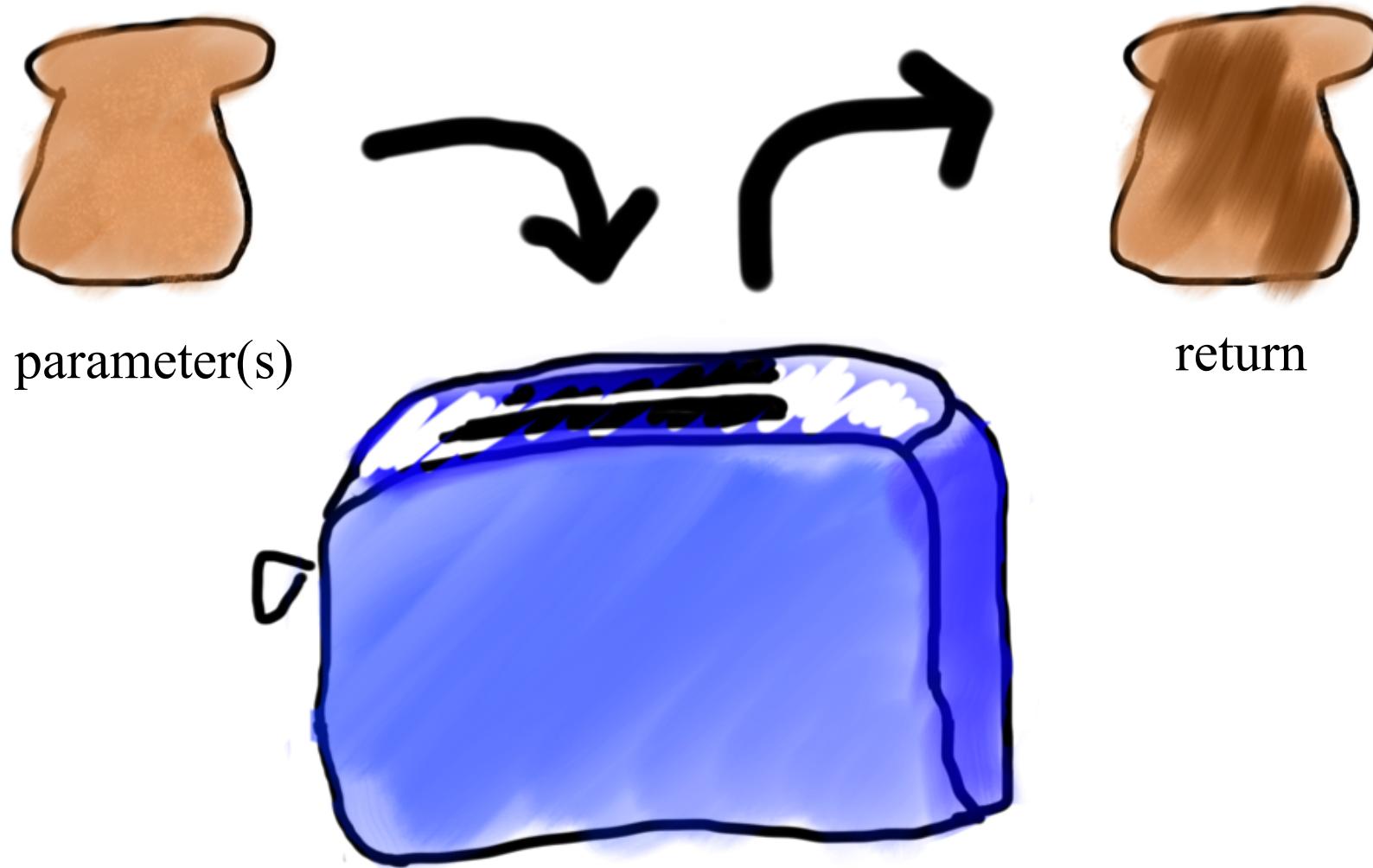
# Methods are Like Toasters



# Methods are Like Toasters



# Methods are Like Toasters



# Formally

```
visibility type nameOfMethod(parameters) {  
    statements  
}
```

- ***visibility*:** usually **private** or **public**
- ***type*:** type returned by method (e.g., **int**, **double**, etc.)
  - Can be **void** to indicate that nothing is returned
- ***parameters*:** information passed into method



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}  
  
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

method “definition”

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

Output expected

Input expected

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

Return Type

Parameters

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

name

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```

body



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```

Ends the method and gives back a single value



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```

This statement is necessary because **average** promised to return a double



# Anatomy of a method

```
public void run() {           method "call"  
    double mid = average(5.0, 10.2);  
    println(mid);  
}  
  
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

Return Type    Parameters

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Anatomy of a method

```
public void run() {  
    double mid = average(5.0, 10.2);  
    println(mid);  
}
```

When a method ends it “returns”

```
private double average(double a, double b) {  
    double sum = a + b;  
    return sum / 2;  
}
```



# Parameters



Parameters let you provide a method some information when you are calling it.



# Learn by Example



# Void Example

```
private void printIntro() {  
    println("Welcome to class");  
    println("It's the best part of my day.");  
}  
  
public void run() {  
    printIntro();  
}
```



# Parameter and Return Example

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}  
  
public void run() {  
    double result = metersToCm(5.2);  
    println(result);  
}
```



# Parameter and Return Example

```
private double metersToCm(double meters) {  
    return 100 * meters;  
}  
  
public void run() {  
    println(metersToCm(5.2));  
    println(metersToCm(9.1));  
}
```



# Parameter Example

```
private void printOpinion(int num) {  
    if(num == 5) {  
        println("I love 5!");  
    } else {  
        println("Whatever");  
    }  
}  
  
public void run() {  
    printOpinion(5);  
}
```

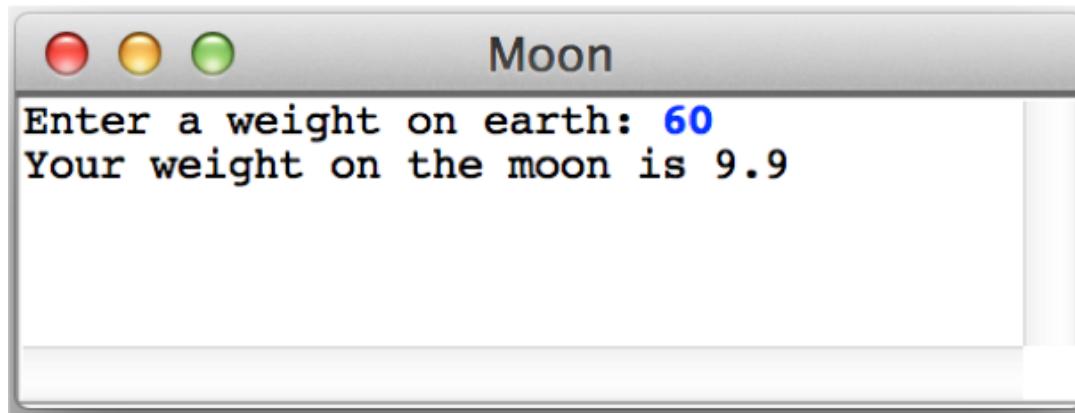


# Multiple Return Statements

```
private void max(int num1, int num2) {  
    if(num1 >= num2) {  
        return num1;  
    }  
    return num2;  
}  
  
public void run() {  
    int larger = max(5, 1);  
}
```



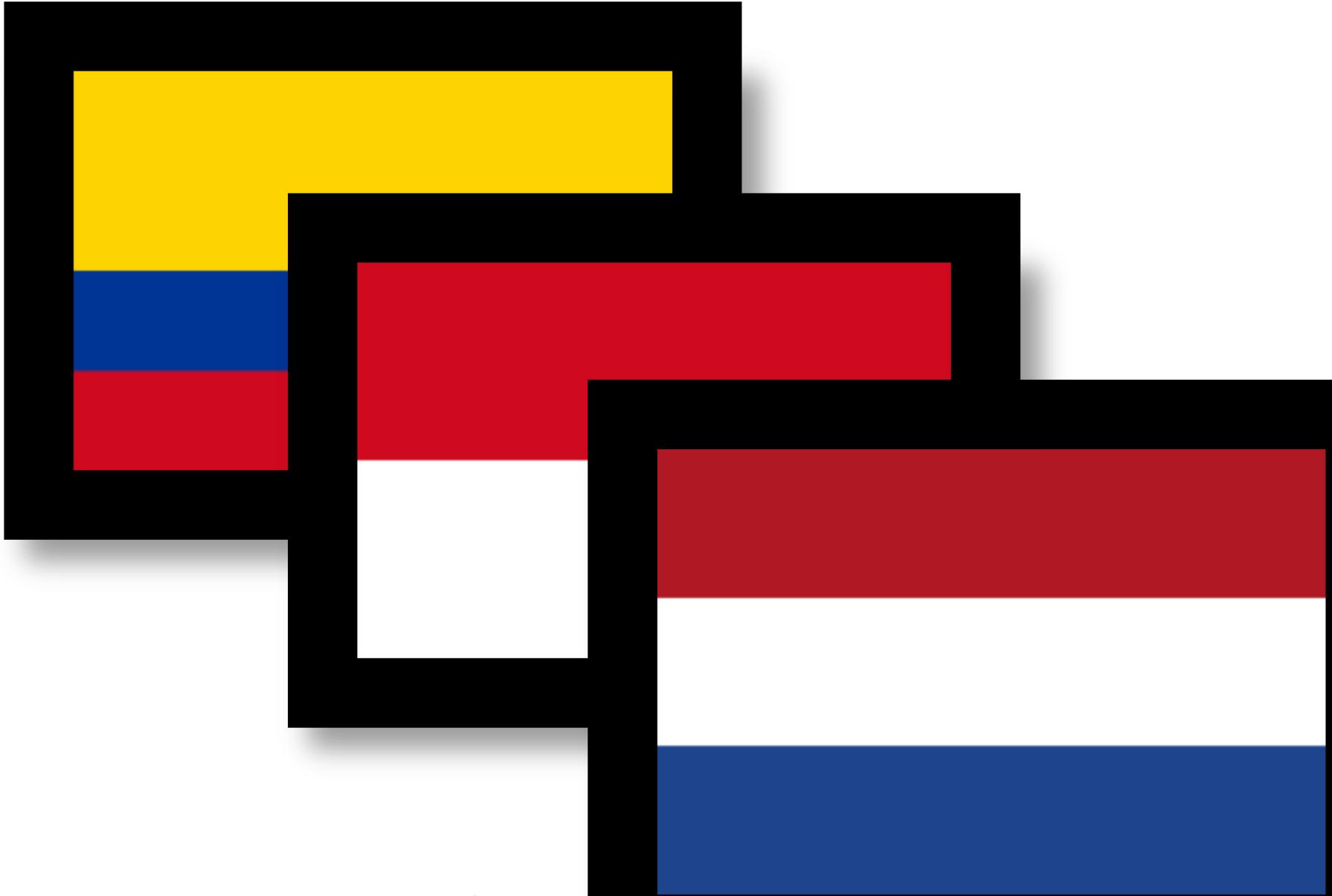
# Method for Weight on Moon



\* Your weight on the moon is 16.5% your weight on the earth



# Passing in Classes



# A Full Program

```
public class FactorialExample extends ConsoleProgram {  
  
    private static final int MAX_NUM = 4;  
  
    public void run() {  
        for(int i = 0; i < MAX_NUM; i++) {  
            println(i + "!" + factorial(i));  
        }  
    }  
  
    private int factorial(int n) {  
        int result = 1;  
        for (int i = 1; i <= n; i++) {  
            result *= i;  
        }  
        return result;  
    }  
}
```

# A Full Program

```
public class FactorialExample extends ConsoleProgram {  
  
    private static final int MAX_NUM = 4;  
  
    public void run() {  
        for(int i = 0; i < MAX_NUM; i++) {  
            println(i + "!" + factorial(i));  
        }  
    }  
  
    private int factorial(int n) {  
        int result = 1;  
        for (int i = 1; i <= n; i++) {  
            result *= i;  
        }  
        return result;  
    }  
}
```

Understand the Mechanism

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 0

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 0

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 0

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n	0	result	1	i	1
---	---	--------	---	---	---

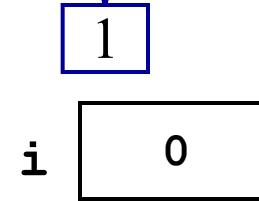
```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n	0	result	1	i	1
---	---	--------	---	---	---

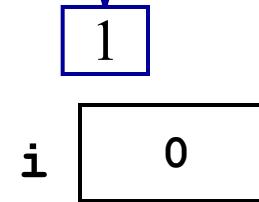
```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```



0! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```

i 1

0 ! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

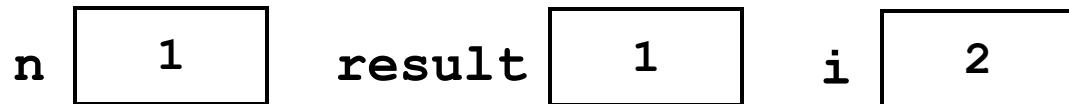
0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```



$0! = 1$

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```



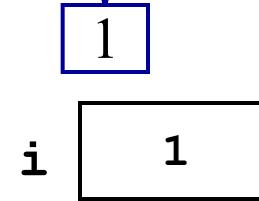
0! = 1

```
private int factorial(int n) {  
    int result = 1;  
    for (int i = 1; i <= n; i++) {  
        result *= i;  
    }  
    return result;  
}
```

n  result  i

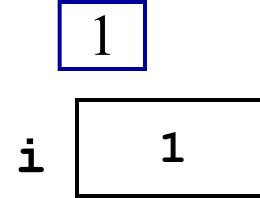
$0! = 1$

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + " ! = " + factorial(i));  
    }  
}
```



0 ! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```

i 2

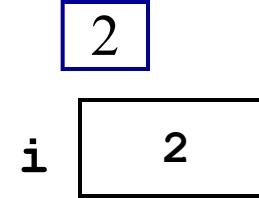
0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 2

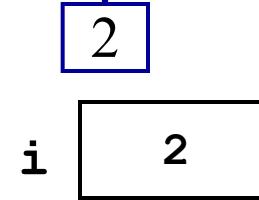
0! = 1  
1! = 1

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```

i 3

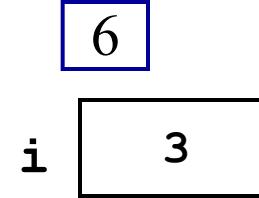
```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 3

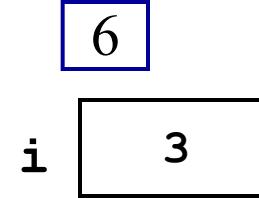
```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1  
2! = 2
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" = " + factorial(i));  
    }  
}
```



```
0! = 1  
1! = 1  
2! = 2  
3! = 6
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 4

```
0! = 1  
1! = 1  
2! = 2  
3! = 6
```

```
public void run() {  
    for(int i = 0; i < MAX_NUM; i++) {  
        println(i + "!" + factorial(i));  
    }  
}
```

i 4

```
0! = 1  
1! = 1  
2! = 2  
3! = 6
```

# Parameters



Every time a method is called, new memory is created for the call.



# Bad Times With Methods

// NOTE: This program is buggy!!

```
private void addFive(int x) {  
    x += 5;  
}
```

```
public void run() {  
    int x = 3;  
    addFive(x);  
    println("x = " + x);  
}
```

Let's "trace" this  
program on the board



# Good Times With Methods

```
// NOTE: This program is feeling just fine...
```

```
private int addFive(int x) {  
    x += 5;  
    return x;  
}  
  
public void run() {  
    int x = 3;  
    x = addFive(x);  
    println("x = " + x);  
}
```





For primitives:  
Variables are **not**  
passed when you  
use parameters.  
Values are passed



# Pass by “Value”



# More Examples

# Changed Name

```
private void run() {  
    int num = 5;  
    cow(num);  
}  
  
private void cow(int grass) {  
    println(grass);  
}
```



# Same Variable Name

```
private void run() {  
    int num = 5;  
    cow();  
    println(num);  
}
```

```
private void cow() {  
    int num = 10;  
    println(num);  
}
```



# Methods Called on Objects

```
GRect rect = new GRect(20, 20);  
rect.setColor(Color.Blue);
```



receiver

\* We will talk about how to define these later in the class

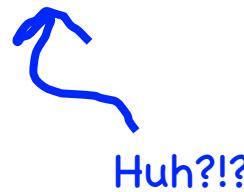


# No Methods in Methods

```
private void run() {  
    println("hello world");  
    private void sayGoodbye() {  
        println("goodbye!");  
    }  
}
```



Illegal modifier for parameter goodbye, only final is permitted



# No Methods in Methods

```
private void run() {  
    println("hello world");  
    sayGoodbye();  
}  
  
private void sayGoodbye() {  
    println("goodbye!");  
}
```



Remember Booleans?

# Boolean Variable

```
boolean karelIsAwesome = true;  
  
boolean myBool = 1 < 2;
```



# Boolean Operations

```
boolean a = true;
```

```
boolean b = false;
```

```
boolean and = a && b;
```

```
boolean or = a || b;
```

```
boolean not = !a;
```





# Is Divisible By

```
private void run() {  
    for(int i = 1; i <= 100; i++) {  
        if(isDivisibleBy(i, 7)) {  
            println(i);  
        }  
    }  
}
```



# Boolean Return

```
private void run() {  
    for(int i = 1; i <= 100; i++) {  
        if(isDivisibleBy(i, 7)) {  
            println(i);  
        }  
    }  
}
```



```
private boolean isDivisibleBy(int a, int b) {  
    if((a % b) == 0) {  
        return true;  
    } else {  
        return false;  
    }  
}
```



# Boolean Return

```
private void run() {  
    for(int i = 1; i <= 100; i++) {  
        if(isDivisibleBy(i, 7)) {  
            println(i);  
        }  
    }  
  
    private boolean isDivisibleBy(int a, int b) {  
        return a % b == 0;  
    }  
}
```



# Learn How To:

1. Write a method that takes in input
2. Write a method that gives back output
3. Trace method calls using stacks



# Extra Exercise

- Greek mathematicians took a special interest in numbers that are equal to the sum of their proper divisors (a proper divisor of  $n$  is any divisor less than  $n$  itself). They called such numbers *perfect numbers*. For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are the integers less than 6 that divide evenly into 6. Similarly, 28 is a perfect number because it is the sum of 1, 2, 4, 7, and 14.
- Design and implement a Java program that finds all the perfect numbers between two limits. For example, if the limits are 1 and 10000, the output should look like this:

