CS 106A, Lecture 7
Parameters and Return

suggested reading:
Java Ch. 5.1-5.4
Plan For Today

• Announcements
• Recap: For Loops
• Recap: Scope
• Parameters
• Return
Plan For Today

• Announcements
• Recap: For Loops
• Recap: Scope
• Parameters
• Return
For Loops in Java

This code is run once, just before the for loop starts.

Repeats the loop if this condition passes.

This code is run each time the code gets to the end of the 'body'.

```
for (int i = 0; i < 3; i++) {
    println("I love CS 106A!");
}
```
• **nested loop**: A loop placed inside another loop.

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 10; j++) {
        print("*");
    }
    println(); // to end the line
}
```

• Output:

```
**********
**********
**********
**********
**********
```

• The outer loop repeats 5 times; the inner one 10 times.
Nested loop question 2

- How would we produce the following output?
  ....1
  ...22
  ..333
  .4444
  55555

- Answer:
  ```java
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5 - i - 1; j++) {
      print(".");
    }
    for (int j = 0; j <= i; j++) {
      print(i + 1);
    }
    println();
  }
  ```
We can define new methods in Java just like in Karel:

```java
private void name() {
    statement;
    statement;
    ...
}
```

For example:
```java
private void printGreeting() {
    println("Hello world!");
    println("I hope you have a great day.");
}
```
public void run() {
    int x = 2;
    printX();
}

private void printX() {
    // ERROR!  "Undefined variable x"
    println("X has the value "+x);
}
Plan For Today

• Announcements
• Recap: For Loops
• Recap: Scope
• Parameters
• Return
A Variable love story

By Chris Piech
Variable Scope

- The **scope** of a variable refers to the section of code where a variable can be accessed.
- **Scope starts** where the variable is declared.
- **Scope ends** at the termination of the code block in which the variable was declared.

- A **code block** is a chunk of code between `{ }` brackets.
Variable Scope

You *cannot* have two variables with the same name in the *same scope*.

```java
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;              // ERROR
    print("/");
}
```
Variable Scope

You *cannot* have two variables with the same name in the *same scope*.

```java
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;              // ERROR
    while (...) {
        int i = 5;           // ERROR
    }
}
```
Variable Scope

You can have two variables with the same name in separate scopes.

```java
class Example {
    public void run() {
        for (int i = 0; i < 5; i++) {
            int w = 2; // w ok here
        }

        for (int i = 0; i < 2; i++) {
            int w = 3; // w ok here
        }
    }
}
```
Variable Scope

You can have two variables with the same name in separate scopes.

```java
public void run() {
    int num = 5;
    cow();
    println(num); // prints 5
}

private void cow() {
    int num = 10;
    println(num); // prints 10
}
```
You can have two variables with the same name in *different scopes*.

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

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

You can have two variables with the same name in different scopes.

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

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

You *can* have two variables with the same name in *different scopes*.

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

private void cow() {
    int num = 10;
    println(num);
}
```
You can have two variables with the same name in different scopes.

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

private void cow() {
    int num = 10;
    println(num);
}
```
You *can* have two variables with the same name in *different scopes*.

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

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

• Announcements
• Recap: For Loops
• Recap: Scope
• Parameters
• Return
Parameters

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

parameter
Example: readInt

readInt("Your guess? ");
Example: readInt

We call readInt

readInt("Your guess? ");

We give readInt some information in parenthesis (the text to print to the user)
Example: printGreeting

printGreeting(5);

(Prints a greeting a certain number of times)
Wouldn’t it be nice if...

We call `printGreeting`.

We give `printGreeting` some information (the number of greetings to print).

```javascript
printGreeting(5);
```
Methods with Parameters

Tells Java this method needs one int in order to execute.

```java
private void printGreeting(int times) {
    // use ‘times’ to print the greeting
}
```
private void printGreeting(int times) {
    // use ‘times’ to print the greeting
}
private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}

printGreeting(5);
public void run() {
    int repeats = 5;
    printGreeting(repeats);
}

private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}
Methods with Parameters

```java
public void run() {
    int repeats = 5;
    printGreeting(repeats);
}

private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}
```

run

5

repeats

printGreeting

times

?
public void run() {
    int repeats = 5;
    printGreeting(repeats);
}

private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}
Methods with Parameters

```java
public void run() {
    int times = 5;
    printGreeting(times);
}

private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}
```

run

5 times

printGreeting

? times
public void run() {
    int times = 5;
    printGreeting(times);
}

private void printGreeting(int times) {
    for (int i = 0; i < times; i++) {
        println("Hello world!");
    }
}
/\ NOTE: This program is **buggy**!!

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

private void addFive(int x) {
    x += 5;
}
```
// NOTE: This program is buggy!!
public void run() {
    int x = 3;
    addFive(x);
    // prints "x = 3"
    println("x = " + x);
}

private void addFive(int x) {
    x += 5;
}
// NOTE: This program is buggy!!
public void run() {
    int x = 3;
    addFive(x);
    // prints "x = 3"
    println("x = " + x);
}

private void addFive(int x) {
    x += 5;
}
// NOTE: This program is buggy!!
public void run() {
    int x = 3;
    addFive(x);
    // prints "x = 3"!
    println("x = " + x);
}

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

- **parameter**: A value passed to a method by its caller.

  - Write a method `drawBox` to draw a box of any size.
    - When *declaring* the method, we will state that it requires the caller to tell it the width and height of the box.
    - When *calling* the method, we will specify the width and height to use.
Declaring a parameter

Stating that a method requires a parameter in order to run

```java
private void name(type name) {
    statements;
}
```

• Example:
  ```java
  private void password(int code) {
    println("The password is: " + code);
  }
  ```
  
  – When password is called, the caller must specify the integer code to print.
Multiple parameters

• A method can accept multiple parameters separated by commas:
  – When calling it, you must pass values for each parameter.

• Declaration:
  
  ```
  private void name(type name, ..., type name) {
    statements;
  }
  ```

• Call:
  
  ```
  name(value, value, ..., value);
  ```
Passing a parameter

Calling a method and specifying values for its parameters

methodName(expression);

• Example:

public void run() {
    password(42);
    password(12345);
}

Output:
The password is 42
The password is 12345

• Illegal to call without passing an int for that parameter.

password(); // Error
password(3.7); // Error
How params are passed

• When the method is called:
  – The value is stored into the parameter variable.
  – The method's code executes using that value.

```java
public void run() {
    chant(7);
}

private void chant(int times) {
    for (int i = 0; i < times; i++) {
        println("Java is great!");
    }
}
```
• Lets write a program that uses methods and parameters to print the following boxes:

```
**********
*    *
*    *
**********

*******
*    *
*    *
*    *
*******
```

– The code to draw each box will be very similar.

• Would variables help? Would constants help?
drawBox(10, 4);
We call `drawBox`.

We give `drawBox` some information (the size of the box we want).

`drawBox(10, 4);`
private void drawBox(int width, int height) {
    // use width and height variables
    // to draw a box
}
private void drawBox(int width, int height) {
    line(width);
    for (int line = 0; line < height - 2; line++) {
        boxSide(width);
    }
    line(width);
}
private void drawBox(int width, int height) {
    line(width);
    for (int line = 0; line < height - 2; line++) {
        boxSide(width);
    }
    line(width);
}
private void drawBox(int width, int height) {
    line(width);
    for (int line = 0; line < height - 2; line++) {
        boxSide(width);
    }
    line(width);
}
private void drawBox(int width, int height) {
    line(width);
    for (int line = 0; line < height - 2; line++) {
        boxSide(width);
    }
    line(width);
}
private void drawBox(int width, int height) {
    line(width);
    for (int line = 0; line < height - 2; line++) {
        boxSide(width);
    }
    line(width);
}
private void line(int count) {
    for (int i = 0; i < count; i++) {
        print("*");
    }
    println();
}
private void boxSide(int width) {
    print("*");
    for (int i = 0; i < width - 2; i++) {
        print(" ");
    }
    println("*");
}
public void run() {
    drawBox(10, 4);
    drawBox(7, 6);
}
Plan For Today

- Announcements
- Recap: For Loops
- Recap: Scope
- Parameters
- Return
Return values let you give back some information when a method is finished.
Methods = Toasters

parameter
Methods = Toasters

parameter
Methods = Toasters
Methods = Toasters
Methods = Toasters

return
Example: readInt

```java
int x = readInt("Your guess? ");
```
Example: readInt

```java
int x = readInt("Your guess? ");
```

We call `readInt`

We give `readInt` some information (the text to print to the user)
Example: readInt

When finished, readInt gives us information back (the user’s number) and we put it in x.

```java
int x = readInt("Your guess? ");
```
Example: readInt

When we set a variable equal to a method, this tells Java to save the return value of the method in that variable.

```java
int x = readInt("Your guess? ");
```
Example: `metersToCm`

double cm = metersToCm(5);

(Returns the given number of m as cm)
Example: metersToCm

```c++
double cm = metersToCm(5);
```

We give `metersToCm` some information (the number of meters).

We call `metersToCm`
Example: metersToCm

When metersToCm finishes, it returns the number of cm, and we put that in this variable.

double cm = metersToCm(5);
private double metersToCm(double meters) {
  ...
}

Tells Java this method needs one double in order to execute.
Methods and Return

Tells Java that, when this method finishes, it will return a double.

```java
private double metersToCm(double meters) {
    ...
}
```
Methods and Return

Tells Java that, when this method finishes, it will return a **double**.
(Void meant returns nothing)

```java
private double metersToCm(double meters) {
    return 100 * meters;
}
```
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
public void run() {
    double meters = readDouble("#meters? ");
    double cm = metersToCm(meters);
    println(cm + " centimeters.");
}

private double metersToCm(double meters) {
    return 100 * meters;
}
If a method returns something, you can use it directly in an expression!
Return

- **return**: To send out a value as the result of a method.
  - Parameters send information *in* from the caller to the method.
  - Return values send information *out* from a method to its caller.
  - A call to the method can be used as part of an expression.

- Q: Why return? Why not just println the result value?

```
run    Math.abs(-42)
  
-42

run    Math.round(2.71)
  
2.71
3

42
```

```
Methods

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
private boolean isEven(int number) {
}

private boolean isEven(int number) {
    if (number % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
private boolean isEven(int number) {
    if (number % 2 == 0) {
        return true;
    } else {
        return false;
    }
}

public void run() {
    int num = readInt("? ");
    if (isEven(num)) {
        println("Even!");
    } else {
        println("Odd!");
    }
}
private boolean isEven(int number) {
    return number % 2 == 0;
}
public void run() {
    for(int i = 1; i <= 100; i++) {
        if(isDivisibleBy(i, 7)) {
            println(i);
        }
    }
}

private void isDivisibleBy(int a, int b) {
    return a % b == 0;
}
Return ends a method’s execution.

```java
private int multiplyByTwo(int num) {
    return num * 2;
    println("Hello world?"); // not executed!
}
```
Return ends a method’s execution.

```java
private int max(int num1, int num2) {
    if (num1 >= num2) {
        return num1;
    }
    return num2; // here only if num1 < num2
}

public void run() {
    println(max(2, 3));
}
```
Revisiting a Bug

// NOTE: This program is buggy!!
public void run() {
    int x = 3;
    addFive(x);
    // prints "x = 3"
    println("x = " + x);
}

private void addFive(int x) {
    x += 5;
}
// NOTE: This program is feeling just fine
public void run() {
    int x = 3;
    x = addFive(x);
    // prints "x = 5"
    println("x = " + x);
}

private int addFive(int x) {
    x += 5;
    return x;
}
Recap

• Announcements
• Recap: For Loops
• Recap: Scope
• Parameters
• Return

Next time: Strings (new variable type!)