Expressions, Statements, Blocks, and Logical Branching

The purpose of this exercise is to reinforce your understanding of expressions, statements, blocks, and logical branching.

This exercise consists of thirty-five small coding problems. Each problem is independent of the others, and they can be solved in any order.

Learning objectives

After completing this exercise, students will understand:

- How to use expressions and statements to solve complex problems.
- How to apply comparison and logical operators to solve complex problems.
- How to organize and group statements within blocks.
- How to choose different paths within code using if/else blocks.

Evaluation criteria and functional requirements

- The project must not have any build errors.
- Unit tests pass as expected.
- Appropriate variable names and data types are being used.

Getting started

1. Import the expressions and control flow exercises project into IntelliJ.
2. Run all tests to see the results of your tests and which ones passed or failed.
3. Provide enough code to get a test passing.
4. Repeat until all tests are passing.

Tips and tricks

Read the problem description carefully

Before each method, there's a description of the problem that needs to be solved and examples with expected output. Use these examples to get an idea of the values you need to write your code around. It may help to keep track of the state of variables on a piece of paper as you work through your code.

For example, in the comments above the `more20` method, there's a section that includes the method name and the expected value that's returned for each method call. The following example shows that when the method is called with 20, it returns false, when it's called with 21, it returns true, and when it's called with 22, it returns true:

```
```
more20(20) → false
more20(21) → true
```
```

```
more20(22) → true
```

Check test output if your tests are failing

If your tests fail, check the output of the test run. It provides helpful clues and information that could be valuable when troubleshooting.

You can also run the tests in debug mode when executing the tests. This allows you to set a "breakpoint", which stops the code at certain points in the editor. You can then look at the values of variables while the test runs, and can also see what code is currently being executed. Don't hesitate to use the debugging capabilities in IntelliJ to help resolve issues.

Don't linger too long on one problem

If you find yourself stuck on a problem for more than fifteen minutes, move on to the next, and try again later. You may figure out the solution after working through another problem or two.