Variables

Introduction to variables

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

```
function randInt(n) {
  return Math.floor(Math.random() * (n + 1));
}

function guessMyNumber(n) {
  if (n > 5) {
    return "Out of bounds! Please try a number between 0 and 5.";
  } else if (n === randInt(5)) {
    return "You guessed my number!";
  }
  return "Nope! That wasn't it!";
}
```

Take a look at the code for this number-guessing game.

```
function randInt(n) {
  return Math.floor(Math.random() * (n + 1));
}

function guessMyNumber(n) {
  if (n > 5) {
    return "Out of bounds! Please try a number between 0 and 5.";
  } else if (n === randInt(5)) {
    return "You guessed my number!";
  }
  return "Nope! That wasn't it!";
}
```

Presently, it accepts guesses between zero and five. If we wanted to accept guesses between zero and ten, what changes would we need to make?

```
function randInt(n) {
  return Math.floor(Math.random() * (n + 1));
}

function guessMyNumber(n) {
  if (n > 5) {
    return "Out of bounds! Please try a number between 0 and 5.";
  } else if (n === randInt(5)) {
    return "You guessed my number!";
  }
  return "Nope! That wasn't it!";
}
```

We need to make three changes - one for each reference to the value 5.

```
function randInt(n) {
  return Math.floor(Math.random() * (n + 1));
}

function guessMyNumber(n) {
  if (n > 5) {
    return "Out of bounds! Please try a number between 0 and 10.";
  } else if (n === randInt(10)) {
    return "You guessed my number!";
  }
  return "Nope! That wasn't it!";
}

guessMyNumber(8);
// => “Out of bounds! Please try a number between 0 and 10.”
```

If we forget to update any of those references, our function might behave in unexpected ways!

```
function randInt(n) {
  return Math.floor(Math.random() * (n + 1));
}

function guessMyNumber(n) {
  if (n > 10) {
    return "Out of bounds! Please try a number between 0 and 10.";
  } else if (n === randInt(10)) {
    return "You guessed my number!";
  }
  return "Nope! That wasn't it!";
}
```

What if there were a way to change the upper bound to our guessing game *without* the need to change three different references in our function? **Variables** will help us solve this problem.

Variables

A variable is a **label** that is associated with a **value**.

We can refer to the same value throughout our program by its label, or name.

Creating Variables in JavaScript

We can use this syntax to create a variable.

```
var x = 10;
```

Creating Variables in JavaScript

We can use this syntax to create a variable.

```
var x = 10;
```

var is a special keyword indicates that we are **creating** a new variable. The **only** time we use this keyword is when we are making a new variable.

Creating Variables in JavaScript

We can use this syntax to create a variable.

```
var x = 10;
```

This is the **name** of our variable. We'll use this name elsewhere in our program to refer to the value we wish to store.

Creating Variables in JavaScript

We can use this syntax to create a variable.

```
var x = 10;
```

This is the **assignment operator**.

Creating Variables in JavaScript

We can use this syntax to create a variable.

```
var x = 10;
```

This is the value we wish to associate with the label x.

Naming Variables

When choosing variable names, follow these conventions:

- Start your name with a lowercase letter

```
var x = 10;
```

Naming Variables

When choosing variable names, follow these conventions:

- Start your name with a lowercase letter
- Use descriptive words

```
var x = 10;

var age = 7;
```

Naming Variables

When choosing variable names, follow these conventions:

- Start your name with a lowercase letter
- Use descriptive words
- If your variable name contains multiple words, use camelCase

```
var x = 10;

var age = 7;

var ageInDogYears = 49;
```

Naming Variables

When choosing variable names, follow these conventions:

- Start your name with a lowercase letter
- Use descriptive words
- If your variable name contains multiple words, use camelCase

These conventions apply to naming functions and function parameters, too!

```
var x = 10;

var age = 7;

var ageInDogYears = 49;

function squareRoot(n) {
  // ...
}
```

Using Variables

Once we have created our variables, we can use their name as a substitute for their value elsewhere in our program.

There are certain regions of your program where you can access the variables you have declared. We'll explore this concept at the end of this lesson.

```
var x = 10;

x + 5; // => 15

x; // => 10

square(x) // => 100
```

Declaring a Variable Without a Value

Sometimes we may want to declare a variable before we know precisely what value we want to store in it.

We can do this by leaving off the assignment operator. If we refer to a variable that we have not assigned a value, it will produce the value undefined.

```
var answer;

answer; // => undefined
```

Changing a Variable's Value

You may want to reassign a variable to refer to a different value. You can use the **assignment operator** to accomplish this task.

Notice that when you change the value of a variable, you don't need to use the var keyword. The var keyword is only needed for creating new variables.

```
var pokemon = 'Charmander';

// charmander has evolved!
pokemon = 'Charmeleon';
```

Where can we use our variables?

Earlier in this lesson, we mentioned that the variables we declare are only available to us in certain regions of our code. These regions are called **scopes**.

Scopes

Every function creates its own **local scope**. Variables declared **inside** a local scope are only available within that scope. We've marked the local scope for our greetSomeone function in pink.

Variables declared outside a function body are in the **global scope**. They can be referred to anywhere in our program.

```
var greeting = 'Hello';

function greetSomeone() {
  var name = 'Josh';
  return greeting + ' ' + name;
}

greetSomeone(); // => ???
name; // => ???
```

Scopes

What value do you expect to see when we invoke greetSomeone?

```
var greeting = 'Hello';

function greetSomeone() {
  var name = 'Josh';
  return greeting + ' ' + name;
}

greetSomeone(); // => ???
name; // => ???
```

Scopes

What value do you expect to see when we invoke greetSomeone?

We'll get a string "Hello Josh" - the variable greeting is in the global scope and is available inside our greetSomeone function.

```
var greeting = 'Hello';

function greetSomeone() {
  var name = 'Josh';
  return greeting + ' ' + name;
}

greetSomeone(); // => "Hello Josh"
name; // => ???
```

Scopes

What do you think will happen when we make a reference to name at the bottom of this code?

```
var greeting = 'Hello';

function greetSomeone() {
  var name = 'Josh';
  return greeting + ' ' + name;
}

greetSomeone(); // => "Hello Josh"
name; // => ???
```

Scopes

What do you think will happen when we make a reference to `name` at the bottom of this code?

We get an error! Because we declared our `name` variable inside the scope of our `greetSomeone` function, we can only access it within that function's scope.

```
var greeting = 'Hello';

function greetSomeone() {
  var name = 'Josh';
  return greeting + ' ' + name;
}

greetSomeone(); // => "Hello Josh"
name; // => ReferenceError
```

Best Practice

When writing functions, we prefer that functions **only** make use of:

- Their **parameters**, or
- Variables **defined inside** of the function (the function's scope)

That's it!