**WSD Lecture Notes**

The Internet

The internet is a world-wide network of computers.

LAN (Local Area Network) - computers and routers

A network of networks

ISP (Internet Service Provider) router to router

Browser checks a local cache or uses a DNS (Domain Naming Server)

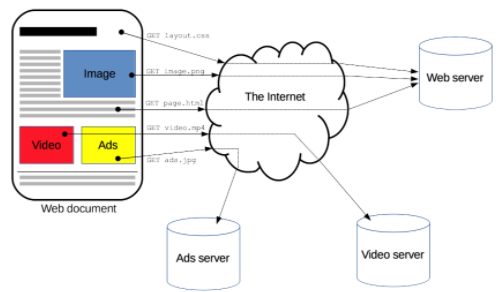
Browser sends a request for the resource from the IP address

**Finding Computers**

IP address (Internet Protocol)

173.194.121.32 - google.com

**HTTP (Hyper Text Transport Protocol)**



## **What is the Hypertext Transfer Protocol (HTTP)?**

HTTP is a protocol which allows you to fetch resources, for example HTML documents.

## **Why is HTTP called a stateless protocol?**

As there isn't a link between two requests being successfully carried out on the same connection. In order for this link to be established the use of cookies is required.

## **Describe the three phases of establishing a session using client-server protocols such as HTTP**

The first phase of establishing a session using client-server protocols is for the client to establish a connection. To open a connection in the HTTP the client has to initiate a connection in the underlying transport layer, usually the TCP. The second phase, of establishing a session using client-server protocols, is when the client sends its request and waits for an answer. A client request is divided into 3 parts. The request must first have a request method, this could be the documents path or the HTTP protocol version. Next, there's a HTTP header which gives the server information. The last part contains further data and isn't required. The final phase of establishing a session using client-server protocol is the server processing the request, sending an answer with a status code and the appropriate data. The server response consists of aknowledging the version of HTTP used, getting the information from the HTTP header and then a final data block containing optional data.

## **Describe the parameters on a get and post request**.

The parameters on a get and post request are located in the URL. For the get request the parameters are separated by an "&" sign and are added to the URL by the use of a "?". The post request sends additional data to the server in the message body of the HTTP request, making it more secure as it can't be bookmarked and isn't stored in the browser history.

## **What are the HTTP response status code and What do the HTTP status codes, 200, and 404 indicate?**

The HTTP response code tells the user whether the request was a success or a failure. The status code 200 indicates that the request was a success. The status code 404 indicates that the request was a failure.

## **What are HTTP cookies? Give an example**.

An HTTP cookie is a small piece of data the server sends to the user's web browser. Cookies are usually used to tell if 2 requests come from the same browser. An example of what cookies are used for is session management. Cookies are used for session management by keeping track of logins, shopping carts, games scores etc.

## **Why are cookies a security issue?**

Cookies are a security issue as they can be used to steal another users session which, if the user is logged in, could give an attacker access to private information, such as payment details.

## **Describe session hijacking and XSS**

Session hijacking is an attack where the user session is taken over by an attacker. The most common method of doing this is by using XSS. XXS is cross-site scripting. The attacker uses this by injecting client-side scripts into web pages causing the users browser to execute code when it loads the page. Without the use of HttpOnly attribute being set in session cookies, those injected scripts can gain access to the session key giving them the information to hijack the session.

## **Describe Cross-site request forgery(CSRF)**

Cross-site request forgery (CSRF) is an attack that impersonates a trusted user and sends the website unwanted commands. This could be done by including malicious parameters in the URL.

## **What are Cross-site HTTP requests (CORS) and what requests use CORS?**

Cross-origin resource sharing (CORS) is an HTTP header based mechanism that allows a server to indicate any other origins than its own from which a browser should allow the loading of resources. An XMLHttpRequest can use CORS.

**MIME Types**

MIME (Multipart Mail Extensions) or media type is a standard that indicates the nature and format of content.

**Structure**

The simplest MIME type consists of type and subtype. When the two strings are combined with a slash, they become a MIME type. No whitespace allowed.

*type/subtype*

The type is the category the data type fall in, I.e. video or text. The subtype identifies the exact kind of data of the specified type the MIME data represents. Parameters can be added.

There are two classes of type: **discrete** and **multipart**. Discrete types are types which represent a single file or medium, such as a single text or music file, or a single video. A multipart type is one which represents a document that's comprised of multiple component parts.

#### Discrete types

**application**

Any kind of binary data that doesn't fall explicitly into one of the other types; either data that will be executed or interpreted in some way or binary data that requires a specific application or category of application to use.

**Audio**

**example**

Reserved for use as a placeholder in examples showing how to use MIME types. These should never be used outside of sample code listings and documentation. example can also be used as a subtype; for instance, in an example related to working with audio on the web, the MIME type audio/example can be used to indicate that the type is a placeholder and should be replaced with an appropriate one when using the code in the real world.

**Font**

**image**

Image or graphical data including both bitmap and vector still images as well as animated versions of still image formats such as animated [GIF](https://developer.mozilla.org/en-US/docs/Glossary/GIF) or APNG.

**model**

Model data for a 3D object or scene

**Text**

Text-only data including any human-readable content, source code, or textual data such as comma-separated value (CSV) formatted data.

**Video**

#### Multipart types

**Multipart** types indicate a category of document broken into pieces, often with different MIME types; they can also be used — especially in email scenarios — to represent multiple, separate files which are all part of the same transaction. They represent a **composite document**.

There are two multipart types:

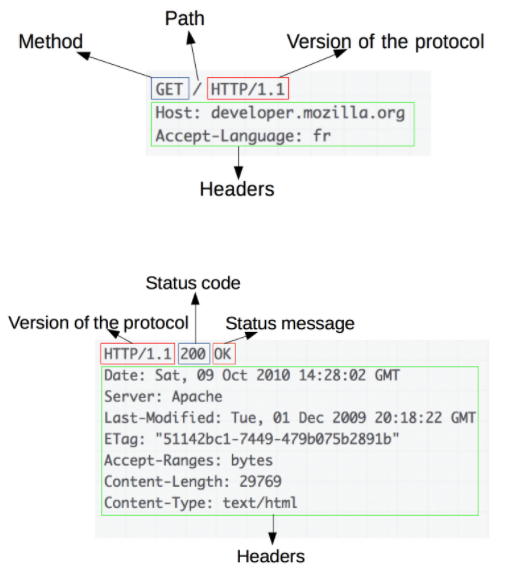
**message**  
A message that encapsulates other messages. This can be used, for instance, to represent an email that includes a forwarded message as part of its data, or to allow sending very large messages in chunks as if it were multiple messages.

**multipart**  
Data that is comprised of multiple components which may individually have different MIME types.

**URL**

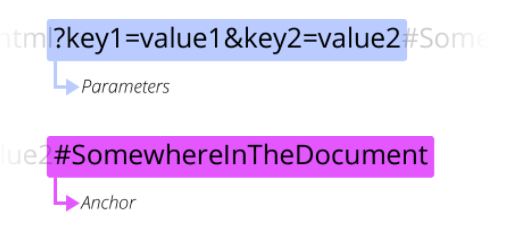
Universal Resource Locator (URL) is a particular type of Universal Resource Identifier(URI) used to surf the web



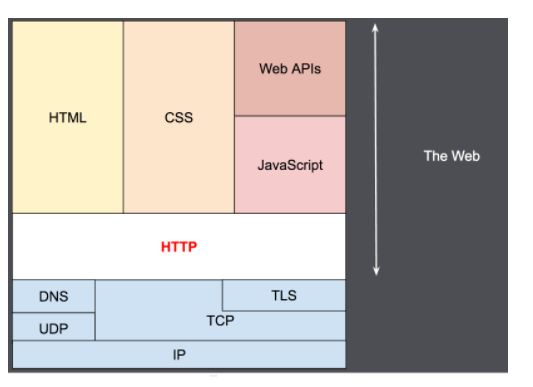


### **URL to a part of a document**

* Target of an HTTP request is called a *resource*
* Each resource has a Uniform Resource Identifier (URI)
* Location of a resource has a Uniform Resource Locator (URL)



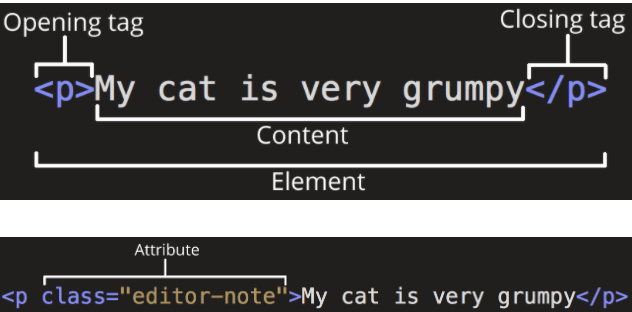
### **Layered Architecture**



**HTML**

HTML is the language used to create web page documents.

**Syntax**



**Element Types**

Block elements are structural or container elements. Block elements are nested to structure a document. Examles of block elements; html, body, nav, section, article, headings, paragraphs, lists, forms, footers, etc.

The main parts of our element are as follows:

1. **The opening tag:** This consists of the name of the element (in this case, p), wrapped in opening and closing **angle brackets**. This states where the element begins or starts to take effect — in this case where the paragraph begins.
2. **The closing tag:** This is the same as the opening tag, except that it includes a *forward slash* before the element name. This states where the element ends — in this case where the paragraph ends. Failing to add a closing tag is one of the standard beginner errors and can lead to strange results.
3. **The content:** This is the content of the element, which in this case, is just text.
4. **The element:** The opening tag, the closing tag, and the content together comprise the element.

**What is an attribute?**

Attributes contain extra information about the element that you don't want to appear in the actual content.

**Nesting Elements**

You can put elements inside other elements too — this is called **nesting**. If we wanted to state that our cat is **very** grumpy, we could wrap the word "very" in a [<strong>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/strong) element, which means that the word is to be strongly emphasized:

**HTML Tags**

* <!DOCTYPE html> — doctype. It is a required preamble. In the mists of time, when HTML was young (around 1991/92), doctypes were meant to act as links to a set of rules that the HTML page had to follow to be considered good HTML, which could mean automatic error checking and other useful things. However these days, they don't do much and are basically just needed to make sure your document behaves correctly. That's all you need to know for now.
* <html></html> — the [<html>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/html) element. This element wraps all the content on the entire page and is sometimes known as the root element.
* <head></head> — the [<head>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/head) element. This element acts as a container for all the stuff you want to include on the HTML page that *isn't* the content you are showing to your page's viewers. This includes things like [keywords](https://developer.mozilla.org/en-US/docs/Glossary/keyword) and a page description that you want to appear in search results, CSS to style our content, character set declarations, and more.
* <meta charset="utf-8"> — This element sets the character set your document should use to UTF-8 which includes most characters from the vast majority of written languages. Essentially, it can now handle any textual content you might put on it. There is no reason not to set this and it can help avoid some problems later on.
* <title></title> — the [<title>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title) element. This sets the title of your page, which is the title that appears in the browser tab the page is loaded in. It is also used to describe the page when you bookmark/favorite it.
* <body></body> — the [<body>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/body) element. This contains *all* the content that you want to show to web users when they visit your page, whether that's text, images, videos, games, playable audio tracks, or whatever else.

alt (alternative) attribute. In this attribute, you specify descriptive text for users who cannot see the image, possibly because of the following reasons:

* They are visually impaired. Users with significant visual impairments often use tools called screen readers to read out the alt text to them.
* Something has gone wrong causing the image not to display. For example, try deliberately changing the path inside your src attribute to make it incorrect. If you save and reload the page, you should see something like this in place of the image:

**Lists**

* **Unordered lists** are for lists where the order of the items doesn't matter, such as a shopping list. These are wrapped in a [<ul>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ul) element.
* **Ordered lists** are for lists where the order of the items does matter, such as a recipe. These are wrapped in an [<ol>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/ol) element.

Each item inside the lists is put inside an [<li>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/li) (list item) element.

Inline elements are inside block elements. An example of an inline element is the <a> element.

**THE HEAD**

The [head](https://developer.mozilla.org/en-US/docs/Glossary/Head) of an HTML document is the part that is not displayed in the web browser when the page is loaded. It contains information such as the page [<title>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title), links to [CSS](https://developer.mozilla.org/en-US/docs/Glossary/CSS) (if you choose to style your HTML content with CSS), links to custom favicons, and other metadata (data about the HTML, such as the author, and important keywords that describe the document.)

**HYPERLINKS**

Hyperlinks allow us to link documents to other documents or resources, link to specific parts of documents, or make apps available at a web address. Almost any web content can be converted to a link so that when clicked or otherwise activated the web browser goes to another web address ([URL](https://developer.mozilla.org/en-US/docs/Glossary/URL)).

Self closing elements only have 1 tag and is closed with />

**XML**

XML is a general purpose markup language. XML is exstensible which means it has no predefined vocabulary XML must have a well formed tree data structure and valid.

**Validating HTML**

Browsers are written to be permissive and allow errors

Without doctype, browser correct it with “quirk mode” which also corrects errors. Browsers do this because if it didn’t the web could break if pages wouldn’t open.

XHTML is well-formed XML and valid to DTD (Document Type Definition)

**Accesibility**

Vision Impairment

People with low or no vision may use an assistive device such as a screen reader to get content from the site. Could make zoom function.

Mobility impairment

Users may use alternative keyboard and mouse, foot pedals, joystick etc. Need to make website compatible with those devices.

Auditory Impairment

Users with limited or no hearing could miss any audio content on the website. Add alternatives such as transcripts for audio and captions for videos.

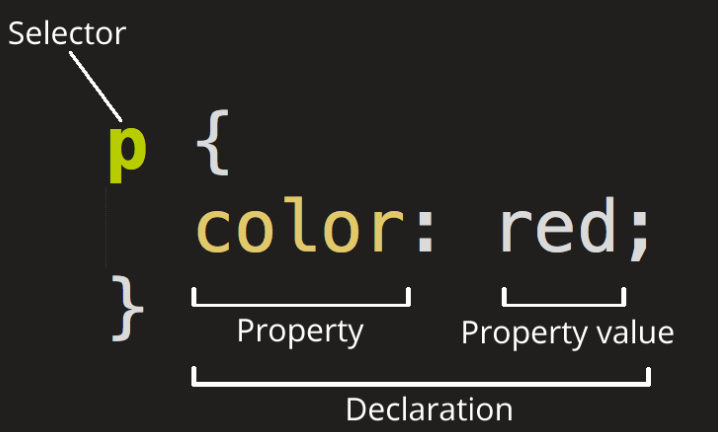
Cognitive Impairment

Users with memory, reading comprehension, problem solving benefit from simple and clear designs.

**CSS**

CSS (Cascading Style Sheets) is the code that styles web content.

**CSS is a style sheet language.** CSS is what you use to selectively style HTML elements.



The whole structure is called a **ruleset.**

**Selector**

This is the HTML element name at the start of the ruleset. It defines the element(s) to be styled (in this example, [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p) elements). To style a different element, change the selector.

**Declaration**

This is a single rule like color: red;. It specifies which of the element's **properties** you want to style.

**Properties**

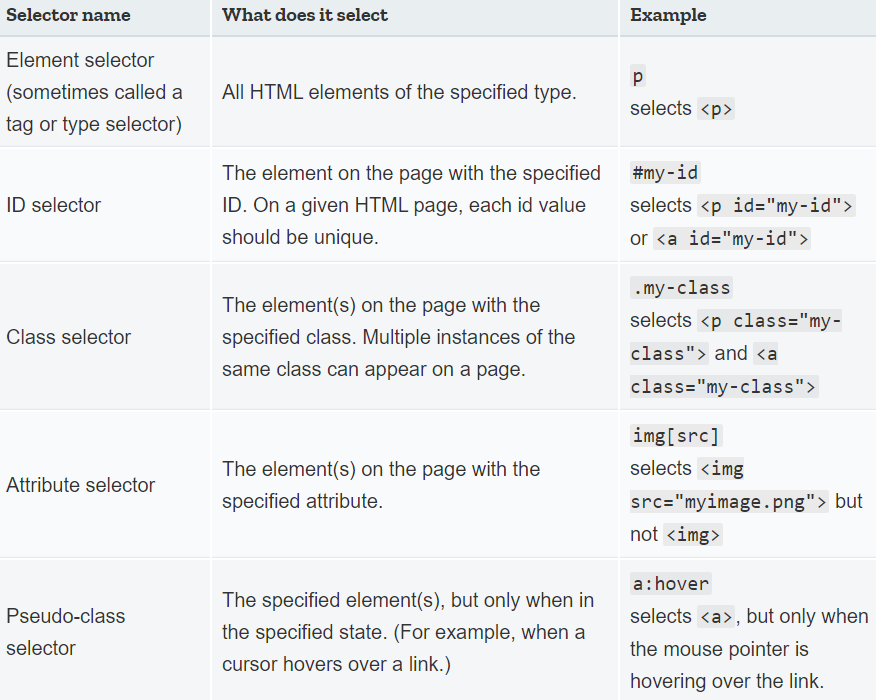
These are ways in which you can style an HTML element. (In this example, color is a property of the [<p>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/p) elements.) In CSS, you choose which properties you want to affect in the rule.

**Property value**

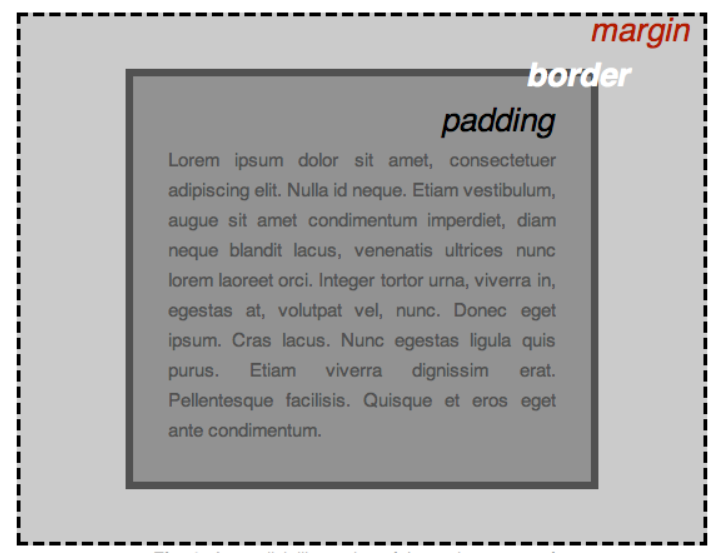
To the right of the property—after the colon—there is the **property value**. This chooses one out of many possible appearances for a given property. (For example, there are many color values in addition to red.)

* Apart from the selector, each ruleset must be wrapped in curly braces. ({})
* Within each declaration, you must use a colon (:) to separate the property from its value or values.
* Within each ruleset, you must use a semicolon (;) to separate each declaration from the next one.

There are many different types of selectors. The examples above use **element selectors**, which select all elements of a given type. But we can make more specific selections as well. Here are some of the more common types of selectors:



CSS layout is mostly based on the *box model.* Each box taking up space on your page has properties like:

* padding, the space around the content. In the example below, it is the space around the paragraph text.
* border, the solid line that is just outside the padding.
* margin, the space around the outside of the border.
* 

width (of an element).

background-color, the color behind an element's content and padding.

color, the color of an element's content (usually text).

text-shadow sets a drop shadow on the text inside an element.

display sets the display mode of an element. (keep reading to learn more)

**Colour / color**  
Can change colour with hex values

**JavaScript**

JavaScript is a programming language that adds interactivity to your website. This happens in games, in the behavior of responses when buttons are pressed or with data entry on forms; with dynamic styling; with animation, etc.

[JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) ("JS" for short) is a full-fledged [dynamic programming language](https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_programming_language) that can add interactivity to a website.

The heading text changed to *Hello world!* using JavaScript. You did this by using a function called [querySelector()](https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector) to grab a reference to your heading, and then store it in a variable called myHeading.

**Variables**

[Variables](https://developer.mozilla.org/en-US/docs/Glossary/Variable) are containers that store values. You start by declaring a variable with the [var](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var) (less recommended, dive deeper for the explanation) or the [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let) keyword, followed by the name you give to the variable. JavaScript is case sensitive.

After declaring a variable, you can give it a value:

myVariable = 'Bob';

Also, you can do both these operations on the same line:

let myVariable = 'Bob';

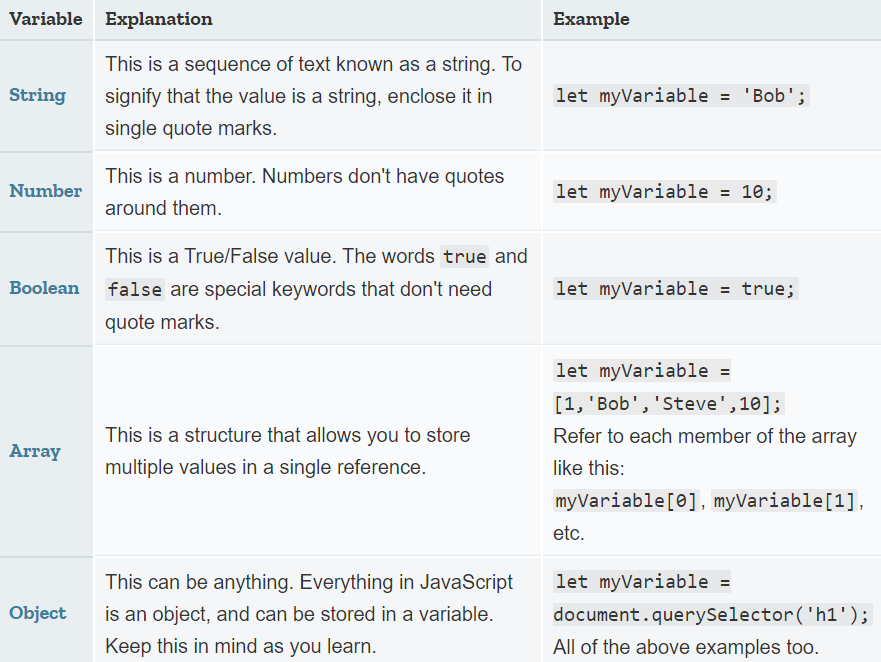
You retrieve the value by calling the variable name:

myVariable;

After assigning a value to a variable, you can change it later in the code:

let myVariable = 'Bob';myVariable = 'Steve';

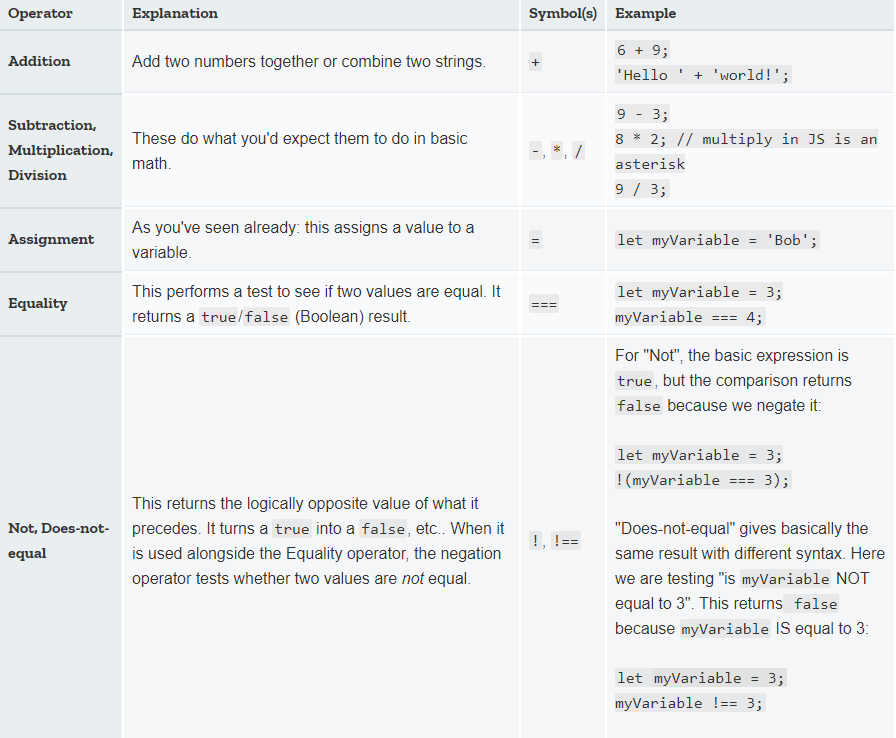
**Variable Table**



**STRING NUMBER BOOLEAN ARRAY OBJECT**

**Operators**

An [operator](https://developer.mozilla.org/en-US/docs/Glossary/operator) is a mathematical symbol which produces a result based on two values (or variables).

 **ADDITION SUBTRACTION MULTIPLICATION DIVISION EQUALITY NOT DOES NOT EQUAL**

**CONDITIONALS – IF STATEMENTS ETC.**

Conditionals are code structures used to test if an expression returns true or not. A very common form of conditionals is the if ... else statement. For example:

let iceCream = 'chocolate';if(iceCream === 'chocolate') { alert('Yay, I love chocolate ice cream!'); } else { alert('Awwww, but chocolate is my favorite...'); }

The expression inside the if( ... ) is the test. This uses the identity operator (as described above) to compare the variable iceCream with the string chocolate to see if the two are equal. If this comparison returns true, the first block of code runs. If the comparison is not true, the second block of code—after the else statement—runs instead.s

**FUNCTIONS**

[Functions](https://developer.mozilla.org/en-US/docs/Glossary/Function) are a way of packaging functionality that you wish to reuse. It's possible to define a body of code as a function that executes when you call the function name in your code. This is a good alternative to repeatedly writing the same code. You have already seen some uses of functions previously. For example:

1. let myVariable = document.querySelector('h1');
2. alert('hello!');

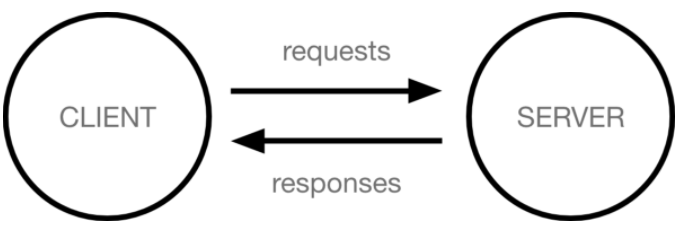
These functions, document.querySelector and alert, are built into the browser.

**EVENTS**

Real interactivity on a website requires events handlers. These are code structures that listen for activity in the browser, and run code in response. The most obvious example is handling the [click event](https://developer.mozilla.org/en-US/docs/Web/Events/click), which is fired by the browser when you click on something with your mouse.

**CLIENTS AND SERVERS**

Computers connected to the web are called **clients** and **servers**. A simplified diagram of how they interact might look like this:



* Clients are the typical web user's internet-connected devices and web-accessing software available on those devices.
* Servers are computers that store webpages, sites, or apps. When a client device wants to access a webpage, a copy of the webpage is downloaded from the server onto the client machine to be displayed in the user's web browser.
* **Your internet connection**: Allows you to send and receive data on the web. It's basically like the street between your house and the shop.
* **TCP/IP**: Transmission Control Protocol and Internet Protocol are communication protocols that define how data should travel across the internet. This is like the transport mechanisms that let you place an order, go to the shop, and buy your goods. In our example, this is like a car or a bike (or however else you might get around).
* **DNS**: Domain Name Servers are like an address book for websites. When you type a web address in your browser, the browser looks at the DNS to find the website's real address before it can retrieve the website. The browser needs to find out which server the website lives on, so it can send HTTP messages to the right place (see below). This is like looking up the address of the shop so you can access it.
* **HTTP**: Hypertext Transfer Protocol is an application [protocol](https://developer.mozilla.org/en-US/docs/Glossary/Protocol) that defines a language for clients and servers to speak to each other. This is like the language you use to order your goods.
* **Component files**: A website is made up of many different files, which are like the different parts of the goods you buy from the shop. These files come in two main types:
  + **Code files**: Websites are built primarily from HTML, CSS, and JavaScript, though you'll meet other technologies a bit later.
  + **Assets**: This is a collective name for all the other stuff that makes up a website, such as images, music, video, Word documents, and PDFs.

**WHAT HAPPENS WHEN YOU TYPE A WEB ADDRESS INTO A BROWSER**

* The browser goes to the DNS server, and finds the real address of the server that the website lives on (you find the address of the shop).
* The browser sends an HTTP request message to the server, asking it to send a copy of the website to the client (you go to the shop and order your goods). This message, and all other data sent between the client and the server, is sent across your internet connection using TCP/IP.
* If the server approves the client's request, the server sends the client a "200 OK" message, which means "Of course you can look at that website! Here it is", and then starts sending the website's files to the browser as a series of small chunks called data packets (the shop gives you your goods, and you bring them back to your house).
* The browser assembles the small chunks into a complete web page and displays it to you (the goods arrive at your door — new shiny stuff, awesome!).

**DNS**

IP addresses represent a unique location on the web. A DNS (Domain Name Server) matches up a web address the user types into the browser, to the websites IP address

**PACKETS**

When data is sent across the web, it is sent as thousands of small chunks, so that many different web users can download the same website at the same time. If websites were sent as single big chunks, only one user could download one at a time, which obviously would make the web very inefficient.

**ERRORS**

* **Syntax errors**: These are spelling errors in your code that actually cause the program not to run, like the Rust error shown above. These are usually easy to fix as long as you are familiar with the language's syntax and know what the error messages mean.
* **Logic errors**: These are errors where the syntax is actually correct, but the code is not what you intended it to be, meaning that the program runs incorrectly. These are often harder to fix than syntax errors, as there isn't an error message to direct you to the source of the error.