Split Variable

previous: Organizing Data

next: Rename Field

```
let temp = 2 * (height + width);
console.log(temp);
temp = height * width;
console.log(temp);
```

```
const perimeter = 2 * (height + width);
console.log(perimeter);
const area = height * width;
console.log(area);
```

formerly: Remove Assignments to Parameters

formerly: Split Temp

Motivation

Variables have various uses. Some of these uses naturally lead to the variable being assigned to several times. Loop variables change for each run of a loop (such as the `i` in `for (let i=0; i<10; i++)`). Collecting variables store a value that is built up during the method.

Many other variables are used to hold the result of a long-winded bit of code for easy reference later. These kinds of variables should be set only once. If they are set more than once, it is a sign that they have more than one responsibility within the method. Any variable with more than one responsibility should be replaced with multiple variables, one for each responsibility. Using a variable for two different things is very confusing for the reader.

Mechanics

- Change the name of the variable at its declaration and first assignment.
  If the later assignments are of the form `i = i + something`, that is a collecting variable, so don't split it. A collecting variable is often used for calculating sums, string concatenation, writing to a stream, or adding to a collection.
- If possible, declare the new variable as immutable.
- Change all references of the variable up to its second assignment.
- Test.
- Repeat in stages, at each stage renaming the variable at the declaration and changing references until the next assignment, until you reach the final assignment.

Example

For this example, I compute the distance traveled by a haggis. From a standing start, a haggis experiences an initial force. After a delay, a secondary force kicks in to further accelerate the haggis. Using the common laws of motion, I can compute the distance traveled as follows:

```
function distanceTravelled (scenario, time) {
  let result;
  let acc = scenario.primaryForce / scenario.mass;
  let primaryTime = Math.min(time, scenario.delay);
  result = 0.5 * acc * primaryTime * primaryTime;
  let secondaryTime = time - scenario.delay;
  if (secondaryTime > 0) {
    let primaryVelocity = acc * scenario.delay;
    acc = (scenario.primaryForce + scenario.secondaryForce) / scenario.mass;
    result += primaryVelocity * secondaryTime + 0.5 * acc * secondaryTime * secondaryTime;
  }
  return result;
}
```

A nice awkward little function. The interesting thing for our example is the way the variable `acc` is set twice. It has two responsibilities: one to hold the initial acceleration from the first force and another later to hold the acceleration from both forces. I want to split this variable.

> When trying to understand how a variable is used, it's handy if my editor can highlight all occurrences of a symbol within a function or file. Most modern editors can do this pretty easily.

I start at the beginning by changing the name of the variable and declaring the new name as `const`. Then, I change all references to the variable from that point up to the next assignment. At the next assignment, I declare it:

```
function distanceTravelled (scenario, time) {
  let result;
  const primaryAcceleration = scenario.primaryForce / scenario.mass;
  let primaryTime = Math.min(time, scenario.delay);
  result = 0.5 * primaryAcceleration * primaryTime * primaryTime;
  let secondaryTime = time - scenario.delay;
  if (secondaryTime > 0) {
    let primaryVelocity = primaryAcceleration * scenario.delay;
    let acc = (scenario.primaryForce + scenario.secondaryForce) / scenario.mass;
    result += primaryVelocity * secondaryTime + 0.5 * acc * secondaryTime * secondaryTime;
  }
  return result;
}
```

I choose the new name to represent only the first use of the variable. I make it `const` to ensure it is only assigned once. I can then declare the original variable at its second assignment. Now I can compile and test, and all should work.

I continue on the second assignment of the variable. This removes the original variable name completely, replacing it with a new variable named for the second use.

```
function distanceTravelled (scenario, time) {
  let result;
  const primaryAcceleration = scenario.primaryForce / scenario.mass;
  let primaryTime = Math.min(time, scenario.delay);
  result = 0.5 * primaryAcceleration * primaryTime * primaryTime;
  let secondaryTime = time - scenario.delay;
  if (secondaryTime > 0) {
    let primaryVelocity = primaryAcceleration * scenario.delay;
    const secondaryAcceleration = (scenario.primaryForce + scenario.secondaryForce) / scenario.mass;
    result += primaryVelocity * secondaryTime +
      0.5 * secondaryAcceleration * secondaryTime * secondaryTime;
  }
  return result;
}
```

I'm sure you can think of a lot more refactoring to be done here. Enjoy it. (I'm sure it's better than eating the haggis—do you know what they put in those things?)

Example: Assigning to an Input Parameter

Another case of splitting a variable is where the variable is declared as an input parameter. Consider something like

```
function discount (inputValue, quantity) {
  if (inputValue > 50) inputValue = inputValue - 2;
  if (quantity > 100) inputValue = inputValue - 1;
  return inputValue;
}
```

Here `inputValue` is used both to supply an input to the function and to hold the result for the caller. (Since JavaScript has call-by-value parameters, any modification of `inputValue` isn't seen by the caller.)

In this situation, I would split that variable.

```
function discount (originalInputValue, quantity) {
  let inputValue = originalInputValue;
  if (inputValue > 50) inputValue = inputValue - 2;
  if (quantity > 100) inputValue = inputValue - 1;
  return inputValue;
}
```

I then perform Rename Variable twice to get better names.

```
function discount (inputValue, quantity) {
  let result = inputValue;
  if (inputValue > 50) result = result - 2;
  if (quantity > 100) result = result - 1;
  return result;
}
```

You'll notice that I changed the second line to use `inputValue` as its data source. Although the two are the same, I think that line is really about applying the modification to the result value based on the original input value, not the (coincidentally same) value of the result accumulator.

previous:
Organizing Data

next:
Rename Field