"Concatenation" is a fancy term for joining two or more things together. In JavaScript, we can use the "+" operator to concatenate strings. For example, if we have two strings, "hello" and "world", we can join them together with the "+" operator like this:
"hello" + "world"
This would give us the string "helloworld". The "+" operator can also be used to add two numbers together, but in this case we're using it to join two strings together.
Alright! So let's say we have two strings, "Hello" and "World". We can concatenate them together with the "+" operator like this:
"Hello" + "World"
This will give us the string "HelloWorld". But what if we want to add a space between the words? We can do that by adding a space character (" ") between the strings, like this:
"Hello " + " " + "World"
Now we have the string "Hello World". It's the same idea as before, but now we've added a space character to make the string easier to read.

JavaScript concatenation has a few different functions. The first is, of course, joining two or more strings together. But it can also be used to create complex strings, like the "Hello World" example I just gave. It can also be used to add characters to the beginning or end of a string, and it can be used to manipulate strings in other ways. For example, you could use it to replace one character with another, or you could use it to find and remove specific characters from a string.
JavaScript concatenation has more than one function, or purpose. The first function is joining strings together. The second function is to create complex strings. This means that you can use concatenation to create strings that have multiple parts, like the "Hello World" string. The third function is adding characters to strings. You can use concatenation to add any character you want to the beginning or end of a string. And the fourth function is manipulating strings?

First, let's talk about replacing one character with another. Let's say you have a string that says "Hello, World!" and you want to replace the comma with a semicolon. You can do that with concatenation. You would write something like this:
"Hello, World!" + ";"
This would create a new string that says "Hello; World!". Does that make sense? The same idea can be used to remove characters from a string. Let's say you have a string that says "Hello, World!" and you want to remove the comma." So, to do that, you would write something like this:
"Hello, World!" - ","
This would give you the string "Hello World!". And that's just the tip of the iceberg when it comes to manipulating strings with concatenation. There are a lot of other things you can do, but let's start with these basic examples.
Let's talk about some of the more complex ways you can use concatenation to manipulate strings. The first example is string slicing. With string slicing, you can extract a subset of a string, based on the start and end positions you specify. So, let's say you have a string that says "Hello, World!" and you want to extract just the word "World" from the string. You could do that with string slicing. You would write something like this:

"Hello, World!"[11:19]
The "11:19" tells the program to extract the characters starting at the 11th
String slicing is the process of extracting a subset of a string based on the start and end positions you specify. So, for example, let's say you have a string called "Hello, World!" and you want to extract just the word "World!" from the string. You could use this code:
var extractedWord = "Hello, World!"[15:21];
This code would take the string "Hello, World!" and extract the characters starting at the 15th position and ending at the 21st position. That would give you the string "World!".
The start position you specify is actually zero-based, so the first character in the string is actually at position zero. And the end position you specify is exclusive, so it won't include that character in the extracted string. So, using the same example, the string "Hello, World!" actually has 16 characters, including the space and the comma. The 15th character is actually the "l" in "World". So, when you use the slicing code, you'll get the characters from the "l" to the "d" in "World" to the "d" in "World". That means the extracted string is just "World".

The start position you specify is actually zero-based, so the first character in the string is actually at position zero. And the end position you specify is exclusive, so it won't include that character in the extracted string. So, using the same example, the string "Hello, World!" actually has 16 characters, including the space and the comma. The 15th character is actually the "l" in "World". So, when you use the slicing code, you'll get the characters from the "l" to the "d" in "World" to the "d" in "World". That means the extracted string is just "World".
You can also use negative numbers in the start and end positions. So, for example, you could use something like this:
var extractedWord = "Hello, World!"[-1:4];
This would extract the last character and the four characters before it, which would give you the string "World!".

The next thing I want to tell you about is string interpolation. This is another way to manipulate strings using concatenation, and it's a bit more flexible than the basic slicing method we just talked about. String interpolation lets you insert variables, expressions, and other values into a string. So, for example, you could do something like this:
var myString = "The price of the item is ${item.price} dollars.";
This code would create a string that says "The price of the item is $10 dollars." if the item.price variable has a value of 10.

Awesome! Let's talk about string concatenation using the template literals method. This is a bit different from the basic slicing method and the string interpolation method. It uses a special syntax called a template literal, which looks like this:
The price of the item is ${item.price} dollars.
This is similar to the interpolation method, but it uses backticks (`) instead of curly braces ({}) to surround the variables and expressions. It also doesn't require you to use the word "var" to define a variable, which makes it a bit more concise.

Let's say we want to create a string that says "Hello, {name}!" and we want to insert a name into the string. We can do that with template literals like this:
const myString = Hello, ${name}!;
This would create a string that says "Hello, John!" if the name variable has a value of "John".
That's great! I'm glad you're following along. Now, the cool thing about template literals is that you can use them to insert not just variables, but also functions and expressions. So, for example, you could do something like this:
const myString = The total price is ${price + tax};
In this example, the price and tax variables are inserted into the string, and then the price and tax are added together. So, if the price variable has a value of 10 and the tax variable has a value of 2, the resulting string would say "The total price is 12."
Great! Let's start with something called tagged template literals. These are template literals that have a tag, which is a piece of code that is executed before the template literal is evaluated. The tag is surrounded by backticks and looks like this:
```
```
In this example, the tag is a JavaScript function called uppercase. So, when the template literal is evaluated, the function is executed first, and then the template literal is evaluated. This means that the string "Hello, {name}" will be converted to uppercase before it is inserted into the template literal.
let's talk about another feature of template literals called string interpolation expressions. These are expressions that are evaluated and inserted into the template literal, like this:
const myString = The value is ${2 + 2};
In this example, the expression "2 + 2" is evaluated and the result (4) is inserted into the template literal. So, the resulting string is "The value is 4."

You can also use template literals to insert multi-line strings. To do this, you just need to use a newline character (\n) at the end of each line you want to include in the string. Like this:
const myString = `This is a multi-line string.
It can be as long as you like.
As long as you want.
And longer still.
It's great!`;
As you can see, each line of the string is separated by a newline character, and the resulting string is a single string with multiple lines.
There's one more feature of template literals called tagged template literals with parameters. These are template literals that have a tag and a list of parameters. The tag can use the parameters to customize the string in different ways. Let's say we have a template literal like this:
(name) {
  return `Hello, ${name}!`;
}
This template literal has a tag that takes a single parameter called "name". The tag returns a string that says "Hello, {name}!" and inserts the value of the "name".

Let's say we want to create a template literal that can say "Hello" to anyone, using their name. We can use a tagged template literal with parameters to do this. First, we need to create a function that takes a name and returns a string that says "Hello" to that name. Let's call the function "helloTo". Here's what it would look like:

```
function helloTo(name) {
  return "Hello, ${name}!";
}
```

Now, we can use this function as the tag for our template literal. So, the whole template literal would look like this: We'll start with the backticks:

Next, we'll add the tag, which is the function we just created:

```
helloTo
```

Inside the curly braces, we'll add the parameters that the tag takes, which is just the "name" parameter:

```
(name)
```

Finally, we'll close the tag with another set of curly braces:

```
}
```

And then we'll close the template literal with another set of backticks:

So, To use it, we just need to add it to our code, like this:

```
const myString = Hello, ${helloTo("Bob")}!;
```

The template literal will call the "helloTo" function and insert the resulting string into the template literal. So, the resulting string will be "Hello, Bob!"