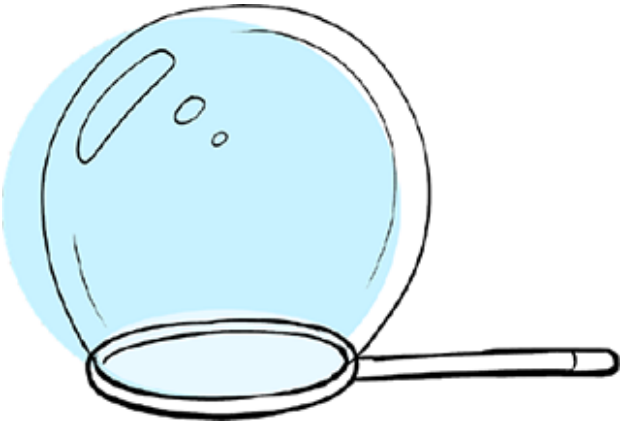
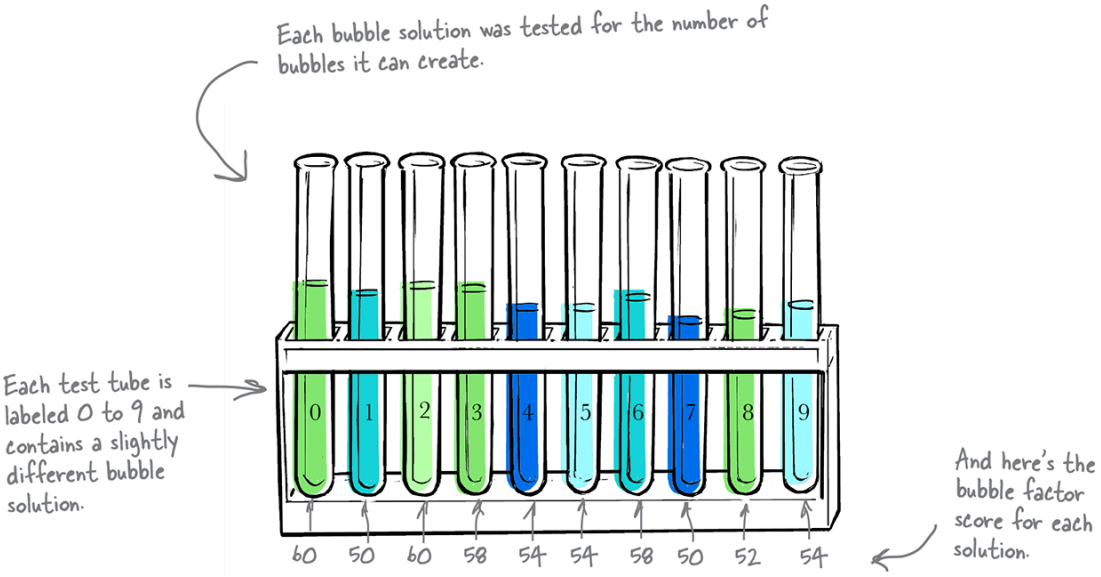
Chapter 4. Putting some order in your Data: *Arrays*

There's more to JavaScript than numbers, strings, and booleans. So far you've been writing JavaScript code with **primitives**—strings, numbers, and booleans, like "Fido", 23, and true. You can do a lot with primitive types, but at some point you've got to deal with **more data**. Say, all the items in a shopping cart, or all the songs in a playlist, or a set of stars and their apparent magnitude, or an entire product catalog. For that you need a little more *oomph*. The type of choice for this kind of ordered data is a JavaScript **array**, and in this chapter we're going to walk through how to put your data into an array, how to pass it around, and how to operate on it. We'll be looking at a few other ways to **structure your data** in later chapters, but let's get started with arrays.

Can you help Bubbles-R-Us?

Meet the Bubbles-R-Us company. Their tireless research makes sure bubble wands and machines everywhere blow the best bubbles. Today, they're testing the "bubble factor" of several variants of their new bubble solution; that is, they're testing how many bubbles a given solution can make. Here's their data:





Of course, you want to get all this data into JavaScript so you can write code to help analyze it. But that's a lot of values. How are you going to construct your code to handle all these values?

How to represent multiple values in

JavaScript

You know how to represent single values like strings, numbers, and booleans with JavaScript, but how do we represent *multiple* values, like all the bubble factor scores from the ten bubble solutions? To do that, we use JavaScript *arrays*. An array is a JavaScript type that can hold many values. Here's a JavaScript array that holds all the bubble factor scores:

```
let scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54];
```

Here are all ten values, grouped together into an array and assigned to the scores variable.

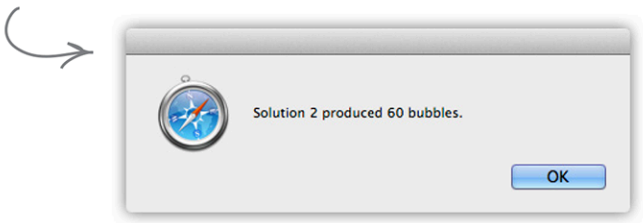
When we define an array like this, we say we're creating an array literal, because we're literally writing out the values.

You can treat all the values as a whole, or you can access the individual scores when you need to. Check this out:

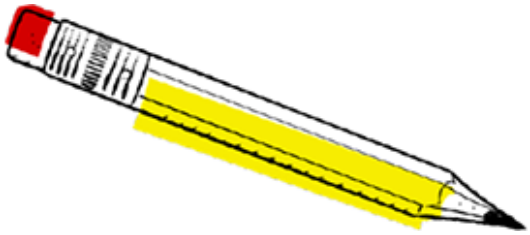
To access an item of the array we use this syntax: the variable name of the array followed by the index of the item, surrounded by square brackets.

Notice that arrays are zero-based. So, the first bubble solution is solution #0 and has the score in scores[0], and likewise, the third bubble solution is solution #2 and has the score in scores[2].

```
let solution2 = scores[2];
alert("Solution 2 produced " + solution2 + " bubbles.");
```

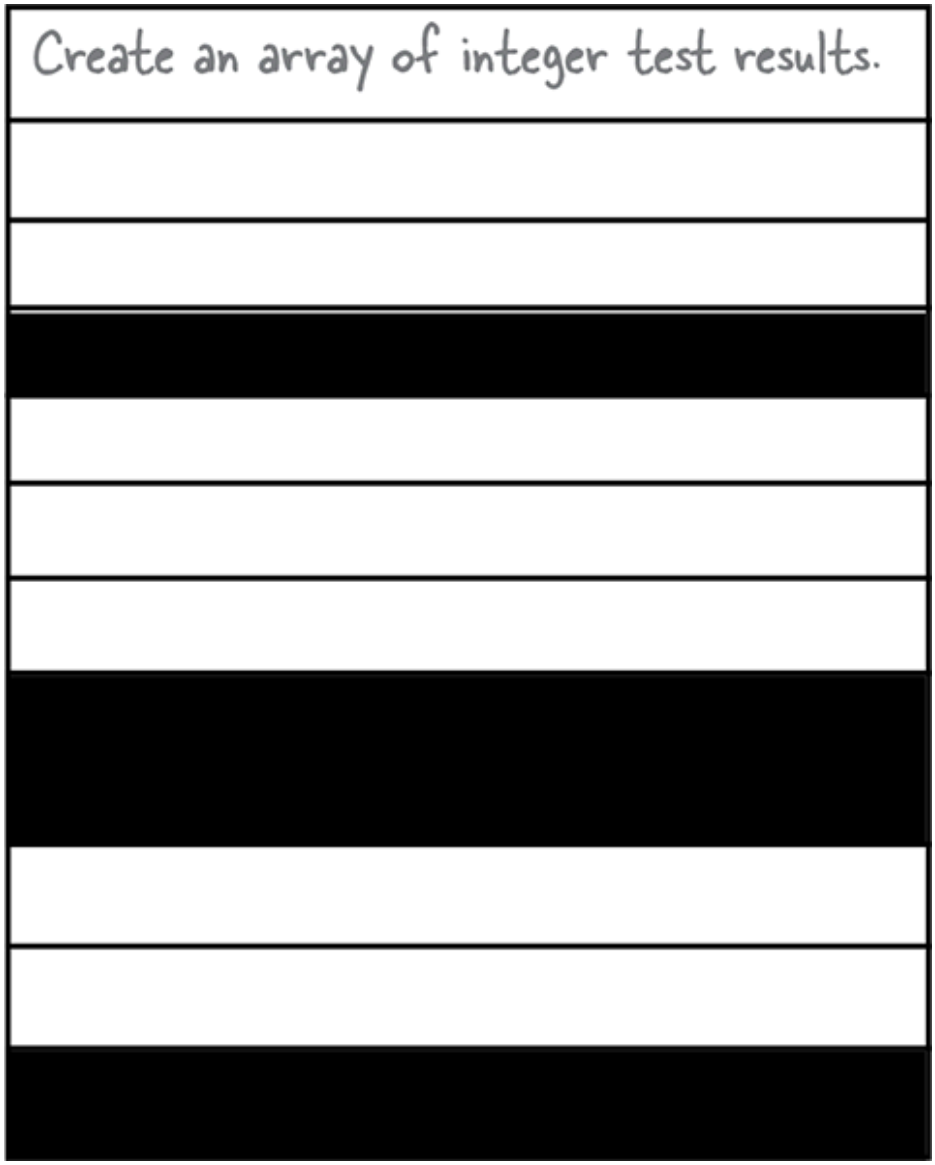


LOOK HOW EASY IT IS TO CREATE ARRAYS



You don’t know much about arrays yet, but we bet you can make some good guesses about how they work. Take a look at each line of code below and see if you can guess what it does. Write in your answers, and check our solution before you continue.

```
let testResults = [91, 84, 100];
let dogs = ["Fido", "Spot"];
let isBarking = [true, false];

let dogBarking1 = isBarking[0];
if (dogBarking1 == true) {
    console.log(dogs[0] + " is barking");
}

if (testResults[2] > 90) {
    console.log("You got an A!");
}
let solution2 = scores[2];
alert("Solution 2 produced " + solution2 + " bubbles.");
```



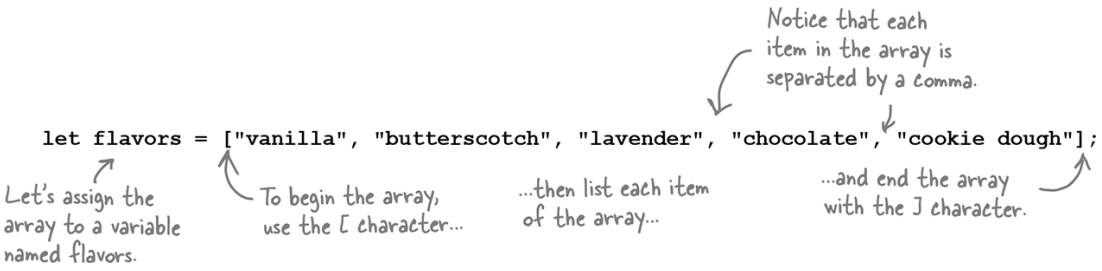
Solution in [“Look how easy it is to create arrays Solution”]

How arrays work

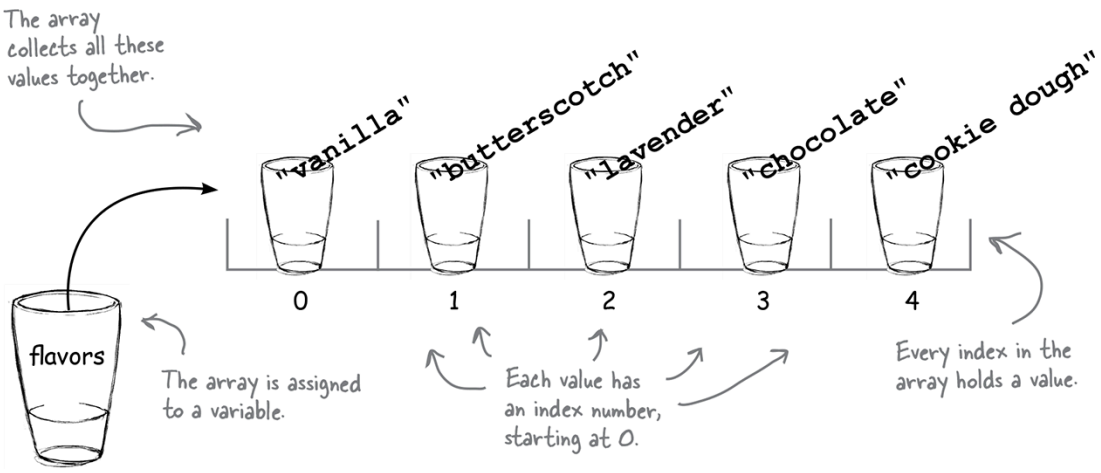
Before we get on to helping Bubbles-R-Us, let’s make sure we’ve got arrays down. As we said, you can use arrays to store *multiple* values (unlike variables that hold just one value, like a number or a string). Most often you’ll use arrays when you want to group together similar things, like bubble factor scores, ice cream flavors, daytime temperatures, or even the answers to a set of true/false questions. Once you have a bunch of values you want to group together, you can create an array that holds them, and then access those values in the array whenever you need them.

How to create an array

Let's say you wanted to create an array that holds ice cream flavors. Here's how you'd do that:

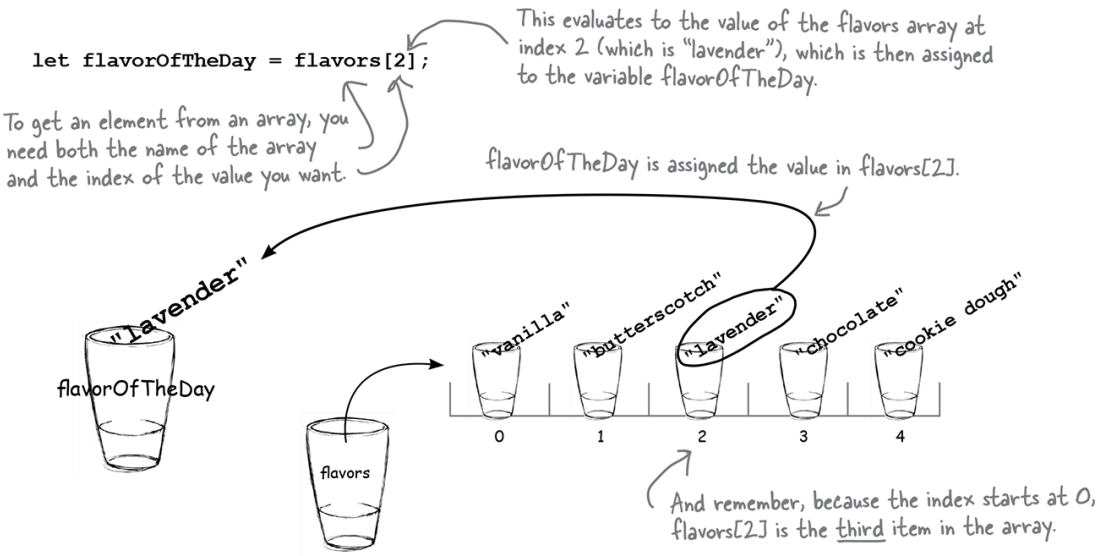


When you create an array, each item is placed at a location, or *index*, in the array. With the flavors array, the first item, "vanilla", is at index 0, the second, "butterscotch", is at index 1, and so on. Here's a way to think about an array:



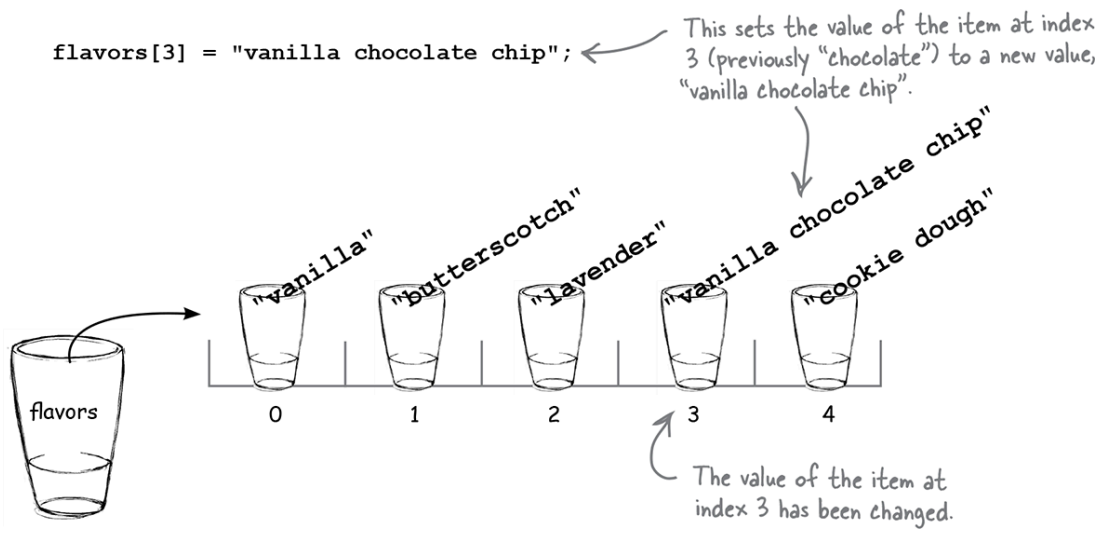
How to access an array item

Each item in the array has its own index, and that's your key to both accessing and changing the values in an array. To access an item, just follow the array variable name with an index, surrounded by square brackets. You can use that notation anywhere you'd use a variable:



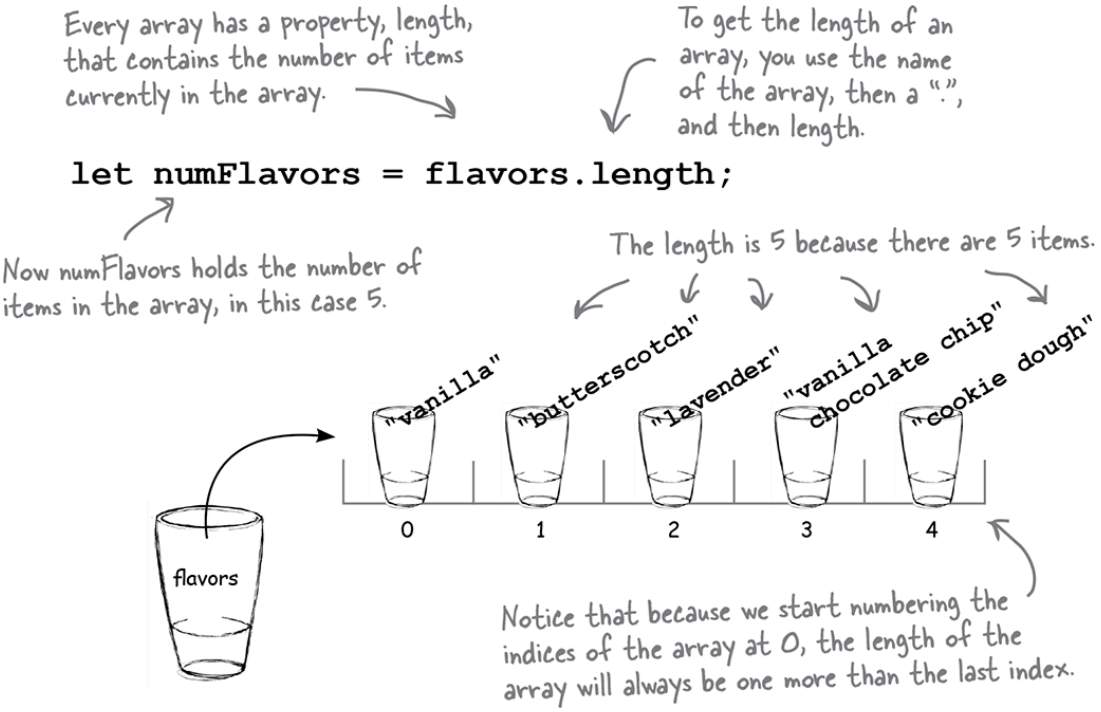
Updating a value in the array

You can also use the array index to change a value in an array. So, after this line of code, your array will look like this:

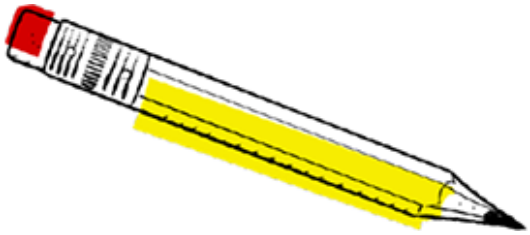


How big is that array anyway?

Say someone hands you a nice big array with important data in it. You know what's in it, but you probably won't know exactly how big it is. Luckily, every array comes with its own property, `length`. We'll talk more about properties and how they work in the next chapter, but for now, a property is just a value associated with an array. Here's how you use the `length` property:



SHARPEN YOUR PENCIL



The products array below holds the Jenn and Berry's ice cream flavors. The ice cream flavors were added to this array in the order of their creation. Finish the code to determine the *most recent* ice cream flavor they created.

```
let products = ["Choo Choo Chocolate", "Icy Mint", "Cake Batter", "Bubblegum"];
let last = _________________;
let recent = products[last];
```

→ **Solution in "Sharpen your pencil Solution"**


```
<!doctype html>
<html lang="en">
<head>
  <title>Phrase-O-Matic</title>
  <meta charset="utf-8">
  <script>
    function makePhrases() {
      let words1 = ["24/7", "multi-tier", "30,000 foot", "B-to-B", "win-win"];
      let words2 = ["empowered", "value-added", "oriented", "focused", "aligned"]
      let words3 = ["process", "solution", "tipping point", "strategy", "vision"]

      let rand1 = Math.floor(Math.random() * words1.length);
      let rand2 = Math.floor(Math.random() * words2.length);
      let rand3 = Math.floor(Math.random() * words3.length);

      let phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
      alert(phrase);
```

```
    }
    makePhrases();
  </script>
</head>
<body></body>
</html>
```

NOTE

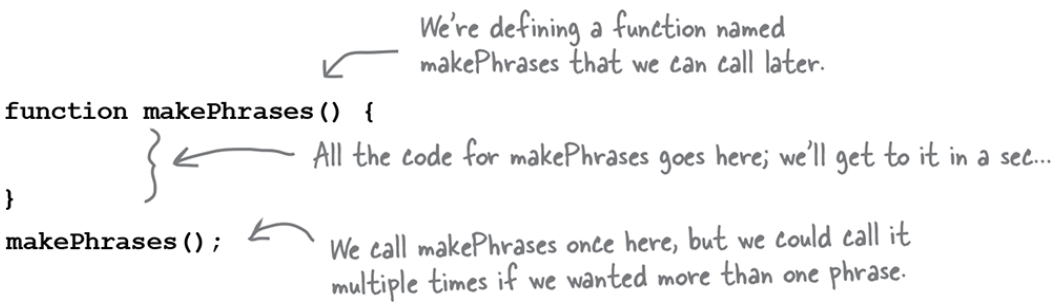
Check out this code for the hot new Phrase-O-Matic app and see if you can figure out what it does before you go on...

You didn't think our serious business application from Chapter 1 was serious enough? Fine. Try this one, if you need something to show the boss.

The Phrase-O-Matic

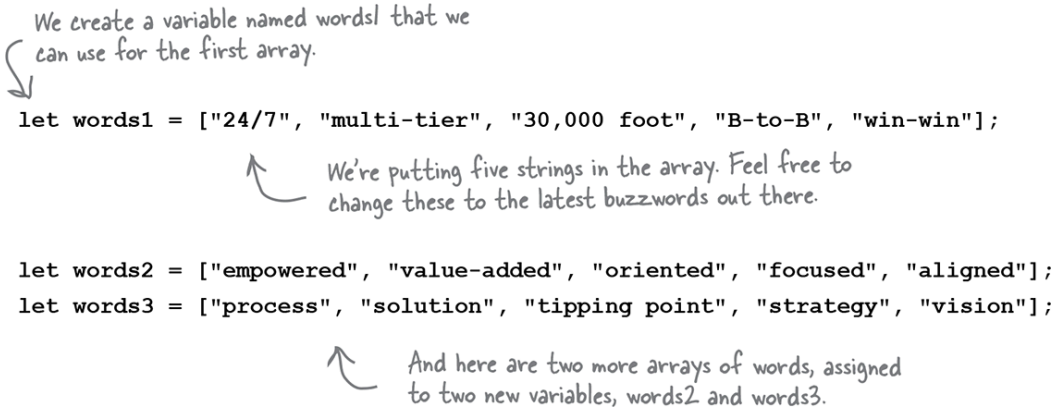
We hope you figured out this code is the perfect tool for creating your next start-up marketing slogan. It has created winners like "B-to-B value-added strategy" and "24/7 empowered process" in the past, and we have high hopes for more winners in the future. Let's see how this thing really works:

1. First, we define the `makePhrases` function, which we can call as many times as we want to generate the phrases we want.

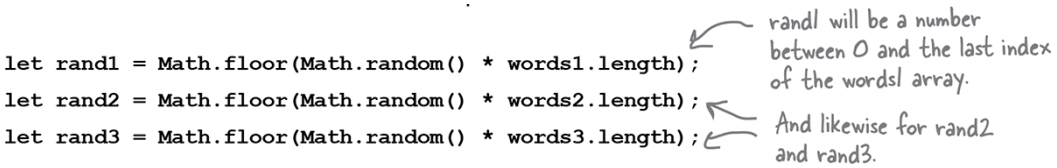


2. With that out of the way, we can write the code for the `makePhrases` function. Let's start by setting up three arrays. Each will

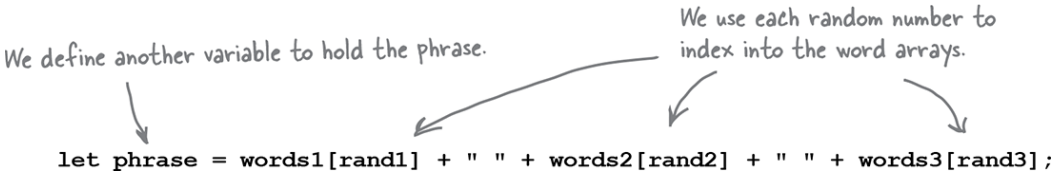
hold words that we'll use to create the phrases. In the next step, we'll pick one word at random from each array to make a three-word phrase.



③ Now we generate three random numbers, one for each of the three random words we want to pick to make a phrase. Remember from Chapter 2 that Math.random generates a number between 0 and 1 (not including 1). If we multiply that by the length of the array and use Math.floor to truncate the number, we get a number between 0 and one less than the length of the array.



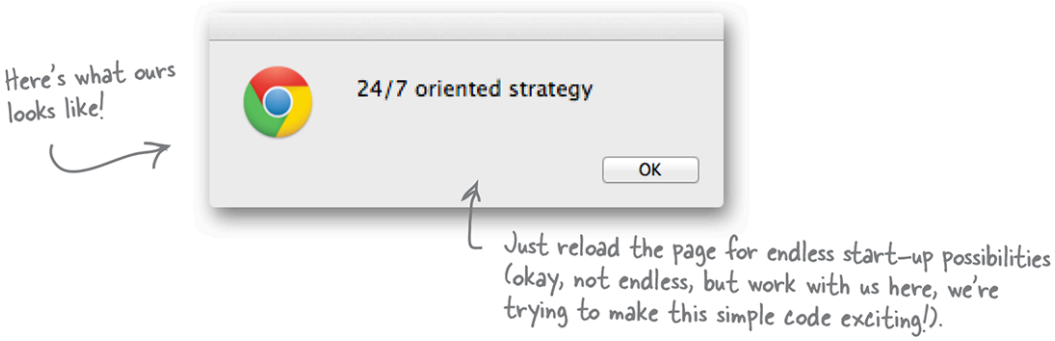
④ Now we create the slick marketing phrase by taking each randomly chosen word and concatenating them all together, with nice spaces in between for readability.



⑤ We're almost done; we have the phrase, now we just have to display it. We're going to use `alert` (but you could also use `console.log` and check your browser's console window).

```
alert(phrase);
```

⑥ Okay, finish that last line of code, have one more look over it all, and feel that sense of accomplishment before you load it into your browser. Give it a test drive and enjoy the phrases.



THERE ARE NO DUMB QUESTIONS

Q: Does the order of items in an array matter?

A: Most of the time, yes, but it depends. In the Bubbles-R-Us scores array, the ordering matters a lot, because the index of the score in the array tells us which bubble solution got that score—bubble solution 0 got score 60, and that score is stored at index 0. If we mixed up the scores in the array, then we'd ruin the experiment! However, in other cases, the order may not matter. For instance, if you're using an array just to keep a list of randomly selected words and you don't care about the order, then it doesn't matter which order they're in in the array. But if you later decide you want the words to be in alphabetical order, then the order will matter. So it really depends on how you're using the array. You'll probably find that ordering matters more often than not when you use an array.

Q: How many things can you put into an array?

A: Theoretically, as many as you want. Practically, however, the number is limited by the memory on your computer. Each array item takes up a little bit of space in memory. Remember that JavaScript runs in a browser, and that browser is one of many programs running on your computer. If you keep adding items to an array, eventually you'll run out of memory space. However, depending on the kinds of items you're putting in your array, the maximum number of items you can put into an array is probably in the many thousands, if not millions, which you're unlikely to need most of the time. And keep in mind that the more items you have the slower your program will run, so you'll want to limit your arrays to reasonable sizes—say, a few hundred—most of the time.

Q: Can you have an empty array?

A: You can, and in fact, you'll see an example of using an empty array shortly. To create an empty array, just write:

```
let emptyArray = [ ];
```

If you start with an empty array, you can add things to it later.

Q: So far we've seen strings and numbers in an array; can you put other things in arrays too?

A: You can; in fact, you can put just about any value you'll find in JavaScript in an array, including numbers, strings, booleans, other arrays, and even objects (we'll get to this later).

Q: Do all the values in an array have to be the same type?

A: No, they don't, although typically we *do* make the values all of the same type. Unlike in many other languages, there is no requirement in JavaScript that all the values in an array be of the same type. However, if you mix up the types of the values in an array, you need to be extra careful when using those values. Here's why: let's say you have an array with the values [1, 2, "fido", 4, 5]. If you then write code that checks to see if the values in the array are greater than, say, 2, what happens when you check to see if "fido" is greater than 2? To make sure you aren't doing something that doesn't make sense, you'd have to check the type of each of the values before you used it in the rest of your code. It's certainly possible to do this (and you'll see how later in the book), but in general, it's a lot easier and safer if you just use the same type for all the values in your arrays.

Q: What happens if you try to access an array with an index that is too big or too small (like less than 0)?

A: If you have an array, like this one:

```
let a = [1, 2, 3];
```

and you try to access a[10] or a[-1], in either case, you'll get the result undefined. So, either you'll want to make sure you're using only valid indices to access items in your array, or you'll need to check that the value you get back is not undefined.

Q: So, I can see how to get the first item in an array using index 0. But how would I get the last item in an array? Do I always have to know precisely how many items are in my array?

A: You can use the length property to get the last item of an array. You know that length is always one greater than the last index of the array, right? So, to get the last item in the array, you can write:

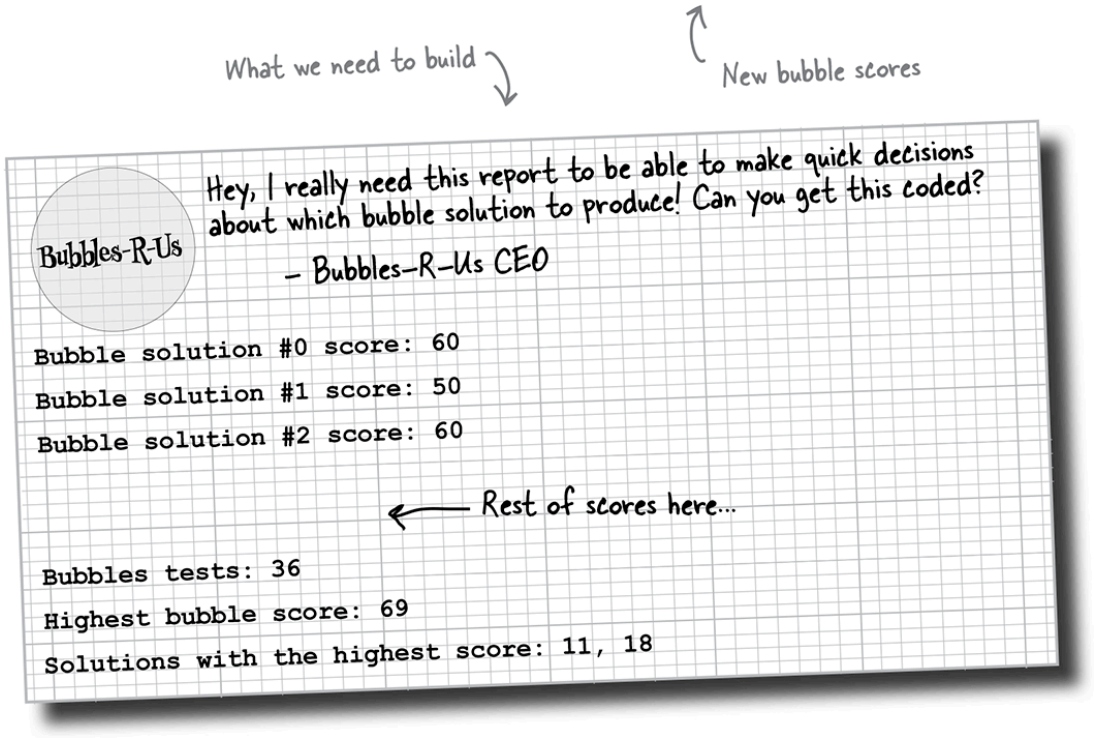
```
myArray[myArray.length - 1];
```

JavaScript gets the length of the array, subtracts one from it, and then gets the item at that index number. So if your array has 10 items, it will get the item at index 9, which is exactly what you want. You'll use this trick all the time to get the last item in an array when you don't know exactly how many items are in it.

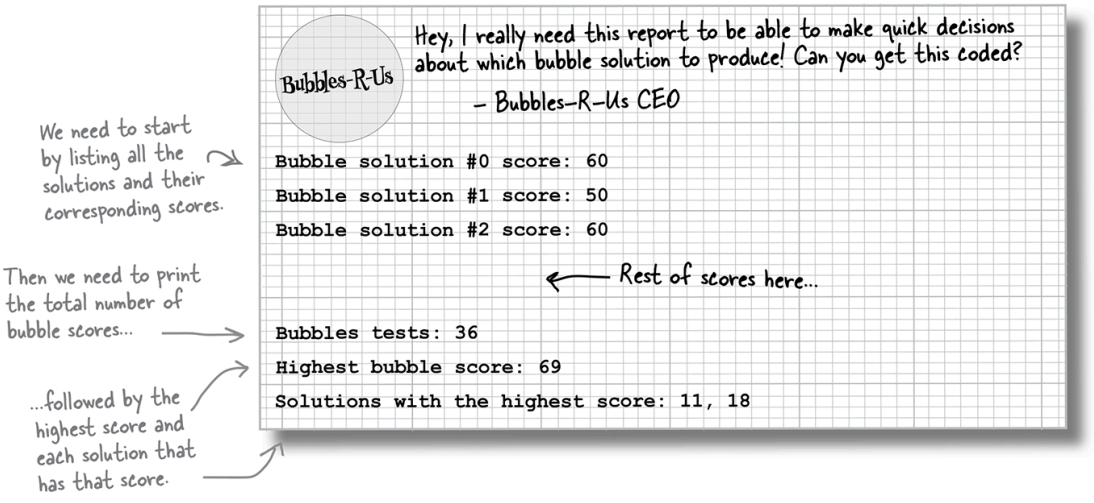
Meanwhile, back at Bubbles-R-Us...


```
let scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
```

```
                34, 55, 51, 52, 44, 51,
                69, 64, 66, 55, 52, 61,
                46, 31, 57, 52, 44, 18,
                41, 53, 55, 61, 51, 44];
```



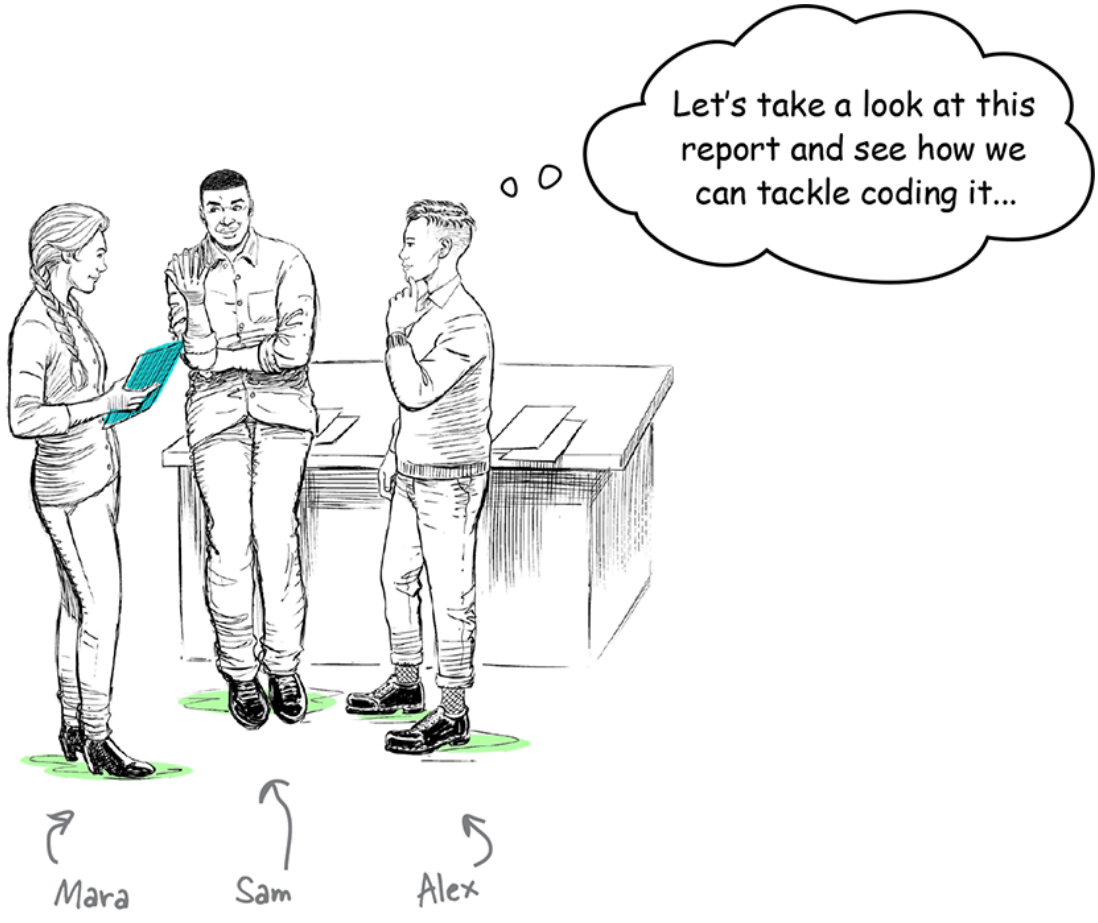
Let's take a closer look at what the CEO is looking for:



BRAIN POWER

Take some time to sketch out your ideas of how you'd create this little bubble score report. Take each item in the report separately and think about how you'd break it down and generate the right output. Make your notes here.

Cubicle Conversation



Mara: The first thing we need to do is display every score along with its solution number.

Alex: And the solution number is just the index of the score in the array, right?

Mara: Oh, yeah, that's totally right.

Sam: Slow down a sec. So we need to take each score, print its index, which is the bubble solution number, and then print the corresponding score.

Mara: You've got it, and the score is just the corresponding value in the array.

Alex: So, for bubble solution #10, its score is just `scores[10]`.

Mara: Right.

Sam: Okay, but there are a lot of scores. How do we write code to output all of them?

Mara: Iteration, my friend.

Sam: Oh, you mean like a while loop?

Mara: Right, we loop through all the values from zero to the length...oh, I mean the length minus one, of course.

Sam: This is starting to sound very doable. Let's write some code; I think we know what we're doing.

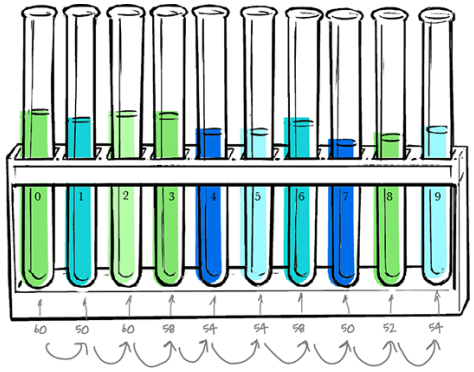
Mara: That works for me! Let's do it, and then we'll come back to the rest of the report.

How to iterate over an array

Your goal is to produce some output that looks like this:

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
.
.
.
Bubble solution #35 score: 44
```

Scores 3 through 34 will be here...we're saving some trees (or bits, depending on which version of the book you have).



We'll do that by outputting the score at index zero, and then we'll do the same for indices one, two, three and so on, until we reach the last index in the array. You already know how to use a while loop. Let's see how we can use that to output all the scores:

NOTE

And then we'll show you a better way in a sec...

```
let scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

let i = 0;

while (i < scores.length) {

  let output = "Bubble solution #" + i + " score: " + scores[i];

  console.log(output);

  i = i + 1;

}
```

Create a variable to keep track of the current index.

And keep looping while the index is less than the length of the array.

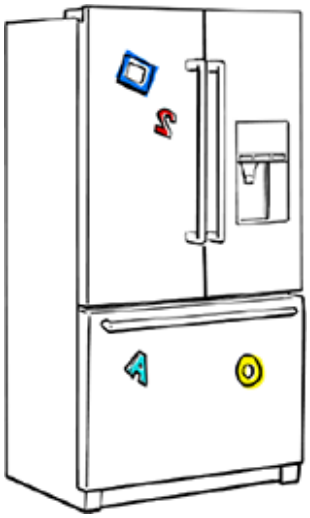
We're using this variable to create a string to output.

Then, create a string to use as a line of output that includes the bubble solution number (which is just the array index) and the score.

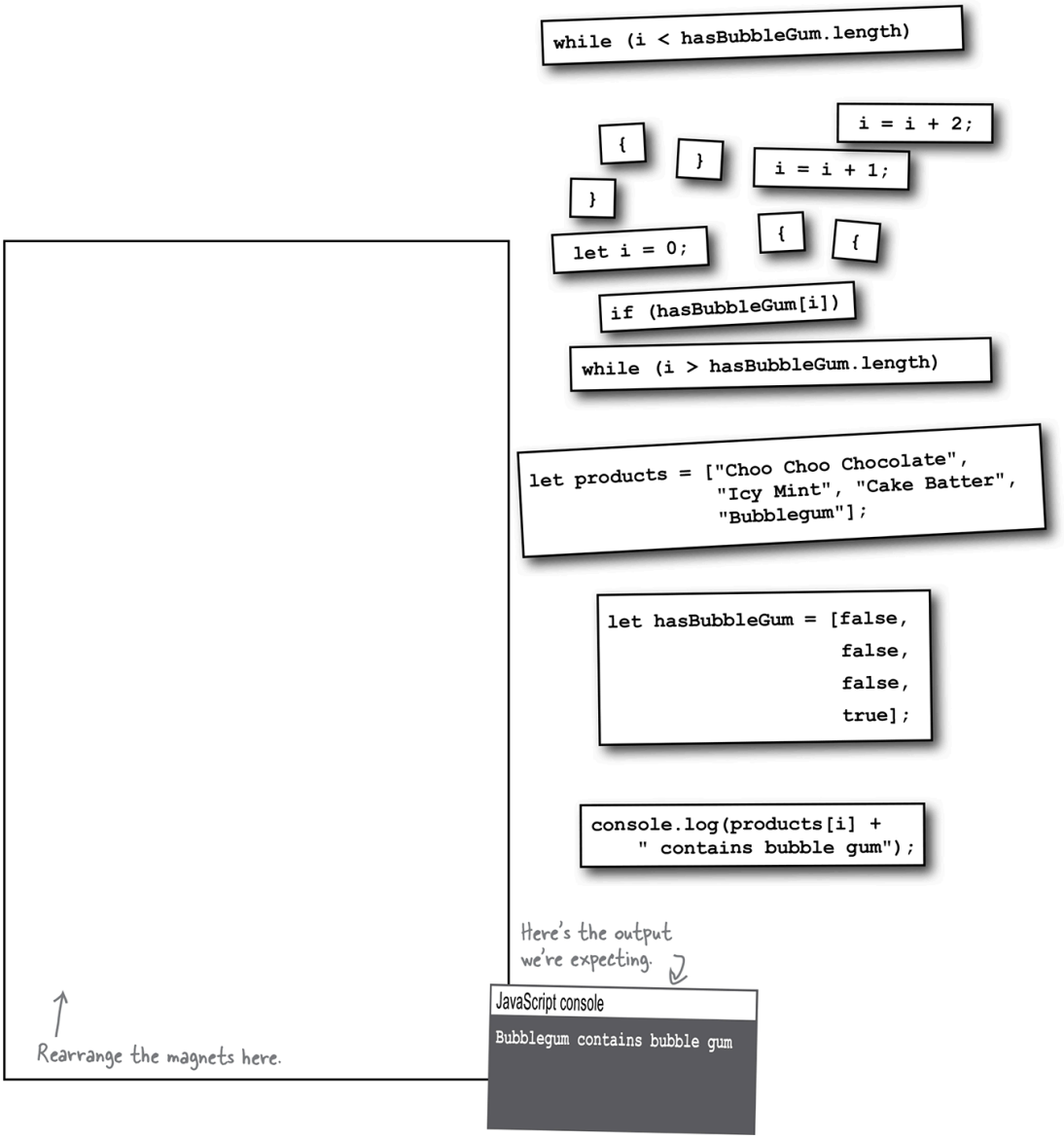
Next, use console.log to output the string.

And finally, increment the index by one before looping again.

CODE MAGNETS



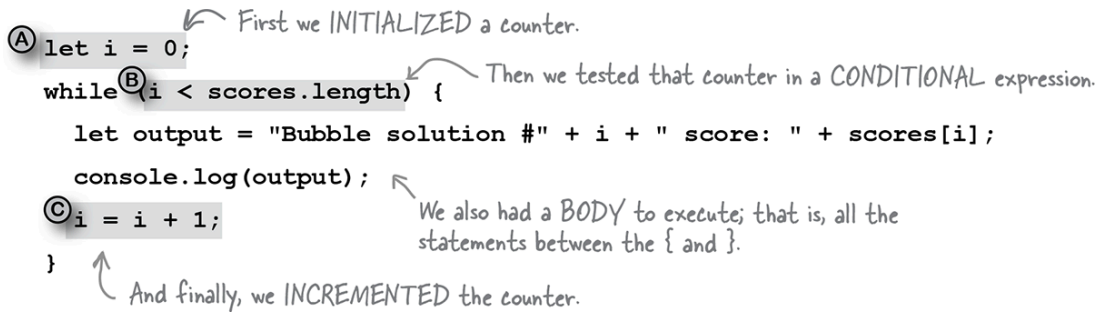
We've got code for testing to see which ice cream flavors have bubblegum pieces in them. We had all the code nicely laid out on our fridge using fridge magnets, but the magnets fell on the floor. It's your job to put them back together. Be careful; a few extra magnets got mixed in. Check your answer at the end of the chapter before you go on.



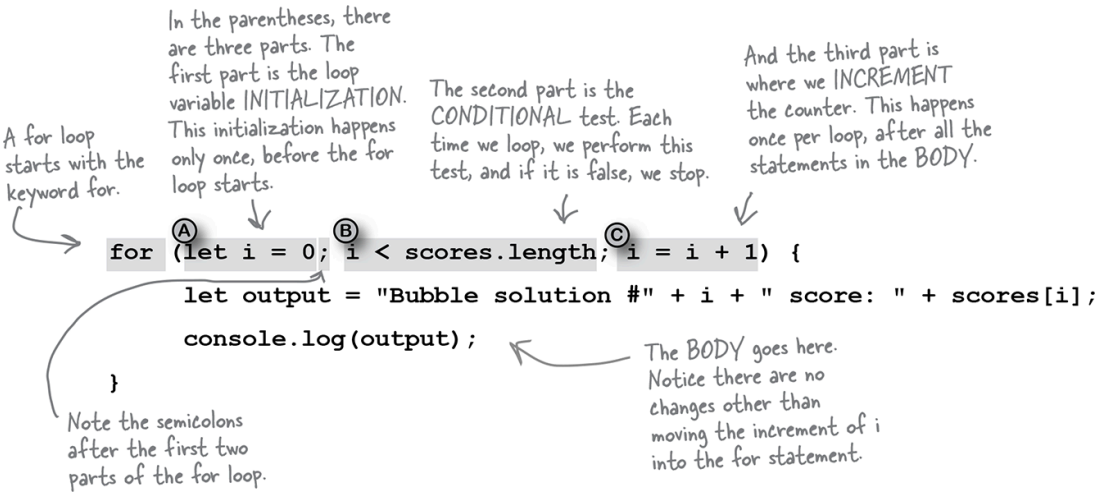
Solution in "Code Magnets Solution"

But wait, there's a better way to iterate over an array

We should really apologize. We can't believe it's already Chapter 4 and we haven't even introduced you to the **for loop**. Think of the for loop as the while loop's cousin. The two basically do the same thing, except the for loop is usually a little more convenient to use. Check out the while loop we just used, and we'll see how that maps into a for loop.



Now let's look at how the for loop makes all that so much easier:



Good catch, and great question.

Yes, a for loop statement creates a block just like a while loop statement does. Every statement between the curly braces is in the for loop block:

This curly brace starts the block...

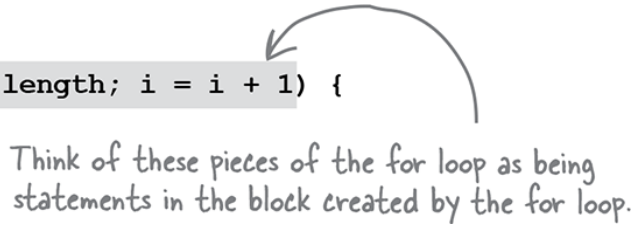
```
for (let i = 0; i < scores.length; i = i + 1) {
   let output = "Bubble solution #" + i + " score: " + scores[i];
   console.log(output);
}
```

Every statement between the curly braces is in the block created by the for loop.

...and this curly brace ends the block.

The for loop statement is interesting because you can also think of the three parts of the statement between the parentheses as being part of the block:

```
for (let i = 0; i < scores.length; i = i + 1) {
   ...
}
```



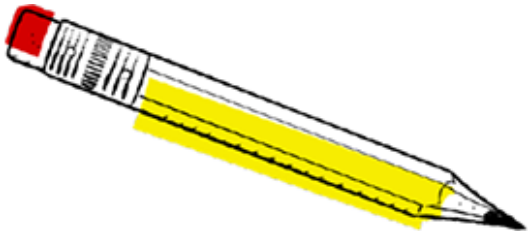
That means the declaration of the `i` loop variable is local to the block, meaning the value of `i` will not be visible outside the block. So you would **not** be able to do this, for instance:

```
for (let i = 0; i < scores.length; i = i + 1) {
   let output = "Bubble solution #" + i + " score: " + scores[i];
   console.log(output);
}
console.log("There are " + i + " bubble solutions in scores");
```

NOTE

If you try this at home, you'll see an error in the console: Uncaught ReferenceError: i is not defined.

SHARPEN YOUR PENCIL



Rewrite your fridge magnet code (from three pages back) so that it uses a for loop instead of a while loop. If you need a hint, refer to each piece of the while loop in "But wait, there's a better way to iterate over an array" and see how it maps to the corresponding location in the for loop.

```
let products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];

let hasBubbleGum = [false,
                    false,
                    false,
                    true];

let i = 0;
while (i < hasBubbleGum.length)
{
if (hasBubbleGum[i])
{
console.log(products[i] +
    " contains bubble gum");
}
i = i + 1;
}
```

Your code goes here.

Solution in "Sharpen your pencil Solution"


```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Bubble Factory Test Lab</title>
  <script>
    let scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];

    for (let i = 0; i < scores.length; i = i + 1) {

        let output = "Bubble solution #" + i +
                     " score: " + scores[i];

        console.log(output);

    }

  </script>
</head>
<body></body>
</html>
```

We've got the standard HTML stuff here for a web page. We don't need much; just enough to create a script.

Here's our bubble scores array.

Here's the for loop we're using to iterate through all the bubble solution scores.

Each time through the loop, we create a string with the value of i, which is the bubble solution number, and scores[i], which is the score that bubble solution got.

(Also notice we split the string up across two lines here. That's okay as long as you don't create a new line in between the quotes that delimit a string. Here, we did it after a concatenation operator (+), so it's okay. Be careful to type it in exactly as you see here.)

Then we display the string in the console. And that's it! Time to run this report.

Test drive the bubble report

Save this file as "bubbles.html" and load it into your browser. Make sure you've got the console visible (you might need to reload the page if you activate the console after you load the page), and check out the brilliant report you just generated for the Bubbles-R-Us CEO.

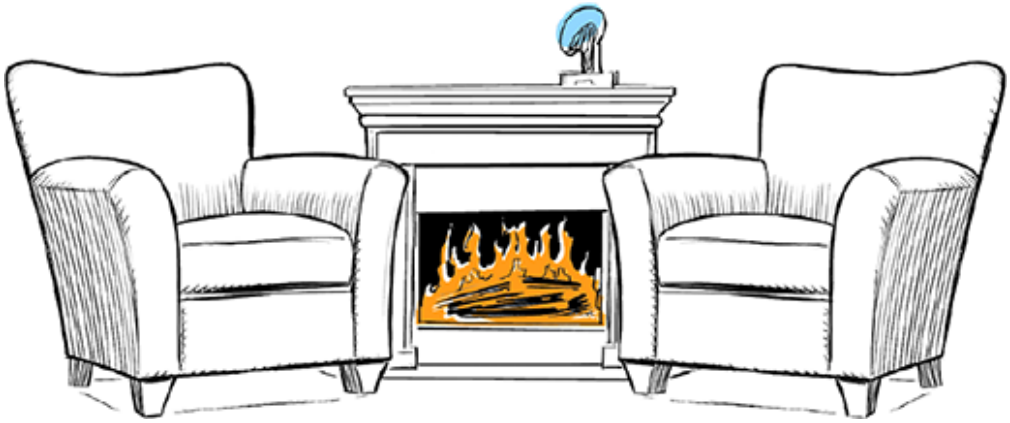
JavaScript console

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
Bubble solution #3 score: 58
Bubble solution #4 score: 54
Bubble solution #5 score: 54
Bubble solution #6 score: 58
Bubble solution #7 score: 50
Bubble solution #8 score: 52
Bubble solution #9 score: 54
Bubble solution #10 score: 48
Bubble solution #11 score: 69
Bubble solution #12 score: 34
Bubble solution #13 score: 55
Bubble solution #14 score: 51
Bubble solution #15 score: 52
Bubble solution #16 score: 44
Bubble solution #17 score: 51
Bubble solution #18 score: 69
Bubble solution #19 score: 64
Bubble solution #20 score: 66
Bubble solution #21 score: 55
Bubble solution #22 score: 52
Bubble solution #23 score: 61
Bubble solution #24 score: 46
Bubble solution #25 score: 31
Bubble solution #26 score: 57
Bubble solution #27 score: 52
Bubble solution #28 score: 44
Bubble solution #29 score: 18
Bubble solution #30 score: 41
Bubble solution #31 score: 53
Bubble solution #32 score: 55
Bubble solution #33 score: 61
Bubble solution #34 score: 51
Bubble solution #35 score: 44
```

Just what the CEO ordered.

It's nice to see all the bubble scores in a report, but it's still hard to find the highest score. We need to work on the rest of the report requirements to make it a little easier to find the winner.

FIRESIDE CHATS



Tonight's talk: **The WHILE and FOR loop answer the question "Who's more important?"**

The WHILE loop:	The FOR loop:
What, are you kidding me? Hello? I'm the *general* looping construct in JavaScript. I'm not married to looping with a silly counter. I can be used with any type of conditional. Did anyone notice I was taught first in this book?	
	I don't appreciate that tone.
And that's another thing, have you noticed that the FOR loop has no sense of humor? I mean, if we all had to do skull-numbing iteration all day I guess we'd all be that way...	
	Cute. But have you noticed that nine times out of ten, coders use FOR loops?
Oh, I don't think that could possibly be true.	
	Not to mention, doing iteration over, say, an array that has a fixed number of items with a WHILE loop is just a bad, clumsy practice.
This book just showed that FOR and WHILE loops are pretty much the same thing, so how could that be?	
	Ah, so you admit we're more equal than you let on, huh? I'll tell you why...
	When you use a WHILE loop you have to initialize your counter and increment your counter in separate statements. If, after lots

	of code changes, you accidentally moved or deleted one of these statements, well, then things could get ugly. But with a FOR loop, everything is packaged right in the FOR statement for all to see and with no chance of things getting changed or lost.
Well, isn't that nice and neat of you. Hey, most of the iteration I see doesn't even include counters; it's stuff like: `while (answer != "forty-two")` Try that with a FOR loop!	
	Okay: `for (;answer != "forty-two";)`
Hah, I can't believe that even works.	
	Oh, it does.
Lipstick on a pig.	
	So that's all you got? You're better when you've got a general conditional?
Not only better, prettier.	
	Oh, I didn't realize this was a beauty contest as well.

It's that time again... can we talk about

your verbosity?

You've been writing lots of code that looks like this:

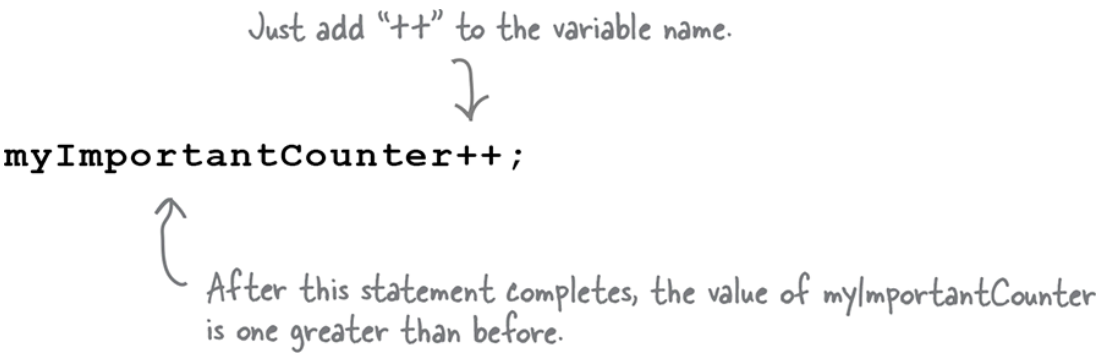
Assume myImportantCounter contains a number, like zero.

Here we're taking the variable and incrementing it by one.

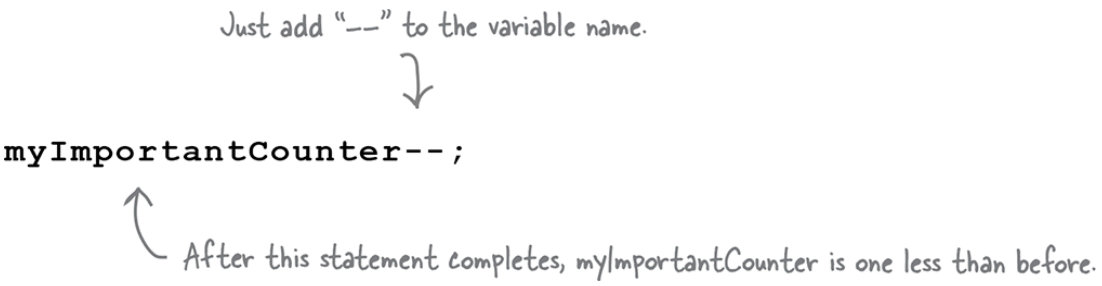
```
myImportantCounter = myImportantCounter + 1;
```

After this statement completes, the value of myImportantCounter is one greater than before.

In fact, this kind of statement is so common there's a shortcut for it in JavaScript. It's called the **post-increment** operator, and despite its fancy name, it's quite simple. Using the post-increment operator, we can replace the above line of code with this:



Of course, it just wouldn't feel right if there wasn't a **post-decrement** operator as well. You can use the post-decrement operator on a variable to reduce its value by one, like this:



And why are we telling you this now? Because these are commonly used with `for` statements. Let's clean up our code a little using the post-increment operator...

Redoing the for loop with the post-increment operator

Let's do a quick rewrite and test to make sure the code works the same as before:

```
    let scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];
    for (let i = 0; i < scores.length; i++) {
        let output = "Bubble solution #" + i +
                        " score: " + scores[i];
```

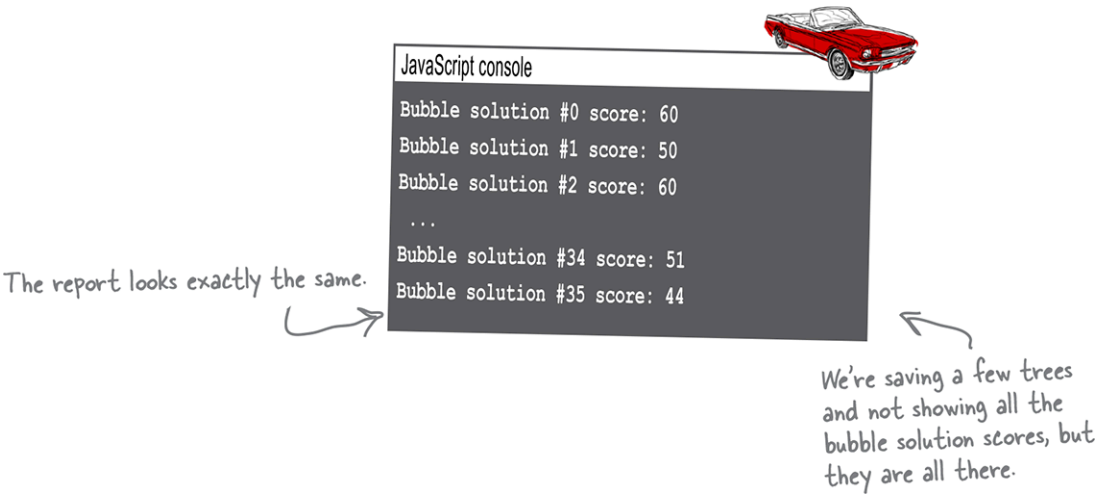
```
        console.log(output);
    }
```

NOTE

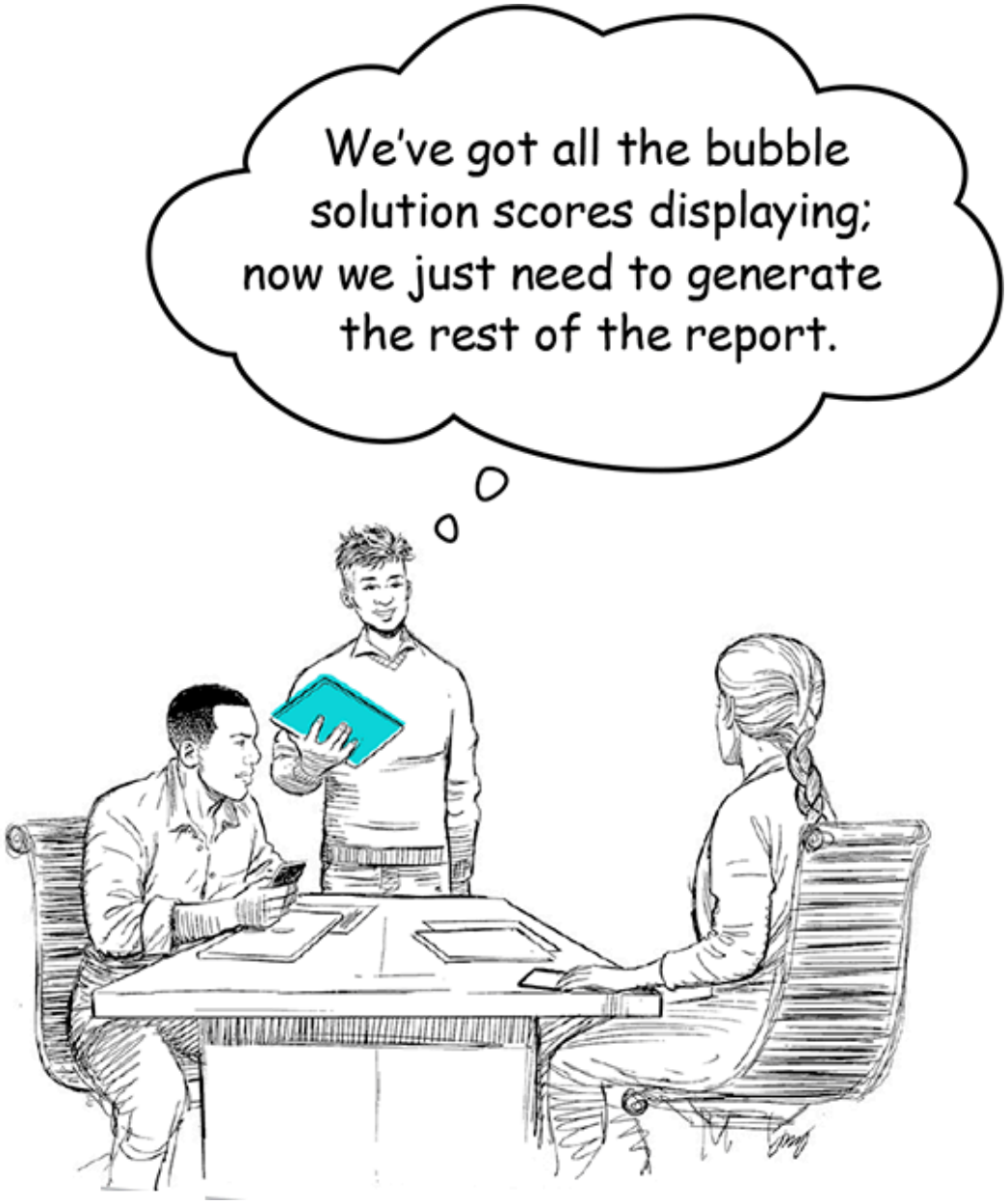
All we've done is update where we increment the loop variable with the post-increment operator.

Another quick test drive

Time to do a quick test drive to make sure the change to use the post-increment operator works. Save your file, "bubbles.html", and reload. You should see the same report you saw before.



Cubicle Conversation continued...



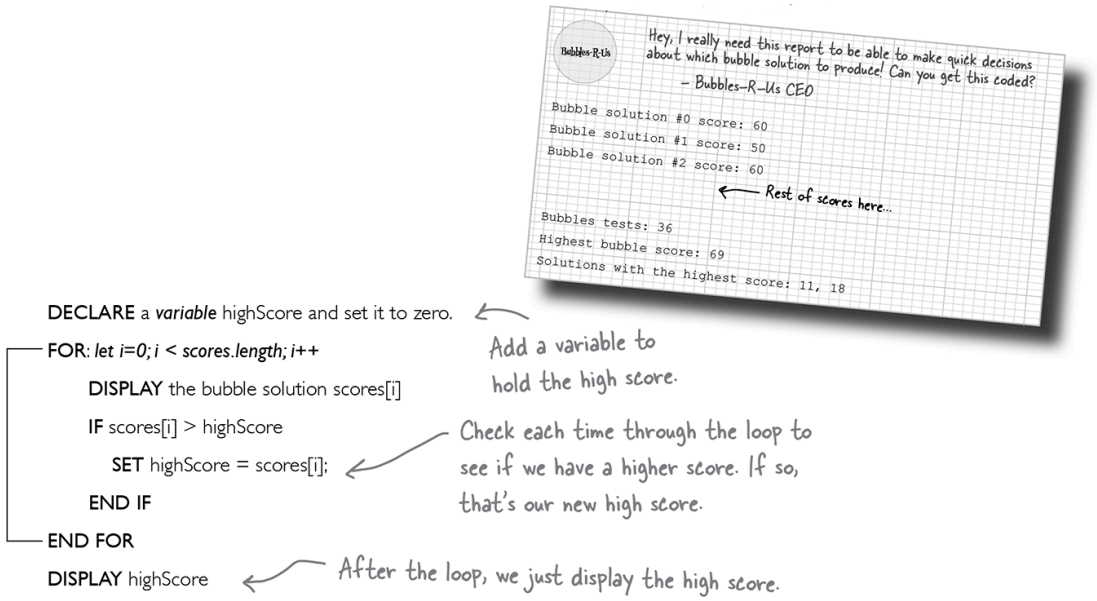
Mara: Right, and the first thing we need to do is determine the total number of bubble tests. That's easy; it's just the length of the scores array.

Alex: Oh, right. We've got to find the highest score too, and then the solutions that have the highest score.

Mara: Yeah, that last one is going to be the toughest. Let's work out finding the highest score first.

Alex: Sounds like a good place to start.

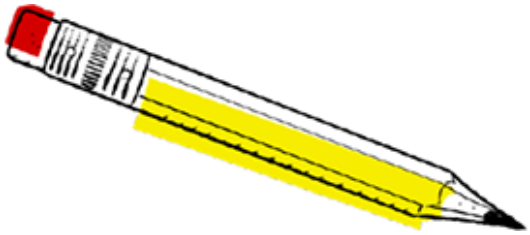
Mara: To do that, I think we just need to maintain a highest score variable that keeps track as we iterate through the array. Here, let me write some pseudocode:



Alex: Oh nice; you did it with just a few lines added to our existing code.

Mara: Yup. Each time through the array we look to see if the current score is greater than `highScore`, and if so, that's our new high score. Then, after the loop ends we just display the high score.

SHARPEN YOUR PENCIL



Go ahead and implement the pseudocode on the previous page to find the highest score by filling in the blanks in the code below. Once you're done, give it a try in the browser by updating the code in "bubbles.html" and reloading the page. Check the results in the console, and fill in the blanks in our console display below with the number of bubble tests and the highest score. Check your answer at the end of the chapter before you go on.

```
let scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

let highScore = _____;
for (let i = 0; i < scores.length; i++) {
    let output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
    if (_____________ > highScore) {
        _____________ = scores[i];
    }
}
console.log("Bubbles tests: " + _____________);
console.log("Highest bubble score: " + _____________);
```

NOTE

Fill in the blanks to complete the code here...

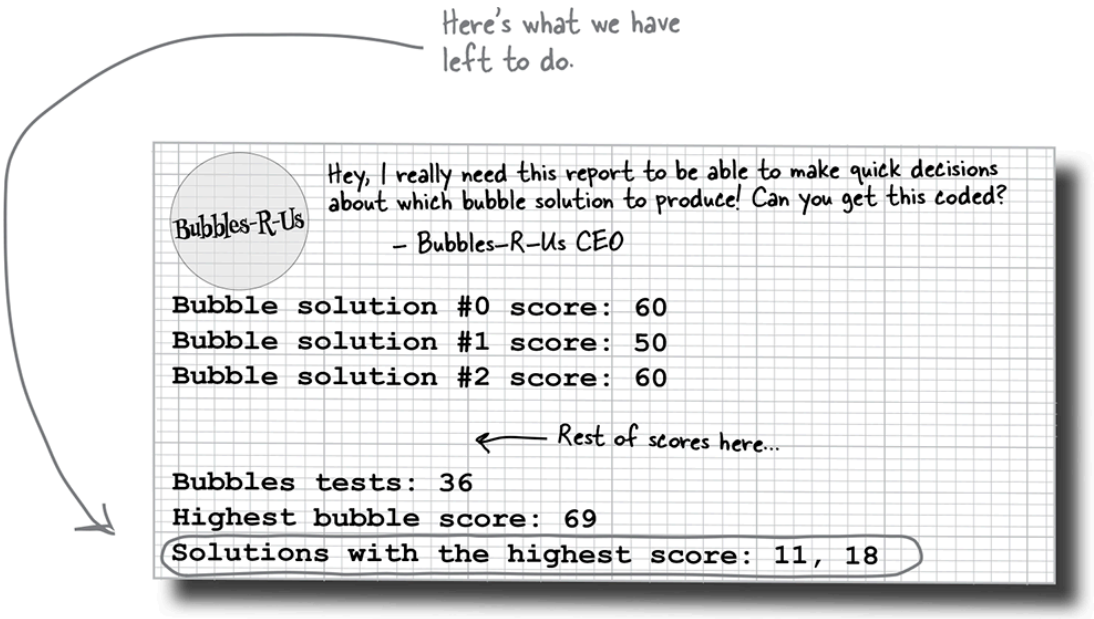
...and then fill in the blanks showing the output you get in the console.

```
JavaScript console
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
Bubbles tests: _____
Highest bubble score: _____
```

→ **Solution in "Sharpen your pencil Solution"**

"More than one"...hmmm. When we need to store more than one thing, what do we use? An array, of course. So can we iterate through the scores array looking for only scores that match the highest score, and then add them to an array that we can later display in the report? You bet we can,

but to do that we'll have to learn how to create a brand new empty array, and then figure out how to add new elements to it.



Creating an array from scratch (and adding to it)

Before we take on finishing this code, let's get a sense of how to create a new array and add new items to it. You already know how to create an array with values, like this:

```
let genres = ["80s", "90s", "Electronic", "Folk"];
```

Remember, this is called an array literal, because we're literally writing out what goes in the array.

But you can also omit the initial items and just create an empty array:

```
let genres = [];
```

A new array, all ready to go with no items and a length of zero.

This is an array literal too, it just doesn't have anything in it (yet).

And you already know how to add new values to an array. To do that you just assign a value to an item at an index, like this:

```
let genres = [];
genres[0] = "Rockabilly";
genres[1] = "Ambient";
let size = genres.length;
```

A new array item is created that holds the string "Rockabilly".

And a second array item is created that holds the string "Ambient".

And here, size holds the value 2, the length of the array.

When adding new items this way, you have to be careful about which index you're adding. Otherwise, you might create a **sparse** array, which is an array with "holes" in it (like an array with values at 0 and 2, but no value at 1). Having a sparse array isn't necessarily a bad thing, but it does require special attention. There's another way to add new items without worrying about the index, and that's `push`. Here's how it works:

```
let genres = [];
genres.push("Rockabilly");
genres.push("Ambient");
let size = genres.length;
```

Creates a new item in the index at the end of the array (which happens to be 0) and sets its value to "Rockabilly".

Creates another new item at the end of the array (in this case, at index 1) and sets the value to "Ambient".

THERE ARE NO DUMB QUESTIONS

Q: The for statement contains a variable declaration and initialization in the first part of the statement. You said we should put our variable declarations at the top. So, what gives?

A: Yes, putting your variable declarations at the top (of your file, if they are global, or of your function if they are local) is a good practice. However, there are times when it makes sense to declare a variable right where you're going to use it, and a for statement is one of those times. Typically, you use a loop variable, like i, just for iterating, and once the loop is done, you're done with that variable. In that case, just declaring it right in the for statement keeps things tidy. And remember that the scope of i is limited to the block of the for loop statement, so that means you could use i again later on in another for loop if you wanted to.

We're doing the same with the variable output too: we're declaring it inside the for loop block, because we don't need it anywhere else.

Q: What does the syntax myarray.push(value) actually mean?

A: Well, we've been keeping a little secret from you: in JavaScript, an array is actually a special kind of object. As you'll learn in the next chapter, an object can have functions associated with it that act on the object. So, think of push as a function that can act on myarray. In this case, what that function does is add a new item to the array—the item that you pass as an argument to push. So, if you write:

```
genres.push("Metal");
```

you're calling the function push and passing it a string argument, "Metal". The push function takes that argument and adds it as a new value on the end of the genres array.

When you see:

```
myarray.push(value)
```

just think, "I'm pushing a new value onto the end of my array."

Q: Can you say a little more about what a sparse array is?

A: A sparse array is just an array that has values at only a few indices and no values in between. You can create a sparse array easily, like this:

```
let sparseArray = [ ];
sparseArray[0] = true;
spraseArray[100] = true;
```

In this example, the sparseArray has only two values, both true, at indices 0 and 100. The values at all the other indices are undefined. The length of the array is 101 even though there are only two values.

Q: Say I have an array of length 10, and I add a new item at index 10000. What happens with indices 10 through 9999?

A: All those array indices get the value undefined. If you remember, undefined is the value assigned to a variable that you haven't initialized. So, think of this as if you're creating 9,989 variables, but not initializing them. Remember that all those variables take up memory in your computer, even if they don't have a value, so make sure you have a good reason to create a sparse array.

Q: So, if I'm iterating through an array, and some of the values are undefined, should I check to make sure before I use them?

A: If you think your array might be sparse, or even have just one undefined value in it, then yes, you should probably check to make sure that the value at an array index is not undefined before you use it. If all you're doing is displaying the value in the console, then it's no big deal, but it's much more likely that you'll actually want to use that value somehow, perhaps in a calculation of some kind. In that case, if you try to use undefined, you might get an error, or at the very least some unexpected behavior. To check for undefined, just write:

```
if (myarray[i] == undefined) {
  ...
}
```

Notice there are no quotes around undefined (because it's not a string, it's a value).

Q: All the arrays we've created so far have been literal. Is there another way to create an array?

A: Yes. You may see this syntax:

```
let myarray = new Array(3);
```

What this does is create a new array with three empty spots in it (that is, an array with length 3, but no values yet). Then you can fill them, just like you normally would, by providing values for myarray at indices 0, 1, and 2. Until you add values yourself, the values in myarray are undefined.

An array created this way is just the same as an array literal, and in practice, you'll find yourself using the literal syntax more often. Don't worry about the details of the syntax above for now (like "new" and why Array is capitalized); we'll get to all that later!

Mara: Yes, we'll start with an empty array to hold the solutions with the highest score and add each solution that has that high score to it one at a time as we iterate through the scores array.

Sam: Great, let's get started.

Mara: But hold on a second...I think we might need a separate loop.

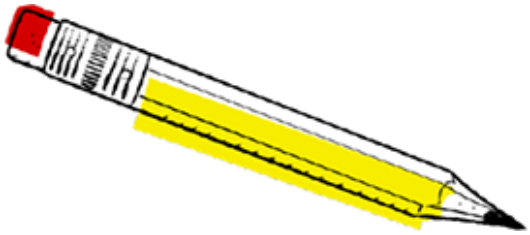
Sam: We do? Seems like there should be a way to do it in our existing loop.

Mara: Yup, I'm sure we do. Here's why: we have to know what the highest score is *before* we can find all the solutions that have that highest score. So we need two loops: one to find the highest score, which we've already written, and then a second one to find all the solutions that have that score.

Sam: Oh, I see. And in the second loop, we'll compare each score to the highest score, and if it matches, we'll add the index of the bubble solution score to the new array we're creating for the solutions with the highest score.

Mara: Exactly! Let's do it.

SHARPEN YOUR PENCIL



Can you write the loop to find all the scores that match the high score? Give it a shot below before you turn the page to see the solution and give it a test drive.

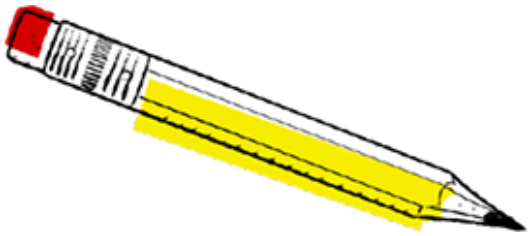
Remember, the variable highScore has the highest score in it; you can use that in the code below.

```
let bestSolutions = [];
for (let i = 0; i < scores.length; i++) {


}
```

Here's the new array we'll use to store the bubble solutions with the highest score.

Your code here.

Solution in "Sharpen your pencil Solution"



From "Sharpen your pencil"

Can you write the loop to find all the scores that match the high score? Here's our solution.

Again, we're starting by creating a new array that will hold all the bubble solutions that match the highest score.

Next, we iterate through the entire scores array, looking for those items with the highest score.

```
let bestSolutions = [];

for (let i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {

        bestSolutions.push(i);

    }
}

console.log("Solutions with the highest score: " + bestSolutions);
```

Each time through the loop, we compare the score at index i with the highScore; if they are equal, then we add that index to the bestSolutions array using push.

And finally, we can display the bubble solutions with the highest score. Notice we're using console.log to display the bestSolutions array. We could create another loop to display the array items one by one, but luckily, console.log will do this for us (and, if you look at the output, it also adds commas between the array values!).

BRAIN POWER

Take a look at the code in the exercise above. What if you woke up and push no longer existed? Could you rewrite this code without using push? Work that code out here:

Test drive the final report

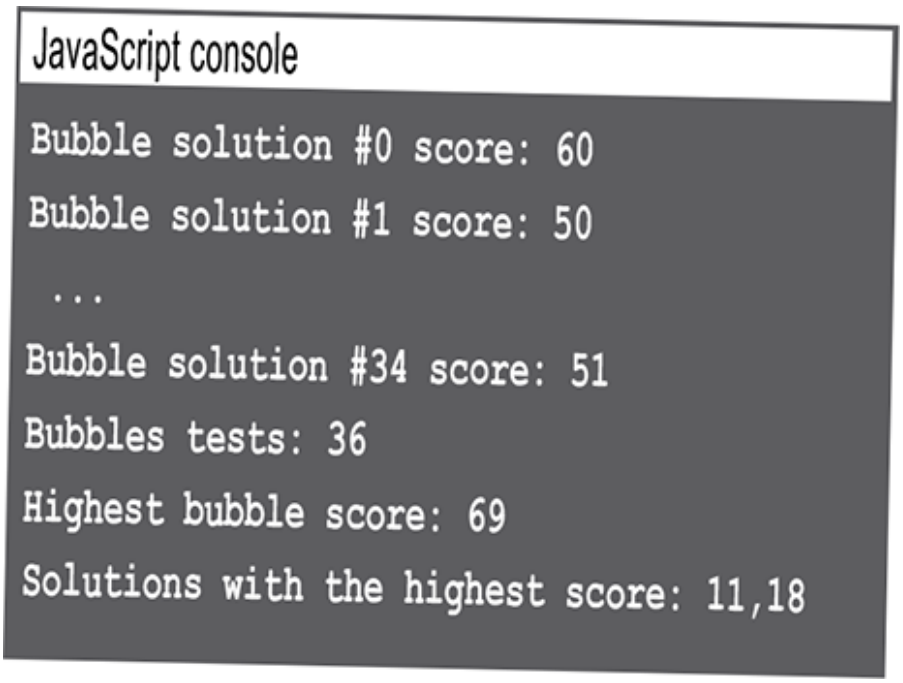
Go ahead and add the code to identify the bubble solutions with the highest score to your code in “bubbles.html”, then run another test drive. All the JavaScript code is shown below:

```
let scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];

let highScore = 0;
for (let i = 0; i < scores.length; i++) {
    let output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
```

```
    if (scores[i] > highScore) {
        highScore = scores[i];
    }
}
console.log("Bubbles tests: " + scores.length);
console.log("Highest bubble score: " + highScore);

let bestSolutions = [];
for (let i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {
        bestSolutions.push(i);
    }
}
console.log("Solutions with the highest score: " + bestSolutions);
```

And the winners are...

Bubble solutions #11 and #18 both have a high score of 69! So they are the best bubble solutions in this batch of test solutions.



You're right, we should be. Given that you just learned about functions, we wanted to get the basics of arrays out of the way before employing them. That said, you always want to think about which parts of your code you can abstract away into a function. Not only that, but say you wanted to reuse, or let others reuse, all the work that went into writing the bub-

ble computations—you'd want to give other developers a nice set of functions they could work with.

Let's go back to the bubble scores code and refactor it into a set of functions. By *refactor*, we mean we're going to rework how it's organized to make it more readable and maintainable, but we're going to do that without altering what the code does. In other words, when we're done, the code will do exactly what it does now, but it'll be a lot better organized.

A quick survey of the code

Let's get an overview of the code we've written and figure out which pieces we want to abstract into functions:

Here's the Bubbles-R-Us code.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Bubble Factory Test Lab</title>
  <script>
    let scores = [60, 50, 60, 58, 54, 54,
                  58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51,
                  69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18,
                  41, 53, 55, 61, 51, 44];

    let highScore = 0;

    for (let i = 0; i < scores.length; i++) {
      let output = "Bubble solution #" + i + " score: " + scores[i];
      console.log(output);
      if (scores[i] > highScore) {
        highScore = scores[i];
      }
    }
    console.log("Bubbles tests: " + scores.length);
    console.log("Highest bubble score: " + highScore);

    let bestSolutions = [];

    for (let i = 0; i < scores.length; i++) {
      if (scores[i] == highScore) {
        bestSolutions.push(i);
      }
    }

    console.log("Solutions with the highest score: " + bestSolutions);
  </script>
</head>
<body> </body>
</html>
```

We don't want to declare scores inside the functions that operate on scores because these are going to be different for each use of the functions. Instead, we'll pass the scores as an argument into the functions, so the functions can use any scores array to generate results.

We use this first chunk of code to output each score and at the same time compute the highest score in the array. We could put this in a printAndGetHighScore function.

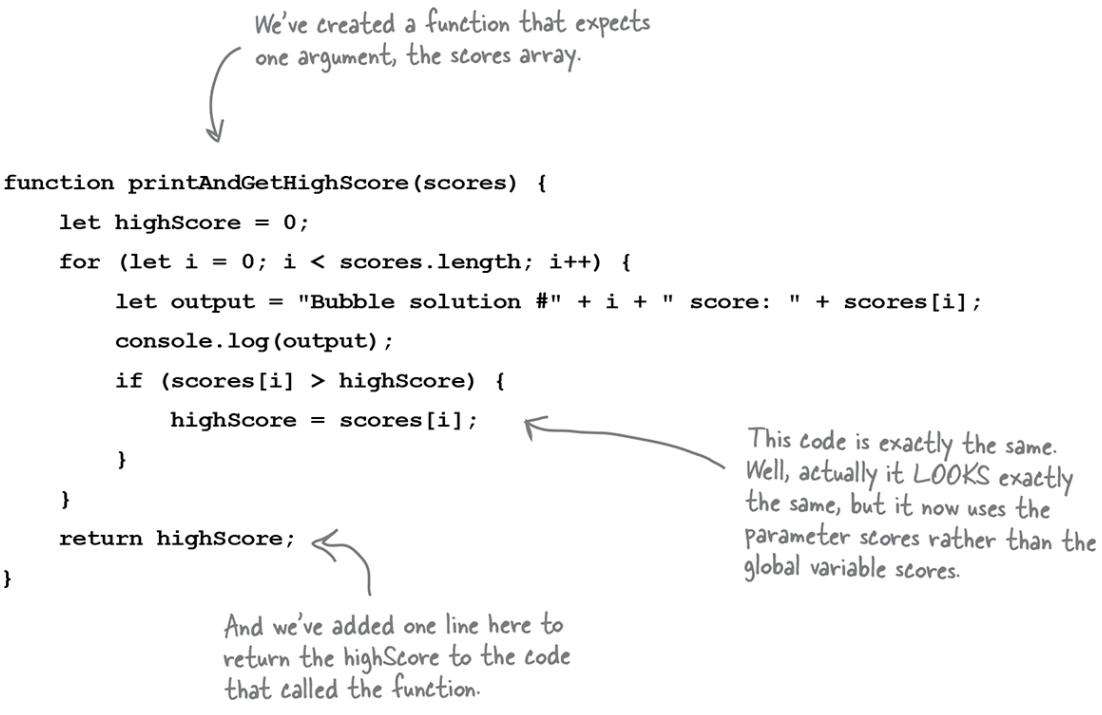
And we use this second chunk of code to figure out the best results given a high score. We could put this in a getBestResults function.

Writing the printAndGetHighScore function

We've got the code for the `printAndGetHighScore` function already. It's just the code we've already written, but to make it a function, we need to think through what arguments we're passing it and if it returns anything back to us.

Passing in the scores array seems like a good idea, because that way we can reuse the function on other arrays of bubble scores. And we want to return the high score that we compute in the function, so the code that calls the function can do interesting things with it (and, after all, we're going to need it to figure out the best solutions).

Oh, and another thing: often you want your functions to do *one thing* well. Here, we're doing two things: we're displaying all the scores in the array, and we're also computing the high score. We might want to consider breaking this into two functions, but given how simple things are right now we're going to resist the temptation. If we were working in a professional environment we might reconsider and break this into two functions, `printScores` and `getHighScore`. For now, though, we'll stick with one function. Let's get this code refactored:



Refactoring the code using printAndGetHighScore

Now, we need to change the rest of the code to use our new function. To do so, we simply call the new function and set the variable `highScore` to the result of the `printAndGetHighScore` function:

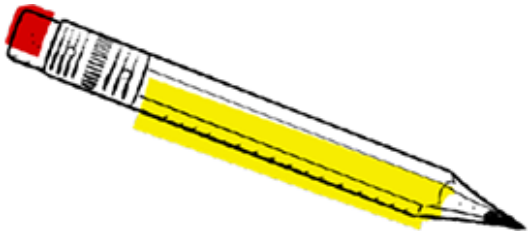
```
<!doctype html>
<html lang="en">
<head>
  <title>Bubble Factory Test Lab</title>
  <meta charset="utf-8">
  <script>
    let scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

    function printAndGetHighScore(scores) {
        let highScore = 0;
        for (let i = 0; i < scores.length; i++) {
            let output = "Bubble solution #" + i + " score: " + scores[i];
            console.log(output);
            if (scores[i] > highScore) {
                highScore = scores[i];
            }
        }
        return highScore;
    }

    let highScore = printAndGetHighScore(scores);
    console.log("Bubbles tests: " + scores.length);
    console.log("Highest bubble score: " + highScore);

    let bestSolutions = [];

    for (let i = 0; i < scores.length; i++) {
      if (scores[i] == highScore) {
        bestSolutions.push(i);
      }
    }

    console.log("Solutions with the highest score: " + bestSolutions);
  </script>
</head>
<body> </body>
</html>
```

Here's our new function, all ready to use.

And now we just call the function, passing in the scores array. We assign the value it returns to the variable highScore.

Now we need to refactor this code into a function and make the appropriate changes to the rest of the code.

SHARPEN YOUR PENCIL



Let's work through this next one together. The goal is to write a function to create an array of bubble solutions that have the high score (there might be more than one, so that's why we're using an array). We're going to pass this function the scores array and the highScore we computed with printAndGetHighScore. Finish the code below. You'll find the answer on the next page, but don't peek! Do the code yourself first, so you really get it.

Here's the original code in case you need to refer to it.

```
let bestSolutions = [];
for (let i = 0; i < scores.length; i++) {
    if (scores[i] == highScore) {
        bestSolutions.push(i);
    }
}
console.log("Solutions with the highest score: " + bestSolutions);
```

We've already started this, but we need your help to finish it!

```
function getBestResults(__________, ____________) {
    let bestSolutions = ______;
    for (let i = 0; i < scores.length; i++) {
        if (___________ == highScore) {
            bestSolutions.__________;
        }
    }
    return _______________;
}

let bestSolutions = _____________(scores, highScore);
console.log("Solutions with the highest score: " + bestSolutions);
```

Solution on page 168

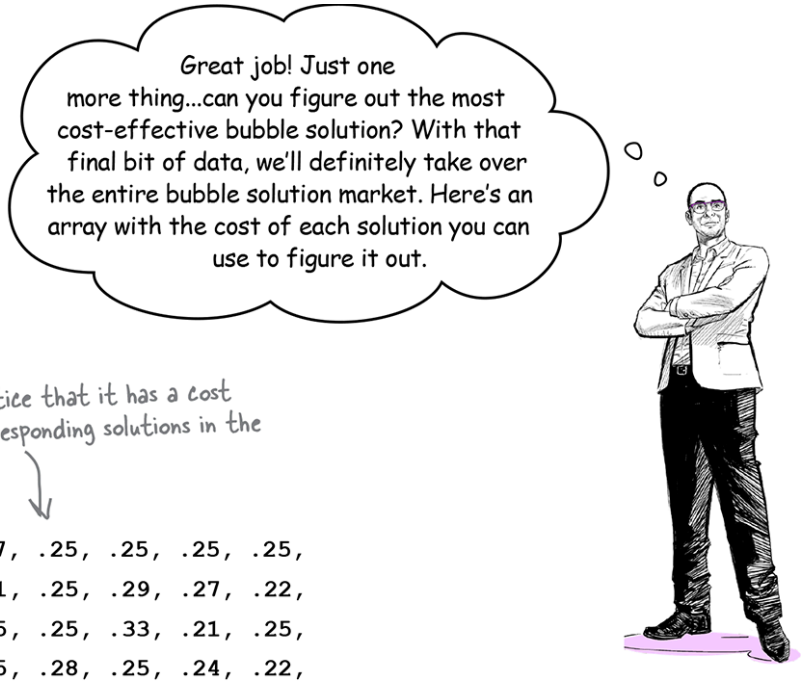
Putting it all together

Once you've completed refactoring your code, make all the changes to "bubbles.html", just like we have below, and reload the bubble report. You should get exactly the same results as before, but now you know your code is more organized and reusable. Create your own scores array and try some reuse!

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Bubble Factory Test Lab</title>
  <script>
    let scores = [60, 50, 60, 58, 54, 54, 58, 50, 52, 54, 48, 69,
                  34, 55, 51, 52, 44, 51, 69, 64, 66, 55, 52, 61,
                  46, 31, 57, 52, 44, 18, 41, 53, 55, 61, 51, 44];

    function printAndGetHighScore(scores) {
        let highScore = 0;
        for (let i = 0; i < scores.length; i++) {
            let output = "Bubble solution #" + i + " score: " + scores[i];
            console.log(output);
            if (scores[i] > highScore) {
                highScore = scores[i];
            }
        }
        return highScore;
    }

    function getBestResults(scores, highScore) {
        let bestSolutions = [];
        for (let i = 0; i < scores.length; i++) {
            if (scores[i] == highScore) {
                bestSolutions.push(i);
            }
        }
        return bestSolutions;
    }

    let highScore = printAndGetHighScore(scores);
    console.log("Bubbles tests: " + scores.length);
    console.log("Highest bubble score: " + highScore);

    let bestSolutions = getBestResults(scores, highScore);
    console.log("Solutions with the highest score: " + bestSolutions);

  </script>
</head>
<body> </body>
</html>
```

Okay, here's the new getBestResults function.

And we use the result of that function to display the best solutions in the report.



Here's the array. Notice that it has a cost for each of the corresponding solutions in the scores array.

```
let costs = [.25, .27, .25, .25, .25, .25,
             .33, .31, .25, .29, .27, .22,
             .31, .25, .25, .33, .21, .25,
             .25, .25, .28, .25, .24, .22,
             .20, .25, .30, .25, .24, .25,
             .25, .25, .27, .25, .26, .29];
```

So, what's the job here? It's to take the leading bubble solutions—that is, the ones with the highest bubble scores—and choose the lowest-cost one. Now, luckily, we've been given a `costs` array that mirrors the `scores` array. That is, the bubble solution score at index 0 in the `scores` array has the cost at index 0 in the `costs` array (.25), the bubble solution at index 1 in the `scores` array has the cost at index 1 in the `costs` array (.27), and so on. So, for any score you'll find its cost in the `costs` array at the same index. Sometimes we call these *parallel* arrays:

scores and costs are parallel arrays because for each score there is a corresponding cost at the same index.

```
let costs = [.25, .27, .25, .25, .25, .25, .33, .31, .25, .29, .27, .22, ..., .29];
let scores = [60,  50,  60,  58,  54,  54,  58,  50,  52,  54,  48,  69,  ..., 44];
```

The cost at 0 is the cost of the bubble solution at 0...

...and likewise for the other cost and score values in the arrays.

Mara: Well, we know the highest score already.

Sam: Right, but how do we use that? And we have these two arrays; how do we get those to work together?

Mara: I'm pretty sure either of us could write a simple for loop that goes through the `scores` array again and picks up the items that match the highest score.

Sam: Yeah, I could do that. But then what?

Mara: Anytime we hit a score that matches the highest score, we need to see if its cost is the lowest we've seen.

Sam: Oh I see, so we'll have a variable that keeps track of the index of the lowest-cost high score. Wow, that's a mouthful.

Mara: Exactly. And once we get through the entire array, whatever index is in that variable is the index of the item that not only matches the highest score, but has the lowest cost.

Sam: What if two items match in cost?

Mara: Hmm, we have to decide how to handle that. I'd say, whatever one we see first is the winner. Of course, we could do something more complex, but let's stick with that unless the CEO says differently.

Sam: This is complicated enough that I think I want to sketch out some pseudocode before writing anything.

Mara: I agree; whenever you are managing indices of multiple arrays things can get tricky. Let's do that. In the long run I'm sure it will be faster to plan it first.

Sam: Okay, I'll take a first stab at it...

EXERCISE

Check out the psuedocode below. Translate it into JavaScript. Make sure to check your answer at the end of the chapter.

```
FUNCTION getMostCostEffectiveSolution (scores, costs, highScore)
DECLARE a variable cost and set it to 100.
DECLARE a variable index.
FOR: let i=0; i < scores.length; i++
    IF the bubble solution at scores[i] has the highest score
        IF the current value of cost is greater than the cost of the bubble solution
        THEN
            SET the value of index to the value of i
            SET the value of cost to the cost of the bubble solution
        END IF
    END IF
END FOR
RETURN index
```

```
function getMostCostEffectiveSolution(scores, costs, highScore) {



}
let mostCostEffective = getMostCostEffectiveSolution(scores, costs, highScore);
console.log("Bubble Solution #" + mostCostEffective + " is the most cost effective");
```

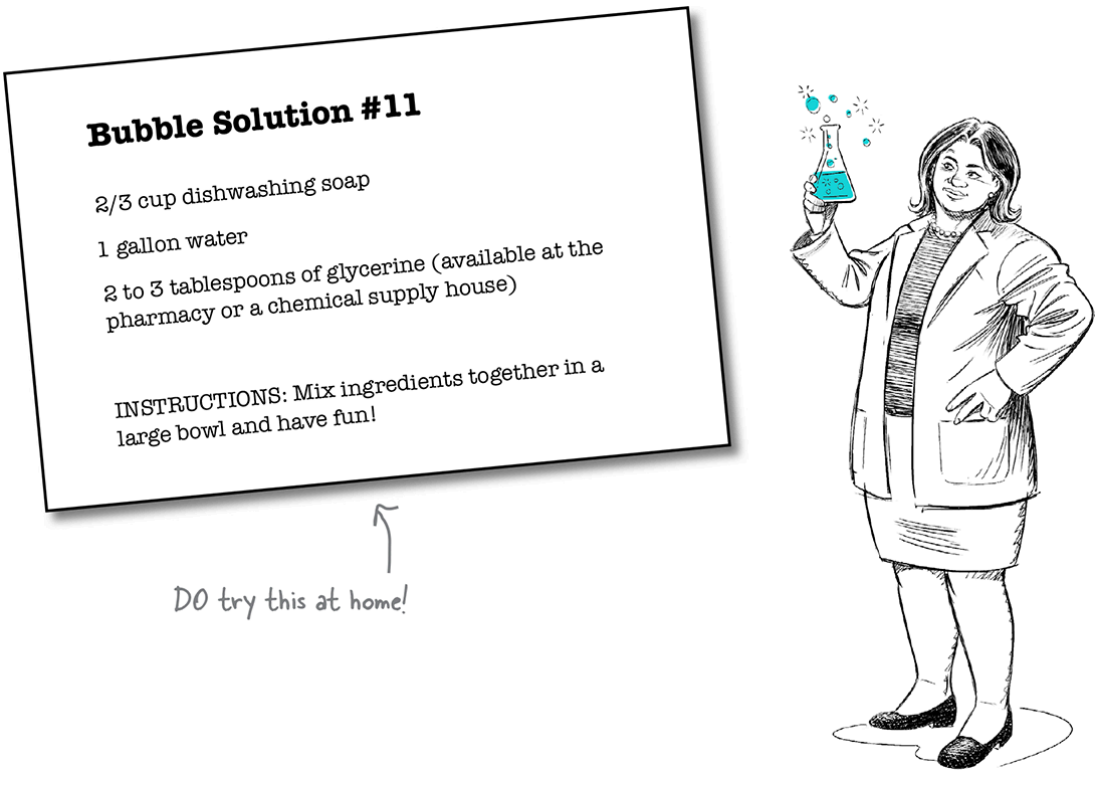
Translate the pseudocode to JavaScript here.

→ Solution in "Exercise Solution"

The last bit of code you wrote really helped determine the TRUE winner; that is, the solution that produces the most bubbles at the lowest cost. Congrats on taking a lot of data and crunching it down to something Bubbles-R-Us can make real business decisions with.

Now, if you're like us, you're dying to know what is in bubble solution #11. Look no further; the Bubbles-R-Us CEO said he'd be delighted to give you the recipe after all your unpaid work.

So, you'll find the recipe for bubble solution #11 below. Take some time to let your brain process arrays by making a batch, getting out, and blowing some bubbles before you begin the next chapter. Oh, but don't forget the bullet points and the crossword before you go!



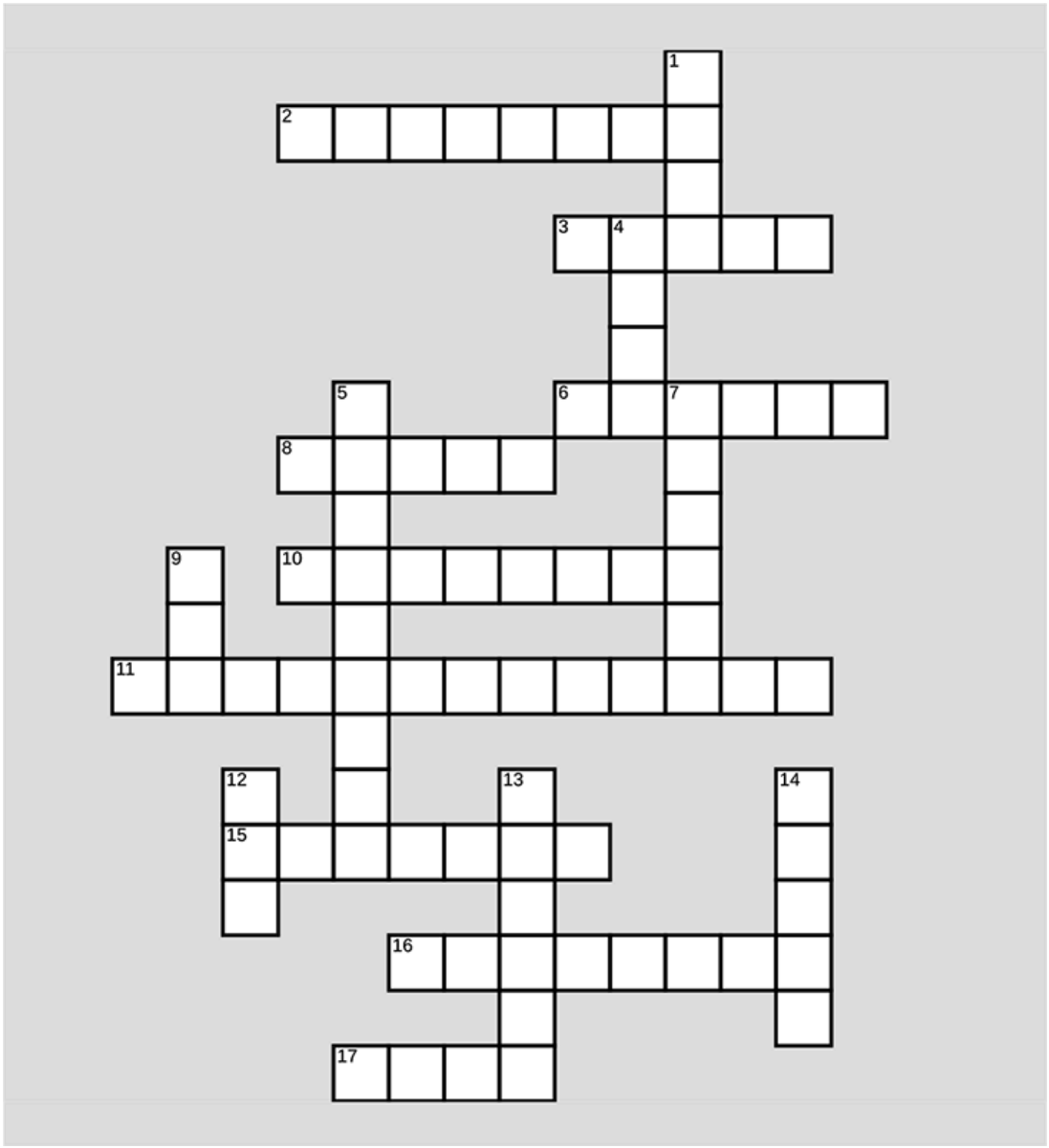
BULLET POINTS

- Arrays are a **data structure** for ordered data.
- An array holds a set of items, each with its own **index**.
- Arrays use a zero-based index, where the first item is at index zero.
- All arrays have a **length** property, which holds a number representing the number of items in the array.
- You can access any item using its index. For example, use myArray[1] to access item one (the second item in the array).
- If an item doesn't exist, trying to access it will result in a value of undefined.
- Assigning a value to an existing item will change its value.
- Assigning a value to an item that doesn't exist in the array will create a new item in the array.
- You can use a value of any type for an array item.
- Not all the values in an array need to be the same type.
- Use the **array literal notation** to create a new array.
- You can create an empty array with:

```
let myArray = [ ];
```

- The **for loop** is commonly used to iterate through arrays.
- A for loop packages up variable initialization, a conditional test, and variable increment into one statement.
- Like a while loop and an if/else statement, a for loop creates a **block** of code. The **scope** of the loop increment variable, often named i, is the block of the for loop.
- The while loop is most often used when you don't know how many times you need to loop, and you're looping until a condition is met. The for loop is most often used when you know the number of times the loop needs to execute.
- Sparse arrays occur when there are undefined items in the middle of an array.
- You can increment a variable by one with the **post-increment** operator, ++.
- You can decrement a variable by one with the **post-decrement** operator, --.
- You can add a new value to the end of an array using **push**.

Let arrays sink into your brain as you do the crossword.



ACROSS

2. Arrays are good for storing _____ values.

3. A for loop creates a _____ scope for the loop variable.

6. To change a value in an array, simply _____ the item a new value.

8. Access an array item using its _____ in square brackets.

10. When you _____ your code, you organize it so it's easier to read and maintain.

11. The operator we use to increment a loop variable.

15. An array is an _____ data structure.

16. Functions can help _____ your code.

17. To add a new value to the end of an existing array, use _____.

DOWN

1. The index of the first item in an array is _____.

4. The last index of an array is always one _____ than the length of the array.

5. The value an array item gets if you don't specify one.

7. An array with undefined values is called a _____ array.

9. How many bubble solutions had the highest score?

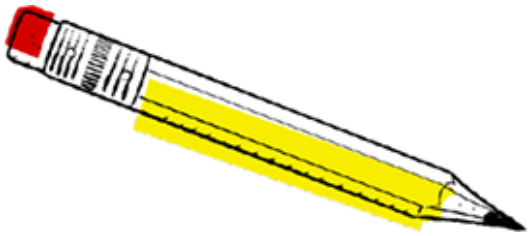
12. We usually use a _____ loop to iterate over an array.

13. When iterating through an array, we usually use the _____ property to know when to stop.

14. Each value in an array is stored at an _____.

Solution in "JavaScript cross Solution"

LOOK HOW EASY IT IS TO CREATE ARRAYS SOLUTION



From “Look how easy it is to create arrays”

You don’t know much about arrays yet, but we bet you can make some good guesses about how they work. Take a look at each line of code below and see if you can guess what it does. Here’s our solution.

```
let testResults = [91, 84, 100];
let dogs = ["Fido", "Spot"];
let isBarking = [true, false];

let dogBarking1 = isBarking[0];

if (dogBarking1 == true) {
    console.log(dogs[0] + " is barking");
}

if (testResults[2] > 90) {
    console.log("You got an A!");
}
```

Create an array of integer test results.

Create an array of string dog names.

Create an array of booleans.

Get the first value in the isBarking array and store it in the variable dogBarking1.

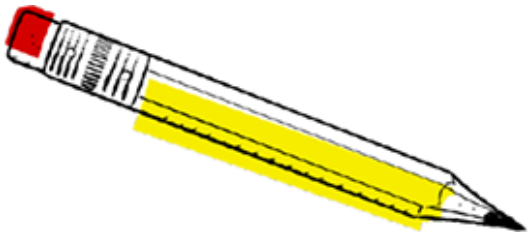
If dogBarking1 is true,

    then print the first dog name in the dogs array to the console.

If the third item in testResults > 90,

    then print the string "You got an A!" to the console.

SHARPEN YOUR PENCIL SOLUTION

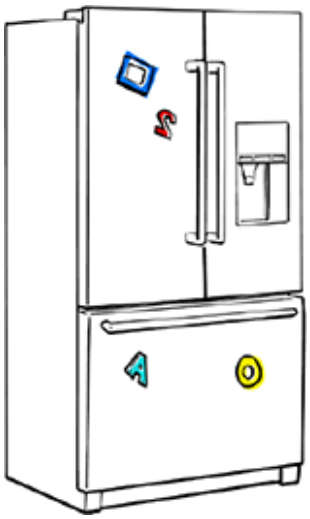


The products array below holds the Jenn and Berry's ice cream flavors. The ice cream flavors were added to this array in the order of their creation. Finish the code to determine the *most recent* ice cream flavor they created. Here's our solution.

```
let products = ["Choo Choo Chocolate", "Icy Mint", "Cake Batter", "Bubblegum"];
let last = products.length - 1;
let recent = products[last];
```

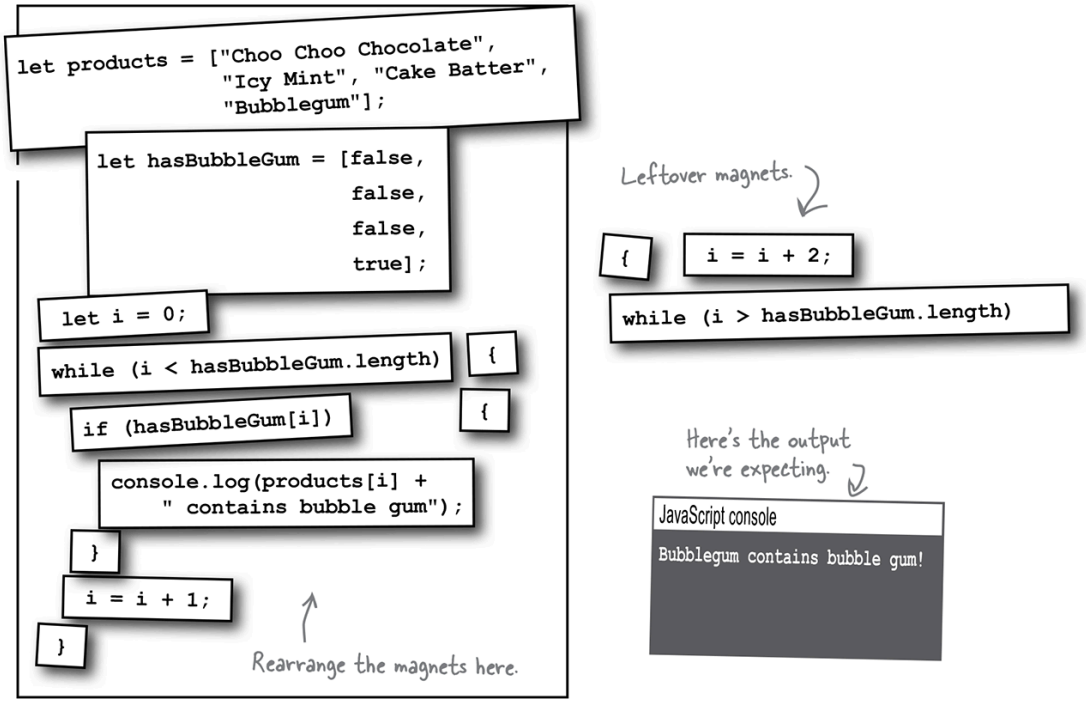
NOTE

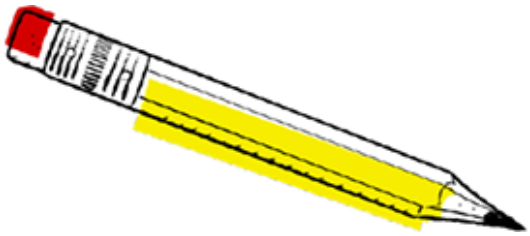
We can use the length of the array minus 1 to get the index of the last item. The length is 4, and the index of the last item is 3, because we start from 0.



From [“Code Magnets”]

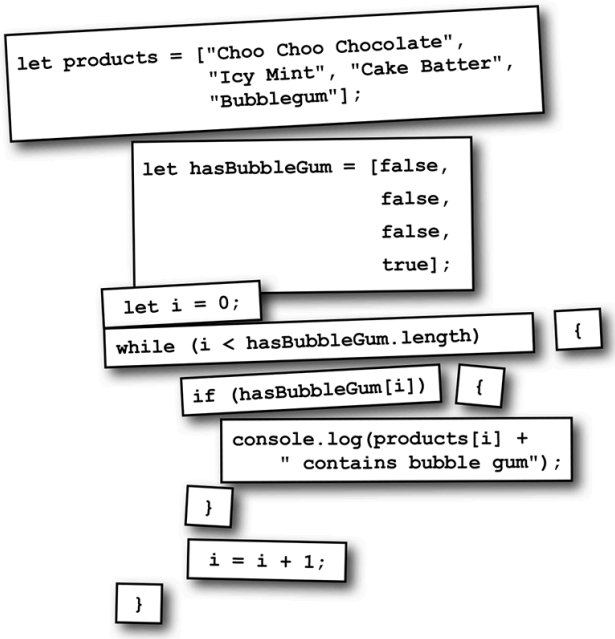
We’ve got code for testing to see which ice cream flavors have bubblegum pieces in them. We had all the code nicely laid out on our fridge using fridge magnets, but the magnets fell on the floor. It’s your job to put them back together. Be careful; a few extra magnets got mixed in. Here’s our solution.





From "Sharpen your pencil"

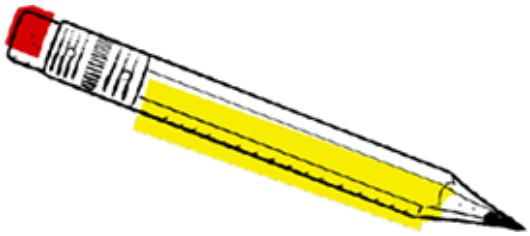
Rewrite your fridge magnet code (from the previous page) so that it uses a for loop instead of a while loop. If you need a hint, refer to each piece of the while loop in "But wait, there's a better way to iterate over an array" and see how it maps to the corresponding location in the for loop. Here's our solution.



Your code goes here.

```
let products = ["Choo Choo Chocolate",
                "Icy Mint", "Cake Batter",
                "Bubblegum"];
let hasBubbleGum = [false,
                    false,
                    false,
                    true];
for (let i = 0; i < hasBubbleGum.length; i = i + 1) {
    if (hasBubbleGum[i]) {
        console.log(products[i] + " contains bubble gum");
    }
}
```

SHARPEN YOUR PENCIL SOLUTION



From “Sharpen your pencil”

Go ahead and implement the pseudocode in “Cubicle Conversation continued...” to find the highest score by filling in the blanks in the code below. Once you’re done, give it a try in the browser by updating the code in “bubbles.html”, and reloading the page. Check the results in the console, and fill in the blanks in our console display below with the number of bubble tests and the highest score. Here’s our solution.

```
let scores = [60, 50, 60, 58, 54, 54,
              58, 50, 52, 54, 48, 69,
              34, 55, 51, 52, 44, 51,
              69, 64, 66, 55, 52, 61,
              46, 31, 57, 52, 44, 18,
              41, 53, 55, 61, 51, 44];
```

Fill in the blanks to complete the code here...

```
let highScore = 0;

for (let i = 0; i < scores.length; i++) {
    let output = "Bubble solution #" + i + " score: " + scores[i];
    console.log(output);
    if (scores[i] > highScore) {
        highScore = scores[i];
    }
}
console.log("Bubbles tests: " + scores.length );
console.log("Highest bubble score: " + highScore );
```

...and then fill in the blanks showing the output you get in the console.

JavaScript console

```
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
Bubbles tests: 36
Highest bubble score: 69
```

From "Exercise"

Here's our solution for the getMostCostEffectiveSolution function, which takes an array of scores, an array of costs, and a high score, and finds the index of the bubble solution with the highest score and lowest cost. Go ahead and test drive all your code in "bubbles.html" and make sure you see the same results.

getMostCostEffectiveSolution takes the array of scores, the array of costs, and the high score.

```
function getMostCostEffectiveSolution(scores, costs, highScore) {
    let cost = 100;
    let index;

    for (let i = 0; i < scores.length; i++) {
        if (scores[i] == highScore) {
            if (cost > costs[i]) {
                index = i;
                cost = costs[i];
            }
        }
    }
    return index;
}
let mostCostEffective = getMostCostEffectiveSolution(scores, costs, highScore);
console.log("Bubble solution #" + mostCostEffective + " is the most cost effective");
```

We'll keep track of the lowest-cost solution in cost...

We start cost at a high number, and we'll lower it each time we find a lowercost solution with the high score.

...and the index of the lowest-cost solution in index.

We iterate through the scores array like before...

...and check to see if the solution has the high score.

If it does, then we can check its cost. If the current cost is greater than the solution's cost, then we've found a lower-cost solution, so we'll make sure we keep track of which solution it is (its index in the array) and store its cost in the cost variable as the lowest cost we've seen so far.

Once the loop is complete, the index of the solution with the lowest cost is stored in index, so we return that to the code that called the function.

We then display the index (which is the bubble solution #) in the console.

The final report showing bubble solution #11 as the winner of the bubble tests with the highest bubble factor at the lowest cost.

```
JavaScript console
Bubble solution #0 score: 60
Bubble solution #1 score: 50
Bubble solution #2 score: 60
 ...
Bubble solution #34 score: 51
Bubble solution #35 score: 44
Bubbles tests: 36
Highest bubble score: 69
Solutions with the highest score: 11,18
Bubble solution #11 is the most cost effective
```

BONUS: We could also implement this using the bestSolutions array so we wouldn't have to iterate through all the scores again. Remember, the bestSolutions array has the indices of the solutions with the highest score. So in that code, we'd use the items in the bestSolutions array to index into the costs array to compare the costs. The resulting code is a little more efficient than this version, but it's also a little bit more difficult to read and understand! If you're interested, we've included the code in the book code download at wickedlysmart.com.

From [“JavaScript cross”]

Let arrays sink into your brain as you do the crossword. Here’s our solution.