**MIDTERMS**

**HyperText Markup Language (HTML)**

**-** by Tim Burners Lee

- is the standard markup language for creating web pages and web applications.

- takes document using markup.

- Structure & Content (Main focus)

-Presentational / aesthetic (old format)

**Website**- is a collection of web resources such as wen pages and web applications.

- it is being host.

**World Wide Web Consortium (W3C) –** Tim Burners Lee

* + - Is an international community that develops open standards to ensure the long-term growth of the web.

Not already recommended codes in HTML:

* Character entities:
  + &nbsp –tab
  + &lt – less than
  + &gt – greater than
* <br>
* Tables –for tabular presentation.

**HTML versions:**

Current – 5.1

HTML 1.0 – 1989

2.0 – Nov 1995 RFC18661L

3.0 – superseded by 3.2

4.0 – December, 1997

4.1 – 1999

**Extensible HyperText Markup Language (XHTML)**

Version: 1.0 – January 26, 2000

Revised August 1, 2002

* + - It is used for data interchange
    - Wrappers up structure of a document

**Ian Hidson** – proposed to go back to HTML

**-** WHATWG

- later becomes HTML5 that is standardized in 2014

HTML 4.0

* **Transitional** – depreciated
* **Strict**
* **Frameset**

**The Elements of HTML**

**Index of Attributes**

Grouping

Categories

Flow:

* Heading
* Sectioning
* Interaction
* Phrasing
* Embedded
* Metadata

Content

* + - Flow content is expected
  + Content Model
    - Phrasing Content
  + Content Attribute
    - Global Attribute

Ex. Id, title, long, alt, src

**ARIA state and property attributes**

* + Global ARIA – attributes

**DOM interface**

* + - interface HTML Paragraph Element : HTML Element

Paragraph in HTML is Thematic a content

Content in w/c this element can be used: Embedded Content

Tag Omission in text HTML: No end tag

**Summary of HTML Elements**

Html  
 head  
 title, base, link, meta, style

Body  
 article, aside, nav, section

header, footer

main

address

div

h1, h2, h3, h4, h5, h6

p

hr

pre

blockquote

ol, ul

li

dl

dt, dd

figure

figcaption

a

em, strong, small

cite, q

dfn, abbr

ruby, rb, rt, rtc, rp --→ ruby anotations

data, time

code, vaar, samp, kbd

sup, sub

l, b, u, mark

bai, bdo

span

table

caption

colgroup, col

**CASCADING STYLE SHEET (CSS)**

CSS

* language used to specify the presentational aspects
* Developed by Hakon Wium Lie (CHSS) and Bert Bos (SSP)

Versions

* css 1 , css 2.1, css3

CSS Preprocessors, CSS Frameworks

* Sass, Less, 960 Grid System, Bootstrap, Foundation, Materializa, etc

**===HTML/XHTML STYLESHEETS===**

* author styles (external stylesheets,embedded styles, inline styles)
* user style
* user agent styles (example default css 2.1 stylesheet for HTML 4)

\* Reset CSS – overide the default stylesheet

**===CSS Statements===**

* At-Rules
  + @charset, @font-face @import ‘global.css’, @namespace, @document, @font-face, @key frames, @media, @page
* css rule sets (a.k.a CSS rules, style rules)
  + consults of a selector, followed by a declaration block.

**===CSS Selectors===**

* Simple Selector
  + either a type selector or universal selector followed by zero or more attribute selector , ID selectors or pseudo classes.
* Selector Syntax
  + Chain of one or more sequence of simple selectors by combinators.
* Type selector
  + matches the name of a document language element type.
* Universal selector
  + written "\*", matches the name of any element type.
* Attribute selector

[attr]

[attr=value]

[attr~=value] – space separated values

[attr|=value] –target long attribute

[attr^=value] -beginning (CSS3)

[attr$=value] –End (CSS3)

[attr\*=value] –Everywhere (CSS3)

* ID selector

- match an element instance based on its identifier.

-A CSS ID selector contains a "#" followed by ID value.

* Pseudo classes
  + classify elements on characteristic other than their name, attributes or content.
  + there's is implicit class associated with.
* Dynamic pseudo class
  + link pseudo-classes
    - link - link that have not yet visited.
    - visited - link that have been visited.
  + user action pseudo class
    - * hover
      * action - active a link
      * focus - move particular element
      * target pseudo classes
      * language pseudo classes

:lang()

* UI element states pseudo classes

-:enabled (CSS3)

-:disabled (CSS3)

-:checked (CSS3)

-:inderminate (css3)

* Structural

-:root

-:first-child

-:last-child

-only-child

-:nth-child

-:nth-last-child

-:first-of-type

-:last-of-type

-:only-of-type

-:nth-of-type

-:nth-last-type

-:empty

negation

:not() - negates the logic of the selector.

* Combinators
  + descendant combinator (whitespace i.e. space, tab, line feed, carriage return, form feed)

--child combinator(>)

adjacent sibling combinator (+)

general sibling combinator (~)

Pseudo elements

::first-letter :first-letter

::first-line :first-line

::before :before

:: after :after

**===CSS Rule Precedence===**

* **by origin and importance**

1. Transition Declarations
2. Override declaration
3. User declaration
4. Animation Declaration
   * user agent important declarations
   * user important declarations
   * author important declarations
   * author normal declarations
   * user normal declarations
   * user agent nor mal declarations

* **by specificity**
  + inline – more specific
  + number of ID selectors
  + number of class selectors, attribute selectors and pseudo selectors
  + number of type selectors and pseudo elements
* **by order**

**===CSS Declarations===**

* **Properties**
  + **short hand properties**
    - background , font
  + **vendor specific extension (aka vendor prefixes)**
* **values**
  + **keywords**
    - **inherit,**
  + **numbers**
  + **measurements**
    - length units
      * font relative
        + em, ex, ch, rem
      * viewport percentage
        + vw, vh,vmin, vmax
      * absolute lengths
        + cm,mm,q, in pt, pc, px
    - angle units
      * deg, grad, rad, turn
    - duration units
      * s, ms
    - frequency units
      * hz, khz
    - resolution units
      * dpi, dpcm, dppx
* **percentages**
* **URLs and URIs**
* **Colors**
  + Rgb( , , ), #rgb,rgba, hsl, hsla
* **Strings**
* **Functions**
  + calc(), attr(), count(), linear-gradient(), translate(), scale(), rotate(), etc.

**Values and Units Module Level 3**

**CSS Preprocessors, CSS Framework**

* SCSS(Sassy Cascading Stylesheet), 60 Grid System, Bootstrap, Foundation, materials, etc.

# JAVASCRIPT CLIENT SIDE SCRIPTING

**Javascript**

* Is a client-side scripting language that dynamically provides interaction to web applications such as
  + games
  + events
  + carousels
  + image galleries
  + responses to button clicks
* used for computations
* It has scripting knowledge, and it is a C base language
* It is a lightweight interpreted or JIT-compiled programming language with first-class functions
* Standard – ECMA 262

ECMA 5 – Almost all browser support this

Different ways to script:

1. console.log(‘externally linked script….’);
2. <script> -(global)

console.log(‘embedded script….’);

function callme(){

console.log(‘you added me’);

}

</script>

1. <button onclick = ‘console.log(“inline script…”);’> click me </button>

**Script –**it should be executing in a ‘sandbox’

* **Defer**- to execute other part of the document
* **Async –** intermix in rendering, don’t wait until the whole document is executed.

**\* <no script>**  - only display if a user doesn’t have script

**Attribute**

* **Global attribute**

Example:

window.navigator.vendor(app version, etc)

window.screen.

window.document.getElementByID(‘h’)

h = document.getElementByID(‘h’)

<h1 id= “h”>

<p> Javascript</p>

</h1>

* **Certain attribute**

W3C:

Document Object Model Core

Document = Node

document extends node

**Interface Node**

**Node**- is the primary datatype for the entire Document Object Model.\

- represents a single mode in the document tree

|  |  |  |
| --- | --- | --- |
| Node Name | Node Type | Node Value |
| “#text” | 3 | “hi” |
| “#document” | 9 | null |

**Child Nodes**

Ex.

document.body.childNodes[0]

document.body.childNodes[length]

document.body.firstChild

document.body.firstChild.nextSibling

others:

insertBefore

replaceChild

removeChild

appendChild

hasChildNode()

**Variable, Let, Constant (var, let, const)**

**Variable-** associated with global content

Example:

<script>

var x = 100;

function f() {

var x =200;

{

var x = 300;

}

} </script>

**let-** not associated w/ global content

Example:

<script>

let x = 100;

function f() {

let x =200;

{

let x = 300;

}

} </script>

**Const-** cannot modify value

- have constant value

**Standard Objects**

**-** get to know standerd built-in objects Array, Booleans, Date, Error, Function, JSON, Math, Number, Object and others

**Expressions and Operatiors**

**-** learn more about the behavior of Java Scripts operators instance of typeOff, new, this, the operator precedence and more

**Array**

Example:

var emptyArray = new Array();

var alsoEmptyArray = [];

var arrayWithLengthFive = new Array(5);

var array = new Array(‘5’);

var sameArray = [‘5’];

**Methods**

* **Mutator methods**

**-** Method use to modify the target.

**-** It does change the target array.

* Accessor

- Don’t modify the target array

ex.

.pop() - remove the last element

.push() – adds into the end of array

.splice

.sort()

.every() – if a condition is satisfied, it wil return true

.map() – returns a certain values of the site.

.reduce() - accumulator

Researches:

**Html**

**Definition**

HTML or HyperText Markup Language is computer language that is used to create webiste. It is use for the content of a website. And the tools besides of HTML are generally used to describe a webpage's appearance/presentation (CSS) or functionality (JavaScript).

According to MDN or Mozilla Developer Network that “*HyperText* refers to links that connect webpages to one another, either within a single website or between websites. Links are a fundamental aspect of the Web. By uploading content to the Internet and linking it to pages created by other people, you become an active participant in the World Wide Web. And HTML uses *markup* to annotate text, images, and other content for display in a Web browser. HTML markup includes special "elements" such as <head>, <title>, <body>, <header>, <article>, <section>, <p>, <div>, <span>, <img>, and many others.”

HTML. (n.d.). Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTML

**History**

Berners-Lee developed and defined the HTML language, which was created and defined using SGML, during the development cycle for the first Web browser/editor from October to December 1990. The first version of the browser initially ran only on the NeXT platform and was only processing text files, but it was a start. Berners-Lee later put the code and specifications for the project (including HTML) on the Internet in the summer of 1991. During the next few years the system introduced by Berners-Lee caught on in the Internet community - and the 'web' of documents available was steadily growing. A common library of code was available to programmers to easily create the needed capabilities to access web documents. Browsers quickly became available for a wide variety of platforms. As the number of implementations grew, the variety did also. The HTML language originally specified by Berners-Lee had developed and extended far beyond its initial form and no real standard had yet been developed.

Wilson, B. (n.d.). HTML Overview. Retrieved from http://www.blooberry.com/indexdot/history/html.htm

**Quick version**

|  |  |  |
| --- | --- | --- |
| **Version** | **Year** | **Specification Document** |
| HTML 1.0 | 1990 | No Official Documentation |
| HTML 2.0 | 1995 | RFC 1866 |
| HTML 3.0 | 1995 | W3C Recommendation |
| HTML 4.0 | 1997 | W3C Recommendation |
| HTML 4.1 | 1999 | W3C Recommendation |
| XHTML 1.0 | 2000 | W3C Recommendation |
| XHTML 1.1 | 2001 | W3C Recommendation |
| HTML 5.0 | 2014 | W3C Recommendation |
| HTML 5.1 | 2016 | W3C Recommendation |

HTML Syntax

In HTML 4.1 there are 3 DTD:

Strict

Strict is a trimmed down version of HTML 4.1. It emphasizes the structure over presentation. In HTML 4 strict, deprecated element and attributes such as attributes use in presentation, frames and link targets are not allowed. Writing in HTML 4 strict developer can achieve accessible, structurally rich documents that easily adapt to style sheets and different browsing situations but documents may look bland on very old browsers that lack support for style sheets.

Transitional

HTML 4 Transitional includes all elements and attributes of HTML 4 Strict but adds presentational attributes, deprecated elements, and link targets.

Frameset

Frameset is a variant of HTML 4 Transitional for documents that use frames.

Quinn, L. (n.d.). Choosing a DOCTYPE. Retrieved from

http://www.htmlhelp.com/tools/validator/doctype.html

|  |  |  |
| --- | --- | --- |
| **Tag** | **What it is** | **When to use it** |
| **<A>** | Anchor (most commonly a link) | Vital. Use to create links in content. Use the title attribute whenever the contents of the **<a>…</a>** pair do not accurately describe what you’ll get from selecting the link. Title attribute often displays as a tooltip in visual browsers, which may be a helpful usability aid. |
| **<ABBR>** | Defines an abbreviation | Works in a similar way to **<dfn>** and **<acronym>**, using a **title** attribute (displays a tooltip in standard visual browsers). e.g. **<abbr title=”Hypertext markup language”>HTML</abbr>** |
| **<ACRONYM>** | Defines an acronym | Works in a similar way to **<abbr>** and **<dfn>**, using a **title** attribute (displays a tooltip in standard visual browsers). |
| **<ADDRESS>** | Used for marking up a physical (e.g. mailing) address | Not commonly used. Recommend looking into microformats, which allow for more detail and interoperability. |
| **<APPLET>** | Inserts a Java applet | The old way to insert a Java app. Use **<object>** instead today. |
| **<AREA>** | Hotspot in image map | Avoid image maps where possible. Occasionally necessary. |
| **<BASE>** | Specifies the base location of the document. | Use only when necessary. Adjusts any relative links and paths within the document. |
| **<BASEFONT>** | Sets default font size | Display info – never use it |
| **<BIG>** | Larger text | Display info – never use it |
| **<BLINK>** | Makes text blink | You go to hell if you use this |
| **<BLOCKQUOTE>** | Large quoted block of text | Use for any quoted text that constitutes one or more paragraphs (note: should contain <p> tags as well). Use **<q>** for quotations within a paragraph. Often used in conjunction with **<cite>** to cite the quotation’s source. |
| **<BODY>** | Document body | Essential (unless you’re using frames) |
| **<BR>** | Line break | This is arguably display information. Still in common use, but use with restraint. |
| **<B>** | Bold text | Display info – never use it |
| **<BUTTON>** | Used for a standard clickable button within a form | Often better than **<input type=”button” />** or **<input type=”submit” />**, as it allows you to assign different styles based on the HTML element alone, whereas differentiating style based on the type of input is less well supported. |
| **<CAPTION>** | Caption for a table: describes the table’s contents | The correct way to assign a title to a table |
| **<CENTER>** | Centred block | Display info – never use it. Use **<div>** or some other block-level tag with the style text-align:center instead |
| **<CITE>** | Defines a citation | Defines the source of a quotation (in conjunction with content in **<q>** or **<blockquote>** pairs). |
| **<CODE>** | Defines an extract of code | Not commonly used. Similar to **<pre>** tag, but collapses consecutive white spaces and line breaks in the source. |
| **<COL>** | Identifies a particular column in a table | Can be very useful. e.g. **<col class=”namecol”>** can be applied to each first column in a series of tables, then the width of each column may be set to be equal in the stylesheet, overriding the table’s natural tendency to adjust its own column widths to fit its contents. |
| **<DFN>** | Definition of a term | Works in a similar way to **<abbr>** and **<acronym>**, using a **title** attribute (displays a tooltip in standard visual browsers). |
| **<DIR>** | Directory list | Now deprecated. Use a standard **<ul>** or other list instead. |
| **<DIV>** | Division | Specifies a logical division within a document. Use it to separate or identify chunks of content that are not otherwise distinguished naturally using other tags.  One of the most common HTML tags. |
| **<DL>** | Definition list | Contains one or more definition-term / definition-description pairs. |
| **<DT>** | Definition term | Used as part of a **<dt></dt><dd></dd>**pair within a definition list (**<dl></dl>**) |
| **<DD>** | Definition description |
| **<EM>** | Emphasis | Commonly used in place of the old **<i>**(italics) tag to indicate emphasis (but less than **<strong>**) |
| **<FONT>** | Font settings | Display info – never use it |
| **<FORM>** | Input form | Essential for data input |
| **<H1>** | Level 1 header | Aim to have one H1 on each page, containing a description of what the page is about. |
| **<H2>** | Level 2 header | Defines a section of the page |
| **<H3>** | Level 3 header | Defines a sub-section of the page (should always follow an H2 in the logical hierarchy) |
| **<H4>** | Level 4 header | Etc. Less commonly used |
| **<H5>** | Level 5 header | Less commonly used. Only complex academic documents will break down to this level of detail. |
| **<H6>** | Level 6 header | Less commonly used |
| **<HEAD>** | Document head | Essential. Contains information about a page that does not constitute content to be communicated as part of the page. |
| **<HR>** | Horizontal rule | Display info with no semantic value – never use it. “Horizontal”, by definition, is a visual attribute. |
| **<HTML>** |  | Core element of every web page. |
| **<IMG >** | Show an image | Vital. Always use the **alt** or **longdesc**attributes when the image has content value |
| **<INPUT>** | Input fields within forms | Vital. (I prefer to use **<button>** for buttons and submit buttons though) |
| **<ISINDEX>** | Old type of search input | Not really used any more. Use **<form>**instead. |
| **<I>** | Italicised text | Display info – never use it |
| **<KBD>** | Keyboard input | Display info – never use it |
| **<LINK>** | Defines a relationship to another document | Commonly used to reference external stylesheets, but has other minor uses |
| **<LI>** | List item | Specifies an item in an unordered or ordered list (**<ul>** or **<ol>**) |
| **<MAP>** | Client-side imagemap | May have occasional value, but only use when absolutely necessary |
| **<MARQUEE>** | Makes text scroll across the screen | See **<blink>** |
| **<MENU>** | Menu item list | Deprecated. Do not use. Use other standard list types instead. |
| **<META>** | Meta-information | Useful way to insert relevant information into the <head> section of the page that does not need to be displayed. |
| **<OL>** | Ordered list | Type of list where the order of elements has some meaning. Generally rendered with item numbers (best managed with CSS). |
| **<OPTION>** | Selection list option | Vital for options within a drop-down control. |
| **<PARAM>** | Parameter for Java applet | Used in conjunction with an **<object>** or **<applet>** tag to pass additional setting information at runtime. |
| **<PRE>** | Preformatted text | Renders text in a pre-formatted style, preserving line breaks and all spaces present in the source. May be useful. *(This one’s a paradox, as it is strictly display info that applies only to visual browsing, but it’s still so commonly used and useful that I’m hesitant to advise against using it.)* |
| **<P>** | Paragraph | Only use to denote a paragraph of text. Never use for spacing alone. |
| **<Q>** | Short quotation | Use for inline quotations (whereas **<blockquote>** should be used for quotations of a paragraph or more). Often used in conjunction with **<cite>** to cite the quotation’s source. |
| **<SAMP>** | Denotes sample output text | Similar to the <code> tag. Rarely used. Avoid. |
| **<SCRIPT>** | Inline script (e.g. JavaScript) | It’s better to have all scripts as separate files than to write inline or in the **<head>**section, however still has its uses. |
| **<SELECT>** | Selection list | A drop-down selector for a form. |
| **<SMALL>** | Smaller text | Display info – never use it |
| **<SPAN>** | An inline span within text | Use to apply meaning (and style) to a span of text that goes with the flow of content (whereas a **<div>** tag is block-level and breaks the flow) |
| **<Strikeout>** |  | Display info – never use it |
| **<STRONG>** | Strong emphasis | Use this instead of the old **<b>** tag. |
| **<STYLE>** | CSS style settings | Normally used in **<head>** section of a page. Try to use external stylesheets, to enable you to apply different styles for different output media. |
| **<SUB>** | Subscript text | Arguably display info – recommend using alternative tags (e.g. **<cite>**). May be required in some academic uses, e.g. Chemical formulas. |
| **<SUP>** | Superscript text |
| **<TABLE>** | Table | Use for repeated data that has a naturally tabular form. Never use for layout purposes. |
| **<TD>** | Table data cell | A cell containing actual data. If a cell actually contains a descriptor or identifier for a row or column, use a **<th>**(table header) tag, not a **<td>.** This usually applies to column headers (within a **<thead>**), column footers (within a **<tfoot>**), as well as row headers (usually the first cell in a row in the **<tbody>**). |
| **<TEXTAREA>** | Multi-line text input area in a form | Essential |
| **<TH>** | Table column or row header cell | May appear in a **<thead>** (to denote a column header cell), **<tbody>** (to denote a row header), and in **<tfoot>** (to denote a column foot cell, e.g. a total) |
| **<TBODY>** | Indicates the main body of a data table | It is always worth using this tag, as well as using **<thead>** and **<tfoot>** where appropriate.  Note that it is permissible to have more than one **<tbody>**, **<thead>**, and **<tfoot>**in the same table. |
| **<THEAD>** | The head section of a table | The place to put column header cells (**<th>**) |
| **<TFOOT>** | The foot section of a table | Good place to put e.g. summary data, such as totals. Note that it goes before the **<tbody>** tag! |
| **<TITLE>** | Document title | Essential |
| **<TR>** | Table row | Essential with tables |
| **<TT>** | “Teletype” – simulates typewriter output | Similar to <pre>, except that it collapses white space like normal HTML (whereas <pre> leaves all consecutive white space intact). Avoid if possible |
| **<UL>** | Unordered list | Essential. Use for lists where the order or items has no particular importance. |
| **<U>** | Underline text | Display info – never use it |
| **<VAR>** | Variable in computer code | Obscure tag, may only be useful in academic documents. Avoid. |

Hunt, B. (2016, January 27). HTML Tags List. Retrieved from http://webdesignfromscratch.com/html-

css/html-tags/

**CSS**

Cascading Style Sheets (CSS) is a style sheet language written in a markup language, is used for formatting Web pages layout. While HTML is for the structure and content of a Web pages, CSS is a language used to specify the presentation aspects of web pages. It is used to design HTML; define text styles, layout display, format table sizes and other formatting aspects for HTML.

**History**

CSS like HTML, W3C standardized CSS and make it open for all and released in 1996. It was first proposed and developed by Hakon Wium Lie (CHSS) on October 10, 1994 and Bert Bos (SSP) who become the co-author of CSS. But it was not a new idea that time, Tim Berners-Lee on 1990 wrote his NeXT browser/editor specify a simple style sheet.

**Versions**

**CSS 1**

Hakon Wium Lie and Bert Bos developed the first CSS and was standardized and recommended officially by W3C. It is the CSS level 1 and it was published December 17, 1996. It is capable of: formatting font properties; Changing of color of some elements such as text and backgrounds; Layout of text, tables, and images; Capable of making and editing paddings, border, margin and positioning of elements. It can also specify foreground and background color, as well as background images.

**CSS 2**

Developed by W3C, CSS level 2 was published and recommended by W3C on May 1998. It added some capabilities like making boxes that behaves like tables, availability of powerful selectors, specification of positioning of elements, availability of media types, style rules, added some font properties and bidirectional text. It was revised and called CSS 2.1, due to the fixing of errors and removes poorly supported features. CSS 2.1 become a Candidate Recommendation first on Febraury 25, 2004 and was developed further more until it was become a W3C recommendation on June 7, 2011.

**CSS 3**

CSS3 was divided into what they called ‘modules,’ meaning it was separated into several documents. Each module can be a recommendation by different people, but some specifications are implemented by any browsers. Some module have new capabilities, some have extended features. Drafts of CSS 3 was released June of 1999. 50 CSS 3 modules was released as of June 2012, but only four of these are considered formal recommendations such as: Media Queries, Namespaces, Selectors Level 3 and Color.

**CSS 4**

It doesn’t have a single CSS 4 specification, and there is no standard name CSS 4, but it is divided into modules, and it does have level 4 modules. These level 4 modules are preceding the functionalities of level 3 modules, like the modules Selectors, Image Values and Background & Borders.

References:

<https://www.w3.org/Style/LieBos2e/history/Overview.html>

http://www.corelangs.com/css/basics/versions.html

**HTML/XHTML STYLESHEETS**

* **AUTHOR STYLES** (External stylesheets, Embedded styles, Inline styles)

Author is the one who made the codes of the CSS and embeds it to use it for websites. Author styles is the CSS file that is linked into the HTML. It is the default CSS style that the author of the code made.

**External Style Sheet**

- External CSS files that is in .css format contains stylesheet rules that will be included together with your HTML document that will be linked using the <link> tag.

**Internal Style Sheet**

- It is the CSS stylesheet rules that is embedded inside of your HTML document. It is located inside of the header section by the <style> tag

**Inline Style Sheet**

- Using the style attribute, you can define the style rules that will be apply directly into your specific HTML elements. It is good to use for making a particular style for one element, because it overrides the rules of an External CSS rules.

* **USER STYLE**

The users or the site’s visitors are the one who is viewing your website. User style allows the users to optionally override the style of the author, by defining their own stylesheets. They can set rules about font sizes, system colors and others to satisfy their own preferences.

* **USER AGENT STYLES** (example default CSS 2.1 stylesheet for HTML 4)

Browser applications such as Google Chrome and Mozilla Firefox are examples of the User Agent. We use these browsers to view websites. User agents has its own default stylesheet, but you can inspect the properties of the website that you are viewing.

**CSS STATEMENTS/ SYNTAX**

CSS style sheets contains statements that has two kinds which are Rulesets and At-rules. It is a building block which starts with characters that are non-space and ends with braces, parenthesis or semi-colon.

**Rule Sets**

- main building block of CSS, which is a collection of CSS declarations that comprises by the selector. The rule is applied to all elements having the same selector that is specified in the declaration block.

**At-rules**

**-** It is an instruction to CSS parser that starts with ‘@’ character, followed by a specified identifier. It was limited by the statement, or by the open and close curly braces or terminated by semicolon. It has its own semantics and also includes internal syntax.

**CSS SELECTORS**

Selectors are patterns that forms several technologies that will be used for the selection of nodes. It is part of the style sheet rule that specifies the content that will be using the style.

References:

<https://developer.mozilla.org/en-US/docs/Web/CSS/Syntax>

<http://reference.sitepoint.com/css/statements>

<https://www.sitepoint.com/web-foundations/css-selectors/>

References:

<http://stackoverflow.com/questions/7022344/whats-difference-between-authors-style-readers-style-agents-style-or-aut>

<https://www.tutorialspoint.com/html/html_style_sheet.htm>

**CSS Declaration**

CSS declaration is composed of value and CSS property name, and it separated and ends with a semi-colon. Declaration block is a container that is composed of one or more declaration, and it is limited inside the curly braces.

* **Property**

**-** it is a specific value that is given by the CSS property. A property is always followed by a semi-colon, and specified the value of that property.

**Shorthand properties –** it allows using of a single shorthand property but specifying several properties on it.

**Vendor-Specific properties-** usually starts with ‘-‘ or ‘\_’ and followed by the browser project extension. It is often use to test features of browsers.

* **Values**
* - It is the style specified to the property. It can be specified using different ways, depending on syntax and restrictions. It can be expressed using keywords, inherit, numbers, measurements, percentages, URLs and URIs, Colors, Strings, Functions and others.

**Values and Units Module Level 3**

**CSS Pre-processors**

Pre-processors are developed to maintain and reduce the CSS codes that the user will do. It extends. It does converts data type into another. It extends functions, variables, operators and others. It is usable for making or maintaining big projects, such as using Ruby in working projects.

* **Haml (HTML abstraction markup language)**

**-** It is a markup language, it promotes dry and well-structured coding without the use of inline code.

* **SASS (Syntactically Awesome Stylesheets)**

**-** It is an extension of CSS preprocessing languages. It is a strict indented syntax, means it is very strict in terms of structuring the codes, even indention could lead you to an error. It keeps stylesheet organized and will keep it run faster.

* **SCSS(Sassy Cascading Stylesheet)**

**-** It like SASS it is a preprocessing language. It is like SASS but it is more flexible that you can write plain CSS. Standard of CSS is much likely the same, or can run into SCSS. It needs curly bracket, and semi-colon, and doesn’t care about indention. It encourages to write with proper nesting of rules.

**CSS Frameworks**

It is a prepared software meant for using an easier and standard design of web pages using CSS. It is an abstract that provides functionalities by user-written code of CSS. It is reusable and universal that provides functionalities. Framework can be defined as a standard code that contains files and folders that is within a package. Some Popular CSS Frameworks are:

* **Bootstrap**

- It is an open-source framework that is a front-end in designing using HTML and CSS templates. It allows web pages dynamically changing, it means the web page is responsive. It is popular among users today and have a large community support, many themes and support Less or Sass.

* **Foundation**

**-** It is an open-source framework that is in front-end and it is dynamic or responsive. It is much similar to Bootstrap framework functionalities but it is more customizable, more grid system and has built-in widgets.

References:

<https://htmlmag.com/article/an-introduction-to-css-preprocessors-sass-less-stylus>

<https://www.codementor.io/css/tutorial/bootstrap-3-vs-foundation-5-front-end-framework-comparison>

JavaScript

JavaScript (JS) is a scripting language that is used to enhance HTML pages or provides interaction to the web applications. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation. JavaScript is a compact but very flexible because of that, it has expanded from its core concepts and developers had developed different extensive functionalities, and these are:

* Browser Application Programming Interfaces (API’s) – Built into a web browser that provides different functionality like manipulating HTML documents and dynamically creating CSS styles, or generating 3D images.
* Third party API’s that allows to integrate the functionality of a site to another content providers.
* Third-party frameworks and libraries that helps developers to create site and application in a more faster than before.

Variables

Variables lets you store values. In declaring a variable, use the “var” keyword then the name of the variable. Variables can be globally or locally scoped. It means that it can be used within a block or anywhere inside the script based on how you declare the variable.

Note: JavaScript is case-sensitive, that means that “webtek” is different from “WEBTEK”. Also, you can name anything to a variable, but there are some restrictions:

* Variable names cannot start with a number(i.e. 1thing, 3ck)
* Variables cannot have any mathematical operators (i.e. you+me, somthing\*27)
* Any punctuation marks are not allowed to be used in declaring a name (i.e. web&tek), but, you can use underscores”\_”.
* Just like in java in declaring a variable names it must not have spaces.
* And also like in java you cannot use the reserve word as your variable name.

In declaring variable there are two way in Javascript you can use “let” and “const”.

A variable defined with “let” is locally scoped. This variable can be created within a function, for and while-loop and many more. A “let” variable can be changed only where you declared the variable.

function varTest() {

var x = 1;

if (true) {

var x = 2; // same variable!

console.log(x); // 2

}

console.log(x); // 2

}

function letTest() {

let x = 1;

if (true) {

let x = 2; // different variable

console.log(x); // 2

}

console.log(x); // 1

}

console.log('Global scope: ', letVar1); // Is undefined

A variable with “const” defined is locally scoped and it creates as read-only reference to a value therefore it can’t be re-assigned or re-declared.

(function() {

const constVar1 = 'const variable 1 value in parent scope';

(function() {

// Scope 1

const constVar1 = 'const variable 1 value in scope 1';

console.log('scope 1: ', constVar1);

// scope 1: const variable 1 value in parent scope

}());

console.log('Patent scope: ', constVar1);

// Patent scope: const variable 1 value in parent scope

}());

Comments

You can put comments to your code using “//” at the top of your code or after an end of a line, and “/\* \*/” for multi-line comments.

*// A single line JavaScript comment*

**var** iNum = 100;

**var** iTwo = 2; *// A comment at the end of line*

*/\**

*multi-line*

*JavaScript comment*

*\*/*

Operators

* Arithmetic Operators – +, -, \*, /, %, ++, and --
* Assignment Operators –=, +=, -=, \*=, /=, %=
* String Operators – + and +=
* Comparison Operators – ==, ===, !=, !==, >, <, >=, <=
* Logical Operators - &&, ||, !
* Bitwise Operators - &, | , ~ , ^ , << , >>
* typeof Operator – returns the type of a variable, object, or function

Functions

In creating a code, you can reuse a block of task by using function where you can call whenever you need it to perform certain task.

function multiply(num1,num2) {

var result = num1 \* num2;

return result;

}

multiply(4,7); //28

multiply(20,20); //400

multiply(0.5,3); //1.5

Events

In JavaScript, you can target certain events in the browser to execute an interaction with the users such as generating report, pop-up dialog box or calculate a certain value.

The example below shows a pop-up in the browser if an element with an id of btn is clicked:

document.getElementById('btn').onclick = function() {

alert('Ouch! Stop poking me!');

}

References:

https://developer.mozilla.org/en-US/docs/Learn/Getting\_started\_with\_the\_web/JavaScript\_basics

http://www.codelifter.com/main/tips/tip\_020.shtml

https://www.w3schools.com/jsref/jsref\_operators.asp

JavaScript Data Types

JavaScript is a dynamic language and there’s no need to specify the data type of a variable whenever you are declaring it. In declaring a variable you need to use the keyword “var” and initialize it to know the type of the variable therefore a variable can be a number, String or even a Boolean.

var sam = 42; // sam is now a Number

var sam = 'bar'; // sam is now a String

var sam = true; // sam is now a Boolean

The latest ECMAScript standard defines seven data types:

* Boolean – Represents the value true or false. And in comparing two value it always return a Boolean value it is either true or false.

var y = 20;

var x = (y == 20);

The value y is compared if it is equal to 20. The result of this comparison is true and that value is assigned to var x.

* Null – Null value has no valid number, String, boolean, Array, or Object.

var person = null;

console.log(typeof person); // returns object

In may be considered to be a bug for developers that the typeof null returns an object, but the reason why typeof null returns null is that the definition of null is the primitive value that represents the intentional absence of any object value.

* Undefined – Undefined value is returned when you explicitly use a variable or an object that is not defined and not existing. You can compare a variable if it is existing to undefined or check the type by using typeof.

var x;

// This method works.

if (x == undefined) {

document.write("comparing x to undefined <br/>");

}

.

// This method doesn't work - you must check for the string "undefined".

if (typeof(x) == undefined) {

document.write("comparing the type of x to undefined <br/>");

}

// This method does work.

if (typeof(x) == "undefined") {

document.write("comparing the type of x to the string 'undefined'");

}

// Output:

// comparing x to undefined

// comparing the type of x to the string 'undefined'

* Number – In JavaScript, integer and floating-point values have no difference to each other. Integer values are positive and negative whole numbers and zero where they can be represented as decimal, hexadecimal, and octal. Floating-point values are whole numbers that contains a decimal point and values can be represented in scientific notation using the lowercase e.
* String – a sequence of characters the represents a message or anything that is textual in contrast. They are written inside double quotes.
* Objects – It is value pair (name:value) that is written with curly brackets (“{}”) and separated by commas. Users can declare an object as an object literal, creating an instance, or using the Object constructor.
* Arrays – Arrays are collection of similar type of elements. Users can declare an array as an array literal (“[]”), creating an instance, or using the Array constructor.

References:

https://docs.microsoft.com/en-us/scripting/javascript/data-types-javascript#see-also

https://www.w3schools.com/js/js\_datatypes.asp

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array

A function is a reusable block of code which can call anywhere therefore it reduces the writing of same code again and again. It helps the developer to divide big program into smaller and manageable code. Developer can also write their own function. And function can be nested.

In creating a function. Type the word function then followed by a function name, a list of parameters and a statement block surrounded by curly braces (“{}”).

A JavaScript function has no return type, and can be any type of value. All functions return a value, and when they don’t, it returns “undefined.”

Example of function:

<script type="text/javascript">

<!--

function functionname(parameter-list){

statements

}

//-->

</script>

Function without name are called Anonymous functions which is another type of JavaScript functions. It is compiled during run time.

The following is an example of an anonymous function:

var sampleHello = function() {

alert("It is a sample");

}

Calling it would be:

sampleHello();

Arrow functions are also anonymous. They make use of the fat arrow “ => “.

Example:

var absValue = (number) => { if (number < 0) { return -number; }

Recursive functions are functions that call itself.

Example:

var x= function sayHello() { // statements go here};

There are three ways a function can call itself:

1. sampleHello
2. arguments.callee()
3. x()

References:

T. (n.d.). JavaScript Functions. Retrieved from

https://www.tutorialspoint.com/javascript/javascript\_functions.htm

Arrow Functions. (n.d.). Retrieved April 5, 2017 from

https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Functions/Arrow\_functions

Functions. (n.d.). Retrieved April 5, 2017 from

https://developer.mozilla.org/en/docs/Web/JavaScript/Guide/Functions

Six Ways to Declare JavaScript Functions. (n.d.). Retrieved April 5, 2017 from https://rainsoft.io/6-ways-

to-declare-javascript-functions/

JavaScript Object Methods. (n.d.). Retrieved April 5, 2017 from

https://www.w3schools.com/js/js\_object\_methods.asp

JavaScript Functions. (n.d.). Retrieved April 5, 2017 from

https://www.w3schools.com/js/js\_functions.asp

Java – Methods. (n.d.). Retrieved April 5, 2017 from

https://www.tutorialspoint.com/java/java\_methods.htm

JavaScript Comparison and Logical Operators. (n.d.). Retrieved April 5, 2017 from

https://www.w3schools.com/js/js\_comparisons.asp

Equality comparisons and sameness. (n.d.). Retrieved April 5, 2017 from

https://developer.mozilla.org/en-

US/docs/Web/JavaScript/Equality\_comparisons\_and\_sameness