MIDTERMS

**<h1>CSS</h1>**

<li>

<ul>The CSS or Cascading Style Sheet was developed by Hakon Wium Lie. </ul>

<ul>The language is used to specify the presentation</ul>

</li>

**<h2>version history</h2>**

<li>

<ul>CSS Level 1 (CSS 1, W3C Recommendation, Dec 1996)</ul>

<ul>CSS Level 2 (CSS 2, W3C Recommendation, May 1998)</ul>

<ul>CSS Level 2 Revision 1 (CSS 2.1, W3C Recommendation June 2018)</ul>

<ul>CSS Level 3

<p>modular approach to CSS development has opposed to the monolithic specification of CSS 2.1</p></ul>

</li>

**<h2>Sources of styles for HTML documents</h2>**

<li>

<ul><h3>author styles<h3>

<li>

<ul>external(s.k.a linked) stylesheets

<p>separating stylesheet to the HTML document</p></ul>

<ul>embedded (a.k.a internal) stylesheets

<p>stylesheet that is located in the header enclosed in the

“<style>” </p></ul>

<ul>inline styles

<p>style that is in the HTML element</p></ul>

</li>

</ul>

<ul><h3>User styles</h3>

</p>It is available to some browser where it provides (non-standard) alternatives through plugins/extensions</p></ul>

**<h2>The source of a file</h2>**

<h3>At - Rules</h3>

<li>

<ul>@charset</ul>

<ul>@import </ul>

<ul>@Namespace</ul>

<ul>@Media</ul>

<ul>@Supports</ul>

<ul>@Page</ul>

<ul>@Font-face</ul>

<ul>@Keyframes</ul>

</li>

**<h2>CSS Rules Sets** ( a.k.a CSS Rules, Styles Rules)</h2>

<p>consists of a selector and then followed a brace-enclosed declaration block that contains zero or more semi-colon separated property.</p>

**<h2>CSS Selectors</h2>**

**<li>**

<ul><h3>Selector</h3>

<p>Structure that is used as a condition in a CSS rule to determine which elements in the document tree are matched</p></ul>

<ul><h3>Selector Syntax</h3>

<p>sequence of simple selectors separated by Combinators</p>

</ul>

<ul><h3>Selector group</h3>

<p>coma-separated list of selectors representing the union of all elements selected by each of the selectors in the list</p></ul>

<ul><h3>Attribute selectors</h3>

<li>

<ul>type-selector</ul>

<ul>univesal selector</ul>

</ul>

**<h2>Combinators</h2>**

<p>Descendant combinators (whitespace , i.e, space tab, line feed, carriage return,form feed)</p>

<li>

<ul>Child combinators(>)</ul>

<ul>Sibling combinator

<li>

<ul>adjacent sibling combinators(+)</ul>

<ul>general sibling combinators(-)</ul>

<div class=”indent”>\*siblings are allowed after the targeted

selector</div>

</ul>

**<h2>pseudo -elements</h2>**

<li>

<ul>:first-letter </ul>

<ul>:first-line</ul>

<ul>:before</ul>

<ul>:after</ul>

</li>

<div class=”otherSide”>

CSS3

<li>

<ul>::first-letter</ul>

<ul>::first-line</ul>

<ul>::before</ul>

<ul>::after</ul>

</li>

</div>

**<h2>Simple Selectors</h2>**

**<li>**

<ul><h3>universal selector(\*)</h3></ul>

<ul><h3>type selector</h3> - different type of selector

ex. p,h1</ul>

<ul><h3>ID selector</h3>

ex. nav, #main, #info</ul>

<ul><h3>class selectors</h3>

ex. .xsample .trial</ul>

<ul><h3>attribute selectors</h3></ul>

ex. *Img[alt]*

*</li>*

<h2>pseudo-classes</h2>

<li>

<ul><h3>dynamic pseudo-classes</h3>

:link, visited</ul>

<ul><h3>target pseudo-classes</h3>

:target (css3)</ul>

<ul><h3>structural pseudo-classes</h3>

:root :empty</ul>

<ul><h3>negation pseudo-classes</h3>

not() &#92;&#92;anything else except</ul>

</li>

**<h2>CSS Rule Precedence</h2>**

<p>An HTML element may be the subject of the selectors of multiple style rules</p>

<h3>Resolution:</h3>

<li>

<ul><h4>by origin and importance</h4>

<div class=”indent”>

!important rule override normal rule</li>

The importance of user and author is reserved in normal declaration</ul>

</div>

<ul><h4>by specificity</h4>

Calculating a selector’s specify</ul>

<ul><h4>by order</h4>

Last is prioritize<ul>

**<h2>CSS Declarations</h2>**

<ul>

<li><h3>properties</h3>

<h4>Shorthand properties</h4>

<p>Allows author specific the values of several properties the values of several properties with a single property ( ex background, border-top)</p></li>

<li><h3>Vendor specific extensions ( a.k.a vendor prefixes )</h3>

<p>Used by browser vendors as a prefix for the names of new experimental or non standards CSS properties, lately , vendors moving only from vendor prefixes in favor of user contrlled flags or preferences</p>

Ex, -webkit-, -moz-, -o-, -ms-</li>

<li><h3>custom properties aka CSS variables (experimental)</h3>

<p>Property names prefixed with --, representing a value that can be reused throughout the document using the var() function</p>

Ex root{--bago: #000012;}<br>

h1{color: var(--bago);}<br>

&#92;&#92;color:var(bago, blue);</li>

<li><h3>values</h3>

<h4>Value processing</h4>

declared , cascaded, specified, computed, used, actual values

<h4>Value types</h4>

<h5>CSS-wide keywords</h5>

<ul>

<li>Initial</li>

<li>Inherit</li>

<li>Unset</li>

</ul>

<h5>Property specific keywords</h5>

</li>

<li><h3>numbers</h3>

<p>integers or reals in (scientific) decimal notation</p></li>

<li><h3>dimensions</h3>

<p>length, angle, duration(or time), frequency, resolution</p>

<ul>

<li>length units

Font-relative: em, cx,ch, rem<br>

Viewport percentage: vw, vh, vmin, vmax<br>

Absolute:cm, mc, in, pt, pc, px</li>

<li>angle units

deg, grad, rad, turn<br>

Used in some gradient and transform functions</li>

<li>duration(or time) units:

s,ms<br>

Used in animation, transition and related protperties</li>

<li>frequency

Hz,kHz<br>

Initially introduces in CSS2 for the (obsolated) aural mediatype

\*reintroduced in (CSS3 but is currently unused</li>

<li>resolution units:

Dpi, dpcm, dppx<br>

Used in media queries

\*svg is resolution proof)</li>

<li>percentages

Number with % suffix <br>

Calculated as a percentiles</li>

</li>

<li><h3>URLs and URIs</h3>

url() function with an absolute or relative</li>

<li><h3>Colors</h3>

<ul>

<li>Color keywords: red</li>

<li>RGB hexadecimal notation:<br>

Ex #ff0000</li>

<li>HSL function<br>

Ex (0,100%,50%)<br>

Color wheel, saturation, lightness</li>

<li>Current color, transparent</li>

</li>

<li><h3>Strings</h3>

<p>delimited be single quotes(‘) or double quotes(“)</p>

Ex: content</li>

<li><h3>functions(CSS3)</h3>

-miscellaneous types</li>

**<h2>CSS Preprocessors, Frameworks, and Polyfills</h2>**

<li><h3>CSS preprocessors</h3>

<p>generate CSS using a custom language syntax that typically includes features that typically includes features that don’t exist in pure CSS(Ex. Variables, control flow, nesting, mixins, functions, etc.)</p>

ex. Sass, Less, Stylus, etc.</li>

<li><h3>CSS Frameworks</h3>

<p>Provides predefined CSS design functionality that can be reused, extended, or customized</p>

Ex. Bootstrap, Foundation, Materialize</li>

<li><h3>Fills</h3>

<p>provide features that developers expect browser to provide natively</p></li>

</ul>

**<h1>JavaScript</h1>**

**<p>Brendan Eich is the inventor of javascript in 1995 and in 1997 becomes an ECMA standard.** a program that runs on a browser environment that allow to manipulate the design of the web page.</p>

Extensions:

<ul>

<li>Defer<br>

Defer the execution of the page/not going to block</li>

<li>Async<br>

Wait for a resource to load and while loading the rest of the page is loaded. When it loads, it can be parsed and loaded</li>

<h2>Javascript Language Fundamentals</h2>

<h3>Variables and constant</h3>

<ul>

<li>var-global scope that becomes a property of the Window object</li>

<li>let-global scope that is not a property of the window object</li>

<li>Const- global constant that does not become a property of the window object</li>

<li>No datatype which is a global reference to window property</li>

<h3>primitive data types(numbers, strings, booleans, null, and undefined)</h3>

<h4>Numbers (in decimal notation)</h4>

Var example = 21; &#92;&#92;decimal integer <br>

exampleB = 0.25; &#92;&#92;decimal floating-point <br>

exampleC = 6.022e23; &#92;&#92;decimal scientific notation

<h4>Number in base 2, 8, and 16</h4>

Var binary = 01101; &#92;&#92;binary numbers <br>

Octal = 0152; &#92;&#92;octal numbers <br>

Hexadecimal = 0xABC &#92;&#92;hexadecimal

<h4>Special numbers</h4>

Positive infinity <br>

Negative infinity <br>

NaN (not a number) <br>

<h4>Strings</h4>

Single = ‘single’; &#92;&#92;single-quote delimited <br>

Double = “double”; &#92;&#92;double-quote delimited <br>

Template = ‘&{first} <br>

&{second} <br>

Third’; &#92;&#92;template literals, that allows multi line <br>

strings and expression interpolation

<h4>Booleans</h4>

<p>0, '', false, null, undefined, and NaN are "false" values.</p> <br>

all other values are "truth" values</p>

<h4>null and undefined</h4>

var nullex=null; <br>

undex = undefined <br>

<h3>Object Types that correspond to the primitive types </h3>

<h4>dates and times</h4>

var ex1 = new Date();

&#92;&#92;show current client date and time<br>

var ex2 = new Date(2018, 8, 23)

&#92;&#92;show specific date<br>

<h4>regular expressions (RegExp object)<h4>

var ex1 = new RegExp('[,.]', 'g'); <br>

var ex2 = /[/ :]/g;

<h4>mathematical functions and constants (Math object)</h4>

<p>Math.sqrt(),Math.PI(),Math.random(),and others</p>

<h4>JavaScript is loose and dynamically-typed for single variable</h4>

var ex=100 &#92;&#92;ex is a number <br>

ex=”letters” &#92;&#92;ex is now a string <br>

ex=true &#92;&#92;ex is now a boolean <br>

<h4>equality and identity or known as strict equality </h4>

1==’1’ &#92;&#92;true <br>

1===’1’ &#92;&#92;false, 3 equals means not equal to <br>

1 != true &#92;&#92;false <br>

1 !== true &#92;&#92;true, exclamation with 2 following = sign means equal to <br>

<h4>implicit type conversions</h4>

1+’2’ &#92;&#92;12 ,adding a string will connect the values<br>

1\*’2’ &#92;&#92;2<br>

1 / false &#92;&#92;Infinity,false is 0<br>

1 + true &#92;&#92;2,true is 1<br>

1 - 'nine' &#92;&#92;NaN,It will not do the operation unless it is a number<br>

1 + null &#92;&#92;1,null is 0<br>

1 + undefined &#92;&#92;NaN,data type not specified<br>

<h4>use object functions for explicit type conversions</h4>

<p>Number(), String()&#92;&#92;and inside the round bracket will be the value which will be parsed</p>

<h3>Functions</h3>

<p>arguments' object that is only valid inside a function</p>

function ex1() { <br>

var result = value[0]; <br>

for (let i = 1; index < value.length; index++) {<br>

result += arguments[index];<br>

}<br>

return result;<br>

}<br>

var res = ex1(1,2,3,4);

<p>default parameter values</p>

Function ex2(a=1,b,c=’a’){<br>

console.log(`a = ${a}, b = ${b}, z = ${z}`);<br>

}

<p>“rest" parameters</p>

Function ex3(example1, example2,example3) <br>

console.log(example1); <br>

console.log(example2); <br>

console.log(example3); <br>

}

<p>function expressions</p>

var ex4 = function(a,b) {return a\*b}; <br>

ex4(2,3);

<p>arrow" function expressions</p>

var ex5 = a => a + a;

<p>nesting functions</p>

function outer{ <br>

Function innerFunc{ <br>

} <br>

}

<h3>Arrays</h3>

var ex = new Array(5); &#92;&#92;the length of the array is specified inside the round bracket <br>

ex[0]=0; <br>

ex[1]=1 <br>

ex[2]=2; <br>

ex[3]=3; <br>

ex[4]=4; <br>

ex[10] &#92;&#92;this is allowed because the array size is dynamic leading the parsed array to null <br>

var arr = new Array(10); &#92;&#92;having “new Array()” will give an array depending on the number inside the round bracket with null value <br>

var arr2=[10]; &#92;&#92;this will lead to 1 element array with value of given <br>

<h3>user-defined objects</h3>

var ex = new Object(); &#92;&#92; Object object constructor<br>

ex.name = Luigi &#92;&#92;properties and methods defining<br>

console.log(ex['name']); &#92;&#92; properties and methods accessing<br>

console.log(ex.unown); &#92;&#92; properties and methods is undefined <br>

for (let property in ex) { &#92;&#92; property enumeration<br>

console.log(`${property} = ${ex[property]}`); <br>

}

<h3>*object literal syntax</h3>*

var xsample = {} <br>

xsample.samp1 = 1; <br>

xsample.samp2 = “two”; <br>

xsample.samp3 = “3”; <br>

var anotherSubject = { <br>

xsample.samp1 = 1; <br>

xsample.samp2 = “two”; <br>

xsample.samp3 = “3”; <br>

}

<h3>Object constructor function</h3>

function xsample2(samp1,samp2 , samp3) { <br>

this.samp1 = 1; <br>

this.samp2 = “two; <br>

this.samp3 = “3”; <br>

<h3>JSON </h3>

<p>allows object conversions JSON-formatted strings</p>

var ex = JSON.stringify(xsample);

<h3>miscellaneous object APIs</h3>

<p>Map(), Set()</p>

<h1>Document Object Model (DOM)</h1>

<ul>

<li>the document itself</li>

<li>the most important API</li>

<li>An API which withn the javascript allow access to the HTML document associated with the webpage</li>

<li>It is represented be a DOM tree which is a parsed document</li>

<h2>DOM API functionalities:<h2>

<h3>Interface Node which retreive references to node collection in the DOM</h3>

<ul>

<li>getElementById()</li>

<li>getElementsByTagName(), getElementsByClassName(), getElementsByName()</li>

<li>querySelector(), querySelectorAll()</li>

</ul>

<h3>From a given node traverse the DOM tree</h3>

<ul>

<li>parentNode, parentElement<li>

<li>childNodes, children<li>

<li>firstChild, lastChild, nextSibling, previousSibling<li>

<li>firstElementChild, lastElementChild, nextElementSibling, previousElementSibling<li>

</ul>

<h3>construct/copy nodes:</h3>

<ul>

<li>createElement(), createTextNode(), createAttribute(), etc.</li>

<li>cloneNode(), importNode()</li>

</ul>

<h3>DOM tree manipulation</h3>

<ul>

<li>nodeType, nodeName, nodeValue</li>

<li>attributes, id, tagName, className, classList</li>

<li>ownerDocument, documentElement, hasChildNodes()</li>

<li>normalize()</li>

</ul>

<h3>miscellaneous attributes and methods</h3>

<ul>

<li>nodeType, nodeName, nodeValue</li>

<li>attributes, id, tagName, className, classList</li>

<li>ownerDocument, documentElement, hasChildNodes()</li>

<li>normalize()</li>

<li>manipulate attributes (for 'Element' nodes):

setAttribute(), getAttribute(), removeAttribute()<br>

setAttributeNode(), getAttributeNode(), removeAttributeNode()

</li>

<li>event handling:

-addEventListener(), removeEventListener(), dispatchEvent()

</li>

<li>style handling:

styleSheets, ownerRule, cssRules, insertRule(), deleteRule()

getComputedStyle()

innerHTML, outerHTML, innerText, outerText

references to oft-used document sections and collections:

head, body, anchors, forms, images

</li>

</ul>

AJAX - provides an object which allows you to fetch something

XML Request

* Create http request object
* Specify resourse
* Do the fetch be using fetch method

\*Javascript is inherently single-threaded\*

Two modes of fetching

1. Synchronous
2. Asynchronous

\*callback function - when events occur, it will be invoked automatically

Event Propagation Order

1. Capturing Phase(Netscape)
2. Target
3. Bubble Phase

Event Handler Registration

Methods:

-Inline HTML event attrib (ex &lt;button id=’button’ onclick=’handler();&gt;CM&lt;/button&gt;

-pre-DOM L2

-DOM:L2 event registration

-standard, well supported

-introduced, addEventListener

-can register more than one event listener

-can register bundles on the bubble or capture phase

-function expressions always have unique identity

- If you register it in capture, close it in capture

-Microsoft-specific, non-standard, IEv6-v10

-determining associated information about event

Javascript promise

&#92;&#92;do operation

&#92;&#92;success resolve(success)

&#92;&#92;failure reject(err)