CSCI 1720

JavaScript – Part 1

JavaScript

A Little History

History

JavaScript, often abbreviated as JS, is a high-level, dynamic, weakly typed, object-based, and interpreted programming language

Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production

It is used to make webpages interactive and provide online programs, including video games

History

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design

Although it was developed under the name Mocha, the language was officially called LiveScript when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995

History

It was renamed JavaScript when it was deployed in the Netscape Navigator 2.0 beta 3 in December

The final choice of name caused confusion, giving the impression that the language was a spin-off of the Java programming language

The choice has been characterized as a marketing ploy by Netscape to give JavaScript the cachet of what was then the hot new Web programming language

History

- 1994: Mosaic Netscape 0.9 released
- By 1995, Netscape commanded ¾ of the browser market / became the main browser of the 1990s
- Subsequently renamed Netscape Navigator (Mosaic became Netscape Communications)

History

- Soon realized that the Web should be more dynamic
- 1995: NSC recruited Brendan Eich / wrote first version prototype in 10 days (May)
- Microsoft got into the game with Jscript and included support for CSS and various extensions to HTML

History

- The competition between Netscape and Microsoft led to non-standard implementations

- November 1996, Netscape submitted JavaScript to Ecma International to carve out a standard specification, which other browser vendors could then implement based on the work done at Netscape

History

- The standards process continued in cycles, with the release of ECMAScript 2 in June 1998
- The release of ECMAScript 3 followed in December 1999, which is the baseline for modern day JavaScript
- At first, Microsoft seemed to participate and even implemented some of the proposals in their JScript .NET language
- With all the haggling back and forth, the Open Source community became involved

History

- 2005: Open source libraries released supporting JavaScript (e.g., Prototype, jQuery, Dojo Toolkit, and others)
- 2008 Oslo Meeting: agreement in early 2009 to standardize

These are the high points…as you can imagine, a lot happened over 14 years, and beyond, to lead us to the JavaScript in use today

First Things First

The Document Object Model (DOM)

11

First Things First: Understanding the DOM

When a web page is loaded, the browser creates a Document Object Model (DOM) of the page

The HTML DOM model is constructed as a tree of Objects

12

First Things First: Understanding the DOM

With the object model, JavaScript gets all the power it needs to create dynamic HTML:

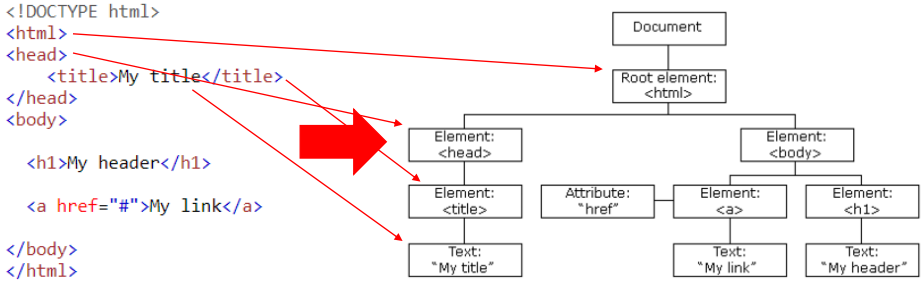
- JavaScript can change all the HTML elements in the page
- JavaScript can change all the HTML attributes in the page
- JavaScript can change all the CSS styles in the page
- JavaScript can remove existing HTML elements and attributes

First Things First: Understanding the DOM

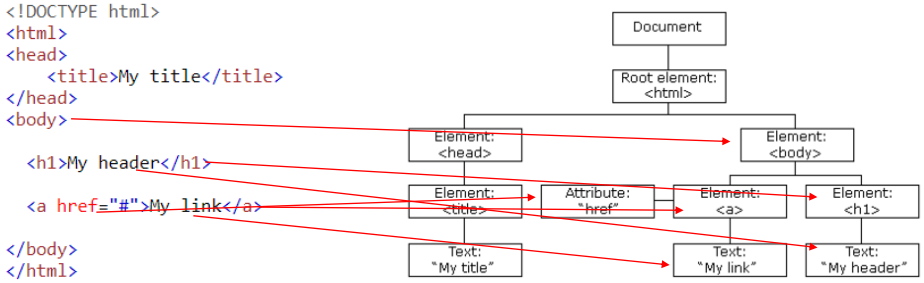
With the object model, JavaScript gets all the power it needs to create dynamic HTML:

- JavaScript can add new HTML elements and attributes
- JavaScript can react to all existing HTML events in the page
- JavaScript can create new HTML events in the page

First Things First: Understanding the DOM

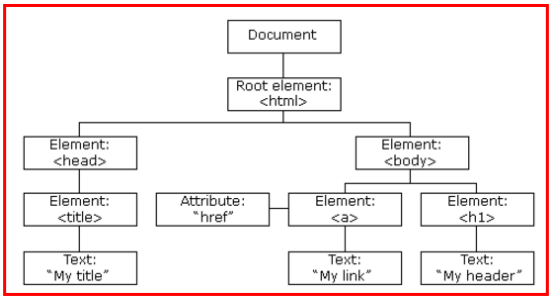


First Things First: Understanding the DOM



First Things First: Understanding the DOM

```
<!DOCTYPE html>
<html>
<head>
    <title>My title</title>
</head>
<body>

  <h1>My header</h1>

  <a href="#">My link</a>

</body>
</html>
```

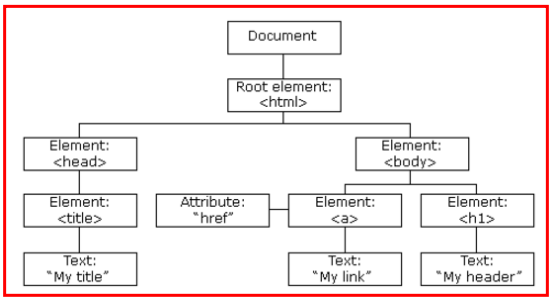


This is how the browser 'sees' the web page

By assigning ids to elements, we can use JavaScript to access (and modify) them

First Things First: Understanding the DOM

```
<!DOCTYPE html>
<html>
<head>
    <title>My title</title>
</head>
<body>

  <h1>My header</h1>

  <a href="#">My link</a>

</body>
</html>
```



It should also be evident that the DOM can grow to be extremely complex as the complexity of a given page increases

Classes and IDs

Classes and IDs

We've seen throughout our exploration of CSS frameworks how important classes are to styling web pages

IDs are the primary way in which JavaScript accesses and dynamically modifies web pages and their elements

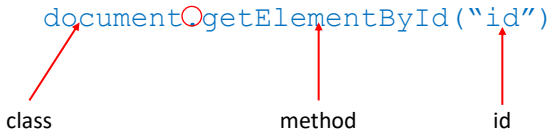
There is some bleed-over, however:

- CSS can style IDs
- JavaScript can access elements by class

Simple Example

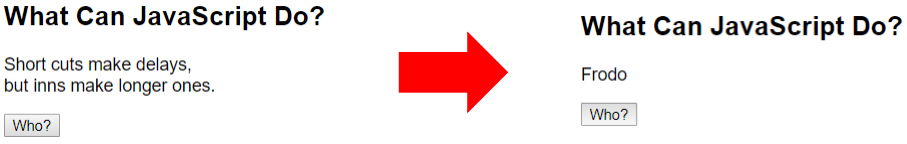
One of the many JavaScript methods is `getElementById()`

This method is part of the `document` class and is thus invoked using dot (“.”) notation (One of the language’s similarities with Java), i.e.,



Simple Example

```
<h2>What Can JavaScript Do?</h2>

<p id="demo">
 Short cuts make delays,<br>
 but inns make longer ones.
</p>

<button type="button" onclick="document.getElementById('demo').innerHTML =
 'Frodo'">Who?
</button>
```



JavaScript - Including With HTML

JS - Including With HTML

The `<script>` Element

In HTML, JavaScript code must be inserted between `<script>` and `</script>` tags

```
<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
  document.getElementById("demo").innerHTML =
  "Do not meddle in the affairs of Wizards, for" +
  "<br>they are subtle and quick to anger.";
</script>

</body>
```

JavaScript in Body

Do not meddle in the affairs of Wizards, for
they are subtle and quick to anger.

JS - Including With HTML

JavaScript Functions and Events

A JavaScript function is a block of JavaScript code, that can be executed when "called" for

For example, a function can be called when an event occurs, like when the user clicks a button

We will talk more about functions and events later

JS - Including With HTML

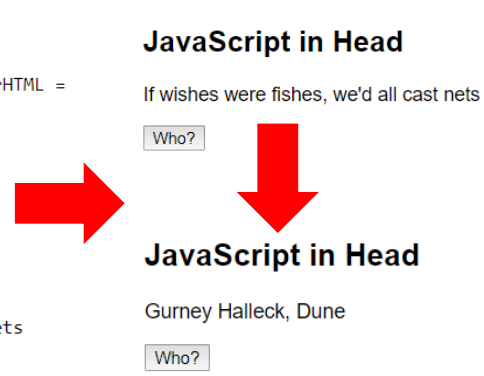
JavaScript in `<head>`

In this example, a JavaScript function is placed in the `<head>` section of an HTML page

The function is invoked (called) when a button is clicked

JS - Including With HTML Example - <head>

```
<head>
<script>
function myFunction() {
    document.getElementById("demo").innerHTML =
    "Gurney Halleck, Dune";
}
</script>
</head>

<body style="font-family:sans-serif;">

<h2>JavaScript in Head</h2>

<p id="demo">
  If wishes were fishes, we'd all cast nets
</p>

<button type="button" onclick="myFunction()">Who?</button>

</body>
```



JS - Including With HTML

JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page

The function is invoked (called) when a button is clicked

JS - Including With HTML Example - <body>

```
<body>
 <h2>JavaScript in Body</h2>

 <p id="demo">
  Happiness consists in getting enough sleep.<br>
  Just that, nothing more.
 </p>

 <button type="button" onclick="myFunction()">
  Who?
 </button>

 <script>
  function myFunction() {
    document.getElementById("demo").innerHTML =
    "Robert A. Heinlein";
  }
 </script>
</body>
```

JS - Including With HTML

External JavaScript

Scripts can also be placed in external files

External scripts are practical when the same code is used in many different web pages

JavaScript files have the file extension **.js**

To use an external script, put the name of the script file in the `src` (source) attribute of a `<script>` tag

JS - Including With HTML - External Example

```
<p id="demo">
 Never attempt to teach a pig to sing; it wastes<br>
 your time and annoys the pig.
</p>

<button type="button" onclick="myFunction()">
 Who?
</button>

<p>(myFunction is stored in an external<br>
file called "myScript.js")</p>

<script src="myScript.js"></script>
```

Never attempt to teach a pig to sing;
it wastes your time and annoys the pig.

Who?

(myFunction is stored in an external
file called "myScript.js")

Robert A. Heinlein

Who?

(myFunction is stored in an external
file called "myScript.js")

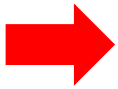
JS - Including With HTML

External scripts can be referenced with a full URL or with a path relative to the current web page

This example uses a relative URL to link to a script

JS - Including With HTML - External Example

```
<p id="demo">
 Progress isn't made by early risers. It's made<br>
 by lazy men trying to find easier ways to do <br>
 something.
</p>

<button type="button" onclick="myFunction()">
 Try it
</button>

<p>
  (myFunction is stored in an external<br>
  file called "myScript.js" in the /js<br>
  folder)
</p>

<script src="js/myScript.js"></script>
```



Progress isn't made by early risers. It's made by lazy men trying to find easier ways to do something.

Try it

(myFunction is stored in an external file called "myScript.js" in the /js folder)

Robert A. Heinlein

Who?

(myFunction is stored in an external file called "myScript.js" in the /js folder)

JS - Including With HTML

External JavaScript Advantages

- Separates HTML and code
- Makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

To add several script files to one page - use several script tags

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

JS - Including With HTML

```
<body>
 <h2>JavaScript in Body</h2>

 <p id="demo">
  Happiness consists in getting enough sleep.<br>
  Just that, nothing more.
 </p>

 <button type="button" onclick="myFunction()">
  Who?
 </button>

 <script>
  function myFunction() {
    document.getElementById("demo").innerHTML =
    "Robert A. Heinlein";
  }
 </script>
</body>
```

Vs.

```
<body>

<h2>JavaScript in Body</h2>

<p id="demo">A Paragraph.</p>

<button type="button"
  onclick="myFunction()">Try it</button>

<script src="js/myScript.js"></script>
```

```
function myFunction() {
    document.getElementById("demo").innerHTML =
    "Paragraph changed.";
}
```

(file js/myScript.js)

JavaScript Output

JS Output

JavaScript can "display" data in different ways:

- Writing into an HTML element, using innerHTML
- Writing into the HTML output using document.write()
- Writing into an alert box, using window.alert()
- Writing into the browser console, using console.log()

JS Output - innerHTML

To access an HTML element, JavaScript can use the document.getElementById(id) method

The id attribute identifies the HTML element

The innerHTML property defines the HTML content

Changing the innerHTML property of an HTML element is a common way to display data in HTML

JS Output - innerHTML Example

```
<h2>Isaac Asimov</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
 "I do not fear computers.<br>" +
 "I fear the lack of them.";
</script>
```

Isaac Asimov

I do not fear computers.
I fear the lack of them.

JS Output - document.write()

For testing purposes, it is convenient to use document.write()

```
<h2>Isaac Asimov</h2>

<script>
 document.write("Isaac Asimov quote<br>" +
 "Should appear here");
</script>
```

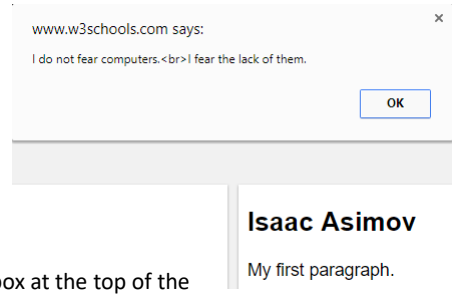
Isaac Asimov

Isaac Asimov quote
Should appear here

JS Output - window.alert()

You can use an alert box to display data

```
<h2>Isaac Asimov</h2>

<script>
window.alert("I do not fear computers.<br>" +
"I fear the lack of them.");
</script>
```



Note: The alert box appears as a drop-down box at the top of the browser window. The button must be clicked in order to return to the page

JavaScript Syntax

JS Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed

- A computer program is a list of instructions to be executed by the computer
- In a programming language, these program instructions are called statements
- JavaScript is a programming language
- JavaScript statements are separated by semicolons

JS Syntax - Statements

JavaScript statements are composed of Values, Operators, Expressions, Keywords, and Comments

The JavaScript syntax defines two types of values: Fixed values and variable values

Fixed values are called literals

Variable values are called variables

JS Syntax - Statements

JavaScript statements are composed of Values, Operators, Expressions, Keywords, and Comments

The JavaScript syntax defines two types of values: Fixed values and variable values

Fixed values are called literals

Numbers are written with or without decimals: `10.50` / `1001`

Strings are text, using either single or double quotes:

`"Hello JS!"` / `'Hello JS!'`

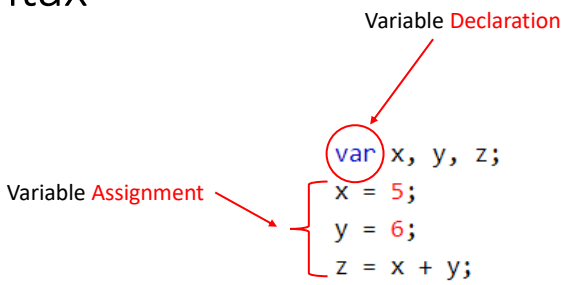
JS Syntax - Statements

In a programming language, variables are used to store data values

JavaScript uses the `var` keyword to declare variables

An equals sign is used to assign values to variables

JS Syntax



In HTML, JavaScript programs are executed by the web browser

JS Syntax - Statements

JavaScript uses arithmetic operators (+ - * /) to compute values

(Again this is identical to how Java and other programming languages depict operators)

The equals sign (=) is referred to as the assignment operator; it 'assigns' a value to a variable

JS Syntax - Statements

An expression is a combination of values, variables, and operators, which computes to a value

The computation is called an evaluation

For example, 5 * 10 evaluates to 50

```
<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = 5 * 10;
</script>
```

Expressions compute to values.

50

JS Syntax - Statements

An expression is a combination of values, variables, and operators, which computes to a value

The computation can also include variables

```
<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>
var x;
x = 5;
document.getElementById("demo").innerHTML = x * 10;
</script>
```

Expressions compute to values.

50

JS Syntax - Statements

An expression is a combination of values, variables, and operators, which computes to a value

The values can be of various types, such as numbers and strings

For example, "John" + " " + "Doe", evaluates to "John Doe"

```
<p>Expressions compute to values.</p>

<p id="demo"></p>

<script>
 document.getElementById("demo").innerHTML =
"John" + " " + "Doe";
</script>
```

Expressions compute to values.

John Doe

JS Syntax - Keywords

JavaScript keywords are used to identify actions to be performed

The `var` keyword tells the browser to create variables

```
var x, y;
x = 5 + 6;
y = x * 10;
```

You can't use a keyword as a variable name

JS Syntax – Keywords

None of these words can be used as a variable name

Some may be legitimately part of a name, however:

```
var in;      NO!
var inTable;  OK
```

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

JS Syntax - Comments

Not all JavaScript statements are "executed"

Code after double slashes `//` or between `/*` and `*/` is treated as a comment

Comments are ignored, and will not be executed

```
var x = 5;   // I will be executed

// var x = 6;   I will NOT be executed
```

Somebody, **please**, tell me why this is important

JS Syntax - Comments

```
function myhide(obj) {
  try {
    obj.style.display = 'none';
  } catch (e) {

  }
}
```

Quick! What does this do?

```
function myhide(obj) {
  /// <summary>This method will hide the DOM object specified in "obj"</summary>
  /// <returns type="void" />
  /// <param name="obj" type="object">Should contain a DOM object</param>
  try {
    obj.style.display = 'none';
  } catch (e) {
    // Something went wrong, we will ignore here
  }
}
```


JS Syntax - Identifiers

Identifiers are names

In JavaScript, identifiers are used to name variables (and keywords, and functions, and labels)

The rules for legal names are much the same in most programming languages

In JavaScript, the first character must be a letter, or an underscore (_), or a dollar sign ($)

Subsequent characters may be letters, digits, underscores, or dollar signs

Numbers are not allowed as the first character (Why?)

JS Syntax - Case Sensitivity

All JavaScript identifiers are case sensitive

The variables lastName and lastname, are two different variables

```
var lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```


JS Syntax - Camel Case

Historically, programmers have used different ways of joining multiple words into one variable name

Hyphens:

first-name, last-name, master-card, inter-city

*Hyphens are not allowed in JavaScript. They are reserved as the subtraction operator

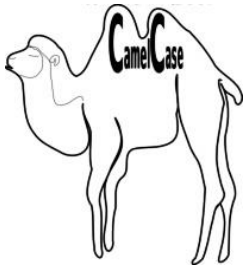
Underscore:

first_name, last_name, master_card, inter_city

JS Syntax - Camel Case

Upper Camel Case (Pascal Case):

FirstName, LastName, MasterCard, InterCity

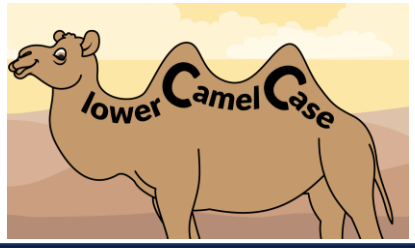


JS Syntax - Camel Case

Lower Camel Case:

JavaScript programmers tend to use camel case that starts with a lowercase letter

firstName, userLastName, masterCard, homeCity



JavaScript Statements

61

JS Statements

In HTML, JavaScript statements are instructions to be executed by the web browser

```
<p>In HTML, JavaScript statements are executed by the browser.</p>
<p id="demo"></p>

<script>
  document.getElementById("demo").innerHTML = "Dave: Open the pod bay doors, HAL." +
  "<br>HAL: I'm sorry, Dave. I'm afraid I can't do that..";
</script>
```

Statement

Execution

In HTML, JavaScript statements are executed by the browser.

Dave: Open the pod bay doors, HAL.
HAL: I'm sorry, Dave. I'm afraid I can't do that..

62

JS Programs

Most JavaScript programs contain many JavaScript statements

The statements are executed, one by one, in the same order as they are written

(Note: function calls)

JS Programs

In this example x, y, and z are given values, and finally z is displayed

```
<p>
  JavaScript code (or just JavaScript) <br>
  is a sequence of JavaScript statements
</p>

<p id="demo"></p>

<script>
  var x, y, z;
  x = 5;
  y = 6;
  z = x + y;
  document.getElementById("demo").innerHTML = z;
</script>
```

JavaScript code (or just JavaScript)
is a sequence of JavaScript statements

11

JavaScript Variables

JS Variables

JavaScript variables are containers for storing data values

In this example, x, y, and z, are variables

```
var x = 5;        x stores the value 5
var y = 6;        y stores the value 6
var z = x + y;    z stores the value 11
```

Much like Algebra

JS Variables

```
var price1 = 5;
var price2 = 6;
var total = price1 + price2;
```

In programming, just like in algebra, we use variables (like price1) to hold values

In programming, just like in algebra, we use variables in expressions (total = price1 + price2)

From the example above, you can calculate the total to be 11

JS Identifiers

All JavaScript variables must be identified with unique names

These unique names are called identifiers

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume)

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names can also begin with $ and _
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names

JS Assignment Operator

In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator

This is different from algebra. The following does not make sense in algebra

$$x = x + 5$$

In JavaScript, however, it makes perfect sense: it assigns the value of x + 5 to x

(It calculates the value of x + 5 and puts the result into x. The value of x is incremented by 5.)

JS Data Types

JavaScript variables can hold numbers like 100 and text values like "John Doe"

In programming, text values are called text strings

JavaScript can handle many types of data, but for now, just think of numbers and strings

Strings are written inside double or single quotes. Numbers are written without quotes

If you put a number in quotes, it will be treated as a text string

JS Data Types

```
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';
```

In this example,

The value 3.14 is assigned to a variable named ‘pi’

The value “John Doe” is assigned to a variable named ‘person’

The value “Yes I am!” is assigned to a variable named ‘answer’

JS Data Types

Creating a variable in JavaScript is called declaring a variable

You declare a JavaScr `var carName;` with the var keyword

After the declaration, the variable has no value (Technically it has t `carName = "Volvo";` efined)

To assign a value to the variable, use the equal sign

JS Data Types

You can also assign a value to the variable when you declare it

```
var carName = "Volvo";
```

```
<p>
  Create a variable,<br>
  assign a value to it,<br>
  and display it:
</p>

<p id="demo"></p>

<script>
var carName = "Volvo";
document.getElementById("demo").innerHTML = carName;
</script>
```

Create a variable,
assign a value to it,
and display it:

Volvo

JS Data Types

You can declare many variables in one statement

Start the statement with `var` and separate the variables by comma

```
var person = "John Doe", carName = "Volvo", price = 200;
```

JS Data Types

In computer programs, variables are often declared without a value

The value can be something that has to be calculated, or something that will be provided later, like user input

A variable declared without a value will have the value undefined

If you re-declare a JavaScript variable, it will not lose its value

Sources

https://www.w3schools.com/js/js_htmldom.asp
https://www.w3schools.com/js/default.asp
http://lotrproject.com/quotes/
https://www.brainyquote.com/quotes/quotes/i/isaacasimo100104.html
https://www.goodreads.com/author/quotes/205.Robert_A_Heinlein
https://en.wikiquote.org/wiki/Dune
http://www.imdb.com/title/tt0062622/quotes

Copyrights