Functions! aka Methods

#5

CS 0007
Introduction to
Computer Programming

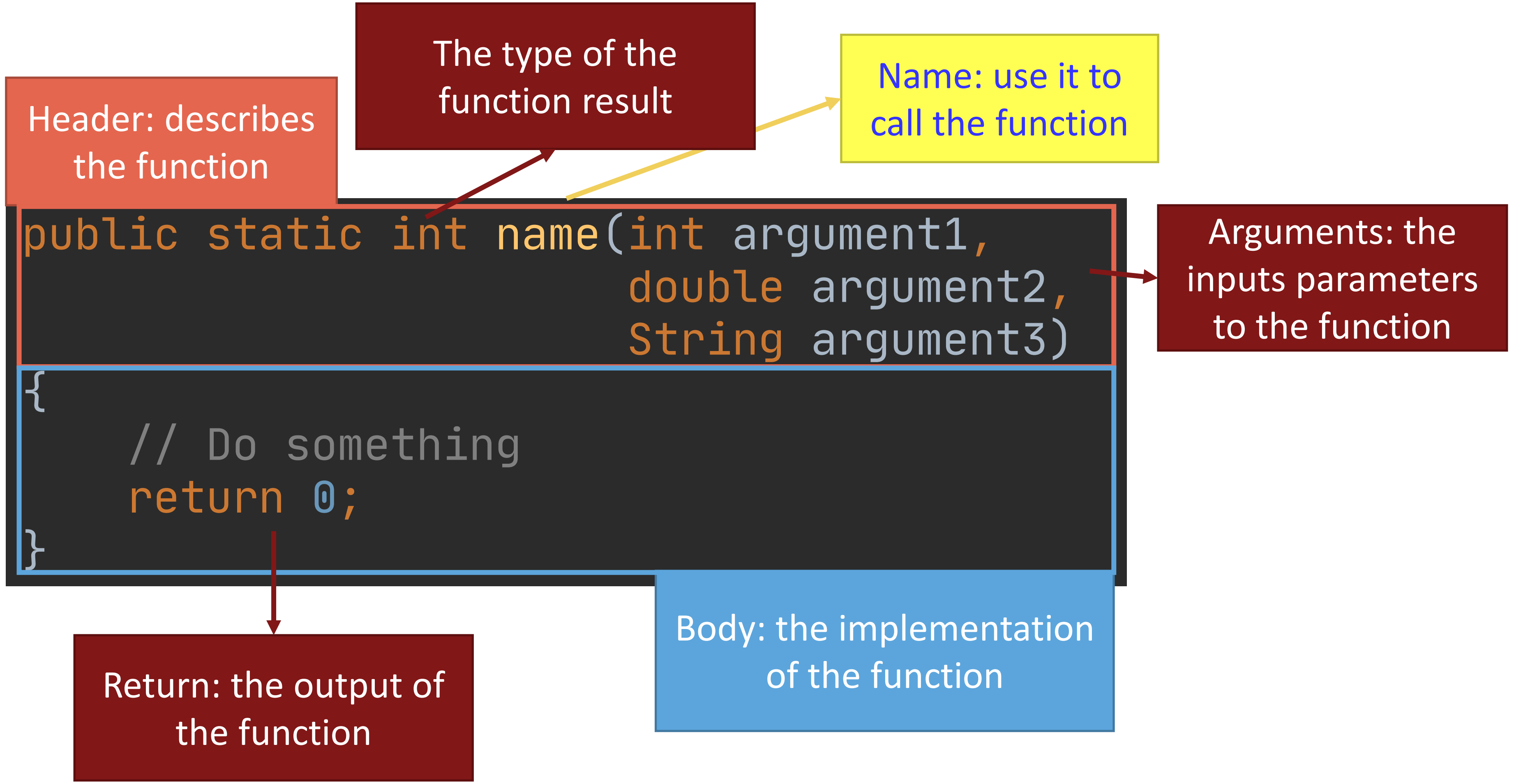
Luís Oliveira

Summer 2020

Functions (Methods)

- “The best thing since if statements” – me
  - They allow you to organize your code into logical sections
  - You can use the same code over and over → without copy paste!
  - It makes code easier to read!
- “Do I need functions?”
  - If your code is 10 lines… No
  - If your code is 100 lines… Probably
  - If your code is 1000 lines… YES!
- Partition your code, and it becomes easier to read and write!
  - You can test your code in chunks too!

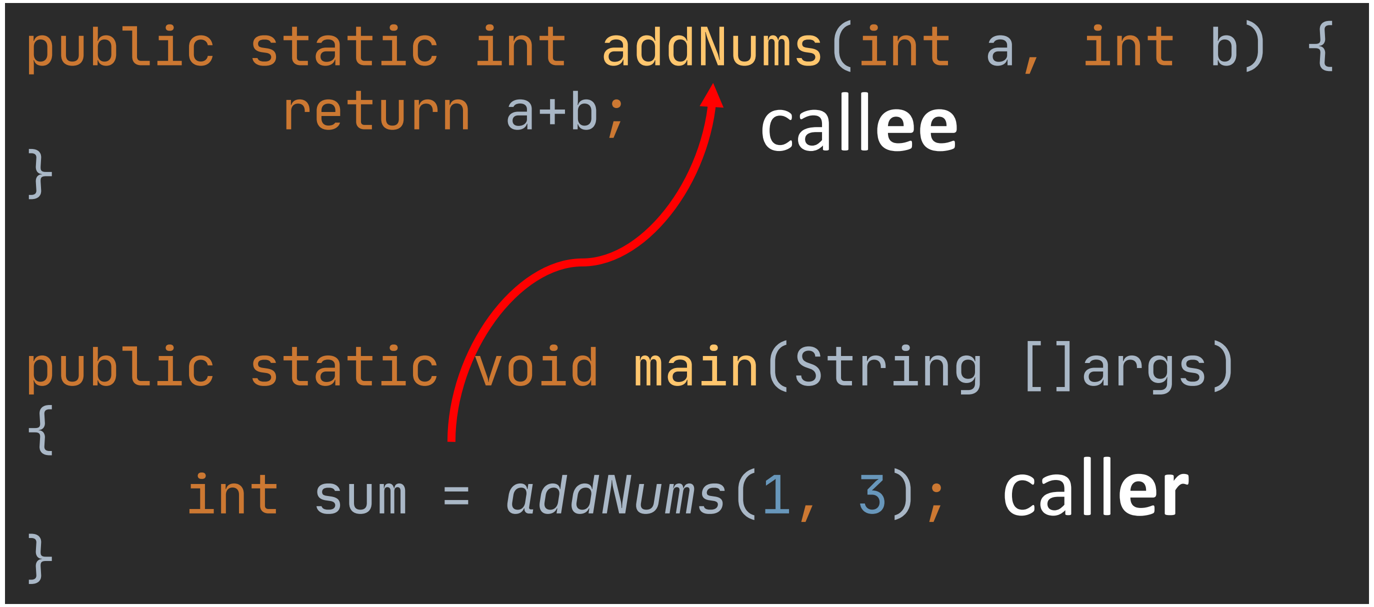
Anatomy of a function


Header: describes the function
The type of the function result
Name: use it to call the function
Arguments: the inputs parameters to the function
Return: the output of the function
Body: the implementation of the function

```java
public static int name(int argument1,
                        double argument2,
                        String argument3)
{
    // Do something
    return 0;
}
```

Flow of program

- Calling a function moves the execution from the caller to the callee
  - It comes back once it is done.



```
public static int addNums(int a, int b) {
        return a+b;
}


public static void main(String []args)
{
     int sum = addNums(1, 3);
}
```

Functions are black boxes

- You don’t need to know how they work internally.
  - Only what they do!

```
public static int addNums(int a, int b) {
        return a+b;
}
```

```
public static int addNums(int a, int b) {
        int sum;
        if (a>b) { // This is silly btw!
            sum = a + b;
        } else {
            sum = b + a;
        }
        return sum;
}
```

addNums

Amazing function that adds two numbers. You do not need to know how it is implemented!!!

Inputs:
1. Number to add
2. Number to add

Outputs:
1. Numbers added together

Arguments

- Arguments allow you to give data to the function
- They will have a type and a name

```
public static int name(int argument1)
```

- They must be explicitly types (even if of the same type!)

```
public static int name(int argument1, int argument2)
```

Return value

- The return value is the response (output) of the function
  - void means nothing is returned!

```
public static void name()
```

- All types can be returned

```
public static int name(int argument1, int argument2)
```

- In Java only one value/type can be returned
  - Other languages have multiple return values
  - We can “trick” java into this, but we’ll look at that later ☺

Use it or lose it!

```
public static int addNums(int a, int b){...}
```

- The return value must be used or stored

```
int result = addNums(1, 2);

int otherResult = addNums(1, 2) * 2;

System.out.println(addNums(1, 2));
```

- If you don't... you will not be able to get it again

```
addNums(1, 2); // gone!
```

Different functions, different scopes

- Functions are sibling scopes! So variable names can be repeated!

```java
public static void sayHello(String input) {
    String name = input;
    System.out.println("Hello" + name);
}
```

These variables are not the same

```java
public static void sayGoodMorning(String input) {
    String name = input;
    System.out.println("Good Morning" + name);
}
```

Same names, but not necessary

- It's common for variable names to match with the function arguments

```java
public static void sayHello(String name) {
    System.out.println("Hello " + name);
}
public static void main(String []args){
    String name = "Luis";
    sayHello(name);
}
```

```java
public static void main(String []args){
    String blargh = "Luis";
    sayHello(blargh);
}
```

Terrible variable name!
But valid!

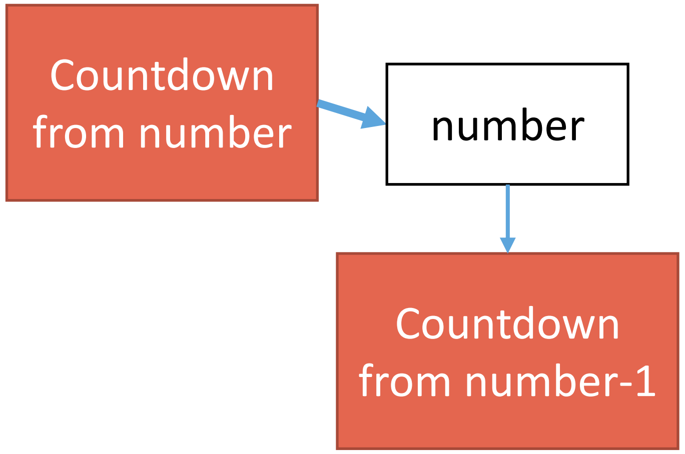
They have no clue!

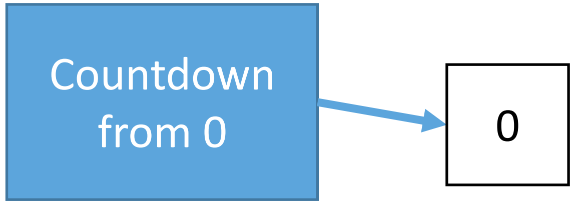
- Functions don’t know values of variables declared in other functions!
  - EVER!
  - Not even when called recursively
- This is on purpose!
  - Imagine having to remember all about variables in your 1,000,000 line code!
- Isolation and abstraction
  - These are the cornerstones (yes, 2!) of functions!
- About recursive……

Recursive functions

Redefining the problem

- Solving problems with **recursion**:
  - The problem can be redefined as a simpler version of the same problem
  - E.g.: Countdown from a number
- In code: The function calls itself
  - Multiple times until it reaches the base case
    Base case: problem with simple solution
- Usually improve readability
  - Occasionally the opposite
- Are limited by number of recursive calls
  - Memory limitation (by the OS → More memory will not help ☹)
    - CS majors: Take CS449 for more details ;)

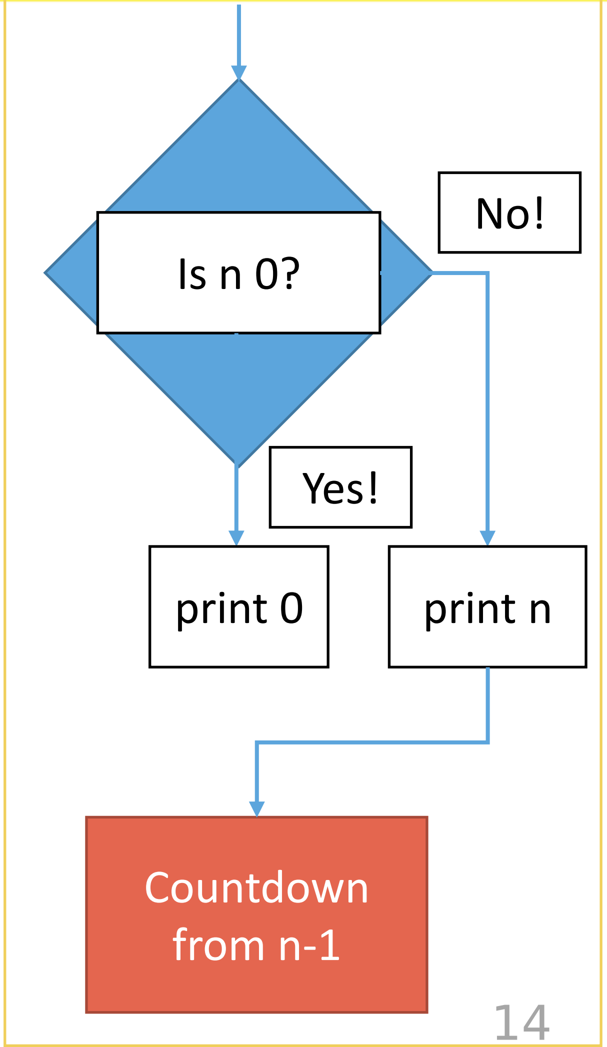




Anatomy of a recursive function

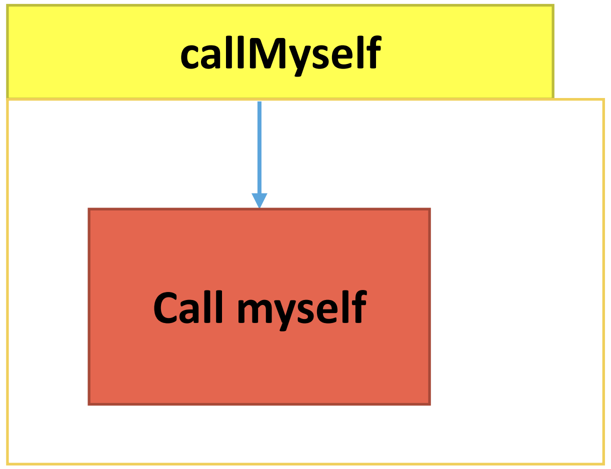
- **A recursive function MUST have a <u>base case</u> and a <u>recursive case</u>**
  - Without the recursive: it's not recursive! duh!
  - Without the base case: it'll never end.
    - Search google for recursion for an example
- **The recursive case MUST (usually) reduce the size of the problem**
  - Otherwise it'll never end!
  - If doing user input validation, that is not true.

Countdown from n



Anatomy of a recursive function

- This is a bad recursive function


callMyself
Call myself