Refactoring

This lesson focuses on code quality. Starting from a working program, students refactor the code to be more readable, writing helper functions thinking structurally about a complex program.

Product Outcomes	• Students refactor existing code to make an emoji image
Materials	• Slides are not yet available for this lesson • Printable Lesson Plan (a PDF of this web page)
Prerequisites	• Simple Data Types • Contracts • Simple Inequalities • Piecewise Functions and Conditionals • Compound Inequalities: Solutions & Non-Solutions • Introduction to Data Structures • Structures, Reactors, and Animations • Key Events
Preparation	• The Robot Emoji file preloaded on student machines • The Emoji Refactoring file preloaded on student machines

Glossary

refactor :: the process of changing the style or structure of a program's code, without changing the program's behavior

Refactoring - a Case Study *40 minutes*

Overview

Student are introduced to the programming concept of *refactoring*, which closely models the Mathematical Practice 7: *Identify and make use of structure*. Students create an emoji generator, and then refactor it to make the code cleaner.

Launch

One of the most common tasks software developers find themselves performing is ***refactoring*** code. This means taking code that is already working and complete, and cleaning it up: adding comments, removing unnecessary expressions, and generally making their code more readable and useable by others. Refactoring does not change the behavior of the program, only the appearance of the code. For instance, a messy expression like:

```
(((4 * 4) + (3 / (8 - 6))) * (9 * 9)) * (1 + 1)
```

could be refactored into:

```
((num-sqr(4) + (3 / 2)) * num-sqr(9)) * 2
```

Both expressions return the same value, but the second is much more readable. Refactoring can involve using existing functions (such as num-sqr in the example above) or writing new functions to perform small tasks.

Open the Robot Emoji file and click "Run". In this file, there are two versions of the same program written by different students.

Take a look at the definitions in this file, and, with your partner, discuss what you notice. Which student's code is easiest to read and understand? Which formatting do you like better? If you were collaborating on a project with another programmer, which version of this code would you rather to receive, and why?

Discuss with students the differences in documentation, formatting, and organization of the two versions of the emoji code.

Next, we're going to practice refactoring an existing program that draws an image.

Investigate

Open the Emoji Refactoring file and click "Run".

This code draws an image of a simple face emoji. Without changing the final image produced, can you see any opportunities to edit the code to make it more readable?

Refactor the code provided. This could include adding comments, more space betwen expressions, or simplifying the existing expressions. Once finished, write one more expression to create a smaller (emoji-sized) version of the original image.

This activity can be done individually or as a class, with students giving suggestions for refactoring code projected at the front of the room. Once the refactoring is completed, students can practice using image functions to create an emoji of their own.