Functions

CREATING A FUNCTION

Functions are a way to group code together with a name so that it can be called from somewhere else. They can also be called methods. Rather than try to build this up from smaller pieces, I'm just going to give an example and then explain the pieces. Keep in mind that this an attempt to be a simple example and may not be something that you would normally write a function for, but you could.

```
import java.util.*;

import java.lang.*;

import java.io.*;


public static class Main {

    public static void main(String[] args) {

        System.out.println(add(5, 4));

        printMessage();

    }

    public static int add(int a, int b) {

        int sum = a + b;
        return sum;

    }

    public static void printMessage() {

        System.out.println(“Hello, World”);

    }

}
```

In this example, we have three different functions, `main()`, `add()` and `printMessage()`. Let’s start by looking at the creation of the function.

Private vs public

You’ll notice the first word in front of each function is `public`. The other main option here is `private`. The meaning of these will be clearer once we get into classes, but it has to do with what code can call the function. If a function is private, it can only be called from inside the same file. If it is public, it can be called by any code.

Static

The next word on each of these is `static`. Again, this has to do with how a function can be accessed and will make more sense once we get into classes. For now, know that if one function is static, the functions it uses will also need to be static. So since `main()` is static, `add()` and `printMessage()` need to be static.

Return types

After the modifiers comes the return type. Since functions are a block of code, they are often used to do some calculations or block of work, so they have to hand back an answer. The return type specifies what type is going to be handed back. So in the case of `add()`, the return type is `int`. This means that in every branch of the code, there needs to be a `return` statement which has an int after the word `return`. You cannot have a way that gets to the end of the function without having a return. You have to hand an `int` back.

You will notice that the other two functions have something unfamiliar as a return type: `void`. This means that you are not returning anything. So you don’t have to have any return statements in that type of functions, although you could if you wanted to.

Naming a function

Next up comes the name of the function. Functions have the same rules as any variable. Note that parentheses follow the function creation to distinguish it from a variable.

Parameters

Just like you have a return type to hand something back from a function, you have parameters to hand variables to a function, if you want. If you have empty parentheses, that means you have no parameters. You can also make a comma-separated list of variables that can then be used inside the function just like any other variables.

Note that any variables created in one function are not accessible in another function unless you send them as parameters or return them. If I were to create a variables named `a` and `b` inside of `main()` and have no parameters for `add()`, I wouldn't able to do the same line of `int sum = a + b;` because the function `add()` doesn't have access to variables from `main()`. For the same reason, I wouldn't be able to change `add()` to a void function and write `add(5, 4); System.out.println(sum);` inside of `main()`, because `main()` doesn't have any access to variables from `add()`.

Inside of a function

A function can include any other code we've seen already, but it cannot include another function inside of it.

CALLING A FUNCTION

To call a function, write the name of the function, followed by parentheses, inside the parens should be a comma-separated list of all params that you need to send. Any value of the type can be used as a parameter value: a variable, another function that has the right return type, a simple value, math that has the right type, a value from an array, anything that has the right type. So we could have something like `add(add(5, i), add(4, 4 + i))` as long as `i` is an int. The return value of a function can be used as a value of that type just like any other value, it can be printed, saved to a variable, sent to another function, anything you can do with an int, you can do with an int function, same for a String and a String function, etc.

Note that a function can call itself, but be careful with it because it is easy to end up with an infinite loop when you do that.

WHY MAKE A FUNCTION

So why make a function? There are three main reasons. The first is reuse. If you want to do the same piece of code over and over again, it makes it really easy to put it into a function and just call that function over again when you need to do it again rather than copy-paste and have multiple places to change when that code changes or has bugs. Like if we had a piece of code that shoots a ball or drives forward, we may need to do that in many different places in code for a robot, but we want it to be consistent, so we can just call driveForward() or shootBall().

The second reason is libraries. It makes it easy for someone to write code that someone else can use. This makes a block of code that is easily accessible elsewhere. There are places that have already written the code that directly interfaces with the electronics on the robot and put it into functions that can be called by you in your code. It makes things easier to share and use in that type of manner if it is in functions.

And even if you are using it in your code and doing it only once, it can make your code more readable. Instead of seeing code that reads something if a user pushes button number 3 and a trigger in the robot is pressed, then turn on motor number 2 to 200 rpm for 3 seconds, you could see code like if button 3 is pressed, shootBall() and the rest of the code could be inside that function, and it will be clearer what is going on (although you should also be using comments!).

FUNCTIONS WE ALREADY KNOW

You've already seen this before several different places. equals(), println(), main() are all functions that you've already been working with.

TRY IT OUT

These questions are roughly ordered in increasing difficulty, but some of them may be easier or harder depending on what you can figure out. It is not expected that you complete all of them. 1 and 2 should be more straightforward. 3 is tricky but may require breaking things down a bit. Do attempt at least one of 4 and 5 as they are similar to each other and worth figuring out. 6 is meant only as something to work on if you have more time and want a challenge.

1. Write a function called blackjack that takes two ints and returns the number that comes the closest to 21 without going over. So blackjack(19, 18) -> 19 but blackjack(4, 22) -> 4.
2. Write a function called loneSum that takes 3 numbers and sums up the unique numbers, and ignores any duplicates. So loneSum (3, 2, 1) -> 6 loneSum(3, 2, 3) -> 2, loneSum(3, 3, 3) -> 0
3. Write a function that takes one number and returns the sum of the digits. sumDigits(123) -> 6, sumDigits(53412) -> 15
4. Write a function that returns that number factorial without using a loop (0!=1, 1! =1, 2!=2, 3!=6, 4!=24, 5!=120) so factorial(4) -> 24

5. Write a function that calculates the nth fibonacci number without using a loop. The fibonacci numbers are created by summing up the two previous numbers. (0 -> 1, 1-> 1, 2-> 2, 3 -> 3, 4 -> 5, 5 -> 8, 6 -> 13)
6. Only if you want a challenge: Create a function `public boolean makeBricks(int small, int big, int goal)`

We want to make a row of bricks that is **goal** inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return true if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops.