Functions

Overview

- When we write programs, it's helpful to be able to break them up into smaller logical pieces
    - Easier to write a few lines of code at a time than hundreds/thousands
    - Can make code easier to read
    - Can make code easier to debug
- Writing functions allows us to name portion of our code to invoke them later
- We've already used all sorts of functions; it's time to write our own

Learning Objectives

- Be able to read a function's signature to identify its return type, its name, and its input types
- Be able to follow program execution through multiple function calls
- Be able to write your own functions to perform specific tasks

Demo: Writing a short function

Functions are named lists of statements

It may help to understand a function if we first see an example:

```
// Function Signature & Declaration
public static void printGreeting() {
    System.out.println("Howdy partner!");
    System.out.println("Welcome to my program!");
}

public static void main(String[] args) {
    printGreeting(); // calling the function
}
```

Vocabulary

- **Functions** are named lists of statements
- **Function definitions** consist of the function's signature as well as a block of statements called its **body**
  - A **function signature** consists of the function's name, its return type, and the list of arguments that it takes as input.
- A **function call** is how we run the statements belonging to a function by invoking its name.
  - e.g. `PennDraw.clear()`

Example: DrawingFlowers.java

Lets see how we can use functions to make code more readable, and easier to debug.

```
public static void drawFlower(double x, double y, double r) {
    PennDraw.setPenColor(242, 121, 208);
    PennDraw.filledCircle(x + r, y, r);
    PennDraw.filledCircle(x - r, y, r);
    PennDraw.filledCircle(x, y + r, r);
    PennDraw.filledCircle(x, y - r, r);

    PennDraw.setPenColor(252, 186, 3);
    PennDraw.filledCircle(x, y, r);
}
```

`main`

- `main` is a **function**
  - by function, we mean a named list of statements
- When we run our Java programs, it invokes the `main` function, which executes all of the statements in `main`.
- Running the program also initializes the `args` array:
  - example: `java MyProgram 3.0 Harry 49.2`
  - sets up `args` with: `{"3.0", "Harry", "49.2"}`
    - note how all these values are read as type `String`

A Closer Look at `main`

```
public class ExampleClass {
    public static void main(String[] args) {
        String out = "";
        for (int i = 0; i < 10; i++) {
            out += "CIS1100";
        }
    }
}
```

- `main`'s function signature is `public static void main(String[] args)`
- The function body consists of the statements between `main`'s curly braces.

Function Signatures

```
public static <ret-type> <name>(<type1> <id1>, <type2> <id2>, ...)
```

Consists of:

- the function's return type, which specified what data type (if any!) this function should produce
- the function's name, which is any valid identifier
- a comma-separated list of the function's inputs. The inputs are each given as a (type, identifier) pair
- We will keep ignoring what `public static` means for now...

`void`

When a function is called, the statements in its body begin executing. When the function is finished executing, it may or may not produce a value.

- When a function does not give some data as a result, we say that it has return type `void`.
  - `PennDraw.circle(double x, double y, double r)` is an example of a function that has return type `void`. No value is produced!

```
PennDraw.circle(0.6, 0.7, 0.1); // Draws a circle to the screen
```

Return Types

Other functions actually produce a value. When writing a function, we must use its signature to tell Java what type it will output.

- A function with a non-`void` return type must ALWAYS return some value of the specified type.
- Examples:
  - `Math.random()` has return type `double`, producing a value between `[0.0, 1.0)`
  - `str.charAt(int idx)` has return type `char`, producing whichever `char` is found at index `idx` in `str`.

Function Signature Practice

```
public static int greatestCommonDivisor(int x, int y) {
    ...
}
```

- What's the function's name?
- What's the function's return type?
- What inputs does this function expect?

Function Signature Practice

```
public static int greatestCommonDivisor(int x, int y) {
    ...
}
```

- What's the function's name? `greatestCommonDivisor`
- What's the function's return type? `int`
- What inputs does this function expect?
    - Two `int` values. The first will be called `x` and the second will be called `y` inside the body of the function.

Function Signature Practice

```
public static int greatestCommonDivisor(int x, int y) {
    ...
}
```

- What's the function's name? `greatestCommonDivisor`
- What's the function's return type? `int`
- What inputs does this function expect?
  - Two `int` values. The first will be called `x` and the second will be called `y` inside the body of the function.
- **From the above, can you guess how this function should work?**

Calling Functions

- To call a function, we write its name, followed by the list of inputs we're giving it in parentheses
  - e.g. `PennDraw.setPenColor(0, 255, 0)` is a call to `PennDraw.setPenColor` with inputs `0`, `255`, and `0`.
- When a function is called, program execution is immediately transferred to the top of the body of that function
- When that function finishes executing, the call is replaced with the value that gets returned (if any).

```
public class FunctionCall {
    public static void sayHi() {
        System.out.println("Hey!");
        System.out.println("It's me!");
    }

    public static void main(String[] args) {
        System.out.println("CALLING sayHi!");
        sayHi();
        System.out.println("BACK IN MAIN!");
    }
}
```

OUTPUT:

```
CALLING sayHi!
Hey!
It's me!
BACK IN MAIN!
```

```
public class FunctionCall {
    public static void sayMessage(String msg) {
        System.out.println("Psst!");
        System.out.println(msg);
    }

    public static void main(String[] args) {
        System.out.println("CALLING sayMessage!");
        sayMessage("Soylent Green is people");
        System.out.println("BACK IN MAIN!");
    }
}
```

OUTPUT:

```
CALLING sayMessage!
Psst!
Soylent Green is people
BACK IN MAIN!
```

The `return` Statement

- The `return` keyword stops the execution of the current function and sends execution back to the line where the function was called.
- If the function returns something, then `return` is paired with the value that actually gets returned.

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

Calling `mean(4.0, 6.0)` thus evaluates to `5.0`!

Rules of `return`

- If a function has a non-`void` return type, then it must include a `return` statement.
- If the function has return type `void`, it may still include a `return` statement, but it doesn't have to.
  - In this case, just write `return` without an accompanying value
  - This use of `return` is just to stop execution

Scoping and Functions

- Any variable declared inside the body of a function is only in scope in that particular function.
- Parameter variables are also in scope only in the body of the function they're declared for.

```
public String duplicate(String s, int n) {
    String output = "";
    for (int i = 0; i < n; i++) {
        output += s;
    }
    return output;
}
```

A Closer Look at Returning

```
public class Averages {
    public static double mean(double a, double b) {
        double sum = a + b;
        double average = sum / 2;
        return average;
    }
    public static void main(String[] args) {
        double outputValue = mean(10.0, 20.0);
        System.out.println(outputValue);
    }
}
```

`mean(10.0, 20.0)` evalutes to `15.0`, which is stored in `outputValue` and then printed.

Common Mistakes in Writing functions

What's wrong here?

```
public class Maxima {
    public static double maximum(double a, double b) {
        if (a >= b) {
            System.out.println(a);
        } else {
            System.out.println(b);
        }
    }
    public static void main(String[] args) {
        double outputValue = maximum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

What's wrong here? `maximum` doesn't return a double!

```
public class Maxima {
    public static double maximum(double a, double b) {
        if (a >= b) {
            System.out.println(a);
        } else {
            System.out.println(b);
        }
    }
    public static void main(String[] args) {
        double outputValue = maximum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

What's wrong here?

```
public class Maxima {
    public static double maximum(double a, double b) {
        if (a >= b) {
            return a;
        } else {
            System.out.println(b);
        }
    }
    public static void main(String[] args) {
        double outputValue = maximum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

What's wrong here? We still don't return a double if `b > a`!

```
public class Maxima {
    public static double maximum(double a, double b) {
        if (a >= b) {
            return a;
        } else {
            System.out.println(b);
        }
    }
    public static void main(String[] args) {
        double outputValue = maximum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

What's wrong here?

```
public class Maxima {
    public static double maximum(double a, double b) {
        if (a >= b) {
            return a;
        } else {
            return b;
            System.out.println(b);
        }
    }
    public static void main(String[] args) {
        double outputValue = maximum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

We'll return before printing `b`. This is called "dead code".

```
public class Maxima {
    public static double maximum(double a, double b) {
        if (a >= b) {
            return a;
        } else {
            return b;
            System.out.println(b);
        }
    }
    public static void main(String[] args) {
        System.out.println(maximum(10, 20));
    }
}
```

Common Mistakes in Writing functions

What's wrong here?

```
public class Summing {
    public static int sum(double a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
        double outputValue = sum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

What's wrong here? `a` and `b` are doubles, and so is `a + b`, but we promised to return an int!

```
public class Summing {
    public static int sum(double a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
        double outputValue = sum(10, 20);
        System.out.println(outputValue);
    }
}
```

Common Mistakes in Writing functions

What's wrong here?

```
public class Repeating {
    // prints a String n times.
    public static void printRepeatedly(String msg, int n) {
        for (int i = 0; i < n; i++) {
            System.out.println(msg);
        }
    }
    public static void main(String[] args) {
        printRepeatedly(4, "Hello!");
    }
}
```

Common Mistakes in Writing functions

What's wrong here? `printRepeatedly` expects a `String` and an `int`, in that order!

```
public class Repeating {
    // prints a String n times.
    public static void printRepeatedly(String msg, int n) {
        for (int i = 0; i < n; i++) {
            System.out.println(msg);
        }
    }
    public static void main(String[] args) {
        printRepeatedly(4, "Hello!");
    }
}
```