Functions

Introduction to functions

Duration: 30 minutes
Q&A: 5 minutes by the end of the lecture

Doing Work in JavaScript

When we write programs, we think about the kind of work we want our program to do. Mathematical calculations, updating data, and presenting information to a user are just a few examples of the kind of work we can perform in the programs we write.

Let's investigate this idea of doing work by writing a program that will take a height expressed in feet and inches and convert it to centimeters.

```
1 in = 2.54 cm

5 ft 10 in = ??? cm
```

Before we write any code, let's think about how we can accomplish this goal given a starting height of 5ft 10in. There are 2.54cm in every inch, so our first step will be to express our initial height in terms of inches.

```
1 in = 2.54 cm

5 ft 10 in = ??? cm

(5 * 12) + 10 = ??? cm
```

We can calculate our original height in inches by multiplying the number of feet by 12, then adding that to our original number of inches.

1 in = 2.54 cm

5 ft 10 in = ??? cm

(5 * 12) + 10 = ??? cm

((5 * 12) + 10) * 2.54 = 177.8 cm

Finally, we can multiply our total inches by 2.54 to arrive at our answer: 177.8 centimeters.

```
((5 * 12) + 10) * 2.54; //=> 177.8
```

Once we've thought about how to solve the problem, we can see that writing code for it is straightforward. JavaScript's arithmetic operators allow us to write an expression that results in the value 177.8.

```
((5 * 12) + 10) * 2.54; //=> 177.8
```

Try the problem: Ask two of your classmates how tall they are (in feet and inches). Write versions of this code that will express their height in centimeters, then write a version for your own height.

```
((5 * 12) + 10) * 2.54; //=> 177.8
((6 * 12) + 4) * 2.54; //=> 193.04
((4 * 12) + 8) * 2.54; //=> 142.24
```

You might have noticed an irritating problem: every time you want to convert a new height, you must type out this same line of code with only two values changed.

```
((5 * 12) + 10) * 2.54; //=> 177.8
((6 * 12) + 4) * 2.54; //=> 193.04
((4 * 12) + 8) * 2.54; //=> 142.24

((<feet> * 12) + <inches>)* 2.54;
```

It would be better to create a version of this code that represents the *pattern* for solving our problem, rather than the *computation*. In JavaScript, we can accomplish this goal using a **function**.

Functions

A Function is a grouping of several lines of code that accepts some kind of input, called **parameters**, and produces a **value** as a result.

We can choose to execute the code contained in a function by **invoking**, or **calling**, the function. When a function is invoked, it may be supplied with some values called **arguments**. The code inside the function will do some work based on the provided arguments, and return a **value**.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
```

This is the function keyword. It's a special word in JavaScript that indicates our intent to define a new function.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

This is our function's **name**. It serves as a label that will allow us to refer to our function later.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

This set of parentheses contains our function's **parameters**. These are the values that we expect to change every time this function is invoked.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

Parameters are labels for the values we want to use when we invoke the function.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

These curly braces indicate the beginning and end of the code we wish to execute when our function is invoked.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

The keyword return indicates the value our function will provide as the result of performing some work.

Creating Functions in JavaScript

We can use this syntax to define a function.

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
```

In this example, our function will return the value that is the result of this expression being evaluated.

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

We'll refer to the function by its name...

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

... and **invoke** it using parentheses.

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

Inside the parentheses, we'll supply some values that will be plugged in to our function. These values are called **arguments**.

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

5

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

For this invocation, any reference to the label **feet** inside our function will be substituted with the value **5**...

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

10

... and any reference to the label **inches** inside our function will be substituted with the value **10**.

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(5, 10); //=> 177.8
```

Our function will return the value `177.8`, the result of some work done based off our provided inputs.

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(3 * 2, 6);
```

One more thing about arguments: we can provide **expressions** as arguments to a function.

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(3 * 2, 6);
```

6

Any expressions supplied as arguments will be evaluated first...

Invoking Functions in JavaScript

To run the code contained within a function, we **invoke** it:

6

```
function convertHeight(feet, inches) {
  return ((feet * 12) + inches) * 2.54;
}
convertHeight(3 * 2, 6);
```

..and their value will be provided to the code inside our function.

A Word About Parameters

Functions can accept as few or as many parameters as necessary. Here is a function which accepts no parameters.

```
function sayHello() {
  return ‘Hello!’;
}

sayHello();
```

A Word About Parameters

Notice that we use parentheses differently when **defining** a function versus when we **invoke** that function.

```
function sayHello() {
  return ‘Hello!’;
}

sayHello();
```

A Word About Parameters

Notice that we use parentheses differently when **defining** a function versus when we **invoke** that function.

These parentheses are for including parameters when **defining** functions.

```
function sayHello() {
  return ‘Hello!’;
}

sayHello();
```

A Word About Parameters

Notice that we use parentheses differently when **defining** a function versus when we **invoke** that function.

```
function sayHello() {
  return ‘Hello!’;
}

sayHello();
```

These parentheses are for **invoking** the function. Notice the absence of the keyword function.

That's it!