Intro to web development

# Key points to cover

* How websites work?

Day 1

Day 2

Day 3

Day 4

Day 5

* Web pages
* Browsers
* Urls, domain names dns
* HtmL
* CSS
* Javascript
* Basics of nodejs

# What is Internet, what is web?

Internet -> giant network of networks. It does not do anything by itself. Its an infrastructure.

From the [definition in the Wikipedia](http://en.wikipedia.org/wiki/Internet): "The Internet is a global system of interconnected computer networks that interchange data by packet switching using the standardized Internet Protocol Suite (TCP/IP)."

Thus, the Internet is a network of networks, defined by the TPC/IP standards.

The Web, on the other hand, is defined in W3C's [Architecture of the World Wide Web, Volume I](https://www.w3.org/TR/webarch/) as follows: "The World Wide Web (WWW, or simply Web) is an information space in which the items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (URI)."

Thus, the Web is an information space. The first three specifications for Web technologies defined URLs, HTTP, and HTML.

“World wide web” or simply “web” is a way of accessing the information through Internet.

It takes a whole set of standards and commonly accepted information-sharing model that make websites work.

Video: <https://youtu.be/scWj1BMRHUA>

Watch till 6 minutes

## A little about standards

Almost everything in Information technology is driven by something called standards. Technology Standards are basically agreements on specifications among manufacturers to achieve compatibility of products and the ability to inter-operate

Optional example >

*Example: Bob is Italian and Alice is Vietnamese. Say Bob and Alice are following each other on Instagram. But they don’t know any details about each others (Cos their profiles are in their own native languages). One day Bob decides to message Alice. If Bob sends the message in Italian, and Alice doesn’t know Italian she wouldn’t bother to reply.*

*In the real world, Bob would assume Alice knows English and send a message in English. Chances are good that Alice know English and may or may not give a reply.*

In this example English is a standard accepted by Alice and Bob. Internet is like Instagram, everything you need to send the message is already there for you to use. But you need a commonly accepted system (English) to make a meaningful communication.

## So what is W3C?

The World Wide Web Consortium (W3C) is an international community where [Member organizations](https://www.w3.org/Consortium/Member/List), a full-time [staff](https://www.w3.org/People/), and the public work together to develop [Web standards](https://www.w3.org/standards/). W3C's mission is to lead the Web to its full potential. More details about w3c is available at <https://www.w3.org/Help/#activity>

## What makes web work

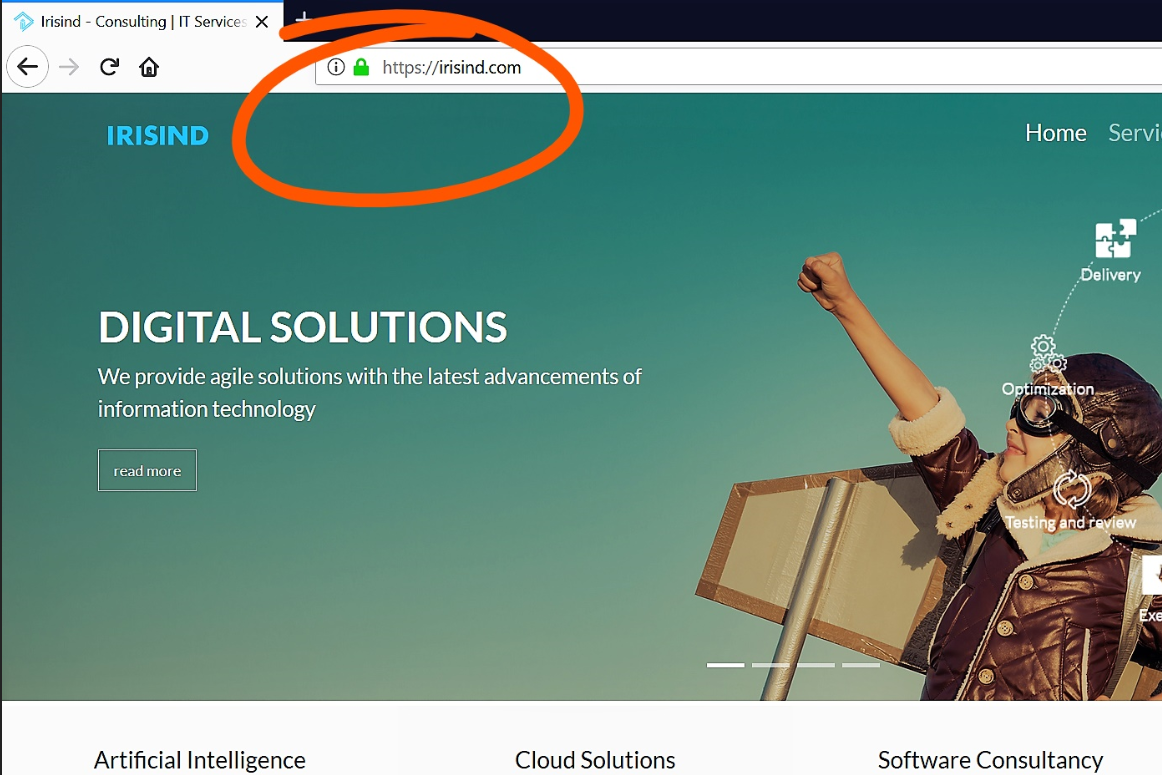
According to w3c there are 3 basic standards that are necessary to make the web work.

They are:

### Web URIs – the addressing standard

A Universal Resource Identifier (Uri), is defined to be an ASCII string used to identify things on the Semantic Web.

What you see in a browser address bar is a web URI, but a more familiar name might be **URL** or a **web address**

****Consider the example [https://**www.w3.org**/Help/](https://www.w3.org/Help/)

A web URI is composed of several parts. The first part is the protocol identifier. For a web link it is usually **http://** or **https://** . Here we are talking about http URI only.

eg: **https://**

The rest of the URI depends on the protocol The next part of an http URI is the **fully qualified domain name.** Eg: www.w3.org.

The 3rd part is the path at which a web page is located. Eg /Help/

### HTTP – the Hypertext standard

A **hypertext** is simply a document that has links in it that points to other resources. **Hypertext Transfer Protocol** (**HTTP**) is the protocol to exchange or transfer hypertext.

HTTP functions as a [request–response](https://en.wikipedia.org/wiki/Request%E2%80%93response) protocol in the client–server computing model. A [web browser](https://en.wikipedia.org/wiki/Web_browser), for example, may be the client and an application running on a computer [hosting](https://en.wikipedia.org/wiki/Host_(network)) a [website](https://en.wikipedia.org/wiki/Website) may be the server. The client submits an HTTP request message to the server. The server, which provides resources such as [HTML](https://en.wikipedia.org/wiki/HTML) files and other content, or performs other functions on behalf of the client, returns a response message to the client. The response contains completion status information about the request and may also contain requested content in its message body.

### Html – The document markup standard

HTML is a language used to create hypertext documents which is key to make web work.

Exercise:

Open any website in your browser (Chrome or Firefox). Press Ctrl + U

(Google.com is a bad example. Try w3.org)

It looks like this:



This is an example of an html document. A browser interprets this to display the original web page.

# Browsers

Chrome, Firefox, Safari, Edge, Opera.

You use it for browsing the web. These are software capable of handling all the web protocols and standards behind the scene. The user enters the URL in the address bar and get the web page displayed. To get there, the browser does a couple of things.

* Resolves the URL, finds out where the server is located
* Sends a HTTP request to the server to get the HTML page to be displayed
* Waits and reads the response from the server
* Parses the HTTP response. In addition to the HTML content, there could be other instructions to the browser
* Parses the html content and starts rendering the page
* Loads all the dependencies files (css, javascript, images etc) embedded in the page.
* Runs javascript

Modern browsers come with a host of developer tools. Most common ones are available through the inspect element context menu. (right click anywhere on the document).

* Elements tab explores the html of the current document. It allows you to modify the DOM directly. You can add , remove or modify htm elements in this view. Click on a single html element and the side panels will show the styles applied to that element
* Console tab displays the Javascript terminal. You can run live javascripts in there and read script outputs from the current document.
* Network tab shows all the http requests done in current document. This include the 1st request to load the html and all the subsequent requests to load linked assets like image files, javascript files, CSS files etc
* There are more tabs but not relevant to this discussion

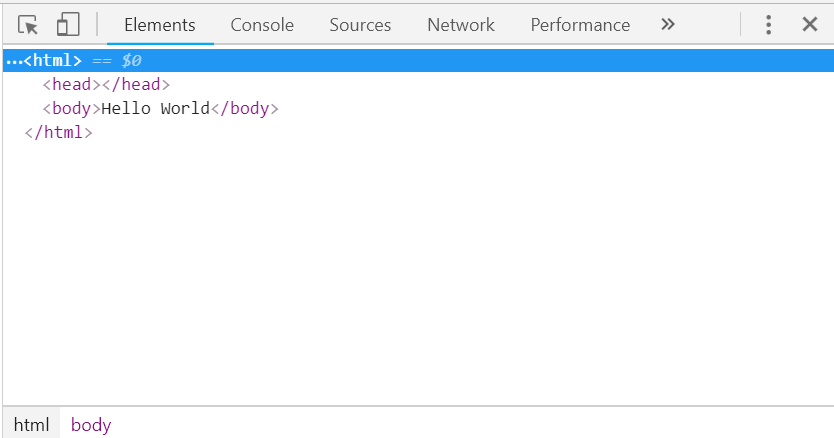
## HTMl

Exercise

Task 1

* Create a project directory
* Create a file named sample.html and save it.
* Open that file in a text editor. Notepad or gedit or sublime, anything
* Type in Hello World and save.
* Open the file in your browser and it prints the text as it is.

Now this is not a valid html file. But your browser doesn’t mind. It corrects your mistakes (Not all, some of your mistakes)

To see this in action, open inspect element on that page (A common short cut is F12 key) and go to elements tab.

Now there are some text inside angle brackets that where not there in your original file. These are called tags. These are added by the browser to make it a valid html document.

A valid Html document is composed of HTML tags and content text inside them

The following section is copied from w3schools HTML introduction

## **A Simple HTML Document**

### **Example**

<!DOCTYPE html>  
<html>  
<head>  
<title>Page Title</title>  
</head>  
<body>  
  
<h1>My First Heading</h1>  
<p>My first paragraph.</p>  
  
</body>  
</html>

Copy this and save it in your sample.html file. Then open it in browser

### **Example Explained**

* The <!DOCTYPE html> declaration defines this document to be HTML5
* The <html> element is the root element of an HTML page
* The <head> element contains meta information about the document
* The <title> element specifies a title for the document. See the text on the browser tab
* The <body> element contains the visible page content
* The <h1> element defines a large heading
* The <p> element defines a paragraph

## **HTML Tags**

HTML tags are element names surrounded by angle brackets:

<tagname>content goes here...</tagname>

* HTML tags normally come **in pairs** like <p> and </p>
* The first tag in a pair is the **start tag,** the second tag is the **end tag**
* The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

**Tip:** The start tag is also called the **opening tag**, and the end tag the **closing tag**.

For deeper html tutorial go to https://www.w3schools.com/html/html\_intro.asp

The most important HTML tag is the <a> tag called a hyperlink. A hyperlink enables the document to link to other HTML documents on the web.

Modify the sample.html to look like this

<title>Page Title</title>  
</head>  
<body>  
  
<h1>My First Heading</h1>  
<p>My first paragraph.</p>  
<a href=”https://google.com”>Go to google.</a>  
  
</body>  
</html>

Notice the line added just before the </body> closing tag. This is how a hyperlink is defined in a web page. You can have multiple html files in a folder and link to them to each other just like this using the relative path instead of https://google.com

Save the file and open in browser.

Exercise

Let students create 2 or more files like sample.1.html andsample.2.html, let them add content and experiment with different tags, let them find new tags. There should be atleast 2 html files and they should be linked to each other like sample.1.html and sample.2.html

## CSS

## Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML. CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.

HTML allows for some basic stylings, but to make it more customized you need CSS

Exercise

open the sample.html in browser

Right click My first paragraph, and inspect element.

In the styles side panel, click the section with element.style section.

Type in color: red

Play with different style options there.

<https://www.w3schools.com/css/css_intro.asp> contains a list of all css styles and a few examples to try out. Try some of those in the inspect element style editor

You’ll notice 2 things

1. The color of text in the document changed to red
2. <p> tag in inspector becomes <p style="color: red;">

The style="color: red;" part is called an attribute, color: red; is the value of style attribute

The value of a style attribute is supposed to be a valid css rule.

Style attribute is not the only place to put css rules. There are 2 other places to do that.

1. You can add a style tag in your HTML ( preferably inside <head>) as follows.  
   <style>h1{color:blue}</style>  
   This will change the heading color to blue
2. Create a new file called style.css and put h1{color:blue} in it and save in the same folder as the sample.html.  
   Add <link rel="stylesheet" type="text/css" href="style.css"> anywhere inside the <head> tag in the html file

When style is written as attribute (also called inline style), the style written applies only to that particular element. To apply the same style as inline, you only need style=”color:blue” applied to <h1> tag.

The other two methods apply the style to entire document. So there should be a way to map the style rule to a specific element. This is where css selectors comes in.

In the above example h1{color:blue} h1 is a selector, it matches all h1 elements in the documents.

3 of the most common selection handles are tagname, id and class

A selector #\_id matches a elements with id=“\_id” attribute. Id selectors match one to one which means the id must exactly match the selector

A selector .className matches all elements that has className in its class attribute. A class attribute takes in multiple classes separated by space. Eg: class=”class1 className anotherClass” If any one matches the selector, the style is applied

A selector without a . or # at start is a tagname selector. It matches any element with that tag name

A composition of these selectors can be used to resolve elements in a very complex document.

There are modifiers and pseudo-classes to even fine tune the selectors

See the w3 schools page to do more interesting things in class

https://www.w3schools.com/css/default.asp

## Javascript

Exercise

Open any web page, open inspect element > console tab

Type in alert(‘Hello World’); and press enter

Alert is the javascript function to show a popup message window.

Javascript enables the webpages to be interactive. But there is a clear distinction between Javascript (JS) and what JS can do in a browser.

JS is a scripting language. [ECMA](http://www.ecma-international.org/) defines the language specifications whereas browser standards are set by W3C (another organization).

So how do they sit together?

### The story of JS

*Beginnings at Netscape // From wikipedia*

*In 1993, the*[*National Center for Supercomputing Applications*](https://en.wikipedia.org/wiki/National_Center_for_Supercomputing_Applications)*(NCSA), a unit of the*[*University of Illinois at Urbana-Champaign*](https://en.wikipedia.org/wiki/University_of_Illinois_at_Urbana%E2%80%93Champaign)*, released*[*NCSA Mosaic*](https://en.wikipedia.org/wiki/Mosaic_(web_browser))*, the first popular graphical*[*Web browser*](https://en.wikipedia.org/wiki/Web_browser)*, which played an important part in expanding the growth of the nascent*[*World Wide Web*](https://en.wikipedia.org/wiki/World_Wide_Web)*. In 1994, a company called*[*Mosaic Communications*](https://en.wikipedia.org/wiki/Netscape)*was founded in*[*Mountain View, California*](https://en.wikipedia.org/wiki/Mountain_View,_California)*and employed many of the original NCSA Mosaic authors to create*[*Mosaic Netscape*](https://en.wikipedia.org/wiki/Netscape_Navigator)*. However, it intentionally shared no code with NCSA Mosaic. The internal codename for the company's browser was Mozilla, which stood for "Mosaic killer", as the company's goal was to displace NCSA Mosaic as the world's number one web browser. The first version of the Web browser, Mosaic Netscape 0.9, was released in late 1994. Within four months it had already taken three-quarters of the browser market and became the main browser for the Internet in the 1990s. To avoid trademark ownership problems with the NCSA, the browser was subsequently renamed Netscape Navigator in the same year, and the company took the name Netscape Communications. Netscape Communications realized that the Web needed to become more dynamic.*[*Marc Andreessen*](https://en.wikipedia.org/wiki/Marc_Andreessen)*, the founder of the company believed that*[*HTML*](https://en.wikipedia.org/wiki/HTML)*needed a "glue language" that was easy to use by Web designers and part-time programmers to assemble components such as images and plugins, where the code could be written directly in the Web page markup.*

*In 1995, Netscape Communications recruited*[*Brendan Eich*](https://en.wikipedia.org/wiki/Brendan_Eich)*with the goal of embedding the*[*Scheme*](https://en.wikipedia.org/wiki/Scheme_(programming_language))*programming language into its Netscape Navigator.*[*[10]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-10)*Before he could get started, Netscape Communications collaborated with*[*Sun Microsystems*](https://en.wikipedia.org/wiki/Sun_Microsystems)*to include in Netscape Navigator Sun's more static programming language*[*Java*](https://en.wikipedia.org/wiki/Java_(programming_language))*, in order to compete with*[*Microsoft*](https://en.wikipedia.org/wiki/Microsoft)*for user adoption of Web technologies and platforms.*[*[11]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-11)*Netscape Communications then decided that the scripting language they wanted to create would complement Java and should have a similar syntax, which excluded adopting other languages such as*[*Perl*](https://en.wikipedia.org/wiki/Perl)*,*[*Python*](https://en.wikipedia.org/wiki/Python_(programming_language))*,*[*TCL*](https://en.wikipedia.org/wiki/Tcl)*, or Scheme. To defend the idea of JavaScript against competing proposals, the company needed a prototype. Eich wrote one in 10 days, in May 1995.*

*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, but it was renamed****JavaScript****[[2]](https://en.wikipedia.org/wiki/JavaScript" \l "cite_note-press_release-2) when it was deployed in the Netscape Navigator 2.0 beta 3 in December.*[*[12]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-techvision-12)*The final choice of name caused confusion, giving the impression that the language was a spin-off of the Java programming language, and the choice has been characterized[[13]](https://en.wikipedia.org/wiki/JavaScript" \l "cite_note-13) as a marketing ploy by Netscape to give JavaScript the cachet of what was then the hot new Web programming language.*

*There is a common misconception that JavaScript was influenced by an earlier Web page scripting language developed by [Nombas](https://en.wikipedia.org/wiki/Openwave" \o "Openwave) named Cmm (not to be confused with the later*[*C--*](https://en.wikipedia.org/wiki/C--)*created in 1997).*[*[14]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-14)[*[15]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-15)*Brendan Eich, however, had never heard of Cmm before he created LiveScript.*[*[16]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-16)*Nombas did pitch their embedded Web page scripting to Netscape, though Web page scripting was not a new concept, as shown by the [ViolaWWW](https://en.wikipedia.org/wiki/ViolaWWW" \o "ViolaWWW) Web browser.*[*[17]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-17)*Nombas later switched to offering JavaScript instead of Cmm in their ScriptEase product and was part of the TC39 group that standardized ECMAScript.*[*[18]*](https://en.wikipedia.org/wiki/JavaScript#cite_note-18)

### Summary

* Started as part of Netscape
* In November 1996, Netscape submitted JavaScript to [ECMA International](https://en.wikipedia.org/wiki/Ecma_International) to carve out a standard specification, which other browser vendors could then implement based on the work done at Netscape
* Competing browsers now have JS capabilities.
* W3C Specifies a set of APIs in JS to make browser do things.

The Difference? Browsers have a complete JS engine built into it so they can run JS code. They also include apis that integrates with the JS Engine.

So all the functions you use in a browser are not Native to JS. Some of them are added by the browser as part of these apis

For Example, the alert() is a Method available in browsers but it may not work in another JS implementation (like NodeJS)

You can use the browser’s console as a playground to try out your scripts. (Open inspect element > go to console tab)

JS code is a sequence of statements. A statement is composed of Values, Operators, Expressions, Keywords, and Comments

### Comments

Not all JavaScript statements are "executed".

There are 2 types of comments

// is a single line comment. Anything after a double forward slash in a line is a comment

/\* … \*/ anything between a ‘/\*’ and a ‘\*/’ is a multi-line comment.

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

var /\* This part is ignored \*/ y = 10;

Semicolons (;) separate JavaScript statements. But there is Automatic Semicolon Insertion in JavaScript, which might be confusing at start. So it is advised to use semicolons after every statement.

Exercise

Try this section in browser console line by line

var a, b, c;     // Declare 3 variables  
a = 5;           // Assign the value 5 to a  
b = 6;           // Assign the value 6 to b  
c = a + b;       // Assign the sum of a and b to c

JavaScript ignores multiple spaces. You can add white space to your script to make it more readable.

The following lines are equivalent:

var person = "Hege";  
var person="Hege";

### JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| break | Terminates a switch or a loop |
| continue | Jumps out of a loop and starts at the top |
| debugger | Stops the execution of JavaScript, and calls (if available) the debugging function |
| do ... while | Executes a block of statements, and repeats the block, while a condition is true |
| for | Marks a block of statements to be executed, as long as a condition is true |
| function | Declares a function |
| if ... else | Marks a block of statements to be executed, depending on a condition |
| return | Exits a function |
| switch | Marks a block of statements to be executed, depending on different cases |
| try ... catch | Implements error handling to a block of statements |
| Var, let, const | Declares a variable |

### JavaScript Values

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

Fixed values are called **literals**. Variable values are called **variables**.

### JavaScript Literals

The most important rules for writing fixed values are:

**Numbers** are written with or without decimals:

10.50  
  
1001

**Strings** are text, written within double or single quotes or backticks:

"John Doe"  
  
'John Doe'

`John Doe`

What is the difference? Well, the 1st and 2nd are the same, they are string literals.

The 3rd one is a Template literal

It allows you to put in multi-line text, it allows you to embed expressions in string. But these are not important now

**JavaScript Variables**

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

JavaScript uses the **var, let or const**keyword to **declare** variables.

An **equal sign** is used to **assign values** to variables.

In this example, x is defined as a variable. Then, x is assigned (given) the value 6:

var x;  
  
x = 6;

Difference between var and let is with the scope of the variable. It differes where you can use a variable defined with each of them.

A var variable is hoisted which means, its available globally. It means you can do something like this and it will still work.   
  
x = 6;  
var x;

You can’t do this with let. ( Its not straight forward to demonstrate this in browser console but not important now)

A const is just like let, but a variable defined with const can’t change its value ever.

var works for now

### JavaScript Operators

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

(5 + 6) \* 10

JavaScript uses an **assignment operator** ( = ) to **assign** values to variables:

var x, y;  
x = 5;  
y = 6;

### JavaScript Expressions

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:

5 \* 10

Expressions can also contain variable values:

x \* 10

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

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

"John" + " " + "Doe"

### JavaScript 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;

### JavaScript 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.  
This way JavaScript can easily distinguish identifiers from numbers.

**JavaScript is Case Sensitive**

All JavaScript identifiers are **case sensitive**.

The variables **lastName** and **lastname**, are two different variables.

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

JavaScript does not interpret **VAR** or **Var** as the keyword **var**.

**JavaScript and 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.

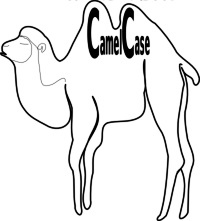
Hyphens are not allowed in JavaScript. They are reserved for subtractions.

**Underscore:**

first\_name, last\_name, master\_card, inter\_city.

**Upper Camel Case (Pascal Case):**

FirstName, LastName, MasterCard, InterCity.



**Lower Camel Case:**

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

firstName, lastName, masterCard, interCity.

### Datatypes in JS

The Concept of Data Types

In programming, data types is an important concept.

To be able to operate on variables, it is important to know something about the type.

Without data types, a computer cannot safely solve this:

var x = 16 + "Volvo";

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result?

JavaScript will treat the example above as:

var x = "16" + "Volvo";

When adding a number and a string, JavaScript will treat the number as a string.

Example

var x = 16 + "Volvo";

var y = "Volvo" + 16;

JavaScript evaluates expressions from left to right. Different sequences can produce different results:

var x = 16 + 4 + "Volvo";

Result:

20Volvo

var x = "Volvo" + 16 + 4;

Result:

Volvo164

In the first example, JavaScript treats 16 and 4 as numbers, until it reaches "Volvo".

In the second example, since the first operand is a string, all operands are treated as strings.

**JavaScript Types are Dynamic.**

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Example

var x;           // Now x is undefined  
x = 5;           // Now x is a Number  
x = "John";      // Now x is a String

**JavaScript Strings**

A string (or a text string) is a series of characters like "John Doe".

Strings are written with quotes. You can use single or double quotes:

Example

var carName = "Volvo XC60";   // Using double quotes  
var carName = 'Volvo XC60';   // Using single quotes

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

Example

var answer = "It's alright";             // Single quote inside double quotes  
var answer = "He is called 'Johnny'";    // Single quotes inside double quotes  
var answer = 'He is called "Johnny"';    // Double quotes inside single quotes

You will learn more about strings later in this tutorial.

**JavaScript Numbers**

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

var x1 = 34.00;     // Written with decimals  
var x2 = 34;        // Written without decimals

Extra large or extra small numbers can be written with scientific (exponential) notation:

Example

var y = 123e5;      // 12300000  
var z = 123e-5;     // 0.00123

You will learn more about numbers later in this tutorial.

**JavaScript Booleans**

Booleans can only have two values: true or false.

Example

var x = 5;  
var y = 5;  
var z = 6;  
(x == y)       // Returns true  
(x == z)       // Returns false

Booleans are often used in conditional testing.

You will learn more about conditional testing later in this tutorial.

**JavaScript Arrays**

JavaScript arrays are written with square brackets.

Array items are separated by commas.

The following code declares (creates) an array called cars, containing three items (car names):

Example

var cars = ["Saab", "Volvo", "BMW"];

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about arrays later in this tutorial.

**JavaScript Objects**

JavaScript objects are written with curly braces.

Object properties are written as name:value pairs, separated by commas.

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

You will learn more about objects later in this tutorial.

**The typeof Operator**

You can use the JavaScript **typeof** operator to find the type of a JavaScript variable.

The **typeof** operator returns the type of a variable or an expression:

Example

typeof ""                  // Returns "string"  
typeof "John"              // Returns "string"  
typeof "John Doe"          // Returns "string"

Example

typeof 0                   // Returns "number"  
typeof 314                 // Returns "number"  
typeof 3.14                // Returns "number"  
typeof (3)                 // Returns "number"  
typeof (3 + 4)             // Returns "number"

**Undefined**

In JavaScript, a variable without a value, has the value**undefined**. The typeof is also **undefined**.

Example

var car;                // Value is undefined, type is undefined

Any variable can be emptied, by setting the value to **undefined**. The type will also be **undefined**.

Example

car = undefined;        // Value is undefined, type is undefined

**Empty Values**

An empty value has nothing to do with undefined.

An empty string has both a legal value and a type.

Example

var car = "";              // The value is "", the typeof is "string"

**Null**

In JavaScript null is "nothing". It is supposed to be something that doesn't exist.

Unfortunately, in JavaScript, the data type of null is an object.

You can consider it a bug in JavaScript that typeof null is an object. It should be null.

You can empty an object by setting it to null:

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
person = null;        // Now value is null, but type is still an object

You can also empty an object by setting it to undefined:

Example

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};  
person = undefined;   // Now both value and type is undefined

**Difference Between Undefined and Null**

Undefined and null are equal in value but different in type:

typeof undefined           // undefined  
typeof null                // object  
  
null === undefined         // false  
null == undefined          // true

**Primitive Data**

A primitive data value is a single simple data value with no additional properties and methods.

**According to the latest ECMAScript release, these are the data types:**

* Boolean. true or false
* Null. null
* Undefined. undefined
* Number. 0,55,3.56, -99 etc
* String. "Hello";
* Symbol.\*new – not important now
* Object. {}, Array() etc

Everything except Object are called primitives.

***Javascript is well known for being , lets say, ”syntactically weird”. So be ready for a few waaaat moments.***

Example

typeof "John"              // Returns "string"   
typeof 3.14                // Returns "number"  
typeof true                // Returns "boolean"  
typeof false               // Returns "boolean"  
typeof x                   // Returns "undefined" (if x has no value)

**Complex Data**

The **typeof** operator can return one of two complex types:

* function
* object

The typeof operator returns object for both objects, arrays, and null.

The typeof operator does not return object for functions.

Example

typeof {name:'John', age:34} // Returns "object"  
typeof [1,2,3,4]             // Returns "object" (not "array", see note below)  
typeof null                  // Returns "object"  
typeof function myFunc(){}   // Returns "function"

The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

So JS has objects… but its not quiet like the objects in Java or C++. JS has prototype based Objects whereas Java has class based objects. The difference?

Class-based object-oriented languages, such as Java and C++, are founded on the concept of two distinct entities: classes and instances.

* A *class* defines all of the properties (considering methods and fields in Java, or members in C++, to be properties) that characterize a certain set of objects. A class is an abstract thing, rather than any particular member of the set of objects it describes. For example, the Employee class could represent the set of all employees.
* An *instance*, on the other hand, is the instantiation of a class; that is, one of its members. For example, Victoria could be an instance of the Employee class, representing a particular individual as an employee. An instance has exactly the same properties of its parent class (no more, no less).

A prototype-based language, such as JavaScript, does not make this distinction: it simply has objects. A prototype-based language has the notion of a *prototypical object*, an object used as a template from which to get the initial properties for a new object. Any object can specify its own properties, either when you create it or at run time. In addition, any object can be associated as the *prototype* for another object, allowing the second object to share the first object's properties.

# JSON

JSON: **J**ava**S**cript **O**bject **N**otation.

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

**Exchanging Data**

When exchanging data between a browser and a server, the data can only be text.

JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.

We can also convert any JSON received from the server into JavaScript objects.

This way we can work with the data as JavaScript objects, with no complicated parsing and translations.

**Sending Data**

If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server:

Example

var myObj = { "name":"John", "age":31, "city":"New York" };  
var myJSON = JSON.stringify(myObj);  
window.location = "demo\_json.php?x=" + myJSON;

You will learn more about the JSON.stringify() function later in this tutorial.

**Receiving Data**

If you receive data in JSON format, you can convert it into a JavaScript object:

Example

var myJSON = '{ "name":"John", "age":31, "city":"New York" }';  
var myObj = JSON.parse(myJSON);  
document.getElementById("demo").innerHTML = myObj.name;

You will learn more about the JSON.parse() function later in this tutorial.

**Storing Data**

When storing data, the data has to be a certain format, and regardless of where you choose to store it, text is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

Example

Storing data in local storage

//Storing data:  
myObj = { "name":"John", "age":31, "city":"New York" };  
myJSON = JSON.stringify(myObj);  
localStorage.setItem("testJSON", myJSON);  
  
//Retrieving data:  
text = localStorage.getItem("testJSON");  
obj = JSON.parse(text);  
document.getElementById("demo").innerHTML = obj.name;

**What is JSON?**

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data-interchange format
* JSON is "self-describing" and easy to understand
* JSON is language independent **\***

\*  
JSON uses JavaScript syntax, but the JSON format is text only.  
Text can be read and used as a data format by any programming language.

The JSON format was originally specified by [Douglas Crockford](http://www.crockford.com/).

**Why use JSON?**

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.