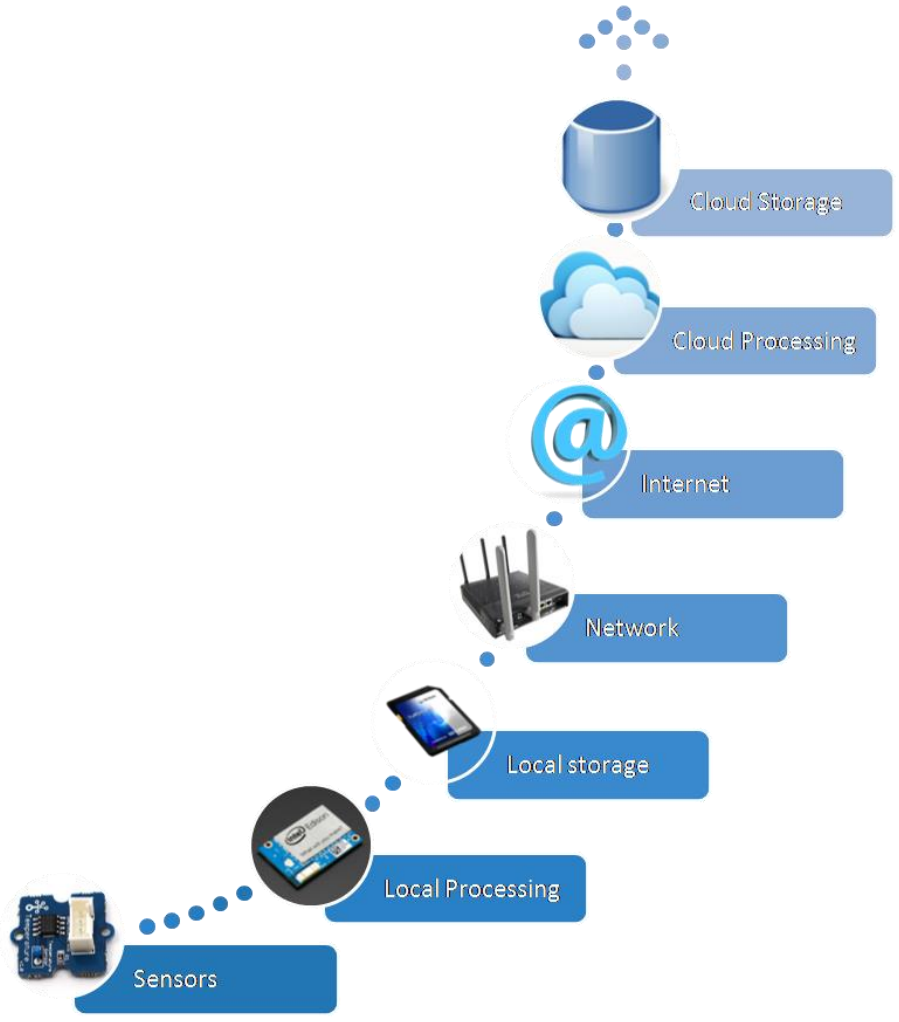
Zagazig University

Faculty of Engineering

Electronics and Communications Engineering

LAB #3
IOT Basics Part#1

(Using HTML, JavaScript, Node.js and Nodemcu)



OBJECTIVE:

- To introduce students with IOT technologies.
- To get familiar with the basics of frontend and Backend web technologies.
- To design simple web UI and a web server using web sockets.

EQUIPMENTS:

- Computer with Node.js installed
- A router or an access point to connect more than one device locally

INTRODUCTION:

The Internet of Things has become a very widely spread concept in the last few years. The reason for this is mainly the need to computerize and control most of the surrounding objects and have access to data in real time. Think about parking sensors, about phones which can check the weather and so on.

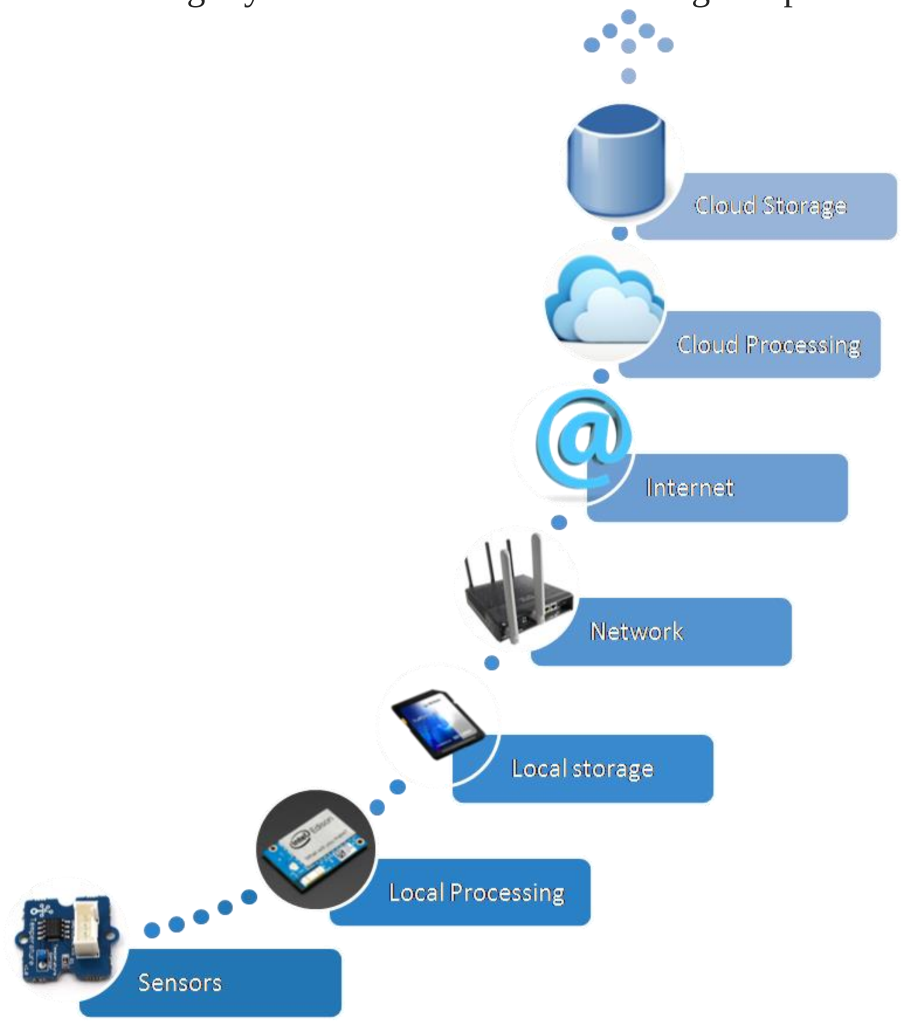
The Internet of Things represents the whole way from collecting data, processing it, taking an action corresponding to the signification of this data to storing everything in the cloud. All this is made possible by the internet.

The IoT is involved in medicine, agriculture or transportation due to sensors and cloud storage.

BASIC CONCEPTS:

IoT Stack

A basic Internet of Things system consists of the following components:



Sensors

They transform analog data given from scanning the environment to digital data, but they merely do any processing. On the bright side, they don't consume much power and can live on batteries for a long time. Sensors are present in everyday life more than you would expect. They improve industry, agriculture, homes, transportation or smart phones for example. They are tools which help monitoring the environment, collecting data about it and, with the help of computers, acting accordingly.

Local processing and storage devices

Local processing devices are the second level and third in IoT. At this point, data is locally stored and processed, ideally not sent forwards unless relevant. This part is explained in detail in the hardware section, as said devices are nothing more than microcontrollers and embeded boards, which handle the data they receive from the sensors.

Network and Internet

There is hardware which connects to the previously described devices, pulls out data and sends it to the cloud to be stored. There are 4 protocols used at this level: CoAP, MQTT (less secure and designed for machine to machine communication), HTTP (web protocol) and XMPP which functions as a chat.

Cloud

In the cloud, which comes next, data is collected and the main goal is for it to reach the point of making predictions based on the stored information. The cloud however, even though it represents one of the most useful features of the internet, is not used properly. Data sent to the cloud didn't reach the level of being formerly processed. Which means there is no preselected data. The cloud is constantly loaded with irrelevant information and thus losing it's property of being practical.

Web Design:
Front End Vs. Back End

The Front End is the stuff that you see and interact with: HTML, CSS , and JS
The Back End is everything else: so many choices!
Restaurant Analogy: The backend is everything that happens in the kitchen; the front end is what is plated and sent to your table

Front End [HTML, ~~CSS~~ , and JS]:
HTML Basics:

What is HTML?

HTML is the standard markup language for creating Web pages.

- HTML stands for Hyper Text Markup Language
- HTML describes the structure of Web pages using markup
- HTML elements are the building blocks of HTML pages
- HTML elements are represented by tags
- HTML tags label pieces of content such as "heading", "paragraph", "table", and so on
- Browsers do not display the HTML tags, but use them to render the content of the page

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>
```

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
- 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

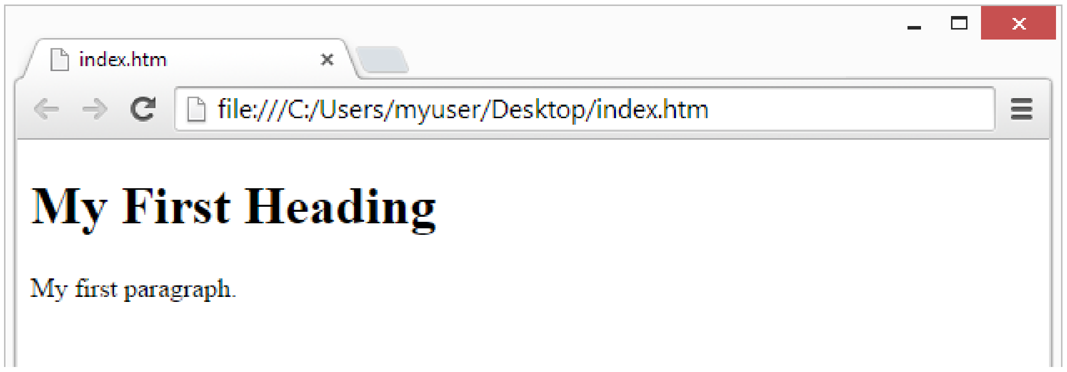
Tip: The start tag is also called the **opening tag**, and the end tag the **closing tag**.

HTML tags are not case sensitive: <P> means the same as <p>.The HTML5 standard does not require lowercase tags, but W3C **recommends** lowercase in HTML

Web Browsers

The purpose of a web browser (Chrome, IE, Firefox, Safari) is to read HTML documents and display them.

The browser does not display the HTML tags, but uses them to determine how to display the document:

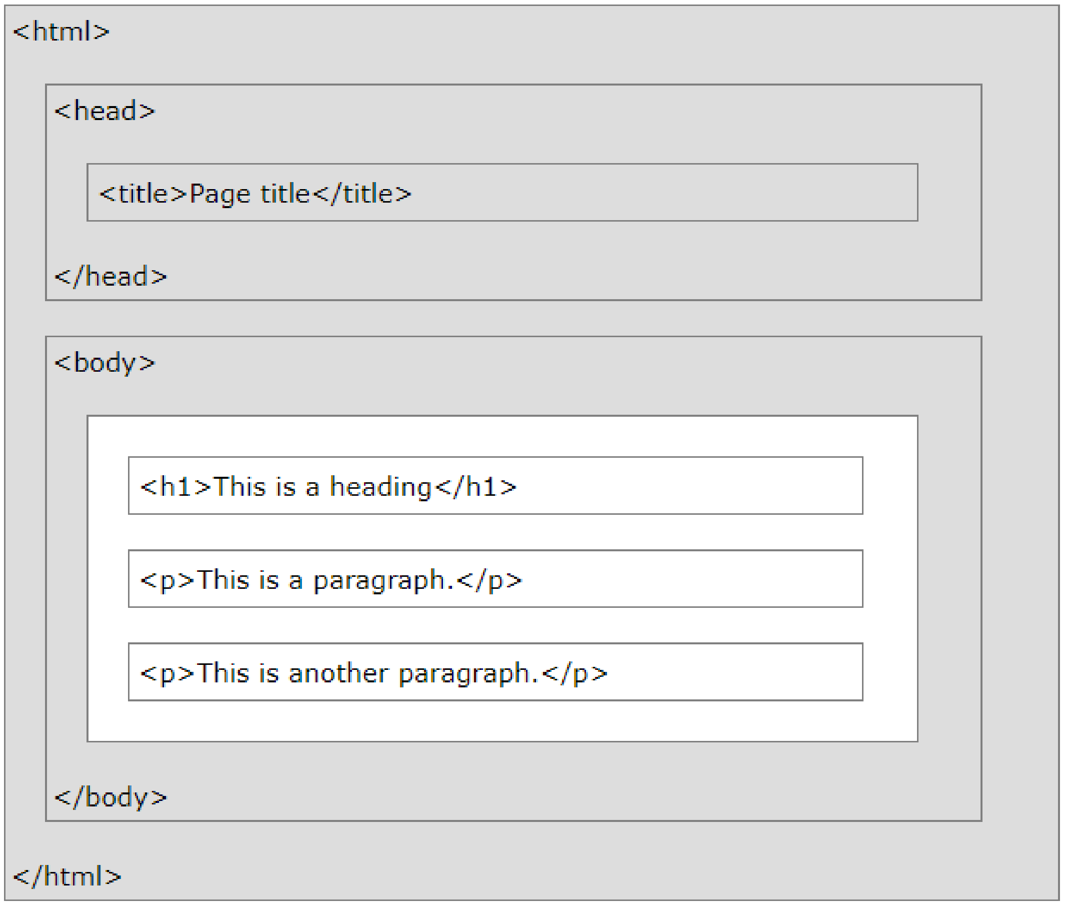


My First Heading

My first paragraph.

HTML Page Structure

Below is a visualization of an HTML page structure:



```
<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
  </body>
</html>
```

Note: Only the content inside the <body> section (the white area above) is displayed in a browser.

The <!DOCTYPE> Declaration

The `<!DOCTYPE>` declaration represents the document type, and helps browsers to display web pages correctly.
It must only appear once, at the top of the page (before any HTML tags).
The `<!DOCTYPE>` declaration is not case sensitive.
The `<!DOCTYPE>` declaration for HTML5 is:

```
<!DOCTYPE html>
```

HTML Documents

All HTML documents must start with a document type declaration: `<!DOCTYPE html>`.
The HTML document itself begins with `<html>` and ends with `</html>`.
The visible part of the HTML document is between `<body>` and `</body>`.

HTML Elements

An HTML element usually consists of a **start** tag and **end** tag, with the content inserted in between:
<tagname>Content goes here...</tagname>
The HTML **element** is everything from the start tag to the end tag:
<p>My first paragraph.</p>

HTML Attributes

Attributes provide additional information about HTML elements.

- All HTML elements can have **attributes**
- Attributes provide **additional information** about an element
- Attributes are always specified in **the start tag**
- Attributes usually come in name/value pairs like: **name="value"**

Ex:

HTML images are defined with the `<img>` tag.
The filename of the image source is specified in the `src` attribute:

```
<img src="img_girl.jpg">
```

The id Attribute

The `id` attribute specifies a unique id for an HTML element (the value must be unique within the HTML document).The `id` value can be used by CSS and JavaScript to perform certain tasks for a unique element with the specified id value

The Style Attribute

Setting the style of an HTML element, can be done with the `style` attribute.

The HTML `style` attribute has the following **syntax**:

```
<tagname style="property:value;">
```

The ***property*** is a CSS property. The ***value*** is a CSS value.

Example

```
<h1 style="color:blue;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
```

HTML Comment Tags

Comment tags are used to insert comments in the HTML source code.

You can add comments to your HTML source by using the following syntax:

```
<!-- Write your comments here -->
```

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.

`<h1>` defines the most important heading. `<h6>` defines the least important heading:

Example

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

Example

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

HTML Links

HTML links are defined with the `<a>` tag:

Example

```
<a href="https://www.w3schools.com">This is a link</a>
```

The link's destination is specified in the `href` attribute.

Attributes are used to provide additional information about HTML elements.

HTML Images

HTML images are defined with the `<img>` tag.

The source file (`src`), alternative text (`alt`), `width`, and `height` are provided as attributes:

Example

```
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

HTML Buttons

HTML buttons are defined with the `<button>` tag:

Example

```
<button>Click me</button>
```

Click me to see a basic elements Example:

HTML with JavaScript

JavaScript makes HTML pages more dynamic and interactive.

Example

The HTML <script> Tag

The `<script>` tag is used to define a client-side script (JavaScript).

The `<script>` element either contains scripting statements, or it points to an external script file through the `src` attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

To select an HTML element, JavaScript very often uses the `document.getElementById()` method.

This JavaScript example writes "Hello JavaScript!" into an HTML element with id="demo":

```
<script>
document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
```

Blinking Button Background Color Example:
Extra Example:

JavaScript Basics:

JavaScript is the programming language of HTML and the Web.

JavaScript is easy to learn.

JavaScript Variables

JavaScript variables are containers for storing data values.

Example

```
var x = 5;
var y = 6;
var z = x + y;
```

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 _ (but we will not use it in this tutorial)
- Names are case sensitive (y and Y are different variables)
- Reserved words (like JavaScript keywords) cannot be used as names
- JavaScript identifiers are case-sensitive.

The 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
```

Declaring (Creating) JavaScript Variables

Creating a variable in JavaScript is called "declaring" a variable.

You declare a JavaScript variable with the `var` keyword:

```
var carName;
```

After the declaration, the variable has no value (technically it has the value of `undefined`).

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

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

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

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

In the example below, we create a variable called `carName` and assign the value "Volvo" to it.

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

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

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 primitive Data Types

A **primitive value** is a value that has no properties or methods.

A **primitive data type** is data that has a primitive value.

JavaScript defines 5 types of primitive data types:

- `string`
- `number`
- `boolean`
- `null`
- `` `undefined ``

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

if x = 3.14, you can change the value of x. But you cannot change the value of 3.14.

Value	Type	Comment
"Hello"	String	"Hello" is always "Hello"
3.14	Number	3.14 is always 3.14
true	Boolean	true is always true
false	Boolean	false is always false
null	null (object)	null is always null
undefined	Undefined	undefined is always undefined

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.

JavaScript Function

avaScript functions are **defined** with the `function` keyword.

You can use a function **declaration** or a function **expression**.

Function Declarations

Earlier in this tutorial, you learned that functions are **declared** with the following syntax:

```
function functionName(parameters) {
  // code to be executed
}
```

Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).

Example

```
function myFunction(a, b) {
  return a * b;
}
```

Function Expressions

A JavaScript function can also be defined using an **expression**.

A function expression can be stored in a variable:

Example

```
var x = function (a, b) {return a * b};
```

anonymous function (a function without a name)

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

The function above ends with a semicolon because it is a part of an executable statement.

JavaScript Objects:

in JavaScript, objects are king. If you understand objects, you understand JavaScript. All JavaScript values, except primitives, are objects.

Real Life Objects, Properties, and Methods

In real life, a car is an **object**.

A car has **properties** like weight and color, and **methods** like start and stop:

Object	Properties	Methods

	car.name = Fiat
	car.model = 500
	car.weight = 850kg
	car.color = white
		car.start()
		car.drive()
		car.brake()
		car.stop()

All cars have the same **properties**, but the property **values** differ from car to car.

All cars have the same **methods**, but the methods are performed **at different times**.

You have already learned that JavaScript variables are containers for data values.

This code assigns a **simple value** (Fiat) to a **variable** named car:

```
var car = "Fiat";
```

Objects are variables too. But objects can contain many values.

This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

```
var car = {type:"Fiat", model:"500", color:"white"};
```

he values are written as **name:value** pairs (name and value separated by a colon).

JavaScript objects are containers for **named values** called properties or methods.

Object Definition

You define (and create) a JavaScript object with an object literal:

Example

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

Object Properties

The **name : values** pairs in JavaScript objects are called **properties**:

Property	Property Value
firstName	John
lastName	Doe
Age	50
eyeColor	blue

Accessing Object Properties

You can access object properties in two ways:

```
objectName.propertyName // person.lastName;
```
or
```
objectName["propertyName"] // person["lastName"];
```

Object Methods

Objects can also have **methods**.

Methods are **actions** that can be performed on objects.

Methods are stored in properties as **function definitions**.

Example

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

The this Keyword

In a function definition, `this` refers to the "owner" of the function.

In the example above, `this` is the **person object** that "owns"
the `fullName` function.

In other words, `this.firstName` means the `firstName` property of **this object**.

Accessing Object Methods

You access an object method with the following syntax:
objectName.methodName()

Example

```
name = person.fullName();
```

If you access a method **without** the () parentheses, it will return the **function definition**:

```
name = person.fullName;
```

JavaScript Object Constructors

The examples from the previous chapters are limited. They only create single objects.

Sometimes we need a "**blueprint**" for creating many objects of the same "type".

The way to create an "object type", is to use an **object constructor function**.

In the example above, `function Person()` is an object constructor function.
It is considered good practice to name constructor functions with an upper-case first letter.

Example

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.name = function() {return this.firstName + " " + this.lastName;};
}
```

Objects of the same type are created by calling the constructor function with the `new` keyword:

```
var myFather = new Person("John", "Doe", 50, "blue");
var myMother = new Person("Sally", "Rally", 48, "green");
```

JavaScript HTML DOM:

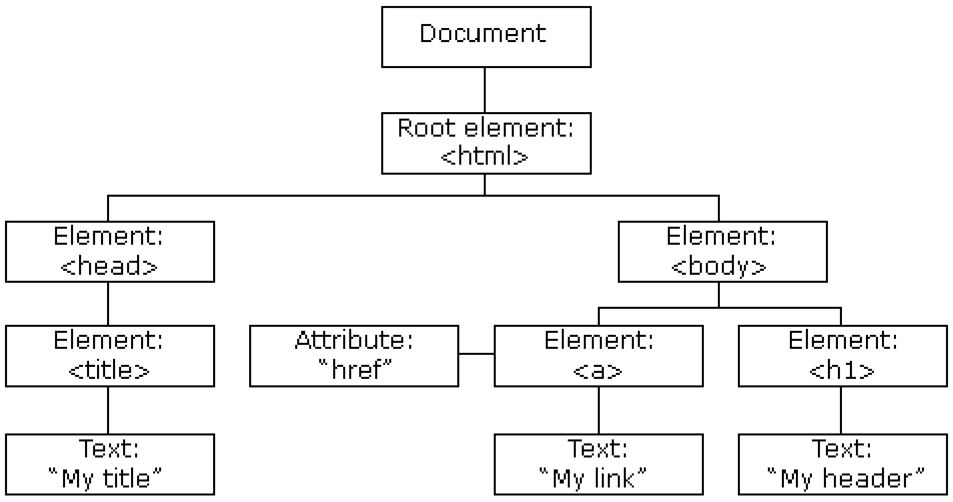
With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

The HTML DOM (Document Object Model)

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



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
- 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

HTML DOM Methods:

Finding HTML Elements

Method	Description
document.getElementById(id)	Find an element by element id
document.getElementsByTagName(name)	Find elements by tag name
document.getElementsByClassName(name)	Find elements by class name

Changing HTML Elements

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.style.property = new style	Change the style of an HTML element
Method	Description
element.setAttribute(attribute, value)	Change the attribute value of an HTML element

Adding and Deleting Elements

Method	Description
document.createElement(element)	Create an HTML element
document.removeChild(element)	Remove an HTML element
document.appendChild(element)	Add an HTML element
document.replaceChild(new, old)	Replace an HTML element
document.write(text)	Write into the HTML output stream

Adding Events Handlers

Method	Description
document.getElementById(id).onclick = function(){code}	Adding event handler code to an onclick event

Back End [Node.js]:

Node.js Introduction

What is Node.js?

- Node.js is an open source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Why Node.js?

Node.js uses asynchronous programming!

A common task for a web server can be to open a file on the server and return the content to the client.

Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.

Node.js runs single-threaded, non-blocking, asynchronously programming, which is very memory efficient.

What Can Node.js Do?

- Node.js can generate dynamic page content
- Node.js can create, open, read, write, delete, and close files on the server
- Node.js can collect form data
- Node.js can add, delete, modify data in your database

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events
- A typical event is someone trying to access a port on the server
- Node.js files must be initiated on the server before having any effect
- Node.js files have extension ".js"

Node.js NPM

What is NPM?

NPM is a package manager for Node.js packages, or modules if you like.

www.npmjs.com hosts thousands of free packages to download and use.

The NPM program is installed on your computer when you install Node.js

What is a Package?

A package in Node.js contains all the files you need for a module.

Modules are JavaScript libraries you can include in your proje

Download a Package

Downloading a package is very easy.

Open the command line interface and tell NPM to download the package you want.

I want to download a package called "upper-case":

Download "upper-case":

```
C:\Users\Your Name>npm install upper-case
```

Now you have downloaded and installed your first package!

NPM creates a folder named "node_modules", where the package will be placed. All packages you install in the future will be placed in this folder.

My project now has a folder structure like this:

```
C:\Users\My Name\node_modules\upper-case
```

Using a Package

Once the package is installed, it is ready to use.

Include the "upper-case" package the same way you include any other module:

```
var uc = require('upper-case');
```

node.js Hello World:

hello world example

sending page example

node.js with WebSockets:

WebSockets are an alternative to HTTP communication in Web Applications.

They offer a long lived, bidirectional communication channel between client and server.

Once established, the channel is kept open, offering a very fast connection with low latency and overhead.

How WebSockets differ from HTTP

HTTP is a very different protocol, and also a different way of communicate.

HTTP is a request/response protocol: the server returns some data when the client requests it.

With WebSockets:

the server can send a message to the client without the client explicitly requesting something

the client and the server can talk to each other simultaneously

very little data overhead needs to be exchanged to send messages. This means a low latency communication.

WebSockets are great for real-time and long-lived communications.

Secured WebSockets

Always use the secure, encrypted protocol for WebSockets, wss://.

ws:// refers to the unsafe WebSockets version (the http:// of WebSockets), and should be avoided for obvious reasons.

Hello World WebSocket:

There is many WebSocket Libraries available we will us **ws** library

ws is a WebSocket client and server implementation, fast, and easy to use Module.

Server Side:

Easily install it using

NPM install ws

Initialization :

```
const SocketServer = require('ws').Server;
const wss = new SocketServer({ server }); // ex : server = port:8000
```

handling connection events:

```
wss.on(‘connection’, function(ws){
        //do something here
        ws.on(‘message’,function(message){ //do something here   });
        ws.on(‘close’,function(client){   //do something here           });
        ws.on(‘error’,function(client){    //do something here          });
});
```

Client Side:

websocket client is a browser supported object.

There are 3 basic must know fucntions:

- ws.onopen : emmited when connected
- ws.send : sending a send event to websocket server
- ws.onmessage : event emmited when receiving message

The code (Hello World)

TASK#1

Each student should to design the following:

- Simple web page contains a button that represent the state of a LED (ON/OFF). The button color would reflect the led state (ON==Green & OFF==Red)
- Simple server using Node.js that can respond to both http and Web-Socket requests from many clients at the same time.
- If a client connects to the server, it should send it back the web page designed in step#1 and opens a web socket between that client and the server
- If a client clicked the Led button, it should toggle its state (i.e. if it was Green toggle it to be Red and vice versa).
- Also the led state should be the same for all connected clients (i.e. if one client toggles the led state it, this change should appear at all connected clients synchronously)