Team: Troll Coders
Members: Thomas Andersen
Jack Lillywhite
William Rauch
Scarlett Winter

1. What is a *loop invariant*?

=> A loop invariant is a condition in a loop that is true both before and after each iteration.

2. Explain how the loop invariant in *power* justifies correctness.

=> In the *power* method, the parameter *(i <= exponent)* is a loop invariant. It justifies correctness by remaining true for all iterations of the loop, until the end point where the loop terminates.

3. Describe a situation where a *for-loop* would be more appropriate than a *while-loop*, and vice versa. Can you provide a code snippet for each scenario?

=> A *for-loop* is more effective in situations where you need to run a loop for a known number of iterations, such as the *multiply* method where we know that it must loop an amount of times equal to the *multiplier* parameter:

```java
for ( int i = 1; i <= multiplier; i++ ) {
	product += multiplicand;
}
```

A *while-loop* is more effective for situations where a loop must iterate an unknown number of times until it achieves a desired result, as in our *log* method:

```java
while ( power( base, (int)exponent ) != (long)argument ) {
	exponent++;
}
```

Team: Troll Coders
Members: Thomas Andersen
Jack Lillywhite
William Rauch
Scarlett Winter

4. Explain how using methods can make a program easier to understand and maintain.

=> By using methods to modularize your program, you prevent repetitive code and make it easier to read. It also narrows the scope of bugfixing and enables you to reuse the method in other programs.

5. Describe what a method signature is in Java. How does it help in method overloading?

=> In Java, the method signature consists of a method's name and its parameters. Using method overloading, you can create multiple methods with the same name but different parameters, allowing for different types of input.

6. Consider a problem using nested loops. Explain how the control variables in the outer and inner loops interact with each other. Could you change the order of these loops without affecting the results? Why or why not?

=> For a problem with nested loops, the control variables of outer loops can interact with those of the inner loops, but the inner variables can't due to their scope. If you changed the order, it would change the variables' scopes and possibly cause syntax errors.