ADDING ARRAYS

To add two arrays together to make a new, single array, you can use *`firstArray`*`.concat(`*`otherArray`*`)`. The term `concat` is short for *concatenate*, a fancy computer science word for joining two values together. The `concat` method will combine both arrays into a new array, with the values from *`firstArray`* added in front of those from *`otherArray`*.

For example, say we have a list of some furry animals and another list of some scaly animals, and we want to combine them. If we put all of our furry animals in an array called `furryAnimals` and all of our scaly animals in an array called `scalyAnimals`, entering `furryAnimals.concat(scalyAnimals)` will create a new array that has the values from the first array at the beginning and the values from the second array at the end.

```
var furryAnimals = ["Alpaca", "Ring-tailed Lemur", "Yeti"];
var scalyAnimals = ["Boa Constrictor", "Godzilla"];
var furryAndScalyAnimals = furryAnimals.concat(scalyAnimals);
furryAndScalyAnimals;
["Alpaca", "Ring-tailed Lemur", "Yeti", "Boa Constrictor", "Godzilla"]
furryAnimals;
["Alpaca", "Ring-tailed Lemur", "Yeti"]
scalyAnimals;
["Boa Constrictor", "Godzilla"]
```

Even though *`firstArray`*`.concat(`*`otherArray`*`)` returns an array containing all the elements from *`firstArray`* and *`secondArray`*, neither of the original arrays is changed. When we look at `furryAnimals` and `scalyAnimals`, they're the same as when we created them.

JOINING MULTIPLE ARRAYS

You can use `concat` to join more than two arrays together. Just put the extra arrays inside the parentheses, separated by commas:

```
var furryAnimals = ["Alpaca", "Ring-tailed Lemur", "Yeti"];
var scalyAnimals = ["Boa Constrictor", "Godzilla"];
var featheredAnimals = ["Macaw", "Dodo"];
var allAnimals = furryAnimals.concat(scalyAnimals, featheredAnimals);
allAnimals;
["Alpaca", "Ring-tailed Lemur", "Yeti", "Boa Constrictor", "Godzilla",
"Macaw", "Dodo"]
```

Here the values from `featheredAnimals` get added to the very end of the new array, since they are listed last in the parentheses after the `concat` method.

`concat` is useful when you have multiple arrays that you want to combine into one. For example, say you have a list of your favorite books, and your friend also has a list of favorite books, and you

want to go see if the books are available to buy all at once at the bookstore. It would be easier if you had only one list of books. All you'd have to do is concat your list with your friend's, and voilà! One list of books.

FINDING THE INDEX OF AN ELEMENT IN AN ARRAY

To find the index of an element in an array, use `.indexOf(`*element*`)`. Here we define the array `colors` and then ask for the index positions of `"blue"` and `"green"` with `colors.indexOf("blue")` and `colors.indexOf("green")`. Because the index of `"blue"` in the array is 2, `colors.indexOf("blue")` returns `2`. The index of `"green"` in the array is 1, so `colors.indexOf("green")` returns `1`.

```
var colors = ["red", "green", "blue"];
colors.indexOf("blue");
2
colors.indexOf("green");
1
```

`indexOf` is like the reverse of using square brackets to get a value at a particular index; `colors[2]` is `"blue"`, so `colors.indexOf("blue")` is `2`:

```
colors[2];
"blue"
colors.indexOf("blue");
2
```

Even though `"blue"` appears third in the array, its index position is 2 because we always start counting from 0. And the same goes for `"green"`, of course, at index 1.

If the element whose position you ask for is not in the array, JavaScript returns `-1`.

```
colors.indexOf("purple");
-1
```

This is JavaScript's way of saying "That doesn't exist here," while still returning a number.

If the element appears more than once in the array, the `indexOf` method will return the first index of that element in the array.

```
var insects = ["Bee", "Ant", "Bee", "Bee", "Ant"];
insects.indexOf("Bee");
0
```

TURNING AN ARRAY INTO A STRING

You can use `.join()` to join all the elements in an array together into one big string.

```
var boringAnimals = ["Monkey", "Cat", "Fish", "Lizard"];
boringAnimals.join();
"Monkey,Cat,Fish,Lizard"
```

When you call the `join` method on an array, it returns a string containing all the elements, separated by commas. But what if you don't want to use commas as the separator?

You can use `.join(`*separator*`)` to do the same thing, but with your own chosen separator between each value. The separator is whatever string you put inside the parentheses. For example, we can use three different separators: a hyphen with spaces on either side, an asterisk, and the word *sees* with spaces on either side. Notice that you need quote marks around the separator, because the separator is a string.

```
var boringAnimals = ["Monkey", "Cat", "Fish", "Lizard"];
boringAnimals.join(" - ");
"Monkey - Cat - Fish - Lizard"
boringAnimals.join("*")
"Monkey*Cat*Fish*Lizard"
boringAnimals.join(" sees ")
"Monkey sees Cat sees Fish sees Lizard"
```

This is useful if you have an array that you want to turn into a string. Say you have lots of middle names and you've got them stored in an array, along with your first and last name. You might

be asked to give your full name as a string. Using `join`, with a single space as the separator, will join all your names together into a single string:

```
var myNames = ["Nicholas", "Andrew", "Maxwell", "Morgan"];
myNames.join(" ");
"Nicholas Andrew Maxwell Morgan"
```

If you didn't have `join`, you'd have to do something like this, which would be really annoying to type out:

```
myNames[0] + " " + myNames[1] + " " + myNames[2] + " " + myNames[3];
"Nicholas Andrew Maxwell Morgan"
```

Also, this code would work only if you had exactly two middle names. If you had one or three middle names, you'd have to change the code. With `join`, you don't have to change anything—it prints out a string with all of the elements of the array, no matter how long the array is.

If the values in the array aren't strings, JavaScript will convert them to strings before joining them together:

```
var ages = [11, 14, 79];
ages.join(" ");
"11 14 79"
```

USEFUL THINGS TO DO WITH ARRAYS

Now you know lots of different ways to create arrays and play around with them. But what can you actually do with all these properties and methods? In this section, we'll write a few short programs that show off some useful things to do with arrays.

FINDING YOUR WAY HOME

Picture this: your friend has come over to your house. Now she wants to show you her house. The only problem is that you've never been to her house before, and later you'll have to find your way back home on your own.

Luckily, you have a clever idea to help you with your problem: on the way to your friend's house, you'll keep a list of all the landmarks you see. On the way back, you'll go through the list in

reverse and check items off the end of the list every time you pass a landmark so you know where to go next.

BUILDING THE ARRAY WITH PUSH

Let's write some code that would do exactly that. We start off by creating an empty array. The array starts off empty because you don't know what landmarks you'll see until you actually start walking to your friend's house. Then, for each landmark on the way to your friend's house, we'll push a description of that landmark onto the end of the array. Then, when it's time to go home, we'll pop each landmark off the array.

```
var landmarks = [];
landmarks.push("My house");
landmarks.push("Front path");
landmarks.push("Flickering streetlamp");
landmarks.push("Leaky fire hydrant");
landmarks.push("Fire station");
landmarks.push("Cat rescue center");
landmarks.push("My old school");
landmarks.push("My friend's house");
```

Here we create an empty array named `landmarks` and then use `push` to store all the landmarks you pass on the way to your friend's house.

GOING IN REVERSE WITH POP

Once you arrive at your friend's house, you can inspect your array of landmarks. Sure enough, the first item is `"My house"`, followed by `"Front path"`, and so on through the end of the array, with the final item `"My friend's house"`. When it's time to go home, all you need to do is pop off the items one by one, and you'll know where to go next.

```
landmarks.pop();
"My friend's house"
landmarks.pop();
"My old school"
landmarks.pop();
"Cat rescue center"
landmarks.pop();
"Fire station"
landmarks.pop();
"Leaky fire hydrant"
```

```
landmarks.pop();
"Flickering streetlamp"
landmarks.pop();
"Front path"
landmarks.pop();
"My house"
```

Phew, you made it home!

Did you notice how the first landmark you put in the array was also the last one you got out of it? And the last landmark you put in the array was the first one that came out? You might have thought that you'd always want the first item you put in to be the first item you get out, but you can see that it's sometimes helpful to go back through an array in reverse.

It's actually very common to use a process like this in larger programs, which is why JavaScript makes pushing and popping so easy.

NOTE

This technique is known as a stack *in computer-speak. Think of it like a stack of pancakes. Every time you cook a new pancake, it goes on top (like* `push`*), and every time you eat one, it comes off the top (like* `pop`*). Popping a stack is like going back in time: the last item you pop is the first one you pushed. It's the same with pancakes: the last pancake you eat is the first one that was cooked. In programming jargon, this is also called* Last In, First Out (LIFO). *The alternative to LIFO is* First In, First Out (FIFO). *This is also known as a* queue*, because it acts like a queue (or line) of people. The first person to join the queue is the first person to be served.*

DECISION MAKER

We can use arrays in JavaScript to build a program to make decisions for us (like a Magic 8-Ball). First, though, we need to find out how to get random numbers.

USING MATH.RANDOM()

We can produce random numbers using a special method called `Math.random()`, which returns a random number between 0 and 1 each time it's called. Here's an example:

```
Math.random();
0.8945409457664937
Math.random();
0.3697543195448816
Math.random();
0.48314980138093233
```

It's important to note that `Math.random()` always returns a number *less than* 1 and will never return 1 itself.

If you want a bigger number, just multiply the result of calling `Math.random()`. For example, if you wanted numbers between 0 and 10, you would multiply `Math.random()` by 10:

```
Math.random() * 10;
7.648027329705656
Math.random() * 10;
9.7565904534421861
Math.random() * 10;
0.21483442978933454
```

ROUNDING DOWN WITH MATH.FLOOR()

We can't use these numbers as array indexes, though, because indexes have to be whole numbers with nothing after the decimal point. To fix that, we need another method called `Math.floor()`. This takes a number and rounds it down to the whole number below it (basically getting rid of everything after the decimal point).

```
Math.floor(3.7463463);
3
Math.floor(9.9999);
9
Math.floor(0.793423451963426);
0
```

We can combine these two techniques to create a random index. All we need to do is multiply `Math.random()` by the length of

the array and then call `Math.floor()` on that value. For example, if the length of the array were 4, we would do this:

```
Math.floor(Math.random() * 4);
2 // could be 0, 1, 2, or 3
```

Every time you call the code above, it returns a random number from 0 to 3 (including 0 and 3). Because `Math.random()` always returns a value less than 1, `Math.random() * 4` will never return 4 or anything higher than 4.

Now, if we use that random number as an index, we can select a random element from an array:

```
var randomWords = ["Explosion", "Cave", "Princess", "Pen"];
var randomIndex = Math.floor(Math.random() * 4);
randomWords[randomIndex];
"Cave"
```

Here we use `Math.floor(Math.random() * 4);` to pick a random number from 0 to 3. Once that random number is saved to the variable `randomIndex`, we use it as an index to ask for a string from the array `randomWords`.

In fact, we could shorten this by doing away with the `randomIndex` variable altogether and just say:

```
randomWords[Math.floor(Math.random() * 4)];
"Princess"
```

THE COMPLETE DECISION MAKER

Now let's create our array of phrases, and we can use this code to pick a random one. This is our decision maker! I'm using comments here to show some questions you might want to ask your computer.

```
var phrases = [
  "That sounds good",
  "Yes, you should definitely do that",
  "I'm not sure that's a great idea",
  "Maybe not today?",
  "Computer says no."
];
```

```
// Should I have another milkshake?
phrases[Math.floor(Math.random() * 5)];
"I'm not sure that's a great idea"
// Should I do my homework?
phrases[Math.floor(Math.random() * 5)];
"Maybe not today?"
```

Here we created an array called `phrases` that stores different pieces of advice. Now, every time we have a question, we can ask for a random value from the `phrases` array, and it will help us make a decision!

Notice that because our array of decisions has five items, we multiply `Math.random()` by 5. This will always return one of five index positions: 0, 1, 2, 3, or 4.

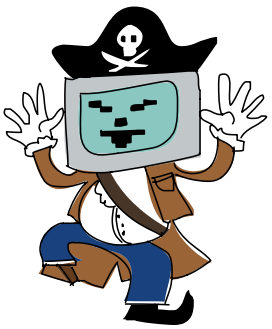
CREATING A RANDOM INSULT GENERATOR

We can extend the decision maker example to create a program that generates a random insult every time you run it!

```
var randomBodyParts = [“Nail”, “Upper Lip”, "Hair"];
var randomAdjectives = [“Upside Down”, “Left instead of Right”, “Funky”];
var randomWords = ["Fly", "Marmot", "Stick", "Monkey", "Rat"];

// Pick a random body part from the randomBodyParts array:
❶ var randomBodyPart = randomBodyParts[Math.floor(Math.random() * 3)];
// Pick a random adjective from the randomAdjectives array:
❷ var randomAdjective = randomAdjectives[Math.floor(Math.random() * 3)];
// Pick a random word from the randomWords array:
❸ var randomWord = randomWords[Math.floor(Math.random() * 5)];
// Join all the random strings into a sentence:
var randomInsult = "Your " + randomBodyPart + " is like a " + ↵
randomAdjective + " " + randomWord + "!!!";
randomInsult;
"Your Nail is like a Funky Marmot!!!"
```

Here we have three arrays, and in lines ❶, ❷, and ❸, we use three indexes to pull a random word from each array. Then, we combine them all in the variable `randomInsult` to create a complete insult. At ❶ and ❷ we're multiplying by 3 because `randomAdjectives` and `randomBodyParts` both contain three elements. Likewise, we're multiplying by 5 at ❸ because `randomWords` is



five elements long. Notice that we add a string with a single space between `randomAdjective` and `randomWord`. Try running this code a few times—you should get a different random insult each time!

TRY IT OUT!

If you wanted to be really clever, you could replace line ❸ with this:

```
var randomWord = randomWords[Math.floor(Math.random() * ↵
randomWords.length)];
```

We know that we always need to multiply `Math.random()` by the length of the array, so using `randomWords.length` means we don't have to change our code if the length of the array changes.

Here's another way to build up our random insult:

```
var randomInsult = ["Your", randomBodyPart, "is", "like", "a", ↵
randomAdjective, randomWord + "!!!"].join(" ");
"Your Hair is like a Upside Down Fly!!!"
```

In this example, each word of the sentence is a separate string in an array, which we join with the space character. There's only one place where we *don't* want a space, which is in between `randomWord` and `"!!!"`. In this case, we use the `+` operator to join those two strings without the space.

WHAT YOU LEARNED

As you've seen, JavaScript arrays are a way to store a list of values. Now you know how to create and work with arrays, and you have many ways of accessing their elements.

Arrays are one of the ways JavaScript gives you to bring multiple values together into one place. In the next chapter, we'll look at objects, which are another way of storing multiple values as a single unit. Objects use *string keys* to access the elements, rather than number indexes.

PROGRAMMING CHALLENGES

Try out these challenges to practice the skills you learned in this chapter.

#1: NEW INSULTS

Make your own random insult generator with your own set of words. Remember, be nice!

#2: MORE SOPHISTICATED INSULTS

Extend the random insult generator so it generates insults like "Your [body part] is more [adjective] than a [animal]'s [animal body part]." (Hint: You'll need to create another array.)

#3: USE + OR JOIN?

Make two versions of your random insult generator: one that uses the `+` operator to create the string, and one that creates an array and joins it with `" "`. Which do you prefer, and why?

#4: JOINING NUMBERS

How could you turn the array `[3, 2, 1]` into the string `"3 is bigger than 2 is bigger than 1"` using the `join` method?