UNIT 1: HTML, CSS & Client Side Scripting

World Wide Web (WWW) also called Web, the Internet's (worldwide computer network) leading information retrieval site. The Internet provides users with access to a vast array of documents that are linked to each other by hypertext or hypermedia links — i.e. hyperlinks, electronic connections that link similar pieces of information to allow an user to access them. Hypermedia files function links to images, sounds, animations, and movies. The Web operates inside the Internet's primary client-server format; servers are pc programs that save and transmit files to different computer systems on the network when asked to, whilst consumers are packages that request files from a server as the person asks for them. Browser software program permits users to view the retrieved documents.

- In Hypertext Mark-up Language (HTML), a hypertext document with its corresponding text and hyperlinks is written and assigned an online address called an Uniform Resource Locator (URL).
- Tim Berners-Lee and his colleagues at CERN, an international research organization, headquartered in Geneva, Switzerland, started creating the World Wide Web in 1989. They developed the Hypertext Transfer Protocol (HTTP) protocol which standardized server-client communication.
- In January 1992, their text-based Web browser was made available for publication.
- The World Wide Web rapidly gained recognition with the launch of a web browser named Mosaic, created in the United States by Marc Andreessen and others at the University of Illinois National Centre for Supercomputing Applications, and launched in September 1993.
- Mosaic allowed people the usage of the Web to use the identical type of "point-and-click" graphical manipulations that had been accessible in private computers for some years.
- In April 1994 Andreessen cofounded Netscape Communications Corporation, whose Netscape Navigator grew to become the dominant Web browser quickly after its launch in December 1994.
- The InternetWorks of BookLink Technologies, its first tabs browser in which a user would be able to visit another website without opening a completely new window, debuted the same year. The World Wide Web was having millions of active users by the mid-1990s.

Microsoft Corporation, the software giant, was interested in promoting internet applications on personal computers and in 1995 created its own web browser (based initially on Mosaic), Internet Explorer (IE), as an add-on to the Windows 95 OS. IE was incorporated into the Windows operating system in 1996 which decreased competition from other Internet browser manufacturers including Netscape. IE gradually became the default Web browser.

Apple's Safari was released on Macintosh personal computers as the default browser in 2003, and later on iPhones (2007) and iPads (2010). Safari 2.0 (2005) was the first privacy-mode browser, Private Browsing, where the client does not save Web sites in its history, save files in its cache, or enter personal information on Web pages.

The first major challenger to IE's dominance was Mozilla's Firefox, released in 2004 and designed to address problems surrounding IE with speed and security. Google launched Chrome in 2008, the first browser featuring isolated tabs, which meant that when one tab crashed, other tabs and the entire browser would still work. In 2013 Chrome had become the

dominant browser, popularly outperforming IE and Firefox. In 2015, Microsoft removed IE, replacing it with Edge.
Smartphone's became more computer-like in the early 21st century, and more modern services, such as Internet access, became possible. Web use on Smartphone's has gradually increased, and it accounted for over half of Internet browsing in 2016.

Internet vs. World Wide Web

The Internet is a comprehensive computer network and was conceptualized by the ARPA or Advanced Research Projects Agency during 1969. The World Wide Web is much newer than the Internet, and was introduced during the 1990s.
The World Wide Web is a series of web pages that follow the http protocol, accessible from any part of the world through the Internet. The http protocol is a type of language used on the Internet to transmit data and to communicate.
It is an application that is used on the Internet and all pages that are part of the World Wide Web start with http:/www, with www being a World Wide Web abbreviation. The World Wide Web is a framework for the knowledge exchange.
It represents a way of accessing information through the Internet. Understanding the differences between the Internet and the World Wide Web is key to understanding the true workings of search engines. Search engines search websites that are accessible on the World Wide Web and not other internet- based sites.
As Web 2.0 web applications seek to brand their domain names and make them easily pronounceable, the use of the www prefix is declining. Given the growing popularity of the mobile web, services such as Gmail.com, MySpace.com, Facebook.com and Twitter.com are most frequently mentioned without adding "www." (Or indeed ".com") to the domain.

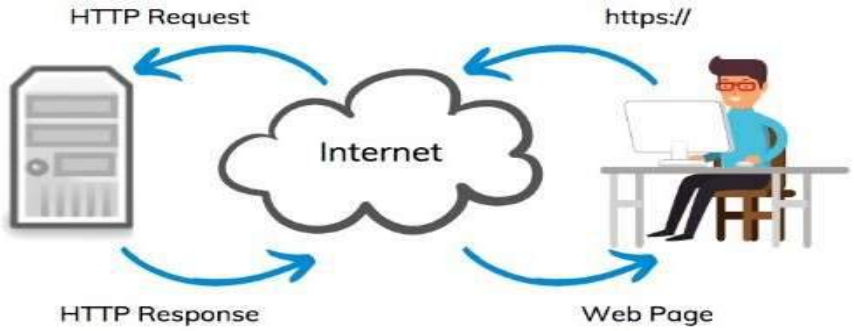
What Is the Web Made of?

The Web consists of:

- Your personal computer
- Web browser software to access the Web
- A connection to an Internet service provider (ISP)
- Servers to host the data
- Routers and switches to direct the flow of data

How the Web Works

Web pages are stored on web servers located around the globe. Entering the Uniform Resource Locator or URL of a web page in your web browser or clicking a link sends a request to the server that hosts the page. The server transmits the web page data to your computer and your web browser displays it on your screen.
A web page is an electronic document written in a computer language called HTML (Hypertext Mark-up Language).Web pages can contain text, graphics, audio, video, and animation, as well as interactive features, such as data entry forms and games. Each page has a unique address known as a URL (Uniform Resource Locator), which identifies its location on the server. Web pages usually contain hyperlinks to other web pages. Hyperlinks are text and images that reference the addresses of other web pages.



A website consists of one or more web pages that relate to a common theme, such as a person, business, organization, or a subject, such as news or sports. The first page is called the home page, which acts like an index, indicating the content on the site. From the home page, you can click links to access other pages on the site or other resources on the Web.

Basic components of the Web

The basic components of the Web are:

- *Web servers*, which are computers that carry distribution information over the Internet. For the example, one Web server may contain the online magazine what's On in Bath's text and images, and another server may contain details on which seats are available for a specific concert. The magazine will use the Web's own publishing language, HTML (Hypertext Mark-up Language), to format. The data on the seating applicable and their prices will be kept in a database with links to different forms published using HTML.
- *Servers* that can also be PCs, Macintosh systems or workstations from UNIX: it is the server software that makes them different, not the machine itself. The servers must be relatively up-to-market devices. Servers must always be left running, so that people can access information about them whenever they prefer. Another significant thing about servers: they are comparatively hard to set up. If you're a non- technical person who wants to be published on the Internet, it's best to rent some room on someone else's site.
- *Web clients* that can be PCs, Macintoshes and other internet enabled devices that can access information from Web servers. The machine at your desk is a Web client. Server applications can be run by PCs, Macintoshes, UNIX workstations and even single terminals. Different lient software for various platforms is marketed .Thus Mosaic has both an implementation of Macintosh and a PC.
- *HTTP protocol* used for transmission of files between servers and clients. When you click on a hypertext link or fill out a form in a Web document, the results must be sent as quickly as possible over the Internet and then understood by a server at the other end. Instructions like 'give me this file' or 'get me the picture' are carried by the HTTP web communication protocol. This protocol is the 'messenger' that gathers files from and to servers, and then delivers results to your computer whenever you click on a button. Among other Internet services, HTTP has its counterparts: FTP, file transfer protocol, and Gopher are protocols which obtain different kinds of information from across the Internet.
- *Browser* software which a Web client requires to view text, pictures, video clips, etc. This is given under the generic name 'browser,' which is perhaps the best-known example of Internet Explorer by Mosaic, Microsoft Corp. and Netscape

Communications Corp.'s Navigator and Communicator browsers. It gives the software ability to search Web server collected information, as you would browse through a book. It also provides you with facilities to save and print information accessed on the Internet.

Navigating the Web
There are three main ways to move between web pages or websites:

1. Clicking a text link.
2. Clicking a hyperlinked graphic, such as a button, photograph, or drawing.
3. Typing the URL of a web page in the location box (also known as the address field) of your web browser and then pressing the Enter or Return key.

Scheme specifiers

At the start of a web URI, the scheme specifiers http:/ and https:/ refer to the Hypertext Transfer Protocol or HTTP Stable, respectively. They specify the protocol of communication to be used for request and reply. The HTTP protocol is fundamental to the operation of the World Wide Web and when browsers send or retrieve sensitive data, such as passwords or banking information, the added encryption layer of HTTPS is vital. Web browsers normally prepend http:/ to user-entered URIs automatically, if omitted.

What is a URL?

URL is the abbreviation for Uniform Resource Locator and is known on the World Wide Web as the global address of documents and other resources. For instance, you'll go to the URL www.google.com to visit a website.
A domain name is part of a URL, which stands for Uniform Resource Locator.

Computers rely on a language consisting of numbers and letters called an IP address, so that computer networks and servers can "speak to each other." Each computer connecting to the Internet has a unique IP address, which looks like this

22.231.113.64 or 3ffe:1900:4545:3:200:f8ff:fe21:67cf

Typing in a long IP address is not ideal, or practical, for an online user to navigate quickly across the Network. That's why domain names have been developed-to cover IP addresses with something more memorable. The domain name may be called a "nickname" to the IP address. A URL, along with other basic information, includes the domain name to create a complete address (or "web address") to guide a visitor to a particular online website called a webpage. Essentially it's a series of directions and every web page has a special one.

Domain Name System

DNS stands for Domain Name System. DNS 'principal role is to convert domain names into IP addresses that computers can recognize. This also lists mail servers that accept Emails for each domain name. Internet machine is assigned a unique address, which is called an IP address. So looks a standard IP address: **199.123.456.7**
It's very hard to keep in mind the IP addresses of all the websites that we visit daily. Words are easier to remember than numeral strings. It's here that domain names get into the picture. What you need to know when you visit a website is their URL.
Computers remember numbers, and DNS helps to translate the URL to an IP address that the machine can use. If you type domain.com into the user, the user wants to get the www.domain.com IP address first. The user contacts a DNS server to ask where the web pages are located. It acts as an IP address directory service.

DNS Servers and IP Addresses

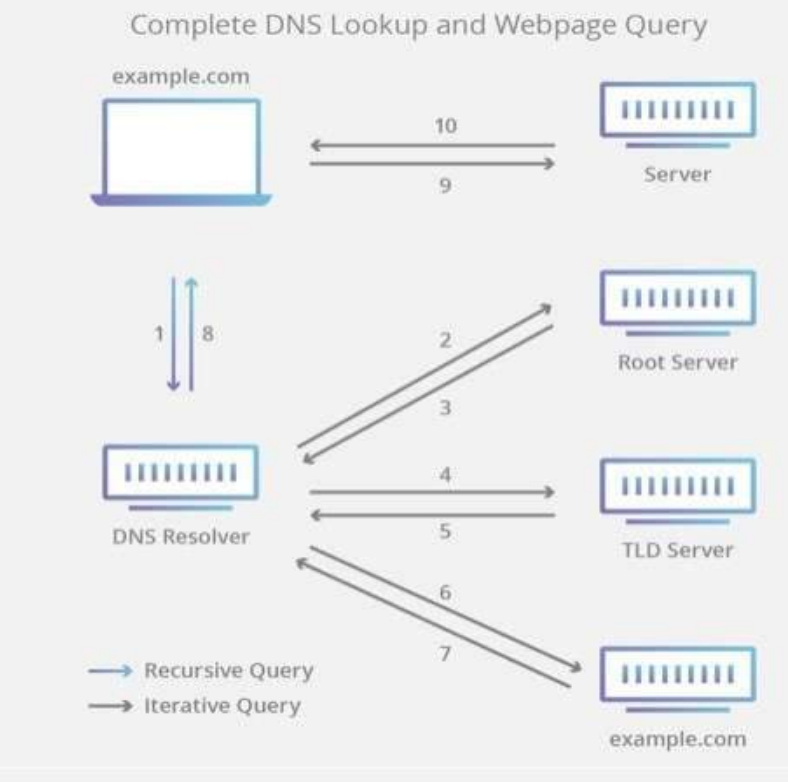
There are billions of IP addresses currently in use, and most machines have a human-readable name as well.DNS servers (cumulatively) are processing billions of requests across the internet at any given time. Millions of people are adding and changing domain names and IP addresses each day.
With so much to handle, DNS servers rely on network efficiency and internet protocols. Part of the IP's effectiveness is that each machine on a network has a unique IP address in both the IPV4 and IPV6 standards managed by the Internet Assigned Numbers Authority (IANA). Here are some ways to recognize an IP address:

- An IP address in the IPV4 standard has four numbers separated by three decimals, as in: 70.74.251.42
- An IP address in the IPV6 standard has eight hexadecimal numbers (base-16) separated by colons, as in 2001:0cb8:85a3:0000:0000:8a2e:0370:7334. Because IPV6 is still a very new standard, we'll concentrate on the more common IPV4 for this article.
- Each number in an IPV4 number is called an "octet" because it's a base-10 equivalent of an 8-digit base-2 (binary) number used in routing network traffic. For example, the octet written as 42 stands for 00101010. Each digit in the binary number is the placeholder for a certain power of two from 2 to 27, reading from right to left. That means that in 00101010, you have one each of 21, 23 and 25. So, to get the base-10 equivalent, just add 21 + 23 + 25 = 2 + 8 + 32 = 42.
- There are only 256 possibilities for the value of each octet: the numbers 0 through 255.
- Certain addresses and ranges are designated by the IANA as reserved IP addresses, which mean they have a specific job in IP. For example, the IP address 127.0.0.1 is reserved to identify the computer you're currently using.

What are the steps in a DNS lookup?
DNS is concerned with a domain name being translated into the appropriate IP address. To learn how this process works, it helps to follow the path of a DNS lookup as it travels from a web browser, through the DNS lookup process, and back again. Let's take a look at the steps.
The 8 steps in a DNS lookup:

1. A user types ‘example.com’ into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.

2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's name server, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.
9. Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:
10. The browser makes a HTTP request to the IP address.
11. The server at that IP returns the webpage to be rendered in the browser (step 10).



Top Level Domain (TLD)

TLD refers to the last part of a domain name. For example, the .com in amazon.com is the Top Level Domain. The most common TLDs include .com,
.net, org, and .info. Country code TLDs represents specific geographic locations. For example: .in represents India. Here are some more examples:

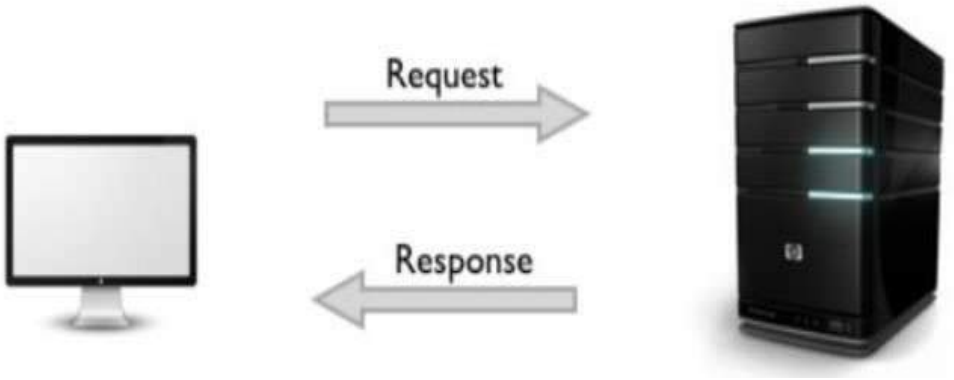
- COM — commercial websites, though open to everyone
- NET — network websites, though open to everyone
- ORG — non-profit organization websites, though open to everyone
- EDU — restricted to schools and educational organizations
- MIL — restricted to the U.S. military
- GOV — restricted to the U.S. government
- US, UK, RU and other two-letter country codes — each is assigned to a domain name authority in the respective country

In a domain name, each word and dot combination you add before a top-level domain indicates a level in the domain structure. Each level refers to a server or a group of servers that manage that domain level. An organization may have a hierarchy of sub-domains further organizing its internet presence, like "bbc.co.uk" which is the BBC's domain under CO, an additional level created by the domain name authority responsible for the U.K. country code.
The left-most word in the domain name, such as www or mail, is a host name. It specifies the name of a specific machine (with a specific IP address) in a domain, typically dedicated to a specific purpose. A given domain can potentially contain millions of host names as long as they're all unique to that domain. (The "http" part stands for Hypertext Transfer Protocol and is the protocol by which information is sent by the user to the website she is visiting. Nowadays, you're more likely to see "https" which is a sign the information is being sent by secure protocol where the information is encrypted.
A subdomain is a subdivision of a domain name, allowing you to put content in your URL before your namespace. For example, blog.companyname.com or shop.companyname.com would be a subdomain of the domain name companyname.com. For example: If a customer buys a domain with 123 Reg,
e.g. yourdomain.co.uk, they can set up subdomains, e.g. site1.yourdomain.co.uk or secure.yourdomain.co.uk. This is an excellent way of breaking up the website if you have different regions, products or even languages.

Protocol

Protocol is the standard means, rules and conventions which are accepted. It is the beginning and the core of the web. HTTP is the Hyper Text Transfer Protocol; it is a Web-based protocol. It is rules for providing to the web user,
and it is a standard delivered to the clients from all common web servers. When the web browser wants a user requesting documents, the server changes the document and the web browser converts this document to the appropriate type again. And it's getting sent to the user. It therefore serves to transmit files which contain web pages to users based on web protocols.
In other words, when communicate with Englishman, he can't recognize our phrases if talk Korean. Likewise, human beings made standards that can speak with all of webs. We called these standards a protocol, and when communicate on the web, it can communicate to suit internet protocols.

How Protocol Works?
The User uses his computer and connects Web server (Google, Yahoo, etc.) through Internet. Consider an example where user trying to access Google.



1. Enter http://www.google.com in the address bar using a web browser.

2. Web browser request information to the Google web server by the HTTP protocols.
3. Web server receives requests, and it sends the answer to the computer.

4. Web browser received the HTTP protocol information represented by texts and pictures.

What is HTTP (Hypertext Transfer Protocol)?
The Hypertext Transfer Protocol is an application protocol for distributed, collaborative, hypermedia information systems that allows users to communicate data on the World Wide Web.
What is the purpose of HTTP?
HTTP was invented alongside HTML to create the first interactive, text-based web browser: the original World Wide Web. Today, the protocol remains one of the primary means of using the Internet.
How does HTTP work?
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 in order to perform various tasks:

- 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

All HTTP servers use the GET and HEAD methods, but not all support the rest of these request methods.

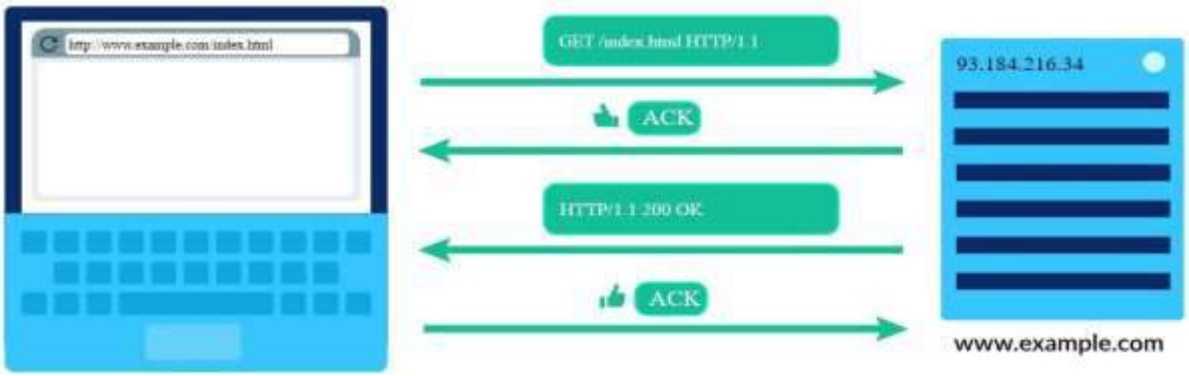
Basic aspects of HTTP

- HTTP is generally designed to be simple and human readable, even with the added complexity introduced in HTTP/2 by encapsulating HTTP messages into frames and reduced complexity for newcomers.
- Introduced in HTTP/1.0, HTTP headers make this protocol easy to extend and experiment with. New functionality can even be introduced by a simple agreement between a client and a server about a new header's semantics.
- HTTP is stateless: there is no link between two requests being successively carried out on the same connection. This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets. But while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.
- HTTP and connection is controlled at the transport layer, and therefore fundamentally out of scope for HTTP. Though HTTP doesn't require the underlying transport protocol to be connection-based; only requiring it to be reliable, or not lose messages (so at minimum presenting an error). Among the two most common transport protocols on the Internet, TCP is reliable and UDP isn't. HTTP therefore relies on the TCP standard, which is connection-based.
- Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips. The default behaviour of HTTP/1.0 is to open a separate TCP connection for each HTTP request/response pair. This is less efficient than sharing a single TCP connection when multiple requests are sent in close succession.

HTTP and TCP/IP

HTTP is a protocol that's built on top of the TCP/IP protocols.

Each HTTP request is inside an IP packet, and each HTTP response is inside another IP packet--or more typically, multiple packets, since the response data can be quite large.



- Diagram with laptop on left and server on right. Laptop has browser window with

"http://www.example.com/index.html" in address bar. Server is labelled with "www.example.com" and its IP address "93.184.216.34". 4 arrows are shown:

- First arrow goes from laptop to server and displays packet with HTTP request inside.
- Second arrow goes from server to laptop and displays packet with "ACK" inside.
- Third arrow goes from server to laptop and displays packet with HTTP response inside.
- Fourth arrow goes from laptop to server and displays packet with "ACK" inside.

There are many other protocols built on top of TCP/IP, like protocols for sending email (SMTP, POP, and IMAP) and uploading files (FTP).
All of these protocols enable us to use the Internet to connect with other computers in useful ways, and to communicate and collaborate across wide distances.

HTTPS

HTTPS stands for Hypertext Transfer Protocol over Secure Socket Layer. Think of it as a secure version of HTTP. HTTPS is used primarily on web pages that ask you to provide personal or sensitive information (such as a password or your credit card details).
When you browse a web page using HTTPS, you are using SSL (Secure Sockets Layer). For a website to use HTTPS it needs to have an SSL certificate installed on the server. These are usually issued by a trusted 3rd party, referred to as a Certificate Authority (CA).
When you browse a web page using HTTPS, you can check the details of the SSL certificate. For example, you could check the validity of it. You could also check that the website does actually belong to the organization you think it does. You can usually do this by double clicking on the browser's padlock icon. The padlock icon only appears when you view a secure site.

Component of HTTP are

1. Transfer time
2. Computer IP
3. Web serverIP
4. Method (get/post)

5. HTTP protocol version (1.0/1.1)

6. File format (flash, file data, etc.)

7. Reference (Previous web page address)

8. Language (Language type)

9. Encoding (Encoding type of English)

10. Information of web browser (IE/Firefox/Chrome etc.)

11. Cookies (Cookie values stored on my computer)

12. Real transfer content (id = iboss/ password=1234, etc.)

What does a typical HTTP request look like?

An HTTP request is just a series of lines of text that follow the HTTP protocol. A GET request might look like this:

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.1 zlib/1.2.11
Host: www.example.com
Accept-Language: en
```

This section of text, generated by the user's browser, gets sent across the Internet. The problem is, it's sent just like this, in plaintext that anyone monitoring the connection can read. (Those who are unfamiliar with the HTTP protocol may find this text hard to understand, but anyone with baseline knowledge of the protocol's commands and syntax can read it easily.)

This is especially an issue when users submit sensitive data via a website or a web application. This could be a password, a credit card number, or any other data entered into a form, and in HTTP all this data is sent in plaintext for anyone to read. (When a user submits a form, the browser translates this into an HTTP POST request instead of an HTTP GET request.)

When an origin server receives an HTTP request, it sends an HTTP response, which is similar:

```
HTTP/1.1 200 OK
Date: Wed, 30 Jan 2019 12:14:39 GMT
Server: Apache
Last-Modified: Mon, 28 Jan 2019 11:17:01 GMT
Accept-Ranges: bytes
Content-Length: 12
Vary: Accept-Encoding
Content-Type: text/plain

Hello World!
```

If a website uses HTTP instead of HTTPS, all requests and responses can be read by anyone who is monitoring the session. Essentially, a malicious actor can just read the text in the request or the response and know exactly what information someone is asking for, sending, or receiving.
When consider with HTTPS, HTTPS uses TLS (or SSL) to encrypt HTTP requests and responses, so in the example above, instead of the text, an attacker would see a bunch of seemingly random characters.
Instead of:

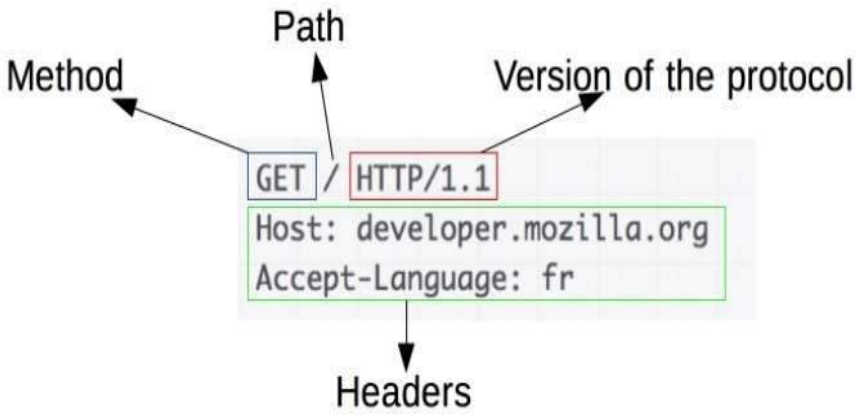
```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.63.0 libcurl/7.63.0 OpenSSL/1.1.1 zlib/1.2.11
Host: www.example.com
Accept-Language: en
```

The attacker sees something like:

```
t8Fw6T8UV81pQfyhDkhebbz7+oiwldr1j2gHBB3L3RFTRsQCpaSnSBZ78Vme+DpDVJPvZdZUZHpzbbcqmSW1+3xXGsERHg9Y
```

HTTP Messages
HTTP messages, as defined in HTTP/1.1 and earlier, are human-readable. In HTTP/2, these messages are embedded into a binary structure, a frame, allowing optimizations like compression of headers and multiplexing. Even if only part of the original HTTP message is sent in this version of HTTP, the semantics of each message is unchanged and the client reconstitutes (virtually) the original HTTP/1.1 request. It is therefore useful to comprehend HTTP/2 messages in the HTTP/1.1 format.
There are two types of HTTP messages, requests and responses, each with its own format.

Requests
An example HTTP request:



Requests consist of the following elements:

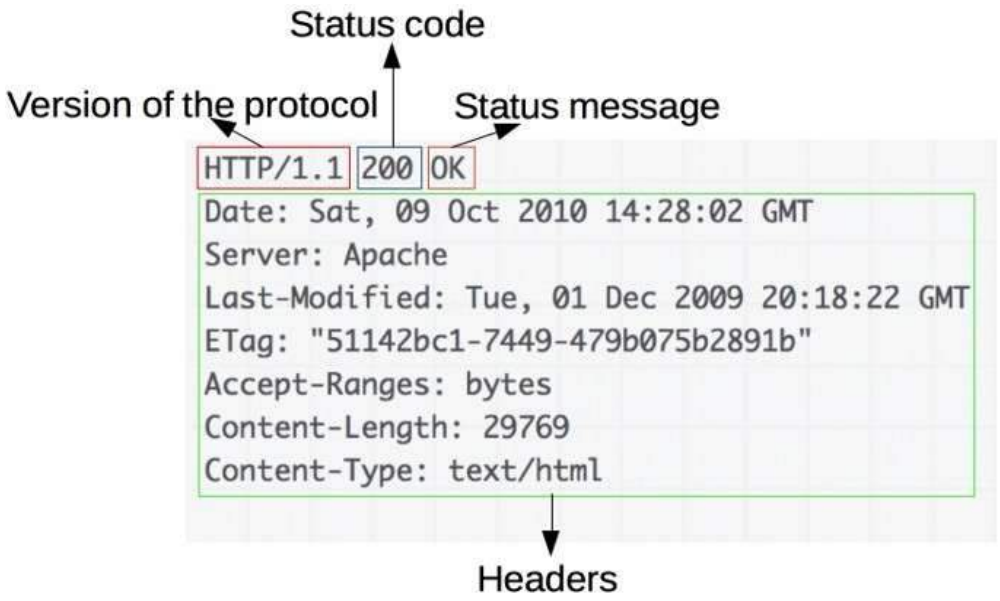
- An HTTP method, usually a verb like GET, POST or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to

fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.

- The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here, google.com), or the TCP port (here, 80).
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- Or a body, for some methods like POST, similar to those in responses, which contain the resource sent.

Responses

An example response:



Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful, or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers like those for requests.
- Optionally, a body containing the fetched resource.

HTTP can use both nonpersistent connections and persistent connections. A nonpersistent connection is the one that is closed after the server sends the requested object to the client. In other words, the connection is used exactly for one request and one response.

With persistent connections, the server leaves the TCP connection open after sending responses and hence the subsequent requests and responses between the same client and server can be sent. The server closes the connection only when it is not used for a certain configurable amount of time. With persistent connections, the performance is improved by 20%.

A persistent connection takes 2 RTT for the connection and then transfers as many objects, as wanted, over this single connection.

Nonpersistent connections are the default mode for HTTP/1.0 and persistent connections are the default mode for HTTP/1.1.
The non-persistent connection takes the connection time of 2RTT + file transmission time. It takes the first RTT (round-trip time) to establish the connection between the server and the client. The second RTT is taken to request and return the object. This case stands for a single object transmission.

HTTP Status Codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (100–199),
2. Successful responses (200–299),
3. . Redirects (300–399),
4. Client errors (400–499),
5. And Server errors (500–599).

1. Information responses

100 Continue : This interim response indicates that everything so far is OK and that the client should continue the request, or ignore the response if the request is already finished.

2. Successful responses

200 OK: The request has succeeded. The meaning of the success depends on the HTTP method:

- GET: The resource has been fetched and is transmitted in the message body.
- HEAD: The entity headers are in the message body.
- PUT or POST: The resource describing the result of the action is transmitted in the message body.
- TRACE: The message body contains the request message as received by the serve

3. Redirection messages

301 Moved Permanently: The URL of the requested resource has been changed permanently. The new URL is given in the response.

4. Client error responses

400 Bad Request: The server could not understand the request due to invalid syntax.

5. Server error responses

500 Internal Server Error: The server has encountered a situation it doesn't know how to handle.

UNIT 1: HTML, CSS & Client Side Scripting

HTML

HTML is the language in which most websites are written. HTML is used to create pages and make them functional. The code used to make them visually appealing is known as CSS.

HTML was first created by Tim Berners-Lee, Robert Cailliau, and others starting in 1989. It stands for Hyper Text Mark-up Language. Hypertext means that the document contains links that allow the reader to jump to other places in the document or to another document altogether. The latest version is known as HTML5.

HTML is a mark-up language which is comprised of a set of tags that describe the document's content. HTML files are simple text files that contain plain text and tags and typically have the file extension .html or .htm. They are commonly referred to as web pages. HTML describes what a page should look like when viewed through a web browser such a Mozilla Firefox, Google Chrome, Safari, and Internet Explorer.

A **Mark-up Language** is a way that computers speak to each other to control how text is processed and presented. To do this HTML uses two things: **tags** and **attributes**.

HTML Editors

HTML can be created or edited in many different types of editors, many of which are free and work incredibly well. In reality, all you need is a simple text editor, like Microsoft Notepad, or Text Edit on a Mac to get started. An HTML editor makes things much easier by colour coding different items making it

simple to find specific items or locate errors.

Free Editors:

- Notepad ++ - http://notepad-plus-plus.org/
- visual studio code - https://code.visualstudio.com/download
- Sublime - https://www.sublimetext.com/3

HTML Tags

HTML Tags are keywords or tag names surrounded by angle brackets or < > and normally come in pairs like this:

<tag> and </tag>.

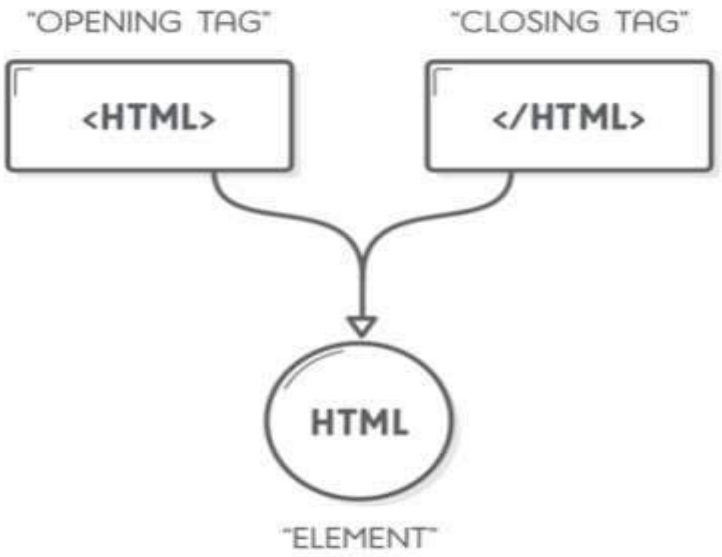
The first tag in a pair is the opening tag and the second tag is the closing tag. The closing tag is written the same way as the opening tag with a forward slash (/) to say “stop doing this.”

For Example:

<tagname> content </tagname>

Your HTML document should always contain <html> to signify the beginning of the HMTL content and </html> to signify the end. Without this tag, the document is only text.

Most tags come in matched "beginning" and "ending" pairs, but this is not an absolute rule.



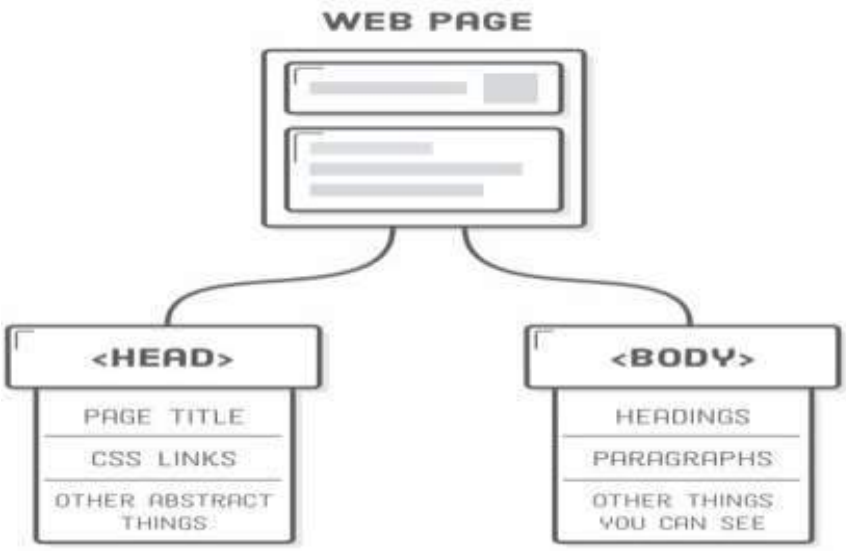
Any Web page you create will contain the following tags at the start of the page:

- <HTML>: tells the Web browser that this is the beginning of an HTML document
- <HEAD>: tells that Web browser that this is the header for the page (you'll learn later what goes between "HEAD" tags)
- <TITLE>: tells the Web browser that this is the title of the page
- <BODY>: tells the Web browser that this is the beginning of the Web page content -- everything you want to say and see on your page will follow this tag.
- The tags needed to end any Web page are:
- </BODY>
- </HTML>

Example:

```
<html>
  <head>
    <title>My First Page</title>
```

```
</head>
<body>
  Hello there. This is my first page!
</body>

</html>
```



Document Language

A web page's default language is defined by the Lang attribute on the top-level <html> element. The document is in English, use the en country code as the attribute value :

```
<html lang='en'>
```

Character Sets

A "character set" is kind of like a digital alphabet for your browser. It's different from the language of your document in that it only affects how the letters themselves are rendered, not the language of the content.

HTML Elements

An HTML Element is everything from the opening tag to the closing tag. The element content is everything between the opening and closing tags. There are also special elements that are closed in the start tag such as a line break. Most HTML elements can have attributes, or special characteristics that describe the element.

Start Tag	Element Content	End Tag
<html>	All of your HTML code	</html>
<head>	Special information about your page	</head>
<body>	The content of your page	</body>
<h1> to <h6>	Headings	</h1> to </h6>
<hr>	Adds a horizontal rule	none
	This is a paragraph	
	This is a link	
 	This is a line break	
<!--	This is a comment	-->

Special Elements

While there are a number of special HTML elements, such as the
 tag mentioned above, there are some other special elements you should know.

HTML Lines: The HTML horizontal rule can be used to divide content areas and uses the <hr> tag. Inserting the <hr> tag will draw a horizontal line across your content area.

HTML Comments: Comments can be inserted into HTML code to make it more readable and to explain to the reader of your code what it is you plan to do or what you have changed. It's always good practice to comment your HTML code.

Comment elements are written as follows and do not show on your rendered page. <!-- This is a comment --> The <!-- is the beginning of the comment

and the --> is the end. Everything typed within these tags will be invisible to the viewer unless the source code is viewed.

HTML "White Space": Browsers will ignore all "white space" in your HTML document. White space can be added to make your code more human readable, but it will be completely ignored when the browser renders the document. Keep this in mind when you write your code. Everything is controlled by a tag. Tags tell the browser what to do, if you instruct nothing, nothing will result.

HTML Attributes

HTML elements can have attributes which provide additional information about an element. Attributes are always assigned in the opening tag and always contain a name and value pair. The value must be contained in double quotes.

<tag name="value"> Content </tag>

Attribute	Description
Class	Specifies one or more classnames for an element (CSS)
Id	Specified a unique id for an element
Style	Specifies an inline CSS style for an element
Title	Specifies extra information about an element (displays as tooltip)

A common example of a complete element:

** google.com **

HTML Headings

HTML headings are defined with <h1> through <h6> tags. <h1> defines the most important heading while <h6> defines the lease important heading. The browser used to view the headings will automatically add space before and after each heading. It is very important that you use headings for your content

headers only and not simply to make any text larger. Search engine crawlers will use your heading tags to organize your content by order of importance.
<h1> should always be used for your most important topics, followed by <h2> and so on.

HTML Paragraphs

HTML Documents are divided into paragraphs. Paragraphs are defined with the tag. Browsers will automatically add white space above and below a paragraph tag. Make sure to include the closing tag to complete the paragraph and start the next. Below is an example paragraph.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras pharetra metus a arcu vulputate aliquet. Nulla ac metus ut neque fringilla posuere. Pellentesque quis viverra nisl.

HTML Formatting

HTML also uses tags for formatting text, much like you would with a word processing program. Text formatting means simply things like bold, italic, and underline. You should note, however, that underlining text in an HTML document is terribly poor form as it can be misconstrued as a link. All formatting tags must be closed.

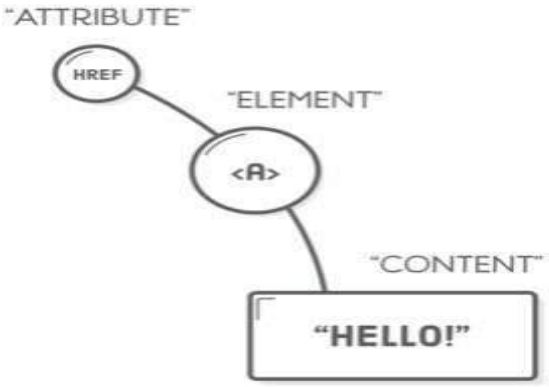
Tag	Description
<b>	Defines bold text
	Also defines bold text
<i>	Defines italic text
	Also defines italic text
<sub>	Defines subscript text
<sup>	Defines superscript text
<blockquote>	Defines a section of text that will be indented

Example:

<p>Text formatting means simply things like bold, italic, and <u>underline.</u>

HTML Links

The HTML <a> tag defines an anchor or hyperlink. An element adds meaning to the content it contains, an HTML "attribute" adds meaning to the element it's attached to.



A hyperlink (or link) is a word, group of words, or image that you can click on to jump to another document. When you move the cursor over a link in a Web page, the arrow will turn into a little hand.

The most important attribute of the <a> element is the href attribute, which indicates the link's destination. By default, links will appear as follows in all browsers:

- An unvisited link is underlined and blue
- A visited link is underlined and purple
- An active link is underlined and red

HTML Head

The HTML <head> element is a special container element to contain all of the head specific elements. Elements inside the head can include scripts, tell the browser where to find external information such as style sheets or JavaScript's

and provide search engines with descriptive information about the content of the page.

Tag	Description
<head>	Defines information about the document
<title>	Defines the title of a document
<base>	Defines the default address of the page
<link>	Links to the document to an external resourse
<meta>	Defines metadata about an HTML document
<script>	Defines a client-side script
<style>	Defines style information for a document

HTML Images

Images are displayed in HTML by use of the <img> tag. The <img> tag does not need to be closed. The <img> tag has multiple attributes to define what to display and how it should be displayed. As with other attributes, values must be contained in quotes.

Attribute	Value	Description
alt	Plain text	Alternate text to describe the image
border	Pixels #	Width of border around an image "0" for none
height	Pixels #	Height in pixels
src	URL	The location of the image file
width	Pixels #	Width in pixels

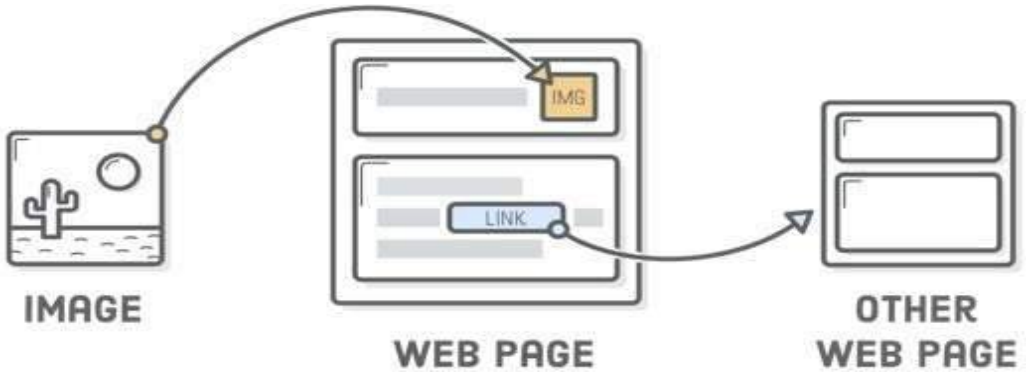


Image Formats

There's four main image formats in use on the web, and they were all designed to do different things.

JPG Images

JPG images are designed for handling large colour palettes without exorbitantly increasing file size. This makes them great for photos and images with lots of gradients in them.

GIF Images

GIFs are the go-to option for simple animations, but the trade off is that they're somewhat limited in terms of colour palette—never use them for photos. Transparent pixels are a binary option for GIFs, meaning you can't have semi-opaque pixels.

PNG Images

PNGs are great for anything that's not a photo or animated. For photos, a PNG file of the same quality (as perceived the human eye) would generally be bigger than an equivalent JPG file.

SVG Images

Unlike the pixel-based image formats above, SVG is a vector-based graphics format, meaning it can scale up or down to any dimension without loss of quality. This property makes SVG images a wonderful tool for responsive design. They're good for pretty much all the same use cases as PNGs, and you should use them whenever you can.

Image Dimensions by default, the <img/> element uses the inherit dimensions of its image file. Our JPG, GIF, and PNG images are actually 150×150 pixels, while SVG is only 75×75 pixels.

Adding alt attributes to your <img/> elements is a best practice. It defines a "text alternative" to the image being displayed. This has an impact on both search engines and users with text-only browsers (e.g., people that use text-to-speech software due to a vision impairment).

Aligning Images

The horizontal alignment of images on web page is formatted using the below tags and techniques

- Use the tag before the image tag to centre or justify right or left.
- Use the tag after the image tag to end the justification.
- Use the <align="..."> tag within the "img src" tag to have text wrap around the image.

Insert below tags **within** the "img src" tag for the **vertical** alignment of images in relation to text:

align="bottom" The text lines up with the bottom of the image.

<img src="https://imageww.com/gif/hsw-logo3.gif" alt="flower" border=0 **align="bottom"**>

align="top" The text lines up with the top of the image.

<img src="https://imageww.com/gif/hsw-logo3.gif" alt="flower" border=0 **align="top"**>

align="middle" The text lines up alongside the middle of the image.

<img src="https://imageww.com/gif/hsw-logo3.gif" alt="flower" border=0 **align="middle"**>

HTML Lists

There are two types of lists in HTML, Ordered and Unordered. Quite simply,

the two are best described as Numbered and Bulleted, respectively. Lists contain two types of tags: The type of list: Ordered <ol> and Unordered <ul> and the List items <li>.

Unordered List
- List Item
- List Item
- List Item

Ordered List
1. List Item
2. List Item
3. List Item

Example:

```
<ul>
    <li>List Item</li>
    <li>List Item</li>
    <li>List Item</li>
</ul>
```

Descriptive Lists

A descriptive list creates a list of text items with an indented second line:

Tony Stark

Founder of Stark Company

Use the following tags in this manner:

<dl>

<dt> Tony Stark

<dd>Founder of Stark Company

</dl>

The <dt> tag should correspond with the text you want lined up on the margin; <dd> corresponds with the line you want indented.

HTML Tables

A table is comprised of rows and columns, similar to a spreadsheet, and can be

quite complex. Tables consist of a number of tags and will always start with the <table> tag. Like many other tags the table tag can have attributes assigned to it such as width and follow the same rules as other attributes. The <table> tag signifies the start of a table but will need other tags to assign rows and columns inside it.

Table Rows and Columns

Table Rows are defined using the <tr> tag and columns are defined using the <td> tag. The <td> tag stands for 'Table Data' and can contain text, images, links, lists or any other HTML element.

Table Tag	Meaning	Location
<thead>	Table Head	Top of the table
<tbody>	Table Body	Content of the table
<tfoot>	Table Foot	Bottom of the table
<colgroup>	Column Group	Within the table
<th>	Table Header	Data cell for the table header

Below is an example of a simple table in HTML.

```
<table>
<tr>
   <td>Row One - Column One</td>
   <td>Row One - Column Two</td>
</tr>
<tr>
   <td>Row Two - Column One</td>
   <td>Row Two - Column Two</td>
</tr>
</table>
```

HTML Output:

Row One –Column One	Row One – Column Two
Row Two – Column One	Row Two – Column Two

Cellpadding

The "cellpadding" tag specifies (in pixels) the amount of space between the edges of each cell and the data inside each cell. Use it within the starting "table" tag:

Example 1: <table border=1 cellpadding=5>

Example 2: <table border=1 cellpadding=15>

Cellspacing

The "cellspacing" tag specifies (in pixels) the amount of space between each cell. Use it within the "table" tag:

Example 1: <table border=1 cellspacing=5>

Example 2: <table border=1 cellspacing=15>

Alignment and Cell Padding

By default, all cell contents within a table (with the exception of table headings) align vertically centered and left justified. To make the contents of a cell align a different way, apply the following tags within the <td>, <th> or <tr> tags:

For horizontal alignment, values can be left, right, or centre:

Example: <tr align="centre">

For vertical alignment, values can be top, bottom, or middle:

Example: <td valign="top">

Cell Spanning

"Spanning" occurs when one cell spans two or more other cells in the table.

For column spanning, the tag <colspan=value> is placed within the <td> tag where it applies. Here is a code example:

<table border=1>

<tr><td colspan=2>This cell spans over two columns </td>

<td>This cell spans over one column </td> </tr>

<tr align="centre"> <td>A </td> <td>B </td> <td>C </td> </tr>

</table>

For row spanning, the tag <rowspan=value> is placed within the <td> tag where it applies. For example:

<table border=1>

<tr><td rowspan=2>This cell spans over two rows </td>

<td>A</td>

<td>B</td>

</tr>

<tr>

<td>C</td>

<td>D</td>

</tr>

</table>

Every web page you create should define the language it's written in and its character set.

Forms

A form is a component of a Web page that has form controls, such as text, buttons, checkboxes, range, or colour picker controls. A user can interact with such a form, providing data that can then be sent to the server for further processing (e.g. returning the results of a search or calculation). No client-side scripting is needed in many cases, though an API is available so that scripts can augment the user experience or use forms for purposes other than submitting data to a server.

HTML forms could have their own category because they have many elements that have to work together in order for the form to work. Forms are the primary way users pass information to the server. The HTML form is encapsulated by the <form> tags and all of the inputs will fall in between the opening and closing form tags.

All forms start with the <FORM> tag and end with </FORM>. All other form objects go between these two tags.

The form tag has two main properties: **METHOD** and **ACTION**.

METHOD refers to post or get. The post attribute will send the information from the form as a text document. The get attribute is used mostly with search engines, and will not be discussed. We will generally set METHOD="post".

ACTION usually specifies the location of the CGI script that will process the form data.

```
<FORM METHOD="post" ACTION="mailto:put.your@email.address.here"></FORM>
```

HTML Form Inputs

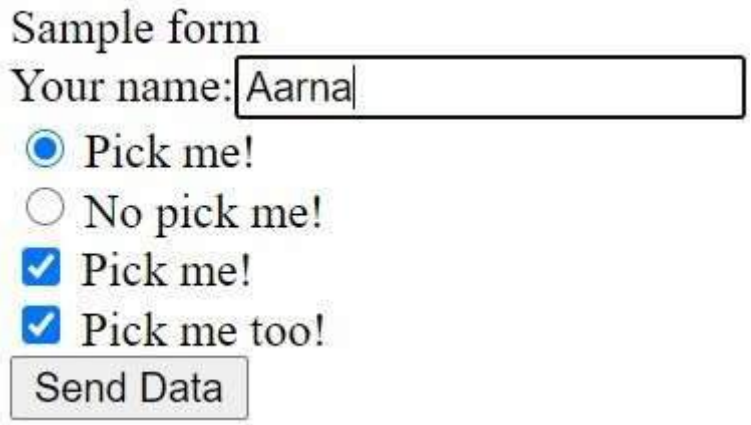
HTML Form Inputs are :

- Text Fields – Allows the user to input text data in a one line field
- Radio Buttons – Allows the user to select one option from multiple options
- Checkboxes – Allows the user to select many options from multiple options
- Submit Button – Allows the user to send the data

Example:

```
<!DOCTYPE html>
<html>
    <head>Sample form</head>
    <form>
        Your name:<input type="text" name="yourName" /><br>
        <input type="radio" name="group1" value="1" /> Pick me! <br>
        <input type="radio" name="group1" value="2" /> No pick me! <br>
        <input type="checkbox" name="checkBox1" value="1" /> Pick me! <br>
        <input type="checkbox" name="checkBox1" value="2" /> Pick me too! <br>
        <input type="submit" value="Send Data"/>
    </form>
</html>
```

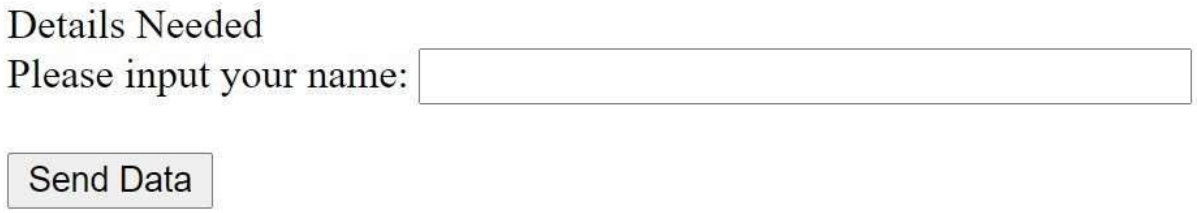
Results:



Typically, the form tag will have several attributes including the name, action, and the method. The name attribute is only to differentiate one form from another on the page. The action attribute is set to the page where you will pass the data (generally, this will not be an HTML page). Finally, the method attribute will determine how you want to send the data.

Single-line Text Entry

A single-line text entry allows the user to input a small amount of text, like a name or an email address.



Syntax:

Please input your name: <INPUT TYPE="text" SIZE="40" MAXLENGTH="30" NAME="personal-name">

The tag has the following elements:

<*INPUT*> is the tag used for most of the form objects.

TYPE="text" sets the object to a single-line text field.

Size="40" sets the field to show 40 characters.

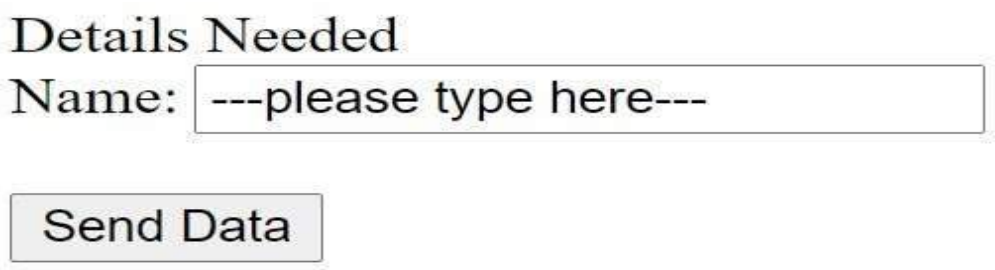
MAXLENGTH="30" means that only a total of 30 characters can be typed in this field.

NAME="personal-name" sets the text field's name to be personal-name (this information is part of the form data sent on for further processing). The name is required to identify the value data which will be associated with it. For example, if the text box was for an email address or telephone number, it might set this attribute with a more suggestive value, e.g. NAME="email" or NAME="tel-no". The easiest way to choose the name is simply to use the purpose of the field.

VALUE=... Another attribute, VALUE, can be used to place an initial text in the field. As might be expected, if this text is left unchanged it becomes the value associated with the NAME when the form is submitted. Putting an initial value in the field is usually not required, but can be used as a prompt. For example, the following HTML produces the figure that comes after it:

Syntax:

Name: <INPUT TYPE="text" NAME="name" SIZE="35" VALUE="---please type here---">



Multi-line Text Entry

While you can type as much text as you need into a single line text field, entering large quantities of text into a multi-line input field is more realistic. HTML calls these fields' text areas.

<TEXTAREA></TEXTAREA> tags are used for the following. They create a scrollable text area for larger amount of text:

<TEXTAREA NAME="movie-comments" COLS="50" ROWS="5">

</TEXTAREA>

NAME="movie-comments" supplies the text field with the given label. Here, because the example allows the user to write about movies, its named the form element "movie-comments".

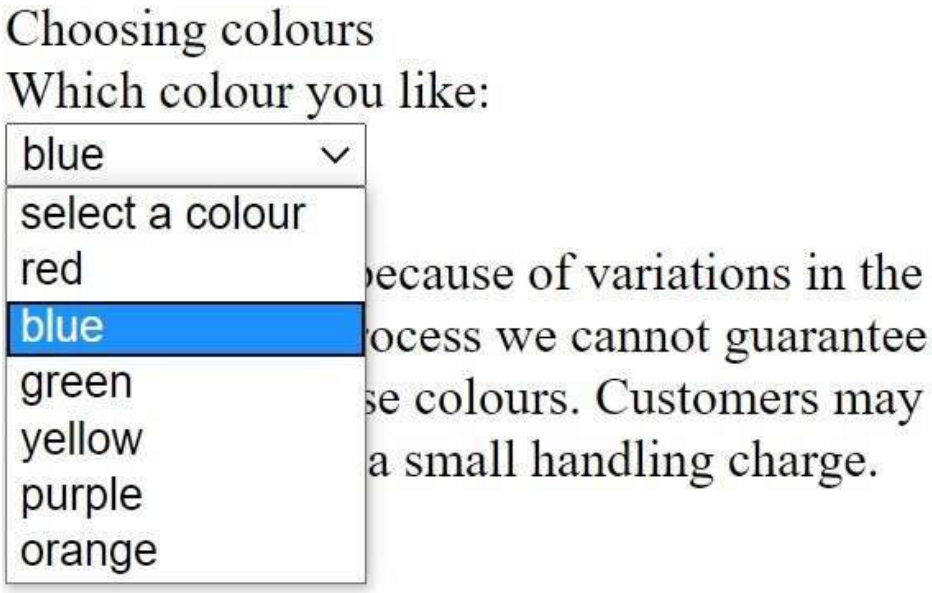
COLS="50" specifies the width, in characters, of the text area. Here 50 characters have been specified. ROWS="5" specifies the height of the text area in rows. This examples specifies five columns. Note that the text that should appear in the multi-line field is placed between the <TEXTAREA> and </TEXTAREA> tags. So, for example, users could be prompted to input their comments on a movie thus:

<TEXTAREA NAME="movie-comments" COLS="50" ROWS="5"> --- please enter your comments here--- </TEXTAREA>

A WRAP attribute is available, and the default value is WRAP="ON". You can change wrapping to off, which will cause a horizontal scroll bar to appear at the bottom edge of the text area.

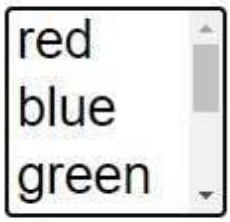
Menu Buttons and Scrolling Lists

Menu buttons and scrolling lists are useful when you have multiple options to choose from. For example, the following shows a menu button (sometimes imprecisely called a pull-down or drop- down menu). Clicking on the 'select a colour' option on the button displays all the other options in the button's menu. Pulling down to the required option will set the value for this element.



Scrolling lists, otherwise known as selection lists, are similar to menu buttons, but they usually display more than one of the available options at a time. They rarely show all options, and the user is required to scroll in order to view them all. If all the options are displayed, no scroll bar is included with the list, and it may not be obvious to the user that they should select an option.

colour:



Please note that because of variations

Unit 1:
HTML5 (New Semantic Tags)

HTML5 semantic elements are here to help web developers to navigate though their web page with ease, since the semantic elements together forms a more structured layout.

The purpose of creating semantic elements in HTML5 is to give meaning to its traditional design layout. It helps browsers quickly and efficiently understand the structure of the layout, and two, it helps web developers to systematically arrange or design web pages and give meaning to each section of the layout. The elements are easy to remember, and fits where it needs.

Technically, do not have to use the DIV element as a container to define every section on the web page. Define the header section using the <header> element and the footer section using <footer> element and so on.

For a example,

Developers have used the DIV element ( ) as a primary container, which served as header, footer, navigation etc, by identifying each element with unique ids or Class attribute.

Header using DIV

```
<div id="header">
   Header content
</div>
```

Footer using DIV

```
<div id="footer">
   Footer content
</div>
```

The above structure cannot be considered as meaningless, since it serves the purpose of separating two different sections as header and footer.

However, HTML5 has added a little meaning (semantic) and identifier to its elements, so web developers can use them wisely in their web pages.

HTML 5 introduces a number of new elements. Some of these are what I've termed "structural"—section, nav, aside, header, and footer. The dialog element is a kind of content element, akin to blockquote. There are also a number of data elements, such as meter, which "represents a scalar measurement within a known range, or a fractional value; for example disk usage," and the time element, which represents a date and/or a time.

Example

Semantic <header> and <footer> elements

Prior to HTML5 semantics, the header and footer would have served as independent sections of a web page. We can define the new semantic <header> and <footer> elements as part of a section or sections.Each section can have its own <header> and <footer> element, allowing us to add multiple semantic elements in a single web page.

HTML 5 is the new major version of HTML. HTML5 brings a host of new elements and attributes to allow developers to make their documents more easily understood by other systems (especially search engines). In addition, HTML 5 will also include fancy APIs for drawing graphics on screen, storing data offline, dragging and dropping, and a lot more. Here are new HTML5 tags that will make it easier to write Web sites.

Doctype

In HTML 5, there is only one doctype. It is declared in the beginning of the page by <!DOCTYPE HTML>. It simply tells the browser that it's dealing with an HTML-document.

<video> and <audio>

One of the biggest uses for Flash, Silverlight, and similar technologies is to get a multimedia item to play. With HTML5 supporting the new video and audio controls, those technologies are now relegated to being used for fallback status. The browser can now natively display the controls, and the content can be manipulated through JavaScript. Don't let the codec confusion scare you away. You can specify multiple sources for content, so you can make sure that your multimedia will play regardless of what codecs the user's browser supports.

<input> type attributes

The venerable <input> element now has a number of new values for the type attribute, and browsers do some pretty slick things depending on its value. For example, set type to "datetime" and browsers can show calendar/clock controls to pick the right time, a trick that used to require JavaScript. There is a wide variety of type attributes, and learning them (and the additional attributes that go with some of them) will eliminate the need for a lot of JavaScript work.

<canvas>

The <canvas> tag gives HTML a bitmapped surface to work with, much like what you would use with GDI+ or the .NET Image object. While <canvas> isn't

perfect (layers need to be replicated by using multiple canvas objects stacked on top of each other, for example), it is a great way to build charts and graphs, which have been a traditional weak spot in HTML, as well as custom graphics.

<header> and <footer>

The <header> and <footer> tags are two of the new semantic tags available. These two tags do not get you anything above and beyond for the actual display. But they will reap long-term rewards for your search engine efforts, since the search engines will be able to tell the difference between "content" and things that are important to the user but that aren't the actual content.

<nav>

The nav-tag is used to contain navigational elements, such as the main navigation on a site or more specialized navigation like next/previous-links.

<article> and <section>

The <article> and <section> tags are two more semantic tags that will boost your search engine visibility. Articles can be composed of multiple sections, and a section can have multiple articles. Confusing? Not really. An article represents a full block of content, and a section is a piece of a bigger whole. For example, if you are looking at a blog, the front page might have a section for the listing of all the posts, and each post would be an article with a section for the actual post and another for comments.

<aside>

The aside tag is used to wrap around content related to the main content of the page that could still stand on it's own and make sense.

<output>

The new <output> tag is unique, in that it expects its content to be generated dynamically with JavaScript. It has a value attribute, which can be manipulated through the DOM with JavaScript to change what is displayed on the screen. This is much more convenient than the current ways of doing things.

<details>

It seems like every Web site needs to have an expanding/collapsing block of text. While this is easy enough to do with JavaScript or server-side code, the <details> tag makes it even easier. It does exactly what all have been doing for years now: makes a simple block that expands and collapses the content when the header is clicked. The <details> tag does not have widespread support yet, but it will soon.

<figure> and <figcaption>

<figure> is a container for content (typically images, but it can be anything), and <figcaption> (which gets put inside the <figure> tag) provides a caption or subtitle for the contents of the <figure> tag. For example, you could have four images representing charts of sales growth within a <figure> tag, and a <figcaption> with text like "Year-to-year sales growth, 1989 – 1993." The images would be shown next to each other with the text running below all four.

<progress>and <meter>

<progress> and <meter> are similar. You use <progress> for a task or a "measure how complete something is" scenario. It also has an indeterminate mode for something that has an unknown duration (like searching a database). The <meter> tag is for gauges and measurements of value (thermometers, quantity used, etc.). While they may look alike on the screen in many cases, they do have different semantic meanings.

<datalist>

The <datalist> tag acts like a combo box, where the system provides a pre-made list of suggestions, but users are free to type in their own input as well. There are tons of possible uses for this, such as a search box pre-populated with items based on the user's history. This is another one of those things that currently requires a bunch of JavaScript (or JavaScript libraries) to handle but that can be done natively with HTML5.

CODE: To add multiple semantic elements in a single web page.

```
<body>
    <section>
        <header>
            <h1>Header in Section1</h1>
        </header>
        <footer>
            <h1>Footer in Section1</h1>
        </footer>
    </section>

    <section>
        <header>
            <h1>Header in Section2</h1>
        </header>
        <footer>
            <h1>Footer in Section2</h1>
        </footer>
    </section>
</body>
```

Results:

Header in Section1

Footer in Section1

Header in Section2

Footer in Section2

CODE: Create navigation links using Semantic <nav> elements

The <nav> element will allow us to group together a list of navigation links in a semantic manner.

A typical navigation link section would have looked like this before.

```
<div id="nav">
    <ul>
        <li>
            <a href="html5.htm">HTML5</a> |
            <a href="css.htm">CSS</a> |
            <a href="sql.htm">SQL</a>
        </li>
    </ul>
</div>
<!-- Now we can encapsulate a group of links in a single semantic element and it looks organized. -->
<section>
    <nav>
        <a href="html5.htm">HTML5</a> |
        <a href="css.htm">CSS</a> |
        <a href="sql.htm">SQL</a>
    </nav>
</section>
```

Results:

- HTML5 | CSS | SQL

HTML5 | CSS | SQL

UNIT 1: HTML, CSS & Client Side Scripting

Cascading Style Sheets

CSS stands for Cascading Style Sheets and it is the language used to style the visual presentation of web pages. CSS is the language that tells web browsers how to render the different parts of a web page.

Every item or element on a web page is part of a document written in a mark-up language. In most cases, HTML is the mark-up language, but there are other languages in use such as XML. CSS is the language that defines the presentation of a web page. It is used to add colour, background images, and textures, and to arrange elements on the page. It is also used to enhance the usability of a website. The style sheet standard supported by modern browsers is called cascading style sheets, or CSS. CSS files contain a set of rules for the formatting of HTML documents.

```
<html>
    <head>
        <title> Example on style sheets</title>
        <style>
         BODY {
    font-family : "times new roman";
    margin-left : 20%;
    margin-right: 20%;
    text-align : justify;
     background : ivory;
    color : black; }
    P { text-indent : 2cm; }
    </style>
    </head>
```

A style sheet is a collection of rules that describe the format of the HTML tags. In the above example there are six rules describing the format of the BODY tag, and one rule for the P tag.

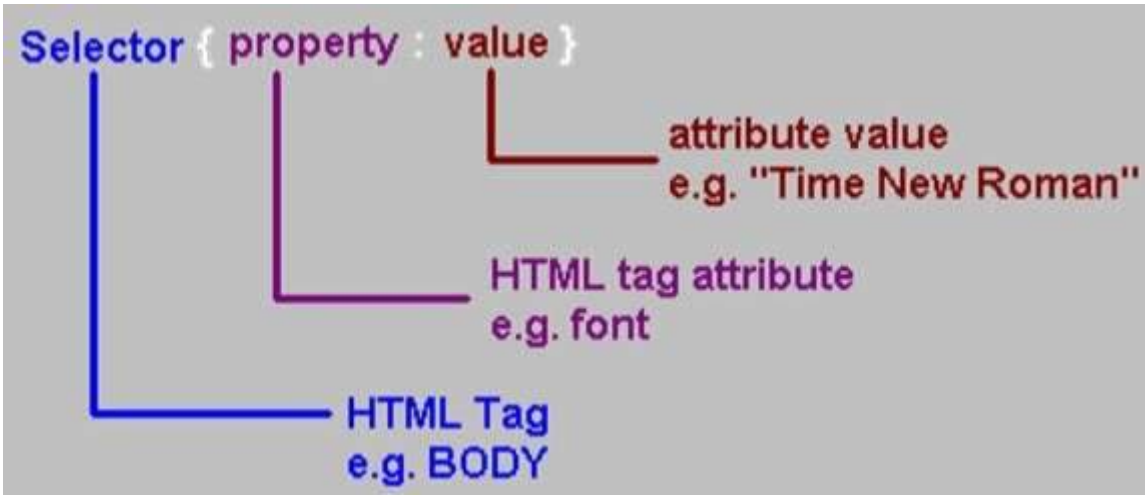
There are two ways to write style sheets:

the technically easier rule-based approach, and an approach that procedurally constructs a style sheet.

Below is a description of the rules used in the above example.

- Body - bgcolor set to "ivory", left and right margins indented relatively by 20%, font set to "Times New Roman" and text colour set to black.
- P to indent the first line by and absolute value of two centimetres.

There are three parts to a style sheet rule, shown in the figure below



The selector represents the HTML element that you want to style. For example:

h1 { color: blue }

This code tells the browser to render all occurrences of the HTML <h1> element in blue.

Grouping Selectors

You can apply a style to many selectors if you like. Just separate the selectors with a comma.

h1, h2, h3, h4, h5, h6 { color: blue }

Applying Multiple Properties

To apply more than one property separate each declaration with a semi-colon.

h1 { color:blue; font-family: arial, helvetica, "sans serif" }

Note:

CSS are a little different than their HTML counterparts. Instead of the <!-----> syntax, CSS ignores everything between /* and */ characters.

There are three ways to apply the above CSS rules to an HTML file:

1. Inline styles: add them in-line to the HTML file itself, attached directly to the relevant HTML tag.
2. Internal stylesheets : embed the rules into the HTML file
3. External stylesheets : link the CSS file to the HTML file

In-line Styles

In-Line styles are added to individual tags and are usually avoided. Like the FONT tag they clog up HTML documents, making them larger and increasing their download times.

An example of an in-line style is given below:

```
<P style="text-indent: 1cm; color: darkred;">
    This paragraph has been formatted <br>
    using the in-line style command.
</P>
<P> This paragraph has not been formatted using the in-line style command. </P>
```

Result:

This paragraph has been formatted
using the in-line style command.

This paragraph has not been formatted using the in-line style command.

Internal Stylesheets

An internal stylesheet is a block of CSS added to an HTML document head element. The style element is used between the opening and closing head tags, and all CSS declarations are added between the style tags.

```
<head>
    <style>
        h1 {
            color: red;
            padding: 10px;
            text-decoration: underline;
            }
    </style>
</head>
<body>
    <h1>Example for Internal Stylesheets</h1>
</body>
```

Results:

External Stylesheets

External stylesheets are documents containing nothing other than CSS statements. The rules defined in the document are linked to one or more HTML documents by using the link tag within the head element of the HTML document.

To use an external stylesheet, first create the CSS document and link it to an HTML document using the link element.

When this HTML document is loaded the link tag will cause the styles in the file styles.css to be loaded into the web page. As a result, all level 1 heading

elements will appear with red text, underlined, and with 10 pixels of padding applied to every side.

```
h1 {
    color: rgb(168, 24, 224);
    padding: 10px;
    text-indent: 2cm;
    }
/* save file as .css */
```

```
<head>
<link rel="stylesheet" type="text/css" href="external.css">
</head>
<body>
    <h1>Example for External Style sheet</h1>
</body>
```

Results:

The cascading style sheet standard supplies very powerful tools to control Web page formatting. For instance, consider a university with many departments each with their own individual design criteria that is producing a website. It is possible to create a hierarchy of style sheets that allows each department's website to maintain formatting consistency with all the other university sites, while allowing each department to deviate from the format where needed.

The hierarchical (cascading) structure of style sheets can be used to do this. The figure below illustrates the style hierarchy design by W3C.

The Style Element

The <style> element is a "metadata" type element, which means its purpose is to provide setup for how the rest of the document will be displayed. This element is used to embed style declarations within our HTML document, rather than linking to an external dedicated stylesheet. It is essentially a form of inline styling.

In addition to the global HTML attributes, the style element accepts the following attributes -

Attribute	Description	Default
type	A valid MIME type that specifies the language for the style	"text/css"
media	A valid media query	"all"
title	Specifies a title for an alternate style element	None
scoped*	Allows us to place a **<style>** element within the body and limit the scope of the styles to the parent element	none/false

By default, most browsers apply a single line of styling to the <style> element -

```
style {
  display: none;
}
```

A simple stylesheet :

In the following example, apply a very simple stylesheet to a document:

```
<!doctype html>
<html>
<head>
  <style>
    p {
      color: rgb(9, 39, 207);
    }
  </style>
</head>
<body>
  <p>This is my paragraph.</p>
</body>
</html
```

Result:

This is my paragraph.

Multiple style elements

Two <style> elements have been included in the example let's see how the conflicting declarations in the later <style> element override those in the earlier one, if they have equal specificity.

```
<!doctype html>
<html>
<head>
  <style>
    p {
      color: white;
      background-color: blue;
      padding: 5px;
      border: 1px solid black;
    }
  </style>
  <style>
    p {
      color: blue;
      background-color: yellow;
    }
  </style>
</head>
<body>
  <p>This is my paragraph.</p>
</body>
</html>
```

Results:

This is my paragraph.

MIME type

MIME stands for "Multipurpose Internet Mail Extensions. It's a way of identifying files on the Internet according to their nature and format. For example, using the "Content-type" header value defined in a HTTP response, the browser can open the file with the proper extension/plug-in.

A "Content-type" is simply a header defined in many protocols, such as HTTP, that makes use of MIME types to specify the nature of the file currently being handled.

Simplest MIME type consists of a type and a subtype; these are each strings which, when concatenated with a slash (/) between them, comprise a MIME type. No whitespace is allowed in a MIME type:

type/subtype

The type represents the general category into which the data type falls, such as video or text. The subtype identifies the exact kind of data of the specified type the MIME type represents. For example, for the MIME type text, the subtype might be plain (plain text), html (HTML source code), or calendar (for iCalendar/.ics) files.

Each type has its own set of possible subtypes, and a MIME type always has both a type and a subtype, never just one or the other.

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, each of which may have its own individual MIME type; or, a multipart type may encapsulate multiple files being sent together in one transaction. For example, multipart MIME types are used when attaching multiple files to an email.

text/plain

This is the default for textual files. Even if it really means "unknown textual file," browsers assume they can display it.

text/css

CSS files used to style a Web page must be sent with text/css. If a server doesn't recognize the .css suffix for CSS files, it may send them with text/plain or application/octet-stream MIME types.

text/html
All HTML content should be served with this type

image/jpeg
JPEG images

image/png
PNG images

Cascading in CSS

CSS: The class selector
The class selector is a way to select all of the elements with the specified class name, and apply styles to each of the matching elements. Declare a CSS class by using a dot (.) followed by the class name. You make up the class name yourself. After the class name you simply enter the properties/values that you want to assign to your class. The browser will look for all tags in the page that have a class attribute containing that class name.

.class-name { property:value; }

If you want to use the same class name for multiple elements, but each with a different style, you can prefix the dot with the HTML element name.

For example:

p.large { font-size: 2em; }

The class name can't contain a space, but it can contain hyphens or underscores. Any tag can have multiple space-separated class names.

```
<!DOCTYPE html>
<title>Example</title>
<style>
  div.css-section {
    border:2px dotted red;
    padding: 10px;
    }
  div.css-section p {
    color: green;
    }
</style>
<div class="css-section">CSS Class
  <p>CSS classes can be very useful</p>
</div>
```

Result:

CSS Class

CSS classes can be very useful

CSS ID Selectors

The id selector is a way to select only the element with the specified id, and apply styles to that element. The selector must start with a pound sign (#) and then the value of the id attribute. The browser will then look for a tag in the page that has an id attribute equal to that id.

The spelling and the casing must be exactly the same - #soon_gone is different from #Soon_Gone. The page should not have multiple tags with the same id- every id should be unique.

#id-name { property:value; }

Again, similar to classes, if you want to use the same id name for multiple elements, but each with a different style, you can prefix the hash with the HTML element name.

p#intro { font-size: 2em; }

```
<!DOCTYPE html>
<title>Example</title>
<style>
  div#css-section {
    /* border:1px dotted red; */
    border: 4mm ridge rgba(170, 50, 220, .6);
    /* border-width: 1px 2em 0 1rem; */
    padding: 20px;
    }
</style>
<div id="css-section">
  This lucky div has ID...
</div>
<div>
  This poor div has no ID...
</div>
```

Result:

Note: ID applies to EVERY single thing with that specified element, while class applies only to the classes you gave to everything, so ID would make every h2 purple in ID for example, but class would only make the one h2 you gave the class to purple.

CSS: The element selector

The element selector is a way to select all the elements with a given tag name in a document, and apply the same styles to each element with the tag name.

Note that you should only write the tag name, and not write brackets around the tag name — h1, not <h1>.

```
    <title>CSS: The element selector</title>
    <style>
        body {
            background-color: rgb(10, 204, 178);
        }

        h1 {
            color: white;
        }

        p {
            color: gold;
            font-size: larger;
        }
    </style>
</head>
<body>
    <h1>Example for element selector</h1>

    <p> element selector</p>

    <p>Another selector</p>
</body>
</html>
```

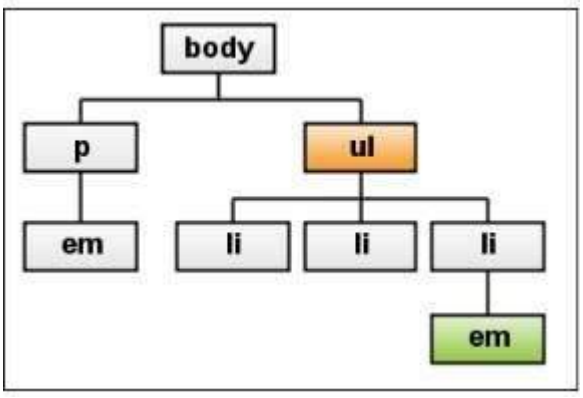
Results:

CSS: The descendant selector

The descendant selector is a way to select elements that are located somewhere underneath other elements, according to the tree structure of the webpage. This selector is actually multiple selectors combined together, but separated by a space. The first selector is the "ancestor", the element(s) that must be located higher in the tree, and the second selector is the "descendant", the element(s) that must be located somewhere underneath the ancestor in the tree.

To understand ancestor/descendant relationships, you need to think about the webpage is a tree.

An example tree structure is shown in the image to the right. The ul is an ancestor of the em below it, but it is not an ancestor of the other em. That means that a ul em selector will only select a single em, not both of them.



```
    <title>CSS: The descendant selector</title>
    <style>
        /* Makes all em tags underneath a ul have a red background */
        ul em {
            background-color: rgb(255, 182, 182);
        }
        /* Makes all strong tags underneath a ul have a violet background */
        ul strong {
            background-color: rgb(237, 221, 255);
        }
    </style>
</head>
<body>

    <h2>The three laws of robotics</h2>

    <p>These laws, also known as <strong>Asimov's Laws</strong> were originally formulated
        by science-fiction author Isaac Asimov in the 1940s, but they're now referred to in movies
        <em>and</em> the news.</p>
    <ul>
        <li> A robot may not injure a human being or, through inaction,
            allow a human being to come to harm.</li>
        <li>A robot must obey the orders given to it by human beings,
            <em>except</em> where such orders would conflict with the First Law.</li>
        <li>A robot must protect its own existence <em>as long as</em>
            such protection <strong>does not conflict</strong> with the First or Second Law.</li>
    </ul>
```

Result:

The three laws of robotics

These laws, also known as **Asimov's Laws** were originally formulated by science-fiction author Isaac Asimov in the 1940s, but they're now referred to in movies *and* the news.

- A robot may not injure a human being or, through inaction, allow a human being to come to harm.
- A robot must obey the orders given to it by human beings, *except* where such orders would conflict with the First Law.
- A robot must protect its own existence *as long as* such protection **does not conflict** with the First or Second Law.

Pseudo-classes

A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected element(s). For example, ***:hover*** can be used to change a button's color when the user's pointer hovers over it.

/* Any button over which the user's pointer is hovering */

```
button:hover {
color: blue;
}
```

Pseudo-classes let you apply a style to an element not only in relation to the content of the document tree, but also in relation to external factors like the history of the navigator (:visited, for example), the status of its content (like :checked on certain form elements), or the position of the mouse (like :hover, which lets you know if the mouse is over an element or not). A few common pseudo-classes are :link, :visited, :hover, :active, :first-child and :nth-child.

CSS pseudo-classes and pseudo-elements can certainly be a handful.

Syntax:

```
selector:pseudo-class {
 property:value;
}
```

Single Or Double Colon For Pseudo-Elements?

The double colon (::) was introduced in CSS3 to differentiate pseudo-elements such as ::before and ::after from pseudo-classes such as :hover and :active. All browsers support double colons for pseudo-elements except Internet Explorer (IE) 8 and below.

Some pseudo-elements, such as ::backdrop, accept only a double colon, though.

Dynamic pseudo-classes

These are the link-related pseudo-class states which were included in CSS1. Each of these states can be applied to an element, usually <a>. They include;

:link - This only selects <a> tags with href attributes. It will not work if it is missing.

:active - Selects the link while it is being activated (being clicked on or otherwise activated). For example, for the “pressed” state of a button-style link.

:visited - Selects links that have already been visited by the current browser.

:hover - This is the most commonly used state. When the mouse cursor rolls over a link, that link is in its hover state and this will select it.

Referring to index.html, it can like to change the background of <li> when hovered, give specific colours to all links, active and visited links.

```
.list-item:hover {
  background-color: aliceblue;
}

.list a:link{
  color: black;
}

.list a:active{
  color: green;
 }

.list a:visited{
  color: red;
 }
```

Structural pseudo-classes

These exciting positioning states/selectors were introduced in CSS2. They target elements according to their position in the document tree and relation to other elements. They include

root - This selects the element that is at the root of the document specifically the <html> element unless you are specifically working in some other environment that somehow also allows CSS.

:first-child - Selects the first element within a parent.

:last-child - Selects the last element within a parent.

:nth-child() - Selects elements based on a provided algebraic expression (e.g. "2n" or "4n-1"). For example, you could use '2n' for selecting even positions and '2n-1' for odd positions. Has the ability to do other things like select "every fourth element", "the first six elements", and things like that.

:first-of-type - Selects the first element of this type within any parent. If for example, you have two divs, each with a paragraph, link, paragraph, link. Then div a:first-of-type would select the first link inside the first div and the first link inside the second div.

:*last-of-type* - This works the same as above but it then selects the last element instead of the first element.

:nth-of-type() - Works like :nth-child, but it is used in places where the elements at the same level are of different types. For example, if inside a div you had a number of paragraphs and links. You wanted to select all the odd paragraphs. :nth-child wouldn't work in this scenario, therefore, you use div p:nth-of-type(odd).

:only-of-type - Selects the element if and only if it is one of its kind within the current parent.

Method one: Using the :nth-child()

```
.list-item:nth-child(2n-1){
 background-color: slategrey;
}
```

Method two: Using the :nth-of-type()

```
.list-item:nth-of-type(odd){
 background-color: slategrey;
}
```

Pseudo-Elements

Content-related pseudo-elements effectively create new elements that are not specified in the mark-up of the document and can be manipulated much like a regular element. This introduces huge benefits for creating cool effects with minimal mark-up, also aiding significantly in keeping the presentation of the document out of the HTML and in CSS where it belongs.

Difference between Pseudo-classes and Pseudo-elements

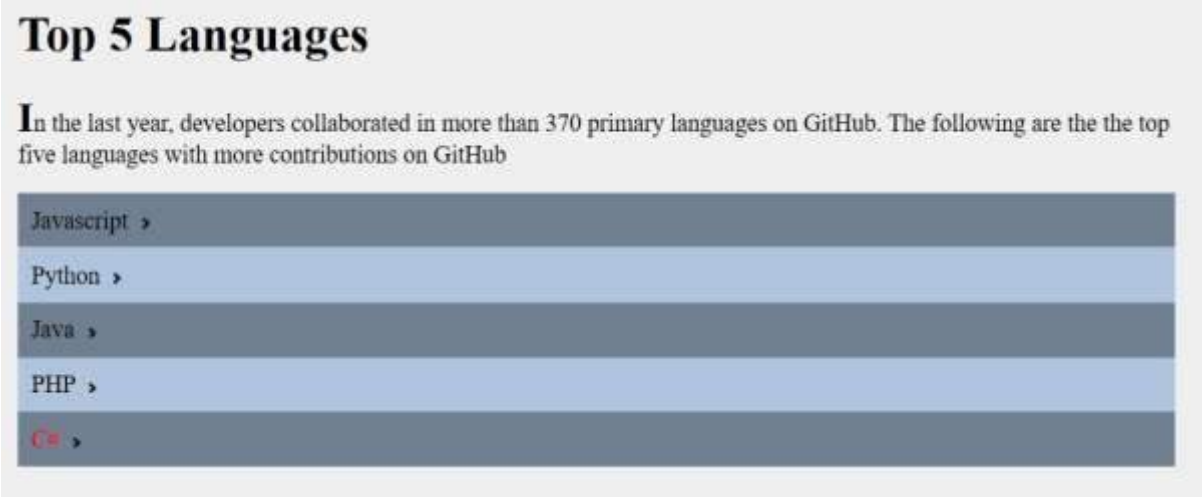
A pseudo-class is a selector that assists in the selection of something that cannot be expressed by a simple selector, for example: hover. A pseudo-element, however, allows us to create items that do not normally exist in the document tree, for example ::after. So you could simply identify a pseudo-class by a single colon (:) and a pseudo-element by two colons (::).

Pseudo-elements include

::before - This enables us to add content before a certain element. For example, adding an opening quote before a blockquote.

::after - This enables us to add content after a certain element. For example, a closing quotes to a blockquote. Also used commonly for the clearfix, where an empty space is added after the element which clears the float without any need for extra HTML mark-up.

::first-letter - This is used to add a style to the first letter of the specified selector. For example, to create a drop cap.



Font and Spacing

CSS has a lot of properties. CSS can do so many things, but often the font and spacing are overlooked. Common font related properties:

- color – sets the font color
- font-size – sets the font size
- font-weight – sets the font weight (skinny to very bold)
- text-align – sets the font position inside element (left, centre, right, justify)
- font-family – sets the actual font type

```
<style type="text/css">
p.special_one {
color: rgb(65, 24, 216);
font-size:3em;
font-weight:100;
text-align:center;
font-family:"Arial", "Myriad Web Pro", Arial, serif;
}
</style>
<p class="special_one"> Sample CSS Demo for font</p>
```

Results:

Sample CSS Demo for font

Text Spacing

Text spacing, refers to characters in relation to other characters. The first important property is letter-spacing, which is the horizontal spacing between characters. The second and final is the line-height that is the vertical spacing between text lines.

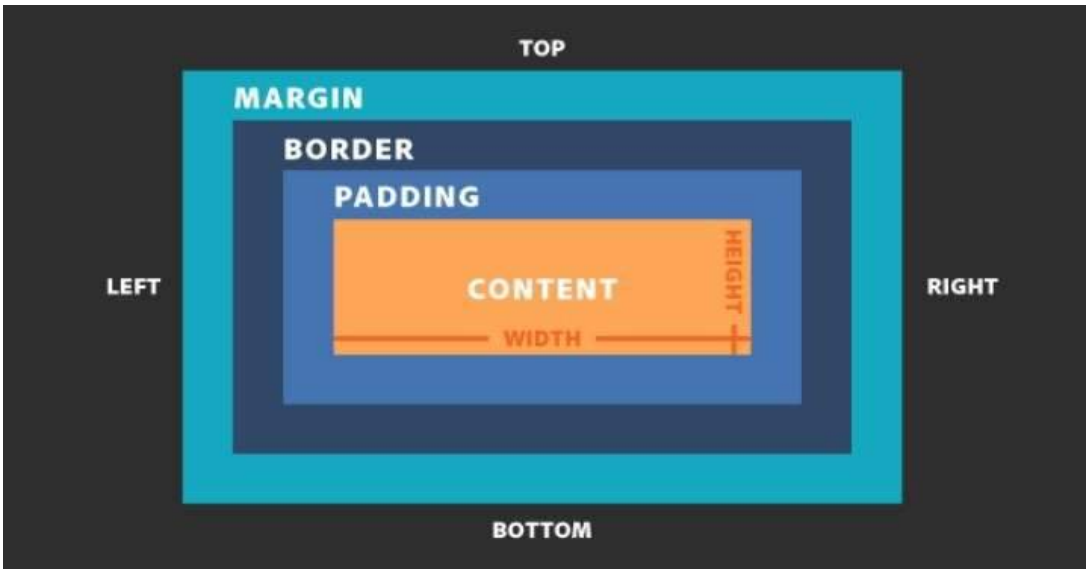
```
<style type="text/css">
p.special_two {
letter-spacing:.1em;
line-height:2em;
}
</style>
<p class="special_two">We will get to the idea of block spacing and margin spacing,<br>
but for now we will only speak of them to have default text.</p>
```

Results:

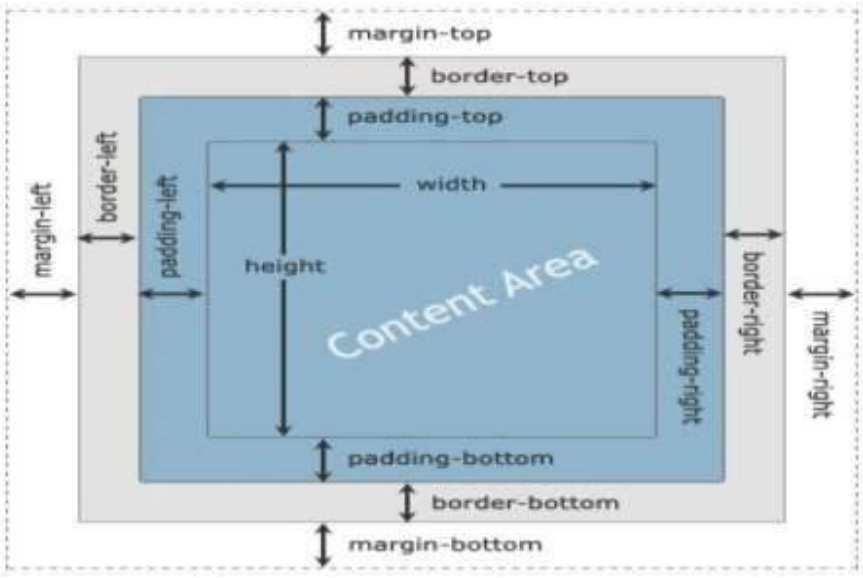
We will get to the idea of block spacing and margin spacing,
but for now we will only speak of them to have default text.

CSS box model

When laying out a document, the browser's rendering engine represents each element as a rectangular box according to the standard CSS basic box model. CSS determines the size, position, and properties (color, background, border size, etc.) of these boxes.



Every box is composed of four parts (or areas), defined by their respective edges: the content edge, padding edge, border edge, and margin edge. Every box has a content area and an optional surrounding margin, padding, and border.

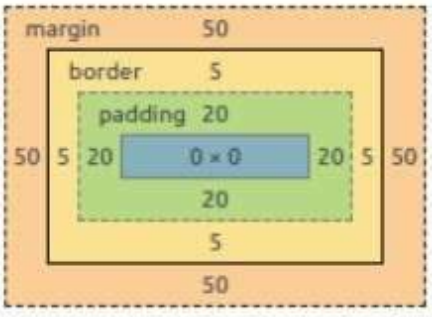


1. The innermost rectangle is the content box. The width and height of this depend on the element's content (text, images, videos, any child elements).
2. Then have the padding box (defined by the padding property). If there is no padding width defined, the padding edge is equal to the content edge.
3. Next, the border box (defined by the border property). If there is no border width defined, the border edge is equal to the padding edge.
4. The outermost rectangle is the margin box. If there is no margin width defined, the margin edge is equal to the border edge.

Consider the following example:

```
div{
 border: 5px solid;
 margin: 50px;
 padding: 20px;
}
```

This CSS styles all div elements to have a top, right, bottom and left border of 5px in width and a top, right, bottom and left margin of 50px; and a top, right, bottom, and left padding of 20px. Ignoring content, our generated box will look like this:



CSS Position

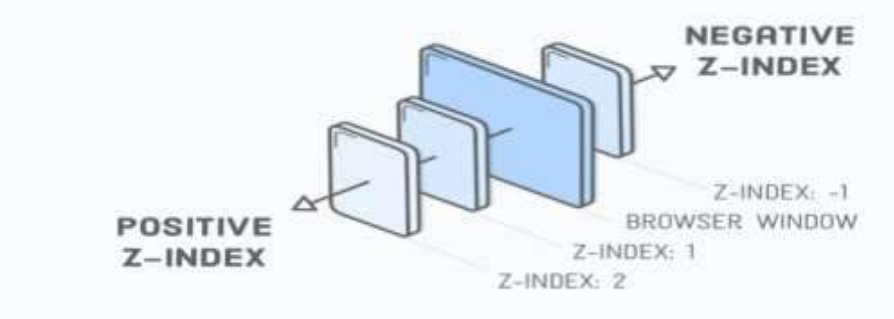
The position CSS property sets how an element is positioned in a document. The top, right, bottom, and left properties determine the final location of positioned elements. There are 5 main values of the Position Property:

position: static , relative , absolute , fixed , sticky

and additional properties for setting the coordinates of an element : **top , right , bottom , left AND the z-index.**

What is this z-index?

Consider height and width (x, y) as 2 dimensions. Z is the 3rd dimension. An element in the webpage comes in front of other elements as its z-index value increases. Z-index doesn't work with position: static or without a declared position.



The .features-menu element needs to have a lower z-index than the Features label. The default z-index value is 0, so make both of them higher than that. It can conveniently wrapped the Features label in a , allowing to style it via a child selector, like this

```
.dropdown > span {
  z-index: 2;
  position: relative; /* This is important! */
  cursor: pointer;
}

.features-menu {
  /* ... */
  z-index: 1;
}
```

The Features label appear on top of the submenu. The position: relative; line. It's required because only positioned elements pay attention to their z-index property.

Now, lets see the Position Property:

1. Static

position: static is the default value. Whether declare it or not, elements are positioned in a normal order on the webpage. Let's give an example:

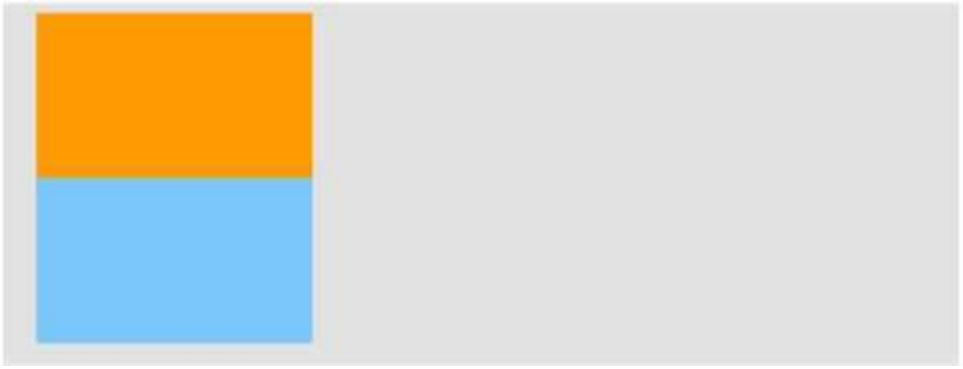
First, define our HTML structure:

```
<body>
    <div class="box-orange"></div>
    <div class="box-blue"></div>
</body>
```

Then, create 2 boxes and define their widths, heights & positions:

```
.box-orange {          // without any position declaration
    background: orange;
    height: 100px;
    width: 100px;
}

.box-blue {
    background: lightskyblue;
    height: 100px;
    width: 100px;
    position: static;    // Declared as static
}
```

Results:



Hence from the result it can observe defining position: static or not doesn't make any difference. The boxes are positioned according to the normal document flow.

2. Relative

position: relative: An element's new position relative to its normal position.

Starting with position: relative and for all non-static position values, able to change an element's default position by using the helper properties. Move the orange box next to the blue one. Orange box is moved 100px to bottom & right, relative to its normal position.

```
.box-orange {
  position: relative;  // We are now ready to move the element
  background: orange;
  width: 100px;
  height: 100px;
  top: 100px;          // 100px from top relative to its old position
  left: 100px;         // 100px from left
}
```

Results:

3.Absolute

In position: relative, the element is positioned relative to itself. However, an absolutely positioned element is relative to its parent.

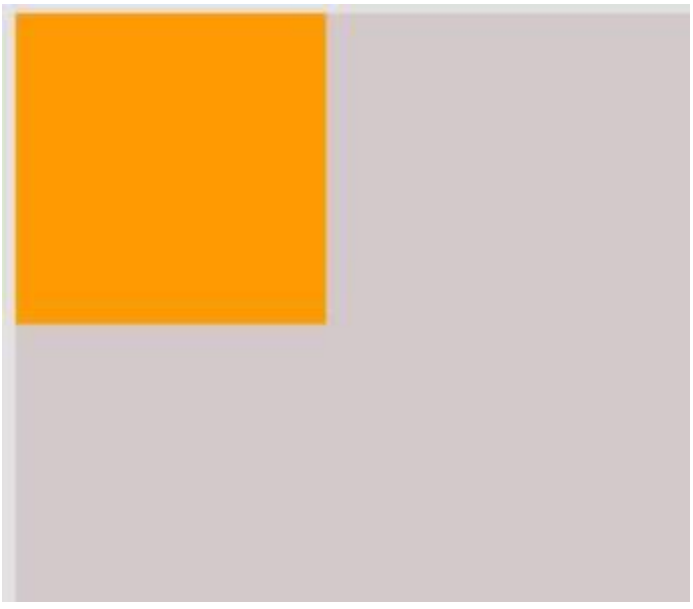
An element with position: absolute is removed from the normal document flow. It is positioned automatically to the starting point (top-left corner) of its parent element. If it doesn't have any parent elements, then the initial document <html> will be its parent.

Since position: absolute removes the element from the document flow, other elements are affected and behave as the element is removed completely from the webpage. Add a container as parent element.

```
<body>
  <div class="container">
    <div class="box-orange"></div>
    <div class="box-blue"></div>
  </div>
</body>
```

```
.box-orange {
  position: absolute;
  background: orange;
  width: 100px;
  height: 100px;
}
```

Results:



It looks like the blue box has disappeared, but it hasn't. The blue box behaves like the orange box is removed, so it shifts up to the orange box's place. So Let's move the orange box 5 pixels:

```
.box-orange {
  position: absolute;
  background: orange;
  width: 100px;
  height: 100px;
  left: 5px;
  top: 5px;
}
```

Results:

The coordinates of an absolute positioned element are relative to its parent if the parent also has a non-static position.

4.Fixed

Like position: absolute, fixed positioned elements are also removed from the normal document flow. The differences are:

- They are only relative to the <html> document, not any other parents.
- They are not affected by scrolling.

```
.container {
  position: relative;
  background: lightgray;
}

.box-orange {
  position: fixed;
  background: orange;
  width: 100px;
  height: 100px;
  right: 5px;     // 5px relative to the most-right of parent
}
```

Here in the example, the orange box's position is changed to fixed, and this time it is relative 5px to the right of the <html>, not its parent (container):

Result :

5.position: sticky

It can be explained as a mix of position: relative and position: fixed.It behaves until a declared point like position: relative, after that it changes its behaviour to position: fixed .

```
<html>
    <head>
        <title>Example Position: sticky</title>
    </head>
  <body>
    <div class="container">
        <div class="box-orange"></div>
        <div class="box-blue"></div>
      <p>Scroll down the page</p>
      <p class="sticky">I am sticky</p>
      </div>
  </body>
</html>
```

```
.container {
  position: relative;
  background: lightgray;
  width: 50%;
  margin: 0 auto;
  height: 1000px;
}
.container p {
  text-align: center;
  font-size: 20px;
}
.box-orange {
  background: orange;
  width: 100px;
  height: 100px;
  position: fixed;
  right: 5px;
}
.box-blue {
  background: lightblue;
  width: 100px;
  height: 100px;
}
.sticky {
  position: sticky;
  background: red;
  top: 0;
  padding: 10px;
  color: white;
}
```

Results:

Background-image:

Background-image defines a pointer to an image resource which is to be placed in the background of an element.

Syntax

object.style.backgroundImage = "Any value as defined above";

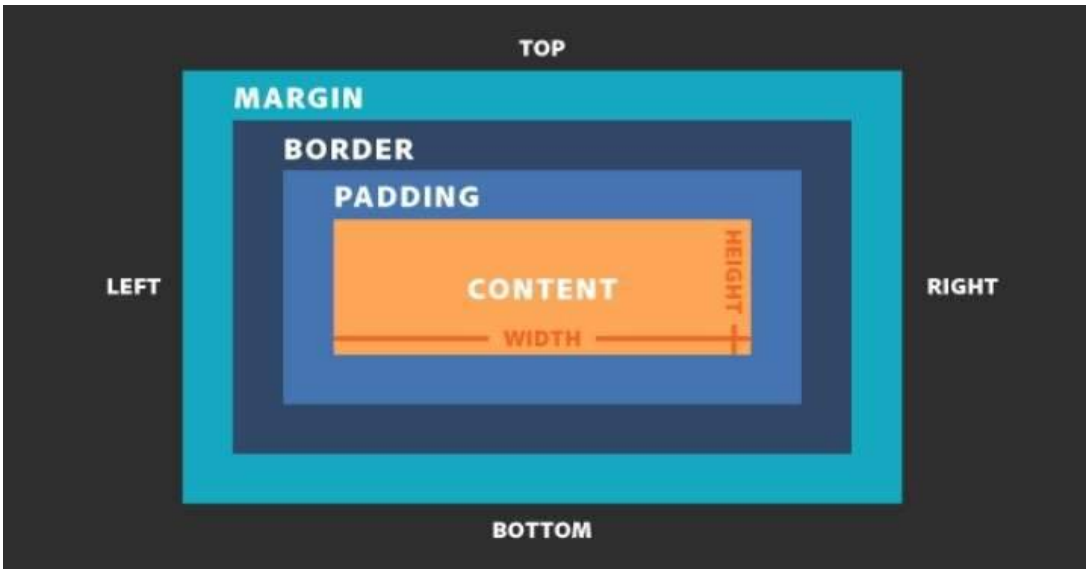
```
<html>
    <head>
        <style>
            body {
                background-image: url('purple.jpg');
                background-color: #cccccc;
            }
        </style>
    </head>

    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Relts:

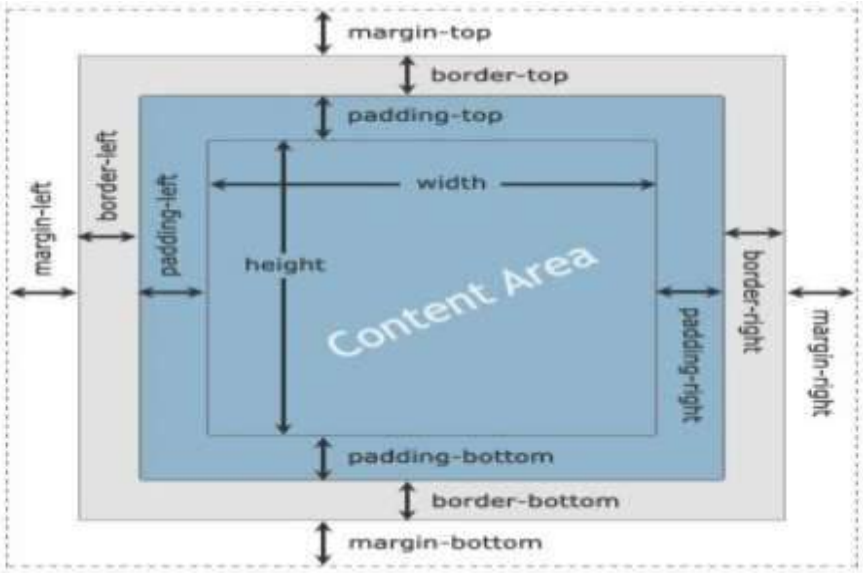
UNIT 1: HTML, CSS & Client Side Scripting

CSS box model

When laying out a document, the browser's rendering engine represents each element as a rectangular box according to the standard CSS basic box model. CSS determines the size, position, and properties (color, background, border size, etc.) of these boxes.



Every box is composed of four parts (or areas), defined by their respective edges: the content edge, padding edge, border edge, and margin edge. Every box has a content area and an optional surrounding margin, padding, and border.

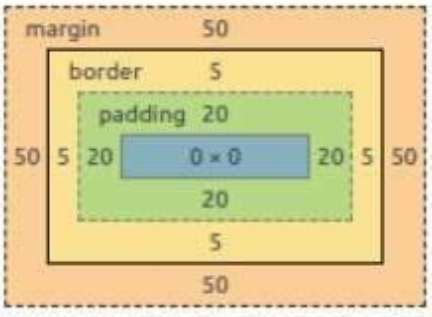


1. The innermost rectangle is the content box. The width and height of this depend on the element's content (text, images, videos, any child elements).
2. Then have the padding box (defined by the padding property). If there is no padding width defined, the padding edge is equal to the content edge.
3. Next, the border box (defined by the border property). If there is no border width defined, the border edge is equal to the padding edge.
4. The outermost rectangle is the margin box. If there is no margin width defined, the margin edge is equal to the border edge.

Consider the following example:

```
div{
 border: 5px solid;
 margin: 50px;
 padding: 20px;
}
```

This CSS styles all div elements to have a top, right, bottom and left border of 5px in width and a top, right, bottom and left margin of 50px; and a top, right, bottom, and left padding of 20px. Ignoring content, our generated box will look like this:



CSS Position

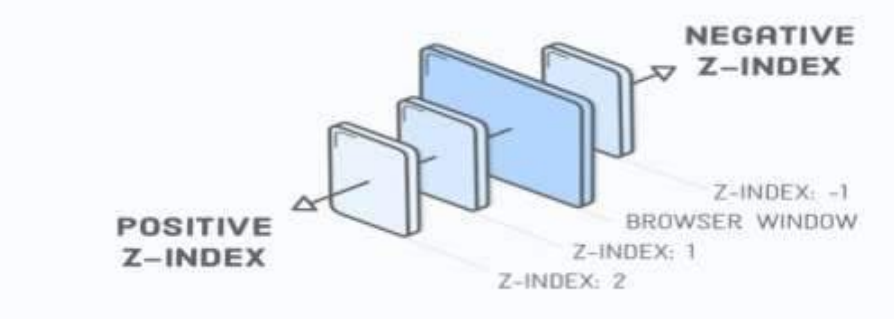
The position CSS property sets how an element is positioned in a document. The top, right, bottom, and left properties determine the final location of positioned elements. There are 5 main values of the Position Property:

position: static , relative , absolute , fixed , sticky

and additional properties for setting the coordinates of an element : **top , right , bottom , left AND the z-index.**

What is this z-index?

Consider height and width (x, y) as 2 dimensions. Z is the 3rd dimension. An element in the webpage comes in front of other elements as its z-index value increases. Z-index doesn't work with position: static or without a declared position.



The .features-menu element needs to have a lower z-index than the Features label. The default z-index value is 0, so make both of them higher than that. It can conveniently wrapped the Features label in a , allowing to style it via a child selector, like this

```
.dropdown > span {
  z-index: 2;
  position: relative;  /* This is important! */
  cursor: pointer;
}

.features-menu {
  /* ... */
  z-index: 1;
}
```

The Features label appear on top of the submenu. The position: relative; line. It's required because only positioned elements pay attention to their z-index property.

Now, lets see the Position Property:

1. Static

position: static is the default value. Whether declare it or not, elements are positioned in a normal order on the webpage. Let's give an example:

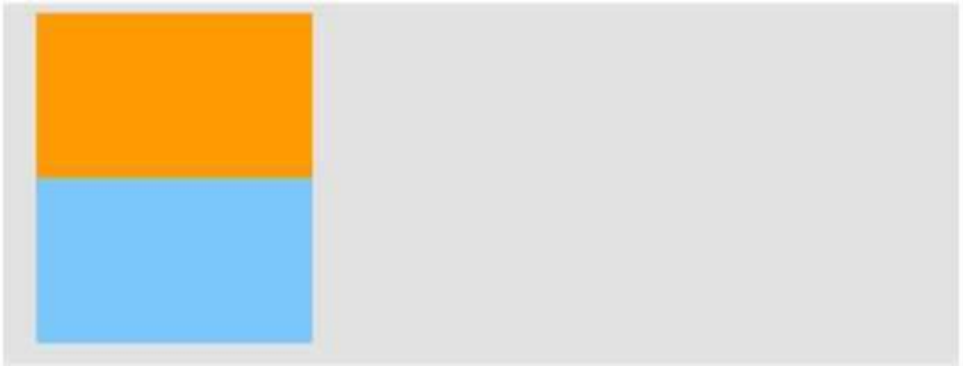
First, define our HTML structure:

```
<body>
    <div class="box-orange"></div>
    <div class="box-blue"></div>
</body>
```

Then, create 2 boxes and define their widths, heights & positions:

```
.box-orange {          // without any position declaration
    background: orange;
    height: 100px;
    width: 100px;
}

.box-blue {
    background: lightskyblue;
    height: 100px;
    width: 100px;
    position: static;    // Declared as static
}
```

Results:



Hence from the result it can observe defining position: static or not doesn't make any difference. The boxes are positioned according to the normal document flow.

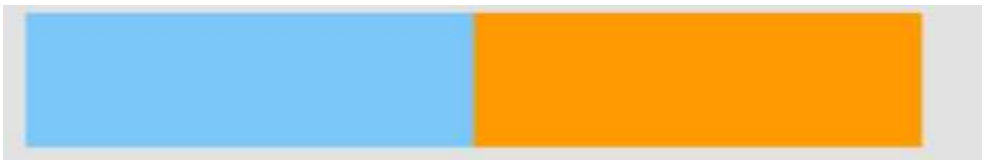
2. Relative

position: relative: An element's new position relative to its normal position.

Starting with position: relative and for all non-static position values, able to change an element's default position by using the helper properties. Move the orange box next to the blue one. Orange box is moved 100px to bottom & right, relative to its normal position.

```
.box-orange {
  position: relative;  // We are now ready to move the element
  background: orange;
  width: 100px;
  height: 100px;
  top: 100px;          // 100px from top relative to its old position
  left: 100px;         // 100px from left
}
```

Results:



3.Absolute

In position: relative, the element is positioned relative to itself. However, an absolutely positioned element is relative to its parent.

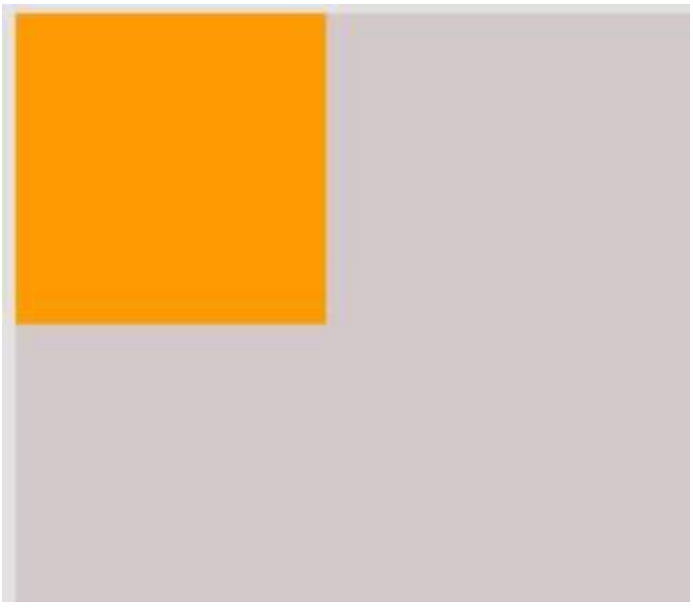
An element with position: absolute is removed from the normal document flow. It is positioned automatically to the starting point (top-left corner) of its parent element. If it doesn't have any parent elements, then the initial document <html> will be its parent.

Since position: absolute removes the element from the document flow, other elements are affected and behave as the element is removed completely from the webpage. Add a container as parent element.

```html
<body>
  <div class="container">
    <div class="box-orange"></div>
    <div class="box-blue"></div>
  </div>
</body>
```

```css
.box-orange {
  position: absolute;
  background: orange;
  width: 100px;
  height: 100px;
}
```

Results:



It looks like the blue box has disappeared, but it hasn't. The blue box behaves like the orange box is removed, so it shifts up to the orange box's place. So Let's move the orange box 5 pixels:

```
.box-orange {
  position: absolute;
  background: orange;
  width: 100px;
  height: 100px;
  left: 5px;
  top: 5px;
}
```

Results:

The coordinates of an absolute positioned element are relative to its parent if the parent also has a non-static position.

4.Fixed

Like position: absolute, fixed positioned elements are also removed from the normal document flow. The differences are:

- They are only relative to the <html> document, not any other parents.
- They are not affected by scrolling.

```css
.container {
  position: relative;
  background: lightgray;
}

.box-orange {
  position: fixed;
  background: orange;
  width: 100px;
  height: 100px;
  right: 5px;     // 5px relative to the most-right of parent
}
```

Here in the example, the orange box's position is changed to fixed, and this time it is relative 5px to the right of the <html>, not its parent (container):

Result :

5.position: sticky

It can be explained as a mix of position: relative and position: fixed.It behaves until a declared point like position: relative, after that it changes its behaviour to position: fixed .

```html
<html>
    <head>
        <title>Example Position: sticky</title>
    </head>
  <body>
    <div class="container">
        <div class="box-orange"></div>
        <div class="box-blue"></div>
      <p>Scroll down the page</p>
      <p class="sticky">I am sticky</p>
    </div>
  </body>
</html>
```

```css
.container {
  position: relative;
  background: lightgray;
  width: 50%;
  margin: 0 auto;
  height: 1000px;
}
.container p {
  text-align: center;
  font-size: 20px;
}
.box-orange {
  background: orange;
  width: 100px;
  height: 100px;
  position: fixed;
  right: 5px;
}
.box-blue {
  background: lightblue;
  width: 100px;
  height: 100px;
}
.sticky {
  position: sticky;
  background: red;
  top: 0;
  padding: 10px;
  color: white;
}
```

Results:

Background-image:

Background-image defines a pointer to an image resource which is to be placed in the background of an element.

Syntax

object.style.backgroundImage = "Any value as defined above";

```
<html>
    <head>
        <style>
            body {
                background-image: url('purple.jpg');
                background-color: #cccccc;
            }
        </style>
    </head>

    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Results:

CSS Property: border-style

The border style, combined with border width and border color, can also be specified with the border shorthand property.

With one value, the border-style property can be used to specify a uniform style border around a box. With two, three, or four values, sides can be specified independently.

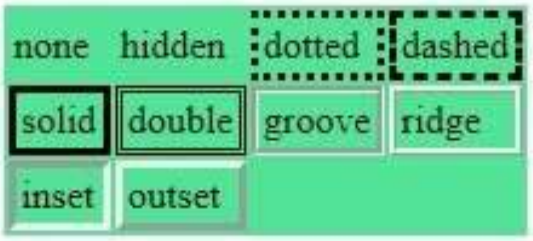
The border-style property may be specified using one, two, three, or four values.

1. When one value is specified, it applies the same style to all four sides.
2. When two values are specified, the first style applies to the top and bottom, the second to the left and right.
3. When three values are specified, the first style applies to the top, the second to the left and right, the third to the bottom.
4. When four values are specified, the styles apply to the top, right, bottom, and left in that order (clockwise).

```
<!DOCTYPE html>
<html>

    <link rel="stylesheet" type="text/css" href="broderstyle.css">
    <table>
        <tr>
            <td class="b1">none</td>
            <td class="b2">hidden</td>
            <td class="b3">dotted</td>
            <td class="b4">dashed</td>
        </tr>
        <tr>
            <td class="b5">solid</td>
            <td class="b6">double</td>
            <td class="b7">groove</td>
            <td class="b8">ridge</td>
        </tr>
        <tr>
            <td class="b9">inset</td>
            <td class="b10">outset</td>
        </tr>
    </table>
</html>
```

```css
/* Define look of the table */
table {
  border-width: 3px;
  background-color: #52E396;
}
tr, td {
  padding: 2px;
}

/* border-style example classes */
.b1 {border-style:none;}
.b2 {border-style:hidden;}
.b3 {border-style:dotted;}
.b4 {border-style:dashed;}
.b5 {border-style:solid;}
.b6 {border-style:double;}
.b7 {border-style:groove;}
.b8 {border-style:ridge;}
.b9 {border-style:inset;}
.b10 {border-style:outset;}
/* save as .css */
```

Results:



Consider the below Table for the same.

Value	Description
none	No border.
solid	Solid line.
dotted	Series of dots.
dashed	Series of dashes.
double	Two solid lines.
groove	Representation of a carved groove. Opposite of ridge.
ridge	Representation of an embossed ridge. Opposite of groove.

Rounded Corners

The border-radius CSS property rounds the corners of an element's outer border edge. You can set a single radius to make circular corners, or two radii to make elliptical corners.

This property is a shorthand to set the four properties border-top-left-radius, border-top-right-radius, border-bottom-right-radius, and border-bottom-left-radius.

The radius applies to the whole background, even if the element has no border; the exact position of the clipping is defined by the background-clip property.

The border-radius property does not apply to table elements when border-collapse is collapse.

```
<!DOCTYPE html>
<html>

<head>

    <link rel="stylesheet" href="roundstyle.css">
</head>

<body>
    <div id="roundc1"></div>
    <div id="roundc2"></div>
    <div id="roundc3"></div>
</body>

</html>


<script type="text/javascript" src="scripts.js">
</script>
```

```css
#roundc1 {
    border-radius: 10px;
    background: #5E37BC;
    padding: 15px;
    width: 50px;
    height: 50px;
}
#roundc2 {
    border-radius: 10px;
    border: 3px solid #5E37BC;
    padding: 15px;
    width: 50px;
    height: 50px;
}
#roundc3 {
    border-radius: 10px;
    background: url('purple.jpg');
    background-size: cover;
    background-repeat: repeat;
    padding: 15px;
    width: 50px;
    height: 50px;
}
```

Results:

UNIT 1: HTML, CSS & Client Side Scripting

JavaScript

JavaScript is a scripting language used to create and control dynamic website content, i.e. anything that moves, refreshes, or otherwise changes on your screen without requiring you to manually reload a web page. Features like:

- animated graphics
- photo slideshows
- auto complete text suggestions
- interactive forms

The three elements together form the backbone of web development.

1. HTML is the structure of your page—the headers, the body text, any images you want to include.
2. CSS controls how that page looks (it's what you'll use to customize fonts, background colors, etc.)
3. JavaScript is the magic third element. Once you've created your structure (HTML) and your aesthetic vibe (CSS), JavaScript makes your site or project dynamic.

JavaScript is a "scripting language." Scripting languages are coding languages used to automate processes that users would otherwise need to execute on their own, step-by-step. Short of scripting, any changes on web pages you visit would require either manually reloading the page, or navigating a series of static menus to get to the content you're after.

JavaScript Used For the following:

- Adding interactivity to websites—Used to make website to be more than a static page of text, you'll need to do some Java Scripting
- Developing mobile applications—JavaScript isn't just for websites…it's used to create those apps you have on your phone and tablet as well
- Creating web browser based games—Used for game to play directly from your web browser. JavaScript probably helped make that happen
- Back end web development—JavaScript is mostly used on the front end of things, but it's a versatile enough scripting language to be used on back end infrastructure, too.

JavaScript is either embedded into a web page or else it's included in a .js file. JavaScript is also a "client-side" language. JavaScript can be added directly to a page's code using <script> tags and giving them the type attribute text/javascript.

Including JavaScript in an HTML Page

```
CSS:
<style>
CSS goes here
</style>

JavaScript:
<script type="text/javascript">
JavaScript code goes here
</script>
```

Call an External JavaScript File

<script src="myscript.js"></script>

<code></code>

Pros and Cons of JavaScript

• Pros: –

- Allows more dynamic HTML pages,
- even complete web applications

• Cons: –

- Requires a JavaScript-enabled browser.
- Requires a client who trusts the server enough to run the code the server provides.
- JavaScript has some protection in place but can still cause security problems for clients.
- Remember JavaScript was invented in 1995 and web browsers have changed a lot since then.

Including Comments

Comments are important because they help other people understand what is going on in the code or remind if forgot something. In JavaScript there are have two different options:

- *Single-line comments* — To include a comment that is limited to a single line, precede it with //
- *Multi-line comments* — In case you want to write longer comments between several lines, wrap it in /* and */ to avoid it from being executed

Variables in JavaScript

Variables are stand-in values that you can use to perform operations. You should be familiar with them from math class.

var, const, let

You have three different possibilities for declaring a variable in JavaScript, each

with their own specialties:

1. var — The most common variable. It can be reassigned but only accessed within a function. Variables defined with var move to the top when the code is executed.

2. const — Cannot be reassigned and not accessible before they appear within the code.
3. let — Similar to const, the let variable can be reassigned but not re-declared.

Data Types

Variables can contain different types of values and data types. Use = to assign them:

- Numbers — var age = 23
- Variables — var x
- Text (strings) — var a = "init"
- Operations — var b = 1 + 2 + 3
- True or false statements — var c = true
- Constant numbers — const PI = 3.14
- Objects — var name = {firstName:"John", lastName:"Doe"}

There are more possibilities. Note that variables are case sensitive. That means *lastname* and *lastName* will be handled as two different variables.

Objects

Objects are certain kinds of variables. They are variables that can have their own values and methods. The latter are actions that you can perform on objects.

Objects are created in a number of ways, although the most common is by the

use of the new operator. This is a unary, prefix operator, which means it is placed before a single operand. The new operator should be followed by a constructor function.

```
new Object();
var accountNumbers = new Array( 5 );
var firstName = new String( "Jose" );
```

Sufficient free memory is located in which to create the new object. The new operator returns a reference to the location in memory where the new object has been created. Variables do not store objects, they store a reference to the object located elsewhere in memory.

Creating String objects

Although the following way of creating String objects is perfectly acceptable JavaScript:

var firstName = new String("Jose");

it is usually much more convenient to used the special syntax provided for creating String literal objects. The same result could be achieved with the following statement:

var firstName = "Jose";

String objects can be created in two different ways because the creation of String objects is such a common feature of programming that the developers of the JavaScript language provided this second, simpler syntax especially for Strings.

Object properties and methods

Objects have both data properties and methods (function properties). Often a

property of one object is an object in its own right or perhaps another kind of collection, such as an array.

Properties

Variables (data) that "belong" to objects are called properties. Example for array's length property:

```
var myArray = new Array( 5 );

alert( "length of array is: " + myArray.length );
```

In the example above the length property of the myArray object is accessed by the dot notation: **myArray.length**

Methods

Methods are object properties containing functions. In earlier units you have seen examples of methods, for example the reverse method of an array:

```
var myArray = new Array( 3 );

myArray[0] = 11; myArray[1] = 22; myArray[2] = 33;

 alert( myArray.reverse() );
```

In the example above the reverse() method of the myArray object is invoked by the dot notation: **myArray.reverse()**

As can be seen, the only difference from (data) properties is when methods are invoked with the use of the () (parentheses) operator — this is a special operator expecting a variable holding a Function object to the left, and optional operands (the function arguments) between the parentheses.

It is important to realise that methods are properties, but ones which can also be invoked since they are properties that hold Function objects.

The 'dot' syntax

The dot notation is used to indicate which properties an object owns. Ownership of properties can exist to any number of levels deep, and the dot notation can be used at every level. fine.

The use of the full stop (dot) can be thought of as meaning "the thing on the right belongs to the object (named collection) on the left". The general form for properties and methods to be invoked is as follows:

<object>.<property>

<object>.<method>()

Common examples include:

```
var myArray = new Array( 3 ) ;

myArray.length;

 var images = new Image();

images.src = "images.jpg";
```

Operators

If you have variables, you can use them to perform different kinds of operations. To do so, you need operators.

Basic Operators

1. + Addition
2. — Subtraction
3. * Multiplication
4. / Division
5. (...) — Grouping operator, operations within brackets are executed earlier than those outside
6. % Modulus (remainder)
7. ++ Increment numbers
8. -- Decrement numbers

Comparison Operators

1. == Equal to
2. === Equal value and equal type
3. != Not equal
4. !== Not equal value or not equal type
5. > Greater than
6. < Less than
7. >= Greater than or equal to
8. <= Less than or equal to
9. ? Ternary operator

Logical Operators

10. && Logical and
11. || Logical or
12. ! Logical not

Bitwise Operators

13. & AND statement
14. | OR statement
15. ~ NOT
16. ^ XOR
17. << Left shift
18. >> Right shift
19. >>> Zero fill right shift

Outputting Data

A common application for functions is the output of data. For the output, you have the following options:

1. alert() — Output data in an alert box in the browser window

2. confirm() — Opens up a yes/no dialog and returns true/false depending on user click
3. console.log() — Writes information to the browser console, good for debugging purposes
4. document.write() — Write directly to the HTML document
5. prompt() — Creates a dialogue for user input

UNIT 1: HTML, CSS & Client Side Scripting

JavaScript Array

Arrays are container-like values that can hold other values. The values inside an array are called elements. Array elements don't all have to be the same type of value. Elements can be any kind of JavaScript value — even other arrays.

```
var hodgepodge = [100, "paint", [200, "brush"],
false];
hodgepodge;
▶ (4) [100, "paint", Array(2), false]
```

Accessing Elements

To access one of the elements inside an array, you'll need to use the brackets and a number like this: myArray[3]. JavaScript arrays begin at 0, so the first element will always be inside [0].

```
var hodgepodge = [100, "paint", [200, "brush"],
false];
hodgepodge[2];
▶ (2) [200, "brush"]
```

reverse

An array's reverse method returns a copy of the array in opposite order.

```
["a", "b", "c"].reverse();
▶ (3) ["c", "b", "a"]
```

array.push():JavaScript array push method to push one or more values into an array. The length of the array will be altered wrt to function.

syntax: array.push(element1[, ...[, elementN]])

Arguments: Let's take a look at the .push() JavaScript arguments. As for the number of arguments permitted in JavaScript array push function, there is no limit as such. It is about the number of elements you want to insert into the array using push JavaScript.

Return value: JavaScript array push function will be returning the new length of the array after you are done with inserting arguments.

```
> var array = [];
  array.push(10, 20, 30, 40, 50, 60, 70);
  console.log(array)
  ▶ (7) [10, 20, 30, 40, 50, 60, 70]
```

Array.pop(): JavaScript is used to remove the last element in an array. Moreover, this function returns the removed element. At the same, it will reduce the length of the array by one. This function is the opposite of the JavaScript array push function.

syntax : array.pop()

Arguments: This function does not pass any arguments.

Return value: As stated before, this function returns the removed element. In case the array is empty, you will be getting undefined as a returned value.

```
var names = ['Blaire', 'Ash', 'Coco', 'Dean', 'Georgia'];
var remove = names.pop();
console.log(names);
console.log(remove);
▶ (4) ["Blaire", "Ash", "Coco", "Dean"]
Georgia
```

Array Shift Method

JavaScript array shift() method to remove an element from the beginning of an array. It returns the item you have removed from the array, and length of the array is also changed. Basically, it excludes the element from the 0th position and shifts the array value to downward and then returns the excluded value.

```
var names = ['Blaire', 'Ash', 'Coco', 'Dean', 'Georgia'];
var initialElement = names.shift();
console.log(names);
console.log(initialElement);
▶ (4) ["Ash", "Coco", "Dean", "Georgia"]
Blaire
```

Unshift JavaScript Method

unshift() JavaScript function will be used to add items to the beginning of an array.

```
var nameArray = ['Ash', 'Coco', 'Dean', 'Georgia'];
nameArray.unshift('Willy', 'Blaire')
console.log(nameArray);
▶ (6) ["Willy", "Blaire", "Ash", "Coco", "Dean", "Georgia"]
```

JavaScript Functions

A function is a subprogram designed to perform a particular task. Functions are executed when they are called. This is known as invoking a function. Values can be passed into functions and used within the function. Functions always return a value. In JavaScript, if no return value is specified, the function will return undefined. Functions are objects.

A Function Declaration defines a named function. To create a function declaration you use the function keyword followed by the name of the function. When using function declarations, the function definition is hoisted, thus allowing the function to be used before it is defined.

```
function name(parameters){
  statements
}
```

A Function Expressions defines a named or anonymous function. An anonymous function is a function that has no name. Function Expressions are not hoisted, and therefore cannot be used before they are defined. In the example below, setting the anonymous function object equal to a variable.

```
let name = function(parameters){
    statements
}
```

An Arrow Function Expression is a shorter syntax for writing function expressions. Arrow functions do not create their own this value.

```
let name = (parameters) => {
  statements
```

```
}
```

Parameters vs. Arguments.

Parameters are used when defining a function, they are the names created in the function definition. In fact, during a function definition, can pass in up to 255 parameters! Parameters are separated by commas in the (). Here's an example with two parameters — **param1** & **param2**:

```
const param1 = true;

const param2 = false;

function twoParams(param1, param2){

  console.log(param1, param2);

}
```

Arguments, on the other hand, are the values the function receives from each parameter when the function is executed (invoked). In the above example, our two arguments are true & false.

Invoking a Function.

Functions execute when the function is called. This process is known as invocation. You can invoke a function by referencing the function name, followed by an open and closed parenthesis: ().

Function Return.

Every function in JavaScript returns undefined unless otherwise specified.

Let's test this by creating and invoking an empty function:

```
function test(){};

test();

// undefined
```

As expected, undefined is returned.

Now, customize what is returned in our function by using the return keyword followed by our return value. Take a look at the code below:

```
function test(){
  return true;
};

test();

// true
```

In this example, explicitly tell the function to return true. When invoked the function, that's exactly what happens.

Hoisting

In most of the examples so far, we've used var to declare a variable, and we have initialized it with a value. After declaring and initializing, we can access or reassign the variable. If we attempt to use a variable before it has been declared and initialized, it will return undefined.

```
// Attempt to use a variable before declaring it
        console.log(x);
// Variable assignment
        var x = 100;
```

Output

```
        undefined
```

However, if we omit the var keyword, we are no longer declaring the variable, only initializing it. It will return a ReferenceError and halt the execution of the script.

```
// Attempt to use a variable before declaring it
        console.log(x);
// Variable assignment without var
        x = 100;
```

Output

```
        ReferenceError: x is not defined
```

The reason for this is due to hoisting, a behavior of JavaScript in which variable and function declarations are moved to the top of their scope. Since only the actual declaration is hoisted, not the initialization, the value in the first example returns undefined.

To demonstrate this concept more clearly, below is the code we wrote and how JavaScript actually interpreted it.

```
// The code we wrote
        console.log(x);
        var x = 100;
```

```
// How JavaScript interpreted it
        var x;
        console.log(x);
        x = 100;
```

JavaScript saved x to memory as a variable before the execution of the script. Since it was still called before it was defined, the result is undefined and not 100. However, it does not cause a ReferenceError and halt the script. Although the var keyword did not actually change location of the var, this is a helpful representation of how hoisting works.

This behavior can cause issues, though, because the programmer who wrote this code likely expects the output of x to be true, when it is instead undefined.

We can also see how hoisting can lead to unpredictable results in the next example:

```
// Initialize x in the global scope
        var x = 100;
function hoist() {
 // A condition that should not affect the outcome of the code
        if (false) {
                var x = 200;
        }
                console.log(x);
        }
        hoist();
        Output
                undefined
```

In this example, we declared x to be 100 globally. Depending on an if statement, x could change to 200, but since the condition was false it should not have affected the value of x. Instead, x was hoisted to the top of the hoist() function, and the value became undefined.

This type of unpredictable behavior can potentially cause bugs in a program. Since let and const are block-scoped, they will not hoist in this manner, as seen below.

```
// Initialize x in the global scope
        let x = true;
        function hoist() {
 // Initialize x in the function scope
        if (3 === 4) {
                let x = false;
        }
         console.log(x);
        }
        hoist();
```

Output

true

Duplicate declaration of variables, which is possible with var, will throw an error with let and const.

```
// Attempt to overwrite a variable declared with var
        var x = 1;
        var x = 2;
        console.log(x);
```

Output

2

```
// Attempt to overwrite a variable declared with let
        let y = 1;
        let y = 2;
        console.log(y);
```

Output

Uncaught SyntaxError: Identifier 'y' has already been declared.

To summarize, variables introduced with var have the potential of being affected by hoisting, a mechanism in JavaScript in which variable declarations are saved to memory. This may result in undefined variables in one's code. The introduction of let and const resolves this issue by throwing an error when attempting to use a variable before declaring it or attempting to declare a variable more than once.

UNIT 1: HTML, CSS & Client Side Scripting

JavaScript Array

Arrays are container-like values that can hold other values. The values inside an array are called elements. Array elements don't all have to be the same type of value. Elements can be any kind of JavaScript value — even other arrays.

```
var hodgepodge = [100, "paint", [200, "brush"],
false];
hodgepodge;
▶ (4) [100, "paint", Array(2), false]
```

Accessing Elements

To access one of the elements inside an array, you'll need to use the brackets and a number like this: myArray[3]. JavaScript arrays begin at 0, so the first element will always be inside [0].

```
var hodgepodge = [100, "paint", [200, "brush"],
false];
hodgepodge[2];
▶ (2) [200, "brush"]
```

reverse

An array's reverse method returns a copy of the array in opposite order.

```
["a", "b", "c"].reverse();
▶ (3) ["c", "b", "a"]
```

array.push():JavaScript array push method to push one or more values into an array. The length of the array will be altered wrt to function.

syntax: array.push(element1[, ...[, elementN]])

Arguments: Let's take a look at the .push() JavaScript arguments. As for the number of arguments permitted in JavaScript array push function, there is no limit as such. It is about the number of elements you want to insert into the array using push JavaScript.

Return value: JavaScript array push function will be returning the new length of the array after you are done with inserting arguments.

```
> var array = [];
  array.push(10, 20, 30, 40, 50, 60, 70);
  console.log(array)
  ▶ (7) [10, 20, 30, 40, 50, 60, 70]
```

Array.pop(): JavaScript is used to remove the last element in an array. Moreover, this function returns the removed element. At the same, it will reduce the length of the array by one. This function is the opposite of the JavaScript array push function.

syntax : array.pop()

Arguments: This function does not pass any arguments.

Return value: As stated before, this function returns the removed element. In case the array is empty, you will be getting undefined as a returned value.

```
var names = ['Blaire', 'Ash', 'Coco', 'Dean', 'Georgia'];
var remove = names.pop();
console.log(names);
console.log(remove);
▶ (4) ["Blaire", "Ash", "Coco", "Dean"]
Georgia
```

Array Shift Method

JavaScript array shift() method to remove an element from the beginning of an array. It returns the item you have removed from the array, and length of the array is also changed. Basically, it excludes the element from the 0th position and shifts the array value to downward and then returns the excluded value.

```
var names = ['Blaire', 'Ash', 'Coco', 'Dean', 'Georgia'];
var initialElement = names.shift();
console.log(names);
console.log(initialElement);
▶ (4) ["Ash", "Coco", "Dean", "Georgia"]
Blaire
```

Unshift JavaScript Method

unshift() JavaScript function will be used to add items to the beginning of an array.

```
var nameArray = ['Ash', 'Coco', 'Dean', 'Georgia'];
nameArray.unshift('Willy', 'Blaire')
console.log(nameArray);
▶ (6) ["Willy", "Blaire", "Ash", "Coco", "Dean", "Georgia"]
```

JavaScript Functions

A function is a subprogram designed to perform a particular task. Functions are executed when they are called. This is known as invoking a function. Values can be passed into functions and used within the function. Functions always return a value. In JavaScript, if no return value is specified, the function will return undefined. Functions are objects.

A Function Declaration defines a named function. To create a function declaration you use the function keyword followed by the name of the function. When using function declarations, the function definition is hoisted, thus allowing the function to be used before it is defined.

```
function name(parameters){
  statements
}
```

A Function Expressions defines a named or anonymous function. An anonymous function is a function that has no name. Function Expressions are not hoisted, and therefore cannot be used before they are defined. In the example below, setting the anonymous function object equal to a variable.

```
let name = function(parameters){
    statements
}
```

An Arrow Function Expression is a shorter syntax for writing function expressions. Arrow functions do not create their own this value.

```
let name = (parameters) => {
  statements
```

```
}
```

Parameters vs. Arguments.

Parameters are used when defining a function, they are the names created in the function definition. In fact, during a function definition, can pass in up to 255 parameters! Parameters are separated by commas in the (). Here's an example with two parameters — **param1** & **param2**:

```
const param1 = true;

const param2 = false;

function twoParams(param1, param2){

  console.log(param1, param2);

}
```

Arguments, on the other hand, are the values the function receives from each parameter when the function is executed (invoked). In the above example, our two arguments are true & false.

Invoking a Function.

Functions execute when the function is called. This process is known as invocation. You can invoke a function by referencing the function name, followed by an open and closed parenthesis: ().

Function Return.

Every function in JavaScript returns undefined unless otherwise specified.

Let's test this by creating and invoking an empty function:

```
function test(){};

test();

// undefined
```

As expected, undefined is returned.

Now, customize what is returned in our function by using the return keyword followed by our return value. Take a look at the code below:

```
function test(){
  return true;
};

test();

// true
```

In this example, explicitly tell the function to return true. When invoked the function, that's exactly what happens.

Hoisting

In most of the examples so far, we've used var to declare a variable, and we have initialized it with a value. After declaring and initializing, we can access or reassign the variable. If we attempt to use a variable before it has been declared and initialized, it will return undefined.

```
// Attempt to use a variable before declaring it
        console.log(x);
// Variable assignment
        var x = 100;
```

Output

```
        undefined
```

However, if we omit the var keyword, we are no longer declaring the variable, only initializing it. It will return a ReferenceError and halt the execution of the script.

```
// Attempt to use a variable before declaring it
        console.log(x);
// Variable assignment without var
        x = 100;
```

Output

```
        ReferenceError: x is not defined
```

The reason for this is due to hoisting, a behavior of JavaScript in which variable and function declarations are moved to the top of their scope. Since only the actual declaration is hoisted, not the initialization, the value in the first example returns undefined.

To demonstrate this concept more clearly, below is the code we wrote and how JavaScript actually interpreted it.

```
// The code we wrote
        console.log(x);
        var x = 100;
```

```
// How JavaScript interpreted it
        var x;
        console.log(x);
        x = 100;
```

JavaScript saved x to memory as a variable before the execution of the script. Since it was still called before it was defined, the result is undefined and not 100. However, it does not cause a ReferenceError and halt the script. Although the var keyword did not actually change location of the var, this is a helpful representation of how hoisting works.

This behavior can cause issues, though, because the programmer who wrote this code likely expects the output of x to be true, when it is instead undefined.

We can also see how hoisting can lead to unpredictable results in the next example:

```
// Initialize x in the global scope
        var x = 100;
function hoist() {
 // A condition that should not affect the outcome of the code
        if (false) {
                var x = 200;
        }
                console.log(x);
        }
        hoist();
        Output
                undefined
```

In this example, we declared x to be 100 globally. Depending on an if statement, x could change to 200, but since the condition was false it should not have affected the value of x. Instead, x was hoisted to the top of the hoist() function, and the value became undefined.

This type of unpredictable behavior can potentially cause bugs in a program. Since let and const are block-scoped, they will not hoist in this manner, as seen below.

```
// Initialize x in the global scope
        let x = true;
        function hoist() {
 // Initialize x in the function scope
        if (3 === 4) {
                let x = false;
        }
         console.log(x);
        }
        hoist();
```

Output

true

Duplicate declaration of variables, which is possible with var, will throw an error with let and const.

```
// Attempt to overwrite a variable declared with var
        var x = 1;
        var x = 2;
        console.log(x);
```

Output

2

```
// Attempt to overwrite a variable declared with let
        let y = 1;
        let y = 2;
        console.log(y);
```

Output

Uncaught SyntaxError: Identifier 'y' has already been declared.

To summarize, variables introduced with var have the potential of being affected by hoisting, a mechanism in JavaScript in which variable declarations are saved to memory. This may result in undefined variables in one's code. The introduction of let and const resolves this issue by throwing an error when attempting to use a variable before declaring it or attempting to declare a variable more than once.

UNIT 1: HTML, CSS & Client Side Scripting

The 'global' object

When the JavaScript interpreter starts up, there is always a single global object instance created. All other objects are properties of this object. Also, all variables and functions are properties of this global object.

For client-side JavaScript, this object is the instance window of the constructor Window.

The Window object for client-side JavaScript: window

When an HTML document is loaded into a browser, a single Window object instance, named window, is created. All other objects are properties of this window object. Since everything is a property of the window object, there is a relaxation of the dot notation when referring to properties. Thus each reference to a variable, function or object is not required to start with "window." although this would be a more "accurate" notation to use, it is far less convenient.

Thus instead of writing:

```
document.write( "Hello" );
```

It can be written:

```
window.document.write( "Hello" );
```

One of the properties of the window object is the status property. This property is a String value that is displayed in the browser window's status bar. The status bar can be changed with, or without, an explicit reference to the window object. Both these lines have the same effect:

```
window.status = "this is a test";
status = "this is a test";
```

The Document object: document

When an HTML document is loaded into a frame of a browser window, a Document object instance named document is created for that frame. This document object, like most objects, has a collection of properties and methods. Perhaps the most frequently used method of the document object is the write() method:

document.write("sorry, that item is out of stock<p>");

One useful property of the document object is the forms property. This is actually an array of Form objects with an element for each form defined in the document. So it could refer to the first form defined in the document as follows: document.forms[0] .

The 'call' object

When a function (or method) is executed, a temporary object is created that exists for as long as the function is executing. This object is called the call object, and the arguments and local variables and functions are properties of this object.

It is through use of this call object that functions/methods are able to use local argument and variable/ function names that are the same as global variables/functions, without confusion. The call object is JavaScript's implementation of the concepts of variable/function scoping i.e. determining which piece of memory is referred to by the name of a variable or function, when there are global and local properties with the same name.

String objects

Strings are objects. A frequently used property of String is length. String includes methods to return a new String containing the same text but in upper or

lower case. So we could create an upper case version of the String "hello" as follows:

var name = "hello"; alert(name.toUpperCase());

It should be noted that none methods belonging to string objects never change the string's value, but they may return a new String object,.

Array objects

Since Arrays are rather an important topic in the own right, Array objects are given their own section at the end of this unit although you may wish to skip ahead to read that section now to help your understanding of object with this more familiar example.

Function objects

Functions are of the type Function, and can be treated as data variables or properties, or they can be treated as sub-programmes and executed using the () operator.

Math objects

The Math object provides a number of useful methods and properties for mathematical processing. For example, Math provides the following property:

Math.PI

that is useful for many geometric calculations.

An example of some other methods provided by the Math object include:

- Math.abs();
- Math.round();
- Math.max(n1, n2);

These may be used in the following way:

document.write("<p>PI is " + Math.PI);

document.write("<p>The signless magnitudes of the numbers -17 and 7 are " + Math.abs(-17) + " and " + Math.abs(7));

document.write("<p>The interger part of 4.25 is " + Math.round(4.25));

document.write("<p>The larger of 17 and 19 is " + Math.max(17, 19));

setTimeout and setInterval

We may decide to execute a function not right now, but at a certain time later. That's called "scheduling a call".

There are two methods for it:

- **setTimeout** allows us to run a function once after the interval of time.
- **setInterval** allows us to run a function repeatedly, starting after the interval of time, then repeating continuously at that interval.

These methods are not a part of JavaScript specification. But most environments have the internal scheduler and provide these methods. In particular, they are supported in all browsers and Node.js.

setTimeout

The syntax:

let timerId = setTimeout(func|code, [delay], [arg1], [arg2], ...)

Parameters:

func|code

Function or a string of code to execute. Usually, that's a function. For historical reasons, a string of code can be passed, but that's not recommended.

delay

The delay before run, in milliseconds (1000 ms = 1 second), by default 0.

arg1, arg2…

Arguments for the function (not supported in IE9-)

For instance, this code calls sayHi() after one second:

```
function sayHi() {
  alert('Hello');
}


setTimeout(sayHi, 1000);
```

With arguments:

```
function sayHi(phrase, who) {
  alert( phrase + ', ' + who );
}
setTimeout(sayHi, 1000, "Hello", "John"); // Hello, John
```

If the first argument is a string, then JavaScript creates a function from it.

So, this will also work:

```
setTimeout("alert('Hello')", 1000);
```

But using strings is not recommended, use arrow functions instead of them, like this:

```
setTimeout(() => alert('Hello'), 1000);
```

Pass a function, but don't run it

Novice developers sometimes make a mistake by adding brackets () after the function:

```
// wrong!
setTimeout(sayHi(), 1000);
```

That doesn't work, because setTimeout expects a reference to a function. And here sayHi() runs the function, and the *result of its execution* is passed to setTimeout. In our case the result of sayHi() is undefined (the function returns nothing), so nothing is scheduled.

Canceling with clearTimeout

A call to setTimeout returns a "timer identifier" timerId that we can use to cancel the execution.

The syntax to cancel:

```
let timerId = setTimeout(...);
clearTimeout(timerId);
```

In the code below, we schedule the function and then cancel it (changed our mind). As a result, nothing happens:

```
let timerId = setTimeout(() => alert("never happens"), 1000);
alert(timerId); // timer identifier

clearTimeout(timerId);
alert(timerId); // same identifier (doesn't become null after canceling)
```

As we can see from alert output, in a browser the timer identifier is a number. In other environments, this can be something else. For instance, Node.js returns a timer object with additional methods.

Again, there is no universal specification for these methods, so that's fine.

For browsers, timers are described in the timers section of HTML5 standard.

setInterval

The setInterval method has the same syntax as setTimeout:

```
let timerId = setInterval(func|code, [delay], [arg1], [arg2], ...)
```

All arguments have the same meaning. But unlike setTimeout it runs the function not only once, but regularly after the given interval of time.

To stop further calls, we should call clearInterval(timerId).

The following example will show the message every 2 seconds. After 5 seconds, the output is stopped:

```
// repeat with the interval of 2 seconds
let timerId = setInterval(() => alert('tick'), 2000);

// after 5 seconds stop
```

```
setTimeout(() => { clearInterval(timerId); alert('stop'); }, 5000);
```

Time goes on while alert is shown

In most browsers, including Chrome and Firefox the internal timer continues "ticking" while showing alert/confirm/prompt.

So if you run the code above and don't dismiss the alert window for some time, then in the next alert will be shown immediately as you do it. The actual interval between alerts will be shorter than 2 seconds.

Nested setTimeout

There are two ways of running something regularly.

One is setInterval. The other one is a nested setTimeout, like this:

```
/** instead of:
let timerId = setInterval(() => alert('tick'), 2000);
*/

let timerId = setTimeout(function tick() {
  alert('tick');
  timerId = setTimeout(tick, 2000); // (*)
}, 2000);
```

The setTimeout above schedules the next call right at the end of the current one (*).

The nested setTimeout is a more flexible method than setInterval. This way the next call may be scheduled differently, depending on the results of the current one.

For instance, we need to write a service that sends a request to the server every 5 seconds asking for data, but in case the server is overloaded, it should increase the interval to 10, 20, 40 seconds…

Here's the pseudocode:

```
let delay = 5000;
```

```
let timerId = setTimeout(function request() {
  ...send request...

  if (request failed due to server overload) {
    // increase the interval to the next run
    delay *= 2;
  }

  timerId = setTimeout(request, delay);

}, delay);
```

And if the functions that we're scheduling are CPU-hungry, then we can measure the time taken by the execution and plan the next call sooner or later.

Nested setTimeout allows to set the delay between the executions more precisely than setInterval.

UNIT 1: HTML, CSS & Client Side Scripting

JavaScript Object Inheritance

Method 1 : Object Literals

Syntax
Object.create(*proto*, [*propertiesObject*])

```
let item1 = {
        iname : "Reebok",
        iprice : "Rs 2000",
        show : function(){
                document.write("iname: "+this.iname+" iprice: "+this.iprice+"<br/>");
        }
};

let item2 = Object.create(item1); // item2 will be replica of item1

let item3 = Object.create(item1, {
                                iname:{value:"PS5"},
                                iprice:{value:"Rs 40000"},
                                idesc:{value:"UHD 1080p"}
                        });

item3.show();
```

Method 2 : Object Constructors

```
        function item(name, price){
                this.iname = name;
                this.iprice = price;
        }

        item.prototype.show = function(){
                document.write("iname: "+this.iname+" iprice: "+this.iprice+"<br/>");
        }

        function game_cons(name, price, desc){
                item.call(this, name, price);
                this.desc = desc;
        }
        //item.show.call(xyz)
        game_cons.prototype = new item();
        game_cons.prototype.constructor = game_cons;
        let gc1 = new game_cons("PS5", "Rs 40000", "UHD 1080p");
```

Object initializer

It is a comma-delimited list of zero or more pairs of property names and associated values of an object, enclosed in curly braces (`{}`):

```
var myCar = {
    make: 'Ford',
    model: 'Mustang',
    year: 1969
};
```

Unassigned properties of an object are `undefined` (and not `null`).

```
myCar.color; // undefined
```

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see property accessors). Objects are sometimes called *associative arrays*, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the `myCar` object as follows:

```
myCar['make'] = 'Ford';
myCar['model'] = 'Mustang';
myCar['year'] = 1969;
```

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

```
// four variables are created and assigned in a single go,
// separated by commas
var myObj = new Object(),
    str = 'myString',
    rand = Math.random(),
    obj = new Object();

myObj.type              = 'Dot syntax';
myObj['date created']   = 'String with space';
myObj[str]              = 'String value';
myObj[rand]             = 'Random Number';
myObj[obj]              = 'Object';
myObj['']               = 'Even an empty string';

console.log(myObj);
```

Please note that all keys in the square bracket notation are converted to string unless they're Symbols, since JavaScript object property names (keys) can only be strings or Symbols (at some point, private names will

also be added as the class fields proposal progresses, but you won't use them with `[]` form). For example, in the above code, when the key `obj` is added to the `myObj`, JavaScript will call the `obj.toString()` method, and use this result string as the new key.

You can also access properties by using a string value that is stored in a variable:

```
var propertyName = 'make';
myCar[propertyName] = 'Ford';

propertyName = 'model';
myCar[propertyName] = 'Mustang';
```

You can use the bracket notation with **for...in** to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function:

```
function showProps(obj, objName) {
  var result = ``;
  for (var i in obj) {
    // obj.hasOwnProperty() is used to filter out properties from the object's
prototype chain
    if (obj.hasOwnProperty(i)) {
      result += `${objName}.${i} = ${obj[i]}\n`;
    }
  }
  return result;
}
```

So, the function call `showProps(myCar, "myCar")` would return the following:

```
myCar.make = Ford
myCar.model = Mustang
myCar.year = 1969
```

Enumerate the properties of an object

Starting with ECMAScript 5, there are three native ways to list/traverse object properties:

- `for...in` loops
  This method traverses all enumerable properties of an object and its prototype chain
- `Object.keys(o)`
  This method returns an array with all the own (not in the prototype chain) enumerable properties' names ("keys") of an object `o`.

Alternatively, you can create an object with these two steps:

1. Define the object type by writing a constructor function. There is a strong convention, with good reason, to use a capital initial letter.
2. Create an instance of the object with `new`.

To define an object type, create a function for the object type that specifies its name, properties, and methods. For example, suppose you want to create an object type for cars. You want this type of object to be called `Car`, and you want it to have properties for make, model, and year. To do this, you would write the following function:

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
```

Notice the use of `this` to assign values to the object's properties based on the values passed to the function.

Now you can create an object called `mycar` as follows:

```
var mycar = new Car('Eagle', 'Talon TSi', 1993);
```

This statement creates `mycar` and assigns it the specified values for its properties. Then the value of `mycar.make` is the string "Eagle", `mycar.year` is the integer 1993, and so on.

You can create any number of `Car` objects by calls to `new`. For example,

```
var kenscar = new Car('Nissan', '300ZX', 1992);
var vpgscar = new Car('Mazda', 'Miata', 1990);
```

An object can have a property that is itself another object. For example, suppose you define an object called `person` as follows:

```
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
```

and then instantiate two new `person` objects as follows:

```
var rand = new Person('Rand McKinnon', 33, 'M');
var ken = new Person('Ken Jones', 39, 'M');
```

Then, you can rewrite the definition of `Car` to include an `owner` property that takes a `person` object, as follows:

```
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
```

To instantiate the new objects, you then use the following:

```
var car1 = new Car('Eagle', 'Talon TSi', 1993, rand);
var car2 = new Car('Nissan', '300ZX', 1992, ken);
```

Notice that instead of passing a literal string or integer value when creating the new objects, the above statements pass the objects `rand` and `ken` as the arguments for the owners. Then if you want to find out the name of the owner of car2, you can access the following property:

```
car2.owner.name
```

Note that you can always add a property to a previously defined object. For example, the statement

```
car1.color = 'black';
```

adds a property `color` to car1, and assigns it a value of "black." However, this does not affect any other objects. To add the new property to all objects of the same type, you have to add the property to the definition of the `Car` object type.

Using `this` for object references

JavaScript has a special keyword, `this`, that you can use within a method to refer to the current object. For example, suppose you have 2 objects, `Manager`and `Intern`. Each object have their own `name`, `age` and `job`. In the function `sayHi()`, notice there is `this.name`. When added to the 2 objects they can be called and returns the `'Hello, My name is'` then adds the `name` value from that specific object. As shown below.

```
const Manager = {
  name: "John",
  age: 27,
  job: "Software Engineer"
}
const Intern= {
  name: "Ben",
  age: 21,
  job: "Software Engineer Intern"
}

function sayHi() {
    console.log('Hello, my name is', this.name)
}

// add sayHi function to both objects
Manager.sayHi = sayHi;
Intern.sayHi = sayHi;

Manager.sayHi() // Hello, my name is John'
Intern.sayHi() // Hello, my name is Ben'
```

The `this` refers to the object that it is in. You can create a new function called `howOldAmI()`which logs a sentence saying how old the person is.

```
function howOldAmI (){
  console.log('I am ' + this.age + ' years old.')
}
Manager.howOldAmI = howOldAmI;
Manager.howOldAmI() // I am 27 years old.
```