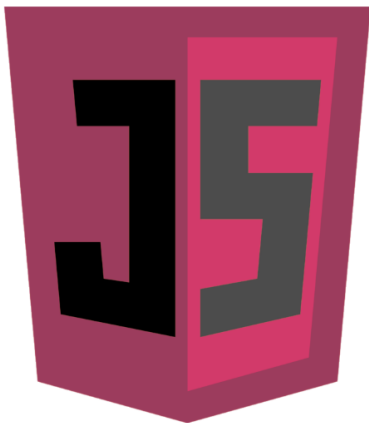


Variables

In this lesson, we will learn about JavaScript variables and how to use them.
Let's begin! :)

WE'LL COVER THE FOLLOWING

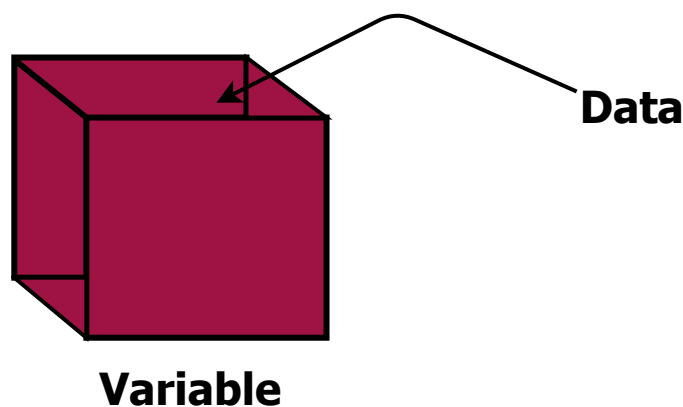
- Variable assignment
- Reference values and primitive values
- **Listing 7-8:** Distinction between primitive and reference values



Variables in JavaScript



As you learned earlier, JavaScript supports objects that can have dynamic properties, and so they can be used to represent compound things.



JavaScript also supports primitive or simple value types that store irreducible values such as numbers, strings, and Boolean flags. In this section, you will learn more about these values and types.

Variable assignment

To store either simple values or objects, you must assign them to variables:

```
var obj = new Object();  
obj.name = "Dave";  
var number = 42;  
var regExp = /(bat)?man/;
```



Here, `obj` and `regExp` variables hold object instances (a regular expression is represented by a `RegExp` object), while `number` holds a value instance (the number `42`).

While many languages (C, C++, C#) are statically-typed languages where a variable can hold only the instances of a well-determined type, **JavaScript is a loosely-typed language**, so the values assigned to the same variable can change its type during execution.

For example, in the next code snippet, the `myValue` variable changes its type several times:

```
var myValue = new Object();  
myValue.name = "Bob";  
console.log(myValue);  
myValue = "Bob";  
console.log(myValue);  
myValue = 12 + 23;  
console.log(myValue);
```



The console output indicates this fact:

```
Object {name: "Bob"}  
Bob  
35
```



In these code snippets you can see that variables can be declared with the **var** keyword. In contrast to other languages, JavaScript allows you to **redefine variables in the same scope**, as the following code snippet indicates:

```
var myValue = new Object();  
myValue.name = "Bob";  
console.log(myValue);  
var myValue = "Bob";  
console.log(myValue);  
var myValue = 12 + 23;  
console.log(myValue);
```



Show Useful Info

Reference values and primitive values

JavaScript makes a distinction between **reference values** and **primitive (single) values**.

While primitive values store simple atomic (irreducible) pieces of data, reference values are objects that may hold multiple values through properties.

The **distinction** the language defines is very important, because reference values and primitive values are handled in different ways.

The first difference between them is the way they are **stored and copied**.

Running the code in Listing 7-8 demonstrates this:

Listing 7-8: Distinction between primitive and

reference values

```
<!DOCTYPE html>
<html>
<head>
  <title>Copying values</title>
  <script>
    // Define two objects
    var obj1 = new Object();
    obj1.name = "Steve";
    var obj2 = obj1;
    obj2.name = "Bob";
    console.log(obj1.name
      + ", " + obj2.name);

    // Define two strings
    var str1 = "Steve";
    var str2 = str1;
    str2 = "Bob";
    console.log(str1 + ", " + str2);
  </script>
</head>
<body>
  Listing 7-8: View the console output
</body>
</html>
```

This code creates an object, stores it in the `obj1` variable, and then sets its `name` property.

After copying the value of `obj1` to `obj2` and updating its value, you may think you have two separate objects named Steve and Bob. However, the first `console.log()` produces this output, showing that you're wrong:



Objects hold reference values. When you define variables with reference values in a context, the value of the object (a set of properties it holds) is stored on the heap, separately from the context that holds only references to the value stored on the heap.

The image below shows how these three lines that copy string values result the console output:

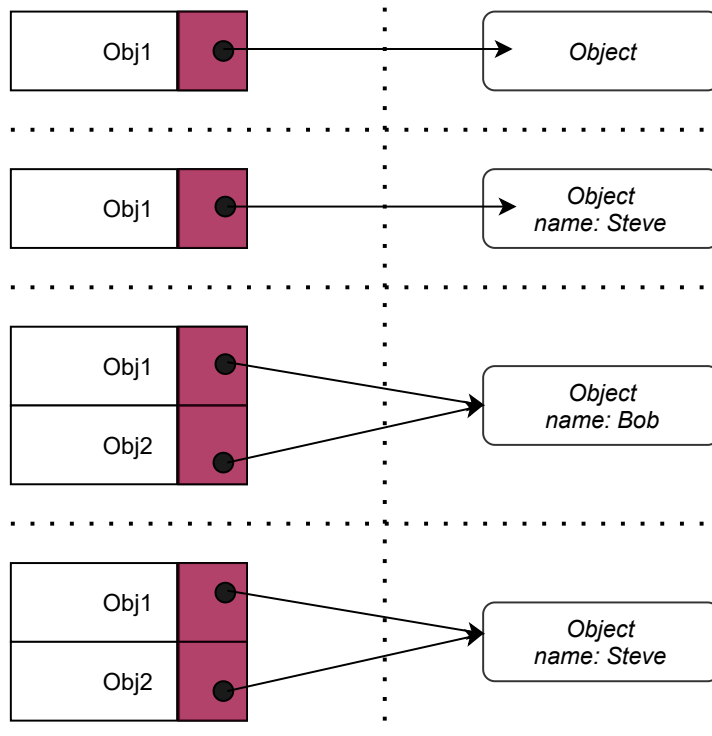
```
var obj1 = new Object();
obj1.name = "Steve";
```

```
obj1.name = "Steve";  
var obj2 = obj1;  
obj2.name = "Bob";
```



Context


Heap



The reference value copy operation steps

As the image above indicates, at the end of the operation both `obj1.name` and `obj2.name` contain “Bob”.

In contrast to reference values, **primitive values are stored in the context**. The second line of the console output shows this:

 console

Steve, Bob



The image below shows how these three code lines—which copy string values—result the console output:

```
var str1 = "Steve";  
var str2 = str1;  
str2 = "Bob";
```



Context

1	<table><tr><td>str1</td><td>Steve</td></tr></table>	str1	Steve
str1	Steve		

.....

2	<table><tr><td>str1</td><td>Steve</td></tr><tr><td>str2</td><td>Steve</td></tr></table>	str1	Steve	str2	Steve
str1	Steve				
str2	Steve				

.....

3	<table><tr><td>str1</td><td>Steve</td></tr><tr><td>str2</td><td>Steve</td></tr></table>	str1	Steve	str2	Steve
str1	Steve				
str2	Steve				

The primitive value copy operation steps

Achievement unlocked! 🎉

Congratulations! You've gained some very useful knowledge on JS variables.

All around amazing work!

Give yourself a round of applause! :)



Now, you can declare variables and know the difference between storing reference and primitive values. But what are the real types you can use in JavaScript?

We'll get to know them in the *next lesson*.

Stay tuned! :)

