# Functions

CS110: Introduction to Computer Science

# Reusing Code

- Ideally, how many times should we find a solution to a problem? Once.
- Right now, we have no way of reusing the **algorithms** we've written.
- **Functions** change that.

# Functions

- Functions are ways to reuse code.
- Functions consist of two components
  - Declaration
  - Utilization

# Declaration

- Declaring a function consists of three components
  - The function signature
  - The body
  - The `return` statement
- The **function signature** consists of 4 components:
  - The keywords `public static` (for now)
  - The `return` type
  - The function name
  - The parameter list.

# Example

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

The Method Signature

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

The Body

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

**The return statement**

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

The keywords

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

The return type

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

The method name

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Example

The parameter list

```
public static int example(int param1, double value)
{
    int rtnValue;
    if (param1 > 10) {
        rtnValue = (int) value;
    } else {
        rtnValue = (int) (value * 2);
    }
    return rtnValue;
}
```

# Utilizing a Function

- To utilize a function you must **call** it.
- To call a function, all you need to do is use its name and pass it the proper number and type of values using the parameter list

```
int a = example(20,30.4);
```

- **Note**: The type of a is the same as the return type of ***example***( )
- **Let's go through some examples**

# Calling From a Different File

- To call a function from a different file, you must must the file name before

```
Magic.println(a);
```

# Function vs Methods

- Sometimes functions that don't need to return a value.
- These functions are called **methods**.
- They have a return type of **`void`**.
- They can have a **`return`** statement, but don't need one
- You've already seen one!

```
public static void main(String[] args){
   Magic.println("Hello World");
}
```

# Scope

- The **scope** of a function is denoted by the `{}`.
- Variables created inside of a function's scope cannot be used outside of it.
- If statements and loops have scope too, albeit they are nested inside of a function's scope.
- **Let's go through some examples**

# Naming Functions

- Since functions perform actions, usually the best name for functions are **verbs** describing what they will do.
- For example:
  - `isBigger()`
  - `drawSquare()`
- Not hard and fast. For example
  - `average()` is better than `calculateAverage()`

# Note About Writing Functions

- When writing a function in an informal description, we will often omit the parameters.
- So, we'd write something like:

*In the following assignment, you should use* `Magic.println()` *to print out the average grade of each student.*

- Instead of the more technically correct but harder to read

*In the following assignment, you should use* `Magic.println(double output)` *to print out the average grade of each student.*

# Procedural Decomposition

# Why Functions?

- Functions are important for lots of reasons
  - Code reuse
  - Single point of maintenance
  - Easier to read
  - **Easier to reason about**

# Easier to Reason About

- Large problems become overwhelming quickly.
- How do you solve it?
- Break it down into smaller subproblems.
- How do you solve the subproblems?
- Break it down again.
- Each time you break it down, you create a new function to correspond to that sub problem.
- This is called P**rocedural Decomposition.**

# Example

- Let's go through an example where you want to calculate student's grade in the class.
- The class has three categories: labs, homework, and tests.
- Labs are 20%, Homework is 20%, and Tests are 60%

# Side Note: Magic is a teaching tool

- `Magic.println()`, `Magic.nextInt()`, `Magic.nextDouble()`, and `Magic.nextLine()` **WILL NOT WORK** inside of functions, other than `main()`
- Why?
- Because you need to get use to the **only way data goes INTO a function is through the parameters** and **the only way data comes OUT of a function is through the return statement**.
- Don't cheat and use the "real" ways (which have no such restriction) . You need to get use to this.

# Recursion

# Recursion

- Recursion is a technique where a function calls itself
- Functions can call other functions, so why not itself
- **Let's try it right now!**

```
public static void silly(int a){
    if(a%2==0) {
        Magic.drawRectangle(a*10, a*10, a * 10, a * 10, "red");
    } else {
        Magic.drawRectangle(a*10, a*10, a * 10, a * 10, "blue");
    }
    silly(a+1);
}
```

# What went wrong?

# Two Parts to Recursion

1. Recursive statement (calling yourself)
2. Base case (when it ends)

# Fixing Silly

```
public static void sillyFixed(int a){
    if(a%2==0) {
        Magic.drawRectangle(a*10, a*10, a * 10, a * 10, "red");
    } else {
        Magic.drawRectangle(a*10, a*10, a * 10, a * 10, "blue");
    }
    if(a < 20){
        sillyFixed(a+1);
    }
}
```

# Recursion in Real Life

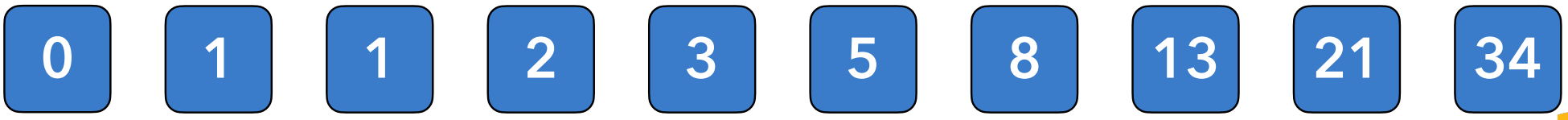
- Suppose you are far back in theater and you want to know what row number you are.
- You could ask the person in front of you what row number THEY are in and add one.
- They can ask the person in front of them and add one to that....
- This continues until the front, when the first person says "row one"

# Why Use?

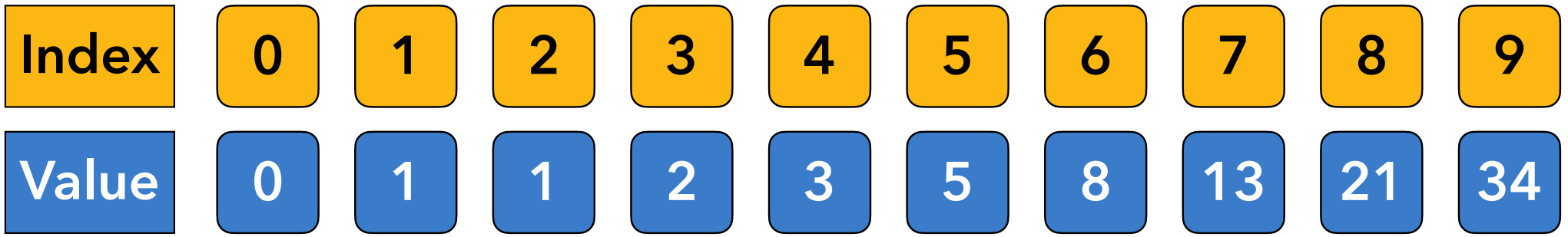
- As we discuss in CS120, recursion and iteration are interchangeable.
- So, why use recursion?
- Some problems are **naturally recursive**.
- Consider the Fibonacci numbers

# Fibonacci Numbers

- Fibonacci numbers are the sequence of numbers created through the following process:
  - The first number is 0
  - The second number is 1
  - Every number after the first two is the sum of the previous two numbers



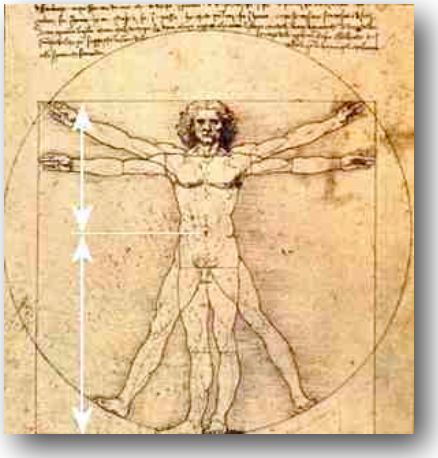

# Index




# Side Note: Golden Ratio

- The most visually appealing ratio
- Frequently appears in nature
- Approximately equal to 1.61803….
- Defined as

$$\lim_{n \to \infty} \frac{\text{fib}(n+1)}{\text{fib}(n)}$$



# Let's Write a Recursive Fibonacci Function

- **Let's do it!**

# Another Example!