# HTTP protocol. Structure of requests and responses, request methods, server response codes, request and response headers.

**How does htttp works?**

As a request-response protocol, HTTP gives users a way to interact with web resources such as HTML files by transmitting hypertext messages between clients and servers. HTTP clients generally use Transmission Control Protocol (TCP) connections to communicate with servers.

HTTP utilizes [specific request methods](https://tools.ietf.org/html/rfc2616#section-5.1.1) in order to perform various tasks. All HTTP servers use the GET and HEAD methods, but not all support the rest of these request methods:

* GET requests a specific resource in its entirety
* HEAD requests a specific resource without the body content
* POST adds content, messages, or data to a new page under an existing web resource
* PUT directly modifies an existing web resource or creates a new URI if need be
* DELETE gets rid of a specified resource
* TRACE shows users any changes or additions made to a web resource
* OPTIONS shows users which HTTP methods are available for a specific URL
* CONNECT converts the request connection to a transparent TCP/IP tunnel
* PATCH partially modifies a web resource

**Request and response:**

HTTP messages are how data is exchanged between a server and a client. There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server.

**Structure**:

HTTP requests, and responses, share similar structure and are composed of:

1. A *start-line* describing the requests to be implemented, or its status of whether successful or a failure. This start-line is always a single line.
2. An optional set of *HTTP headers* specifying the request, or describing the body included in the message.
3. A blank line indicating all meta-information for the request has been sent.
4. An optional *body* containing data associated with the request (like content of an HTML form), or the document associated with a response. The presence of the body and its size is specified by the start-line and HTTP headers.

**Http request:**

HTTP requests are messages sent by the client to initiate an action on the server. Their *start-line* contain three elements:

1. An [*HTTP method*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods), a verb (like [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET), [PUT](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT) or [POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST)) or a noun (like [HEAD](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD) or [OPTIONS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)), that describes the action to be performed. For example, GET indicates that a resource should be fetched or POST means that data is pushed to the server (creating or modifying a resource, or generating a temporary document to send back).
2. The *request target*, usually a [URL](https://developer.mozilla.org/en-US/docs/Glossary/URL), or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods. It can be

An absolute path, ultimately followed by a '?' and query string.

POST / HTTP/1.1 GET /background.png HTTP/1.0 HEAD /test.html?query=alibaba HTTP/1.1 OPTIONS /anypage.html HTTP/1.0

* + A complete URL, known as the *absolute form*, is mostly used with GET when connected to a proxy. GET [https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages%20HTTP/1.1)
  + The authority component of a URL, consisting of the domain name and optionally the port (prefixed by a ':'), is called the *authority form*. It is only used with CONNECT when setting up an HTTP tunnel. CONNECT developer.mozilla.org:80 HTTP/1.1
  + The *asterisk form*, a simple asterisk ('\*') is used with OPTIONS, representing the server as a whole. OPTIONS \* HTTP/1.1

3, The *HTTP version*, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

There is also header and body of http request.

[Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#headers)

[HTTP headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consist of one single line, which can be quite long.

Many different headers can appear in requests. They can be divided in several groups:

* [General headers](https://developer.mozilla.org/en-US/docs/Glossary/General_header), like [Via](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Via), apply to the message as a whole.
* [Request headers](https://developer.mozilla.org/en-US/docs/Glossary/Request_header), like [User-Agent](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent) or [Accept](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept), modify the request by specifying it further (like [Accept-Language](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Language)), by giving context (like [Referer](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referer)), or by conditionally restricting it (like [If-None](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/If-None)).
* [Representation headers](https://developer.mozilla.org/en-US/docs/Glossary/Representation_header) like [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) that describe the original format of the message data and any encoding applied (only present if the message has a body).

[Body](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#body)

The final part of the request is its body. Not all requests have one: requests fetching resources, like GET, HEAD, DELETE, or OPTIONS, usually don't need one. Some requests send data to the server in order to update it: as often the case with POST requests (containing HTML form data).

Bodies can be broadly divided into two categories:

* Single-resource bodies, consisting of one single file, defined by the two headers: [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) and [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length).
* [Multiple-resource bodies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#multipartform-data), consisting of a multipart body, each containing a different bit of information. This is typically associated with [HTML Forms](https://developer.mozilla.org/en-US/docs/Learn/Forms).

[**HTTP Responses**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#http_responses)

[Status line](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#status_line)

The start line of an HTTP response, called the *status line*, contains the following information:

1. The *protocol version*, usually HTTP/1.1.
2. A *status code*, indicating success or failure of the request. Common status codes are [200](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200), [404](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/404), or [302](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/302)
3. A *status text*. A brief, purely informational, textual description of the status code to help a human understand the HTTP message.

A typical status line looks like: HTTP/1.1 404 Not Found.

And same there is also header and body of http responses

[Headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#headers_2)

[HTTP headers](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) for responses follow the same structure as any other header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including its value, presents as a single line.

Many different headers can appear in responses. These can be divided into several groups:

* [General headers](https://developer.mozilla.org/en-US/docs/Glossary/General_header), like [Via](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Via), apply to the whole message.
* [Response headers](https://developer.mozilla.org/en-US/docs/Glossary/Response_header), like [Vary](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Vary) and [Accept-Ranges](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept-Ranges), give additional information about the server which doesn't fit in the status line.
* [Representation headers](https://developer.mozilla.org/en-US/docs/Glossary/Representation_header) like [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) that describe the original format of the message data and any encoding applied (only present if the message has a body).

[Body](https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages#body_2)

The last part of a response is the body. Not all responses have one: responses with a status code that sufficiently answers the request without the need for corresponding payload (like [201](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/201) **Created** or [204](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/204) **No Content**) usually don't.

Bodies can be broadly divided into three categories:

* Single-resource bodies, consisting of a single file of known length, defined by the two headers: [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) and [Content-Length](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Length).
* Single-resource bodies, consisting of a single file of unknown length, encoded by chunks with [Transfer-Encoding](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Transfer-Encoding) set to chunked.
* [Multiple-resource bodies](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types#multipartform-data), consisting of a multipart body, each containing a different section of information. These are relatively rare.

**Server response codes:**

1. [Informational responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#information_responses) (100–199)
2. [Successful responses](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#successful_responses) (200–299)
3. [Redirects](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#redirection_messages) (300–399)
4. [Client errors](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#client_error_responses) (400–499)
5. [Server errors](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status#server_error_responses) (500–599)

# 2: HTML markup language. Features, main tags and tag attributes.

Html is a hyper text markup language which is used to create webpages:

**Features of HTML:**

* It is easy to learn and easy to use.
* It is platform-independent.
* Images, videos, and audio can be added to a web page.
* Hypertext can be added to text.
* It is a markup language.

**Main tags:**

There are four required tags in HTML. These are **html, title, head and body**.

**Html attributes:**

HTML attributes are special words used inside the opening tag to control the element's behavior.

<a href =””…..

# 3.The structure of the HTML page. Document Object Model (DOM).

An HTML 4 document is composed of three parts:

1. a line containing [HTML version information](https://www.w3.org/TR/html401/struct/global.html#version-info),
2. a declarative header section (delimited by the [HEAD](https://www.w3.org/TR/html401/struct/global.html#edef-HEAD) element),
3. a body, which contains the document's actual content. The body may be implemented by the [BODY](https://www.w3.org/TR/html401/struct/global.html#edef-BODY) element or the [FRAMESET](https://www.w3.org/TR/html401/present/frames.html#edef-FRAMESET) element.

**What is Document object model ?**

Изображение выглядит как текст

Автоматически созданное описание

# 4: HTML forms. Setting the HTTP request method. Rules for placing forms on pages, types of input fields.?

The <**form**> **HTML** element represents a document section containing interactive controls for submitting information.

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

# 5: Cascading Style Sheets (CSS). Structure - rules, selectors. Types of selectors, features of their use. Priorities of rules. Advantages of CSS over directly styling through tag attributes?

Cascading Style Sheets (**CSS**) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.

Rules and structure:

A ***Cascading Style Sheet (CSS) rule*** is a statement that defines the style of one or more elements in your web page. These rules follow a specific ***structure***.

The format or syntax for CSS rules consists of a *selector* and a *declaration*. A *declaration block* consists of several declarations for s given selector. Multiple selectors can be combined in a rule.

**Rule format or syntax**

A rule consists of a *selector* and a *declaration* that is surrounded by curly parentheses:

**selector {declaration}**

***Selector***

A CSS rule defines styles to elements of all web pages using the style-sheet, including the **<body>** of your pages and HTML tags such as **<h1>**, **<p>**, and **<ul>.** When stated in a CSS rule, these elements are called *selectors*.

**h1 {declaration}**

***Declaration***

The declaration contains the *property* of the selector and the *value* of the property. The property always is followed by a colon ( **:** ) and each value ends in a semicolon ( **;** ).

**selector {property: value;}**

For example:

**h1 {font-weight: bold;}**

where

* **h1** is the selector
* **font-weight** is the property of the selector for this rule
* **bold** is the value of the property
* **{font-weight: bold;**} is the declaration

**Types of Selector:**

**Element selector:** it select based on selector name:

p {  
  text-align: center;  
  color: red;  
}

**Id selector:** the id of an element is unique within a page, so the id selector is used to select one unique element.

#para1 {  
  text-align: center;  
  color: red;  
}

**Class selector:** The class selector selects HTML elements with a specific class attribute.

.center {  
  text-align: center;  
  color: red;  
}

**Universal Selector:** The universal selector (\*) selects all HTML elements on the page.

\* {  
  text-align: center;  
  color: blue;  
}

**Grouping Selector:** The grouping selector selects all the HTML elements with the same style definitions.

h1, h2, p {  
  text-align: center;  
  color: red;  
}

**priorities of rules of selector in css**

|  |  |
| --- | --- |
| ID | Higher specificity than class selector. |

**Web page style tag or css:**

**The layout of a web page is better controlled**. ... **Style (CSS) kept separate from structure (HTML)**, means smaller file size. Reduced file size means reduced bandwidth, which means faster loading time.

# LESS, Sass, SCSS. Key features, comparative characteristics. Browser compatibility, translation to "normal" CSS.

This stylesheet language is strongly oriented around CSS and resembles SCSS in its syntax. Both preprocessors share some of the same properties: Both SASS and LESS allow for the use of mixins and variables. One difference, though, is that SASS is based on Ruby, while LESS uses JavaScript.

# Client scripts. Features, scope. JavaScript language.

A client script is **JavaScript code which runs on the client**, rather than the server. Well-designed client scripts can reduce the amount of time it takes to complete a form and improve the user experience. However, improperly implemented client scripts can significantly slow down form load times. Follow these best practices to ensure Client Scripts work efficiently.

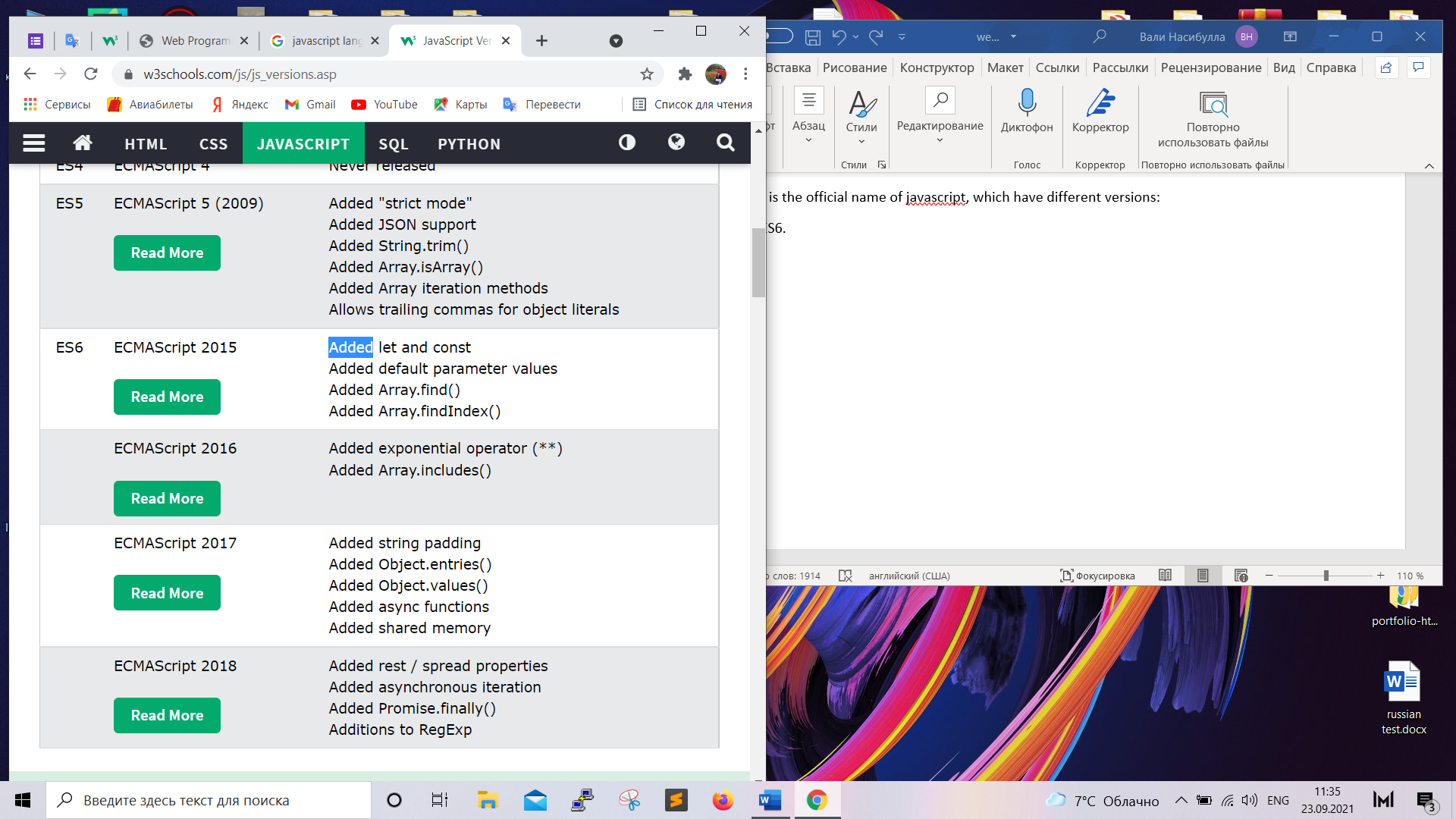
**JavaScript**: JavaScript is commonly used for **creating web pages**. It allows us to add dynamic behavior to the webpage and add special effects to the webpage.

# ECMAScript versions, new ES6 and ES7 features.

K

EsmaScript is the official name of javascript, which have different versions:

ES1, ES2,..ES6.



# Synchronous and asynchronous processing of HTTP requests. AJAX.

J

A **synchronous client constructs an HTTP structure, sends a request, and waits for a response**. An asynchronous client constructs an HTTP structure, sends a request, and moves on. In this case, the client is notified when the response arrives. The original thread, or another thread, can then process the response.

What does AJAX do in JavaScript?

AJAX stands for Asynchronous JavaScript And XML. In a nutshell, it is **the use of the XMLHttpRequest object to communicate with servers**. It can send and receive information in various formats, including JSON, XML, HTML, and text files.

# JQuery library. Purpose, main API. Use to implement AJAX and work with the DOM.

What is a jQuery library?

What is jQuery? jQuery is **a fast, small, and feature-rich JavaScript library**. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers.

What is API and how it works?

API stands for Application Programming Interface. An API is a software intermediary that allows two applications to talk to each other.

The AJAX technique makes **web pages more responsive by exchanging data with the web server behind the scenes**, instead of reloading an entire web page each time a user makes a change.

# Implementing AJAX with SuperAgent.

#### 6. SuperAgent

SuperAgent is a lightweight and progressive AJAX library that’s focused more on readability and flexibility. SuperAgent also boasts of a gentle learning curve unlike other libraries out there. SuperAgent has a request object that accepts methods such as GET, POST, PUT, DELETE, and HEAD.

Pros of SuperAgent

* It has a plugin-based environment and ecosystem where plugins could be built and developed for extra or additional functionality.
* Easily Configurable.
* Nice interface for making HTTP requests.
* Multiple functions chaining to send requests.
* Has to support for upload and download progress.
* Has support for chunked transfer encoding.
* Old-style callbacks are supported
* Numerous plugins available for many common features

Performing Get Request

request  
 .get('/user')  
 .query({ id: 1 })  
 .then(res => { });

Performing Post Request

request.post('/user')  
 .set('Content-Type', 'application/json')  
 .send('{"name":"Ipseeta","id":1}')  
 .then(callback)  
 .catch(errorCallback)

# Server scripts. CGI - definition, purpose, key features.

What is server scripting language?

Server-side scripting languages are **programming languages developed especially for creating HTML pages (or Web pages) on the server side**. These languages usually provide special libraries that facilitate creating HTML pages. ... They provide an interface for creating HTML tags.

CGI (**Common Gateway Interface**) is a standard way of running programs from a Web server. Often, CGI programs are used to generate pages dynamically or to perform some other action when someone fills out an HTML form and clicks the submit button.

**Features of CGI:**

* It is a very well defined and supported standard.
* CGI scripts are generally written in either Perl, C, or maybe just a simple shell script.
* CGI is a technology that interfaces with HTML.
* CGI is the best method to create a counter because it is currently the quickest.

# FastCGI - technology features, advantages and disadvantages in relation to CGI.

What is the difference between CGI and FastCGI?

**FCGI** is a newer variation on **CGI** that helps to reduce overhead associated with **CGI** programs, allowing a server to handle more web page requests at once. ... Each individual **FastCGI** process can handle many requests over its lifetime, thereby avoiding the overhead of per-request process creation and termination.

# PHP language - syntax, data types, embedding into web pages, rules for processing HTTP requests. Features of the implementation of OOP principles in PHP.

**PHP** is a general-purpose scripting **language** geared towards web development.

Php Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with <?php and ends with ?>:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1>My first PHP page</h1>  
  
<?php  
echo "Hello World!";  
?>  
  
</body>  
</html>

Php is not case sensitive in some keywords of it. But it is sensitive with var names.

PHP supports the following data types:

* String
* Integer
* Float (floating point numbers - also called double)
* Boolean
* Array
* Object
* NULL
* Resource

### Object-Oriented Features in PHP 5.0

An object is an encapsulation of data and methods. The data goes by various names - attributes, fields, properties. And the methods have various names - methods, functions, operations. A class is the definition of an object. It is the template used to create objects. Objects are instances of a class. Object oriented programming is built around 4 principles:

* **Abstraction:** The process of extracting the properties and behavior of real world objects and representing them in software.
* **Encapsulation:** Combining data and methods as one entity. There is a corollary concept *information hiding* where one hides the data from the user but expose the behavior of objects.
* **Inheritance:** Simulate the real world ordering of objects. Group shared properties and behaviors into one class that other classes can inherit from.
* **Polymorphism:** Ability to use or replace objects of one type (class) with another.