Web Interfaces

Eric Lecolinet, James Eagan, Cyril Concolato

eric.lecolinet@telecom.paris.fr, james.eagan@telecom-paris.fr

TELECOM ParisTech

Note : to be continued in **IGR203**

2023

Web Applications

Advantages compared to desktop and mobile apps:

- Available on the Web
- No software to install
- Always up-to-date
- Same user interface everywhere (Windows, Mac, Linux, etc.)

Web Technologies

```
<!DOCTYPE html>
<html>
 <head>
  <title>Hello</title>
  <script>
   window.onload=function(e) { alert("Page loaded!"); };
  </script>
  <style type="text/css">
   body { width: 30%; margin: auto; }
   p {
    font-size: 30px;
    font-family: sans-serif;
   }
  </style>
 </head>
 <body>
  <p>
   <img src="../pesto/image01.png" style="float: left; margin-right: 5px" onclick="alert('He
   This is a simple image, but next is a vector graphics image
   <svg style="float: right; width: 100px; height: 100px">
    <rect rx="5" width="50" height="50" fill="lightblue" onclick="alert('Rect click');"/>
   </svg>
  </p>
 </body>
</html>
```

- **HTML / XHTML:**
  Content, structuration, metadata
- **CSS:**
  Presentation, layout,animations...
- **JavaScript / ECMAScript:**
  Programmatic behavior

Web Technologies

```
<!DOCTYPE html>
<html>
 <head>
  <title>Hello</title>
  <script>
   window.onload=function(e) { alert("Page loaded!"); };
  </script>
  <style type="text/css">
   body { width: 30%; margin: auto; }
   p {
    font-size: 30px;
    font-family: sans-serif;
   }
  </style>
 </head>
 <body>
  <p>
   <img src="../pesto/image01.png" style="float: left; margin-right: 5px" onclick="alert('He
   This is a simple image, but next is a vector graphics image
   <svg style="float: right; width: 100px; height: 100px">
    <rect rx="5" width="50" height="50" fill="lightblue" onclick="alert('Rect click');"/>
   </svg>
  </p>
 </body>
</html>
```

- **SVG:** Rich graphical content (vectorial graphics)
- **XML:** Data exchange, validation...
- **JSON:** Data exchange
- etc.

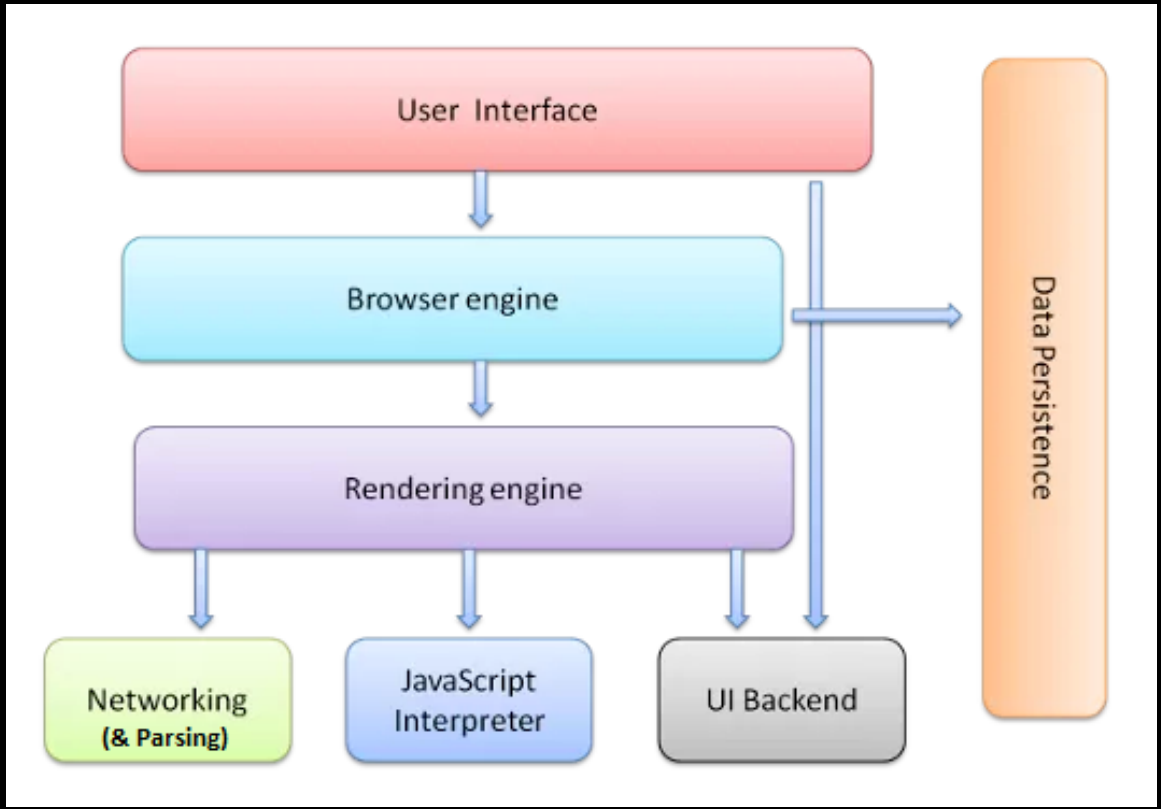
Web browsers

Download documents
using HTTP

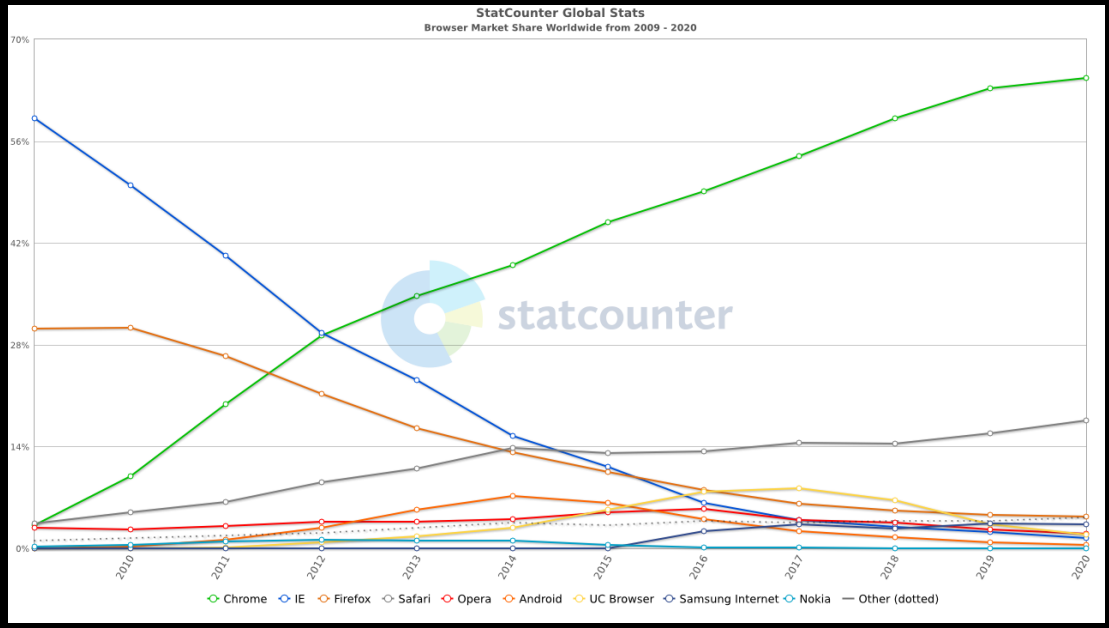
Render the page

Dynamic aspects:

- User interaction
- Animations
- Networking



Web browsers



Browser	Rendering Engine	Scripting Engine
Edge	EdgeHTML	Chakra
Internet Explorer	Trident	Chakra
Firefox and alike (IceWeasel, Seamonkey...)	Gecko	(Spider)Monkey
Safari	WebKit	JavaScript Core
Chrome	Blink (previously WebKit)	V8
Opera	Blink (previously Presto)	V8 (previously Carakan)
Browser	**Rendering Engine**	**Scripting Engine**

HTML

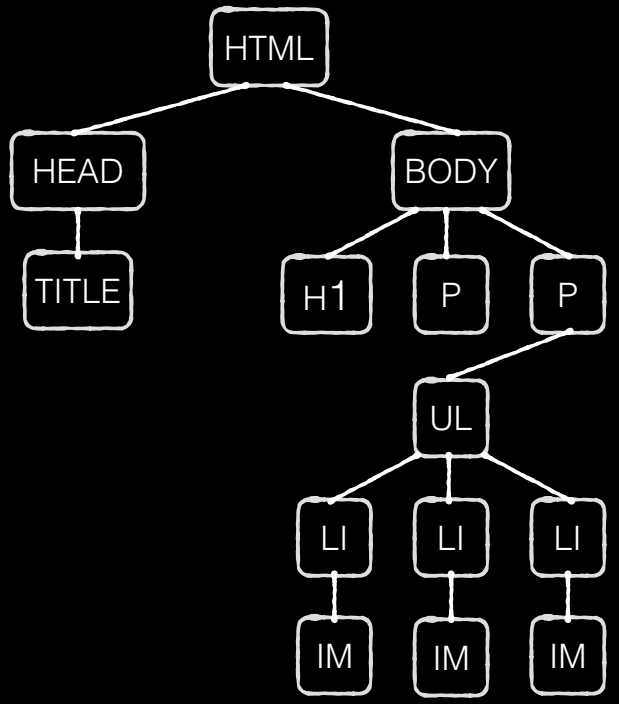
```
<html>
   <head>
      <title>Hello HTML</title>
   </head>

   <body>
      <h1>Hello HTML!</h1>
      <p>Hello HTML. This is an HTML document.</p>
      <p>I like very much :
         <ul>
            <li> Oranges <img src="orange.jpg"> </li>
            <li> Apples <img src="apple.png"> </li>
            <li> Mangoes <img src="mangoe.jpg"> </li>
         </ul>
      </p>
   </body>
</html>
```

Hello HTML!

Hello HTML. This is an HTML document.

I like very much :

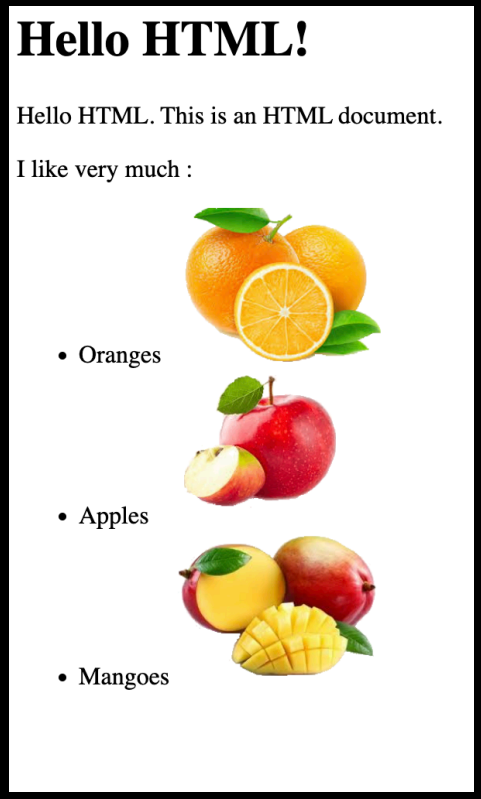
- Oranges 
- Apples 
- Mangoes 



A **HTML** page consists of **elements** organized as a **tree**

HTML

```
<html>
   <head>
      <title>Hello HTML</title>
   </head>

   <body>
      <h1>Hello HTML!</h1>
      <p>Hello HTML. This is an HTML document.</p>
      <p>I like very much :
         <ul>
            <li> Oranges <img src="orange.jpg"> </li>
            <li> Apples <img src="apple.png"> </li>
            <li> Mangoes <img src="mangoe.jpg"> </li>
         </ul>
      </p>
   </body>
</html>
```



- In **HTML**, **most** elements start / end with a **tag**
- In **XHTML**, **all** elements start / end with a **tag**

Head section

```
<html>
   <head>
      <title>Hello HTML</title>
   </head>

   <body>
      <h1>Hello HTML!</h1>
      <p>Hello HTML. This is an HTML document.</p>
      <p>I like very much :
         <ul>
            <li> Oranges <img src="orange.jpg"> </li>
            <li> Apples <img src="apple.png"> </li>
            <li> Mangoes <img src="mangoe.jpg"> </li>
         </ul>
      </p>
   </body>
</html>
```

- **head** = metadata
- (don't confuse with **header**)

- **title** = title of window (or tab)

Head section

```
<head>

<title>My great website</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<link href="file.css" type="text/css" rel="stylesheet"/>

<link rel="stylesheet" media="screen and (max-width: 1280px)" href="file.css" />

<style>
p { color: red; } </style>
</head>

<script src="...">

</head>
```

- **meta** = meta information
  e.g. content encoding
- **link**: related data
  e.g. CSS file

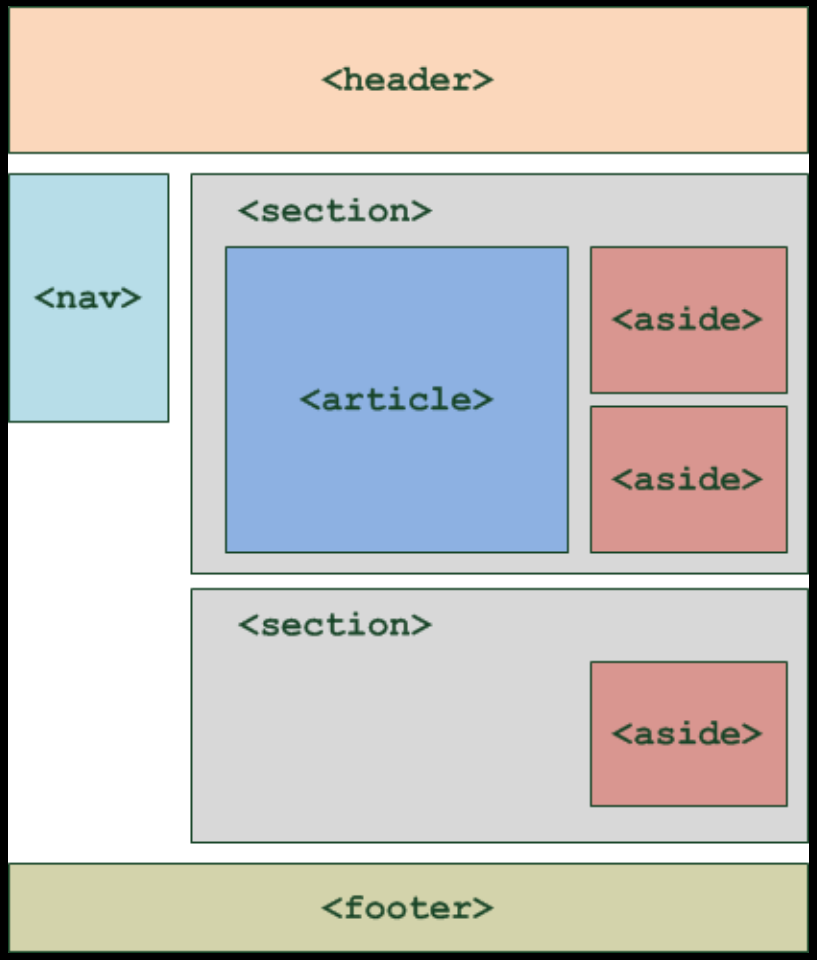
Body

```
<html>
   <head>
      <title>Hello HTML</title>
   </head>

   <body>
      <h1>Hello HTML!</h1>
      <p>Hello HTML. This is an HTML document.</p>
      <p>I like very much :
         <ul>
            <li> Oranges <img src="orange.jpg"> </li>
            <li> Apples <img src="apple.png"> </li>
            <li> Mangoes <img src="mangoe.jpg"> </li>
         </ul>
      </p>
   </body>
</html>
```

- **body** = content of the document
- specifies its **structure**
  - **h1, h2**, etc. titles
  - **p** : paragraph
  - **ul** : list (witout numbers)
  - **li** : list item

Body structure

```
<body>
   <header> <!-- ... --> </header>
   <nav> <!-- ... --> </nav>

   <section>
      <article>
         <!-- ... -->
      </article>
      <aside> <!-- ... --> </aside>
      <aside> <!-- ... --> </aside>
   </section>

   <section>
      <aside> <!-- .. --> </aside>
   </section>

   <footer> <!-- ... --> </footer>
</body>
```



[Cyril Concolato]

Attributes

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>

  <body>
    <h1>Hello HTML!</h1>
    <p>Hello HTML. This is an HTML document.</p>
    <p>I like very much :
      <ul>
        <li> Oranges <img src="orange.jpg"> </li>
        <li> Apples <img src="apple.png"> </li>
        <li> Mangoes <img src="mangoe.jpg"> </li>
      </ul>
    </p>
  </body>
</html>
```

<img src="orange.jpg">

- **img** = an element
- **src** = an attribute

class and *id* attributes

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>

  <body>
    <h1 class="titles" id="main-title">Hello HTML!</h1>
    <p>Hello HTML. This is an HTML document.</p>
    <p>I like very much :
      <ul>
        <li> Oranges <img src="orange.jpg"> </li>
        <li> Apples <img src="apple.png"> </li>
        <li> Mangoes <img src="mangoe.jpg"> </li>
      </ul>
    </p>
  </body>
</html>
```

id

- Identifies a specific (unique) **element**

class

- Identifies a **class name**
- Can be shared by **many** elements

Both are **case-sensitive**

CSS (Cascading Style Sheets)

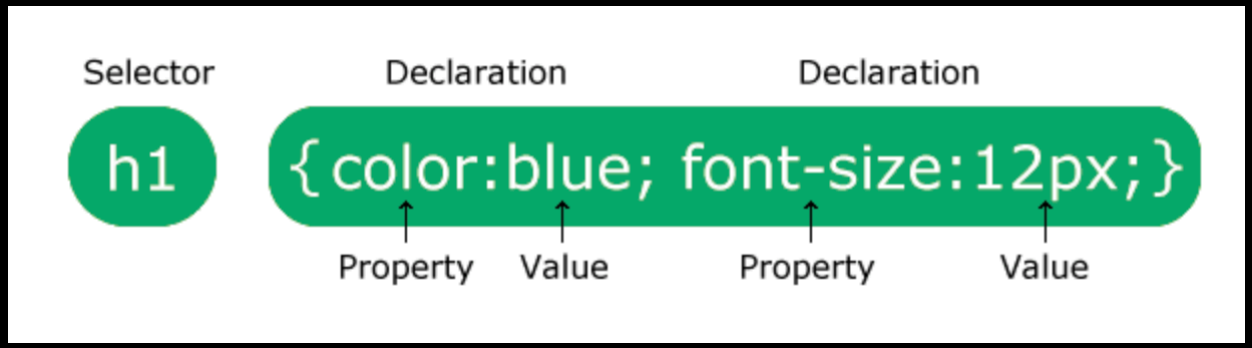
HTML vs. CSS

```html
<html>
  <head>
    <title>Hello HTML</title>
    <link href="file.css" type="text/css" rel="stylesheet"/>
  </head>

  <body>
    <h1 class="titles" id="main-title">Hello HTML!</h1>
    <p>Hello HTML. This is an HTML document.</p>
    <p>I like very much :
      <ul>
        <li> Oranges <img src="orange.jpg"> </li>
        <li> Apples <img src="apple.png"> </li>
        <li> Mangoes <img src="mangoe.jpg"> </li>
      </ul>
    </p>
  </body>
</html>
```

```css
.titles {
    font-family: Georgia, serif;
}

#main-title {
    font-size: 2em;
    text-align: center;
}
```

- **HTML** = content, structure
- **CSS** = presentation, layout, animations
  - **cascaded style sheet** files
  - strong **separation!**

CSS Syntax



https://www.w3schools.com/

```
body, html {
  height: 100%;
}

body {
  font-family: Georgia, serif;
}

h1 {
  font-size: 2em;
  text-align: center;
}
```

selectors can have **many forms!**

Absolute & relative sizes

```
body, html {
  height: 100%;
}

body {
  font-family: Georgia, serif;
}

h1 {
  font-size: 2em;
  text-align: center;
}
```

```
height: 100%;
```

Absolute sizes:
px: pixels
pt: points
cm, **mm**, **in**

```
font-size: 2em;
```

Relative sizes:
em: height of M
ex: height of x
ch: width of 0
%: container percent

Prefer **relative sizes!**

Selectors

```
h1 {
  color: red;
}

.myHeaders {
  color: blue;
}

#firstHeader {
  color: green;
}
```

```
h1.myHeaders {
  color: orange;
}

h1#firstHeader {
  color: brown;
}
```

Selectors can refer to:

- **elements:** *h1*
- **classes** (note the dot): *.myHeaders*
- **ids** (note the #): *#firstHeader*
- **pseudo-classes:** *:active*
- **attributes**

And combinations of them

Pseudo-class selector

```
/* unvisited link */
a:link {
  color: #FF0000;
}

/* visited link */
a:visited {
  color: #00FF00;
}

/* mouse over link */
a:hover {
  color: #FF00FF;
}

/* selected link */
a:active {
  color: #0000FF;
}
```

Attribute Selectors

```css
/* Les éléments <a> avec un attribut title */
a[title] {
  color: purple;
}

/* Les éléments <a> avec un href qui correspond */
/* à "https://example.org" */
a[href="https://example.org"] {
  color: green;
}

/* Les éléments <a> dont href contient "example" */
a[href*="example"] {
  font-size: 2em;
}
```

attributes can be selected depending of their **value**

various **operators**

Combinators

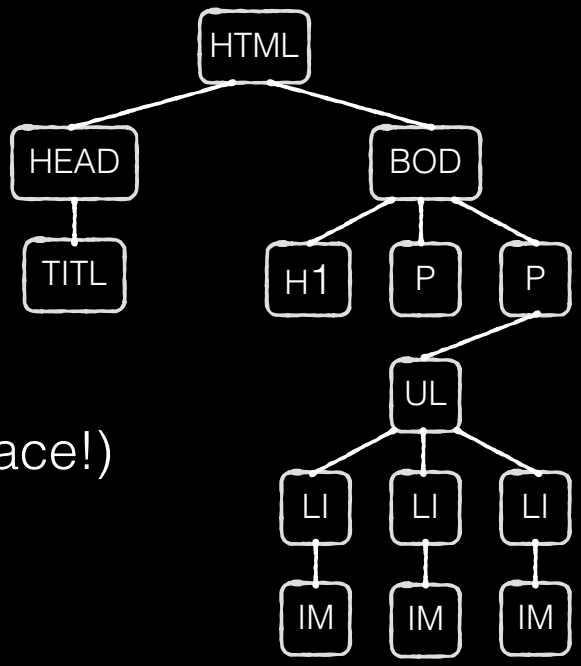
```
ul li {
  color: red;
}

.myHeaders li {
  color: blue;
}

h1.myHeaders li {
  color: green;
}
```

Elements inside elements

- **ul li** descendants (note the space!)
- **ul > li** direct children

Elements after elements

- **div + p** all p immediately after div
- **p ~ ul** all ul preceeded by p



Selector	Example	Example description
.class	.intro	Selects all elements with class="intro"
.class1.class2	.name1.name2	Selects all elements with both *name1* and *name2* set within its class attribute
.class1 .class2	.name1 .name2	Selects all elements with *name2* that is a descendant of an element with *name1*
#id	#firstname	Selects the element with id="firstname"
*	*	Selects all elements
element	p	Selects all <p> elements
element.class	p.intro	Selects all <p> elements with class="intro"
element,element	div, p	Selects all elements and all <p> elements
element element	div p	Selects all <p> elements inside elements
element>element	div > p	Selects all <p> elements where the parent is a element
element+element	div + p	Selects all <p> elements that are placed immediately after elements
element1~element2	p ~ ul	Selects every element that are preceded by a <p> element

Combinators

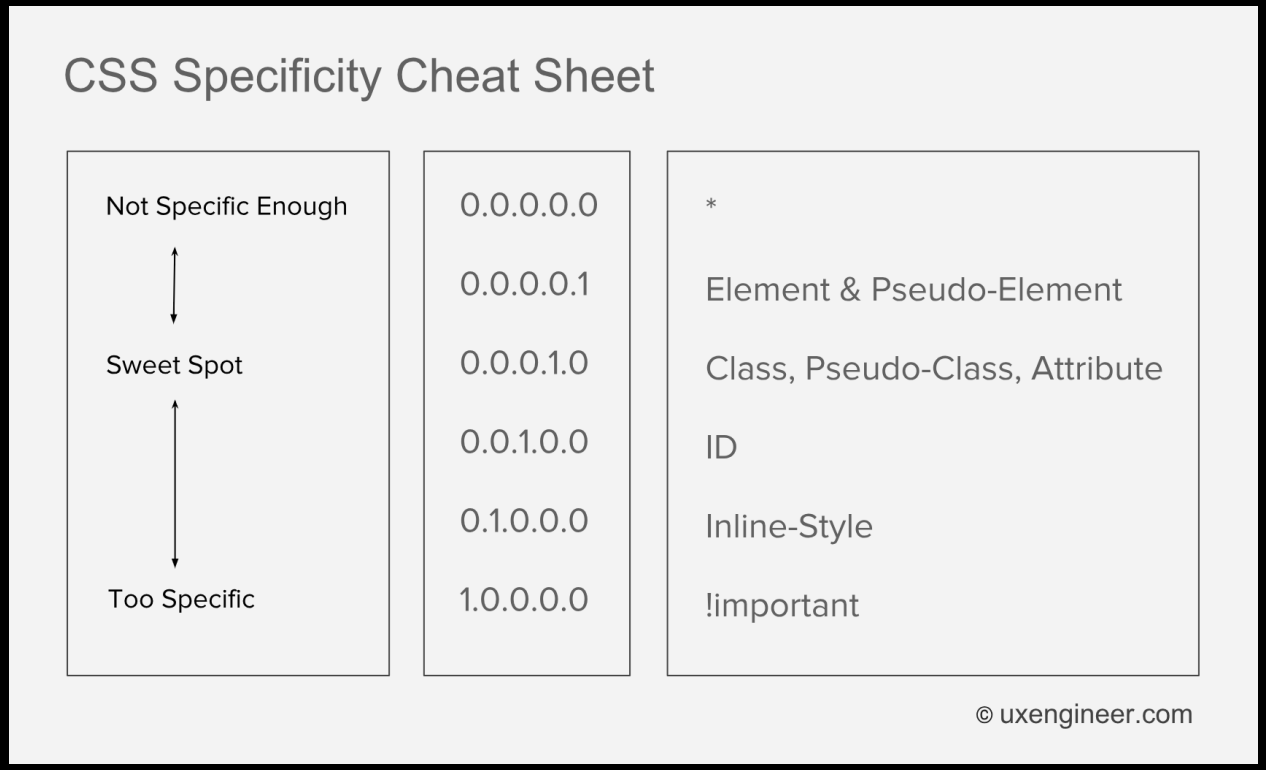
More info:

https://developer.mozilla.org/

https://www.w3schools.com

https://www.w3schools.com

CSS Specificity



More specific
= higher priority

ex: increasing priority:

```
ul li {
    color: red;
}

.myHeaders li {
    color: blue;
}

h1.myHeaders li {
    color: green;
}
```

See: https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity()

Inline styles

```
<h1 id="toto" style="color:pink">Hello HTML!</h1>
```

- Styles normally in separate **style sheet** files
- Can also be:
  - in **head** section
  - in elements = **inline** styles (discouradged)

CSS Inheritance

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>

  <body>
    <h1>Hello HTML!</h1>
    <p>Hello HTML. This is an HTML document.</p>
    <p>I like very much :
      <ul id="my-list">
        <li> Oranges <img src="orange.jpg"> </li>
        <li> Apples <img src="apple.png"> </li>
        <li> Mangoes <img src="mangoe.jpg"> </li>
      </ul>
    </p>
  </body>
</html>
```

```
#my-list {
   font-family: Georgia, serif;
    color: blue;
}
```

some properties are **inherited** in the **instance tree**

ex : font, color

CSS Inheritance

- For a given element, if the value for a given property is **not specified**, the value is obtained as follows:
  - if the property is "inheritable" (i.e. "inherited: yes" in its definition),
    - if the element has a parent in the DOM tree, the **computed value** on that parent is use

```
p { color: green }
```

```
<p>The text and the span will be <span>green</span> because 'color' is inheritable.
```

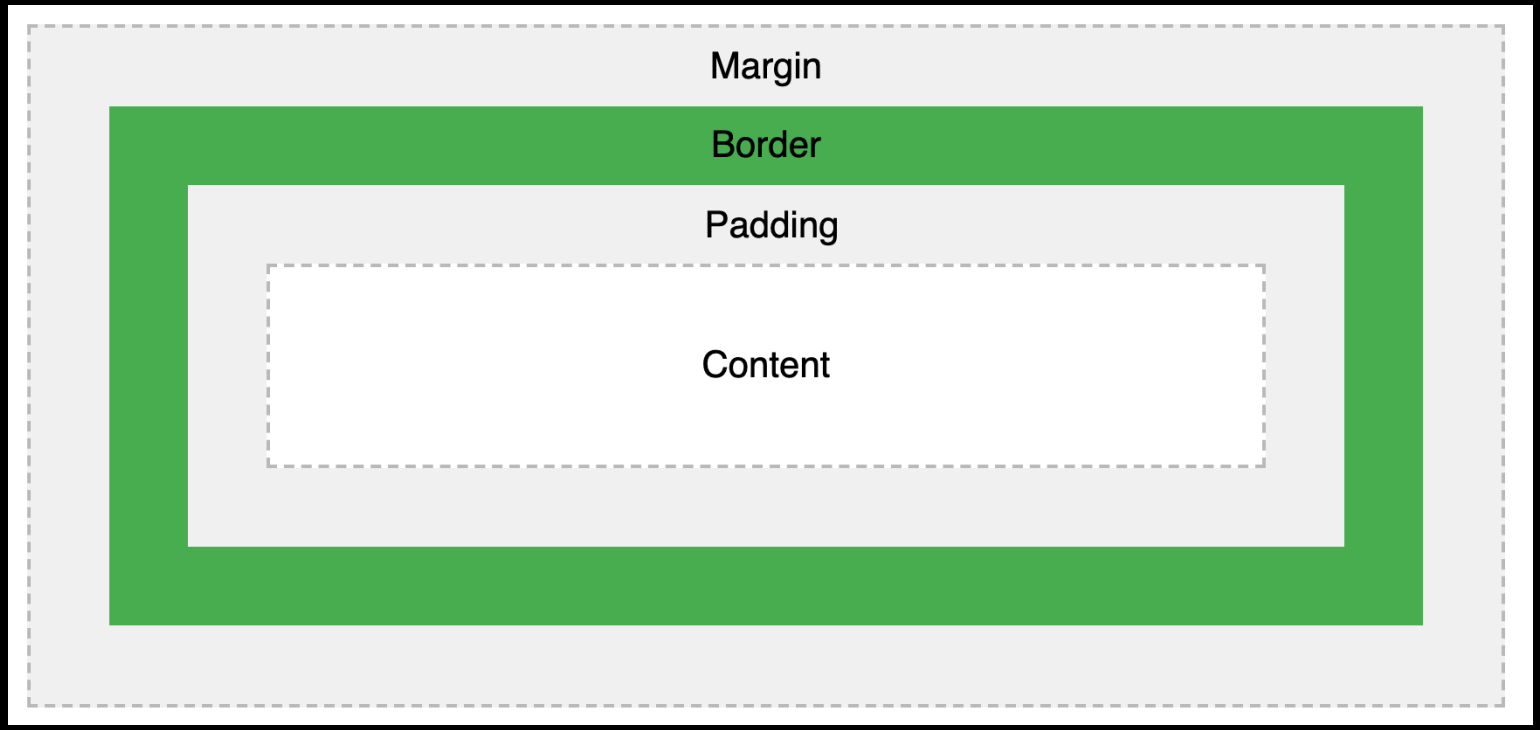
    - otherwise (for the root), the **initial value** is used.
  - if not (i.e. "inherited: no"), the **initial value** is used

```
p { border-width: 1px }
```

```
<p>Only the text will have <span>a border</span> because 'border-width' is not inhe
```

- The computed value is obtained:
  - by converting a relative value (when possible) to an absolute value
  - otherwise (% values when layout is involved), using the relative value

CSS Box model



Page layout

display property (main values):

- **block**
  - starts new line, uses full page width
  - **ex:** , <p>, <h1>
- **inline**
  - in currrent flow
  - **ex:** , <b>, <i>,

```
<p>
This is a paragraph containing <span style="color:red">a span</span>.
A span is <b>inline</b> (so as the "b" element)
</p>
```

This is a paragraph containing a span. A span is **inline** (so as the "b" element)

Page layout

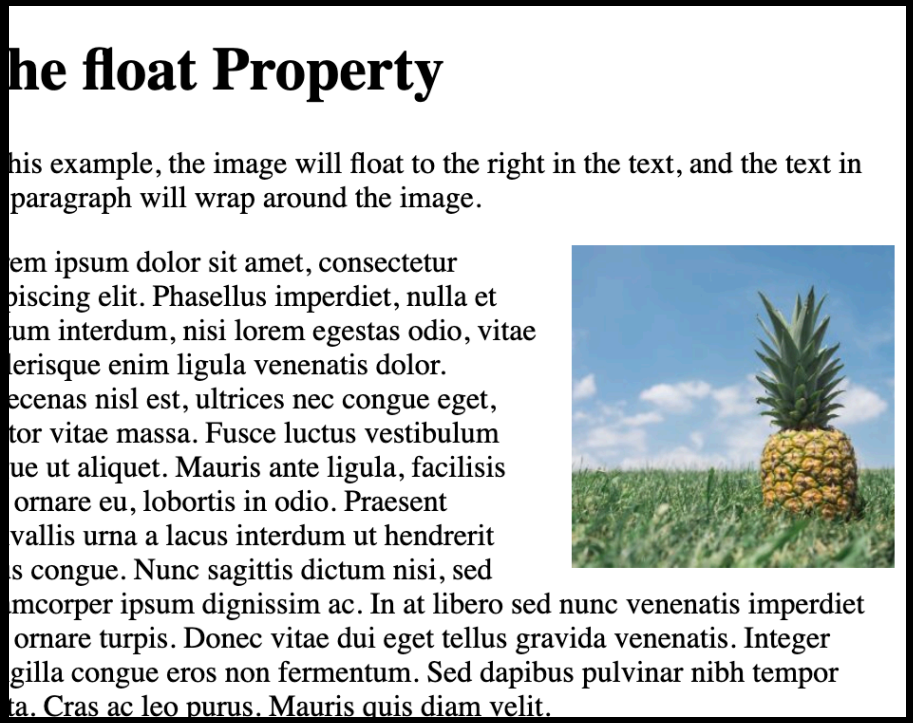
float: left

float: right

- lets an element float left (or right) to the text in a container

clear

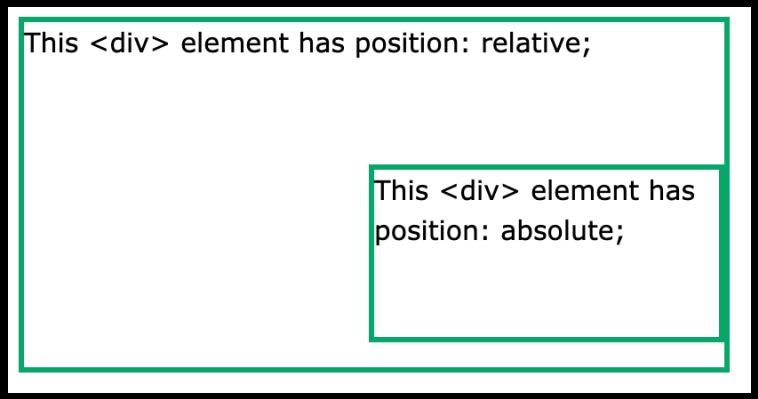
- for clearing **float**s



Page layout

position

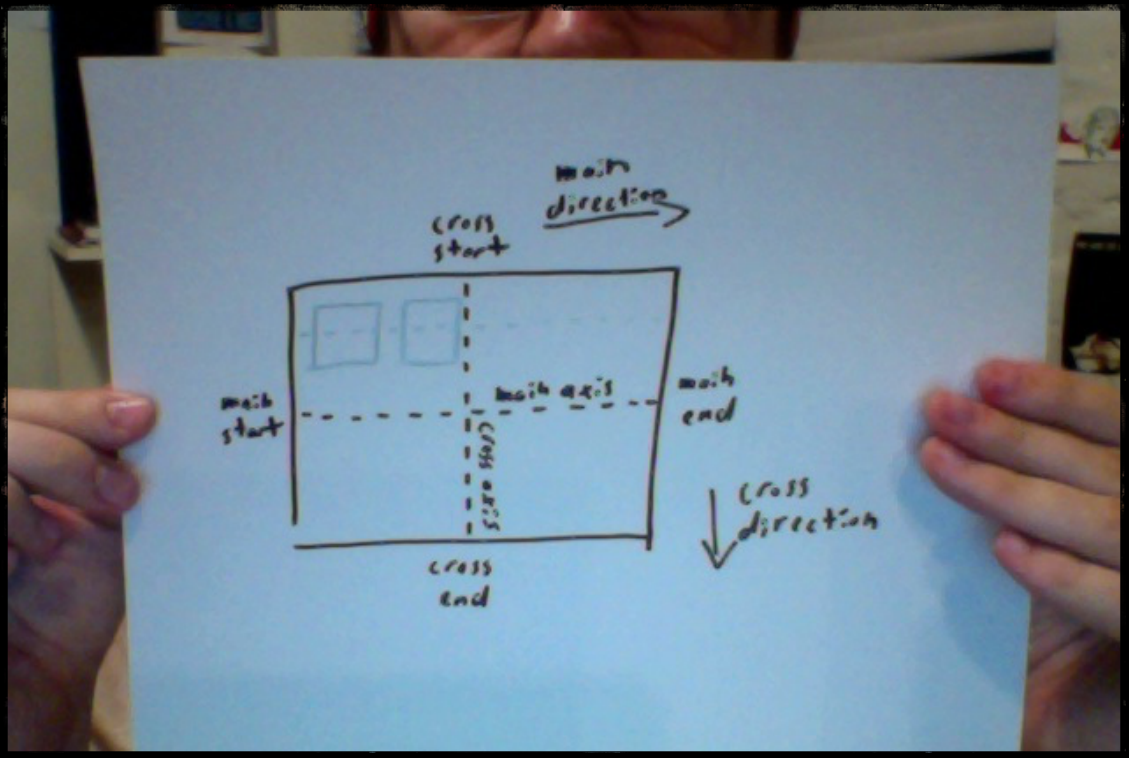
- **static** (default)
- **fixed :** relative to window
- **relative :** relative to normal position (shifted)
- **absolute :** relative to first non static ancestor
- **sticky** : depends on user's scroll position



Flexbox

display: flex[-inline]

- For containers & Items
- Each descendant is a **flex item**
- **Main** and **cross** axes



[https://bocoup.com/blog/dive-into-flexbox]

Direction

Main / Cross direction

flex-direction: row-reverse;

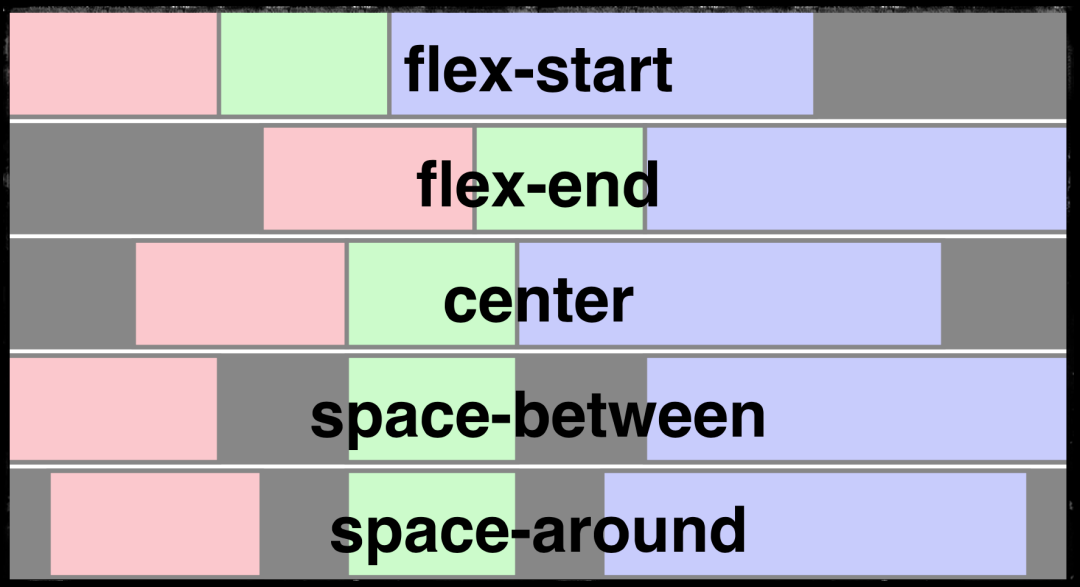
flex-direction: column-reverse;

Main alignement

Position on Main axis

justify-content:

one among :

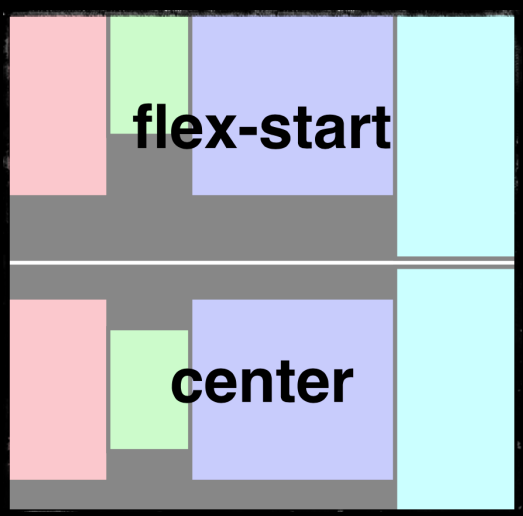


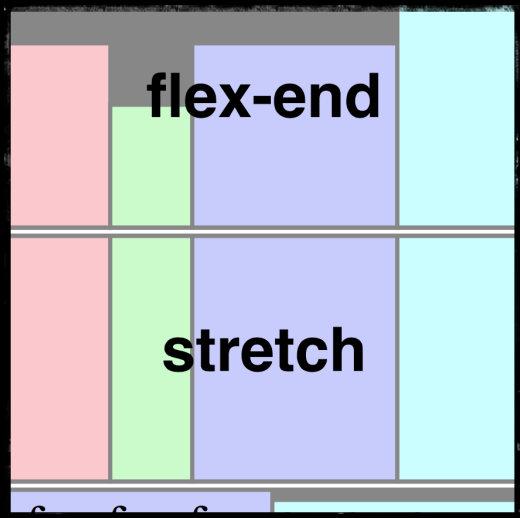
Cross alignement

Position on **Cross** axis

align-items:

one among :

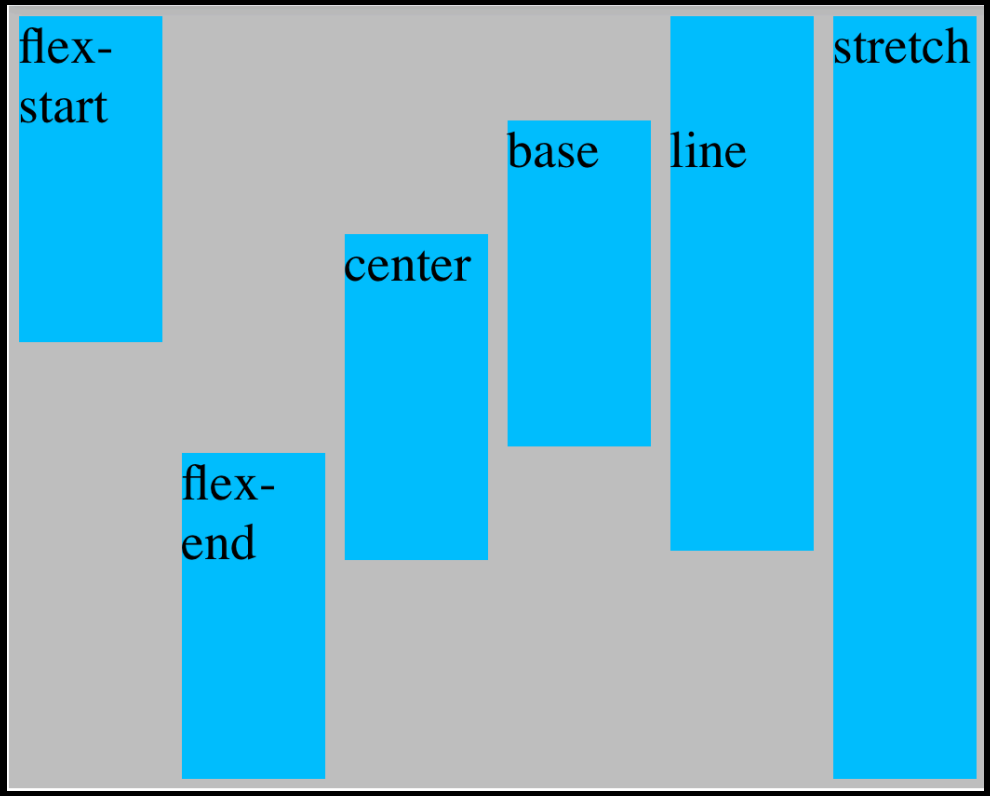




Individual alignment

For a specific item

align-self:
one among:

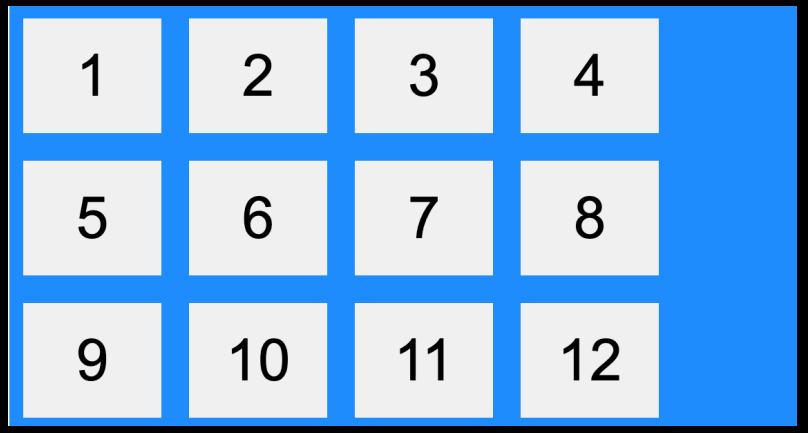


[https://jsfiddle.net/gsmith/REtEG/2/]

Wrap and flex-flow

Line breaks

flex-wrap: nowrap;
flex-wrap: wrap;
flex-wrap: wrap-reverse;



flex-flow = flex-direction + flex-wrap

flex-flow: *direction wrap*;

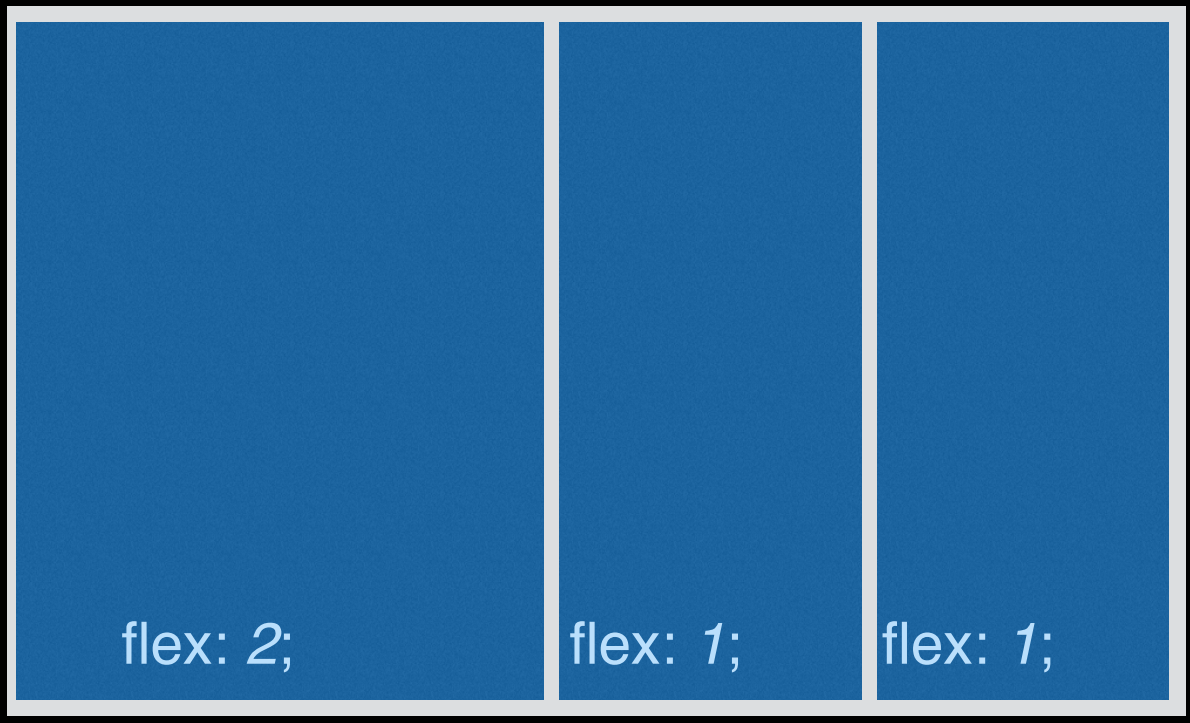
flex-flow: row wrap-reverse;

Sea also: **align-content**:
controls **lignes** instead of **items**

Flex

Relative size compared to siblings

flex: *grow*;
flex: *grow shrink basis*;
flex: initial;
flex: auto;
flex: none;



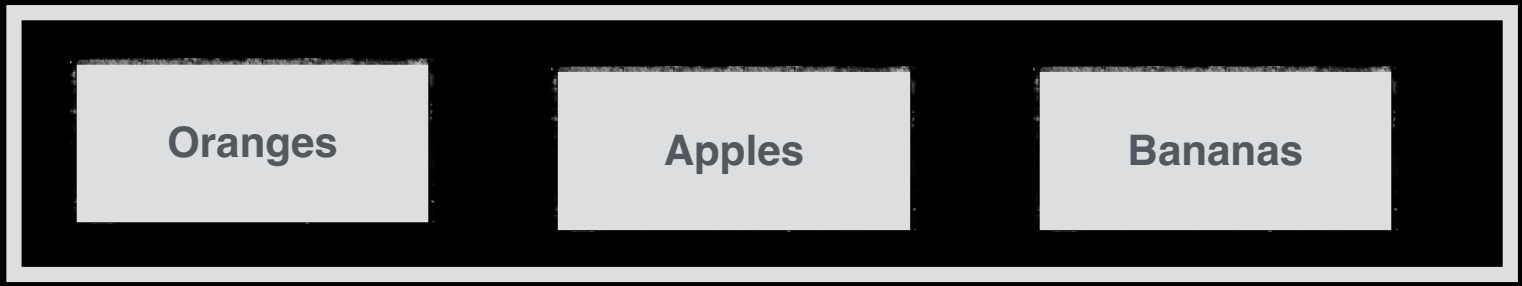
For more detail see: https://developer.mozilla.org/fr/docs/Web/CSS/flex

Order

Changes painting order

order: *int*;

```
<item>Apples</item>
<item>Bananas</item>
<item class="my-favorite">Oranges</item>

.my-favorite { order: -1; }
```



Margin

margin: *size*;
margin: auto;

[https://jsfiddle.net/gsmith/sQAFY/2/]

Integrated debuggers

- **Source Code** Inspector (HTML, CSS ...)
- **Advanced** Inspectors (DOM Tree...)
- **JavaScript** debug
- **Network** monitoring
- **Performance** (frame rate, CPU,

- Chrome Dev Tools
- Mozilla Firebug
- Microsoft Developer Tools
- Safari Developer tools

JavaScript / ECMAScript

- Two eras: before vs. after **ECMAScript 6**
- In this document: version >= **ECMAScript 6**

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello HTML!</h1>
    <p>Hi HTML.  Ceci est un exemple d'un document HTML.</p>
    <script>
      const adjectives = ["glamorous", "evil", "nimble", "tragic", "weird"];
      const animals = ['aardvark', 'cobra', 'eel', 'flamingo', 'wombat'];

      function specialAnimal() {
        let adjective = adjectives[ Math.floor(Math.random() * adjectives.length) ];
        let animal = animals[ Math.floor(Math.random() * animals.length) ];
        return `${adjective}-${animal}`;
      }

    function updateName(name) {
      const titleTag = document.getElementsByTagName("title")[0];
      const headerTag = document.getElementsByTagName("h1")[0];

      titleTag.textContent = "Hi " + name;
      headerTag.textContent = `Hi ${name}`;
    }
    </script>
    <button onclick="updateName(specialAnimal());">Who am I?</button>
  </body>
</html>
```

Variables and functions

```
let euros = 1;
const rate = 1.221;

function eurosToDollars(euros) {
    return euros * rate;
}
```

let

declares a variable in the **current scope**

const

value **can't be changed**

types are inferred **dynamically**

Note: in C++ / Java they can be inferred **statically**

Variables and functions

```
euros = 1;
var rate = 1.221;

function eurosToDollars(euros) {
    return euros * rate;
}
```

old syntax (before ECMAScript 6):

- ***no keyword***
  global variable
- **var**
  function or global scope

prefer **let** and **const** when possible

Tables

```
const adjectives = ["glamorous", "evil", "nimble", "tragic", "weird"];
const animals = ['aardvark', 'cobra', 'eel', 'flamingo', 'wombat'];

function specialAnimal() {
    let adjective = adjectives[ Math.floor(Math.random() * adjectives.length) ];
    let animal = animals[ Math.floor(Math.random() * animals.length) ];

    return `${adjective}-${animal}`;
}
```

adjectives.length

`${*adjective*}-${*animal*}`

Element's content

```
function updateName(name) {
    const titleTag = document.getElementsByTagName("title")[0];
    const headerTag = document.getElementsByTagName("h1")[0];

    titleTag.textContent = "Hi " + name;
    headerTag.textContent = `Hi ${name}`;
}
```

```
<head>
    <title>Hello HTML</title>
</head>

<body>
    <h1>Hello HTML!</h1>
    ....
```

To retrieve an element:

- getElementsById()
- getElementsByTagName()

Element's content:

- **textContent**: text format
- **innerHTML**: HTML format

Element's content

```
headerTag.textContent = `Hi <i>${name}</i>`;
```

Hi evil-wombat

```
headerTag.innerHTML = `Hi <i>${name}</i>`;
```

Hi *evil-wombat*

Element's content:

- **textContent**: text format
- **innerHTML**: HTML format

Events

```
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello HTML!</h1>
    <p>Hi HTML.  Ceci est un exemple d'un document HTML.</p>
    <script>
      const adjectives = ["glamorous", "evil", "nimble", "tragic", "weird"];
      const animals = ['aardvark', 'cobra', 'eel', 'flamingo', 'wombat'];

      function specialAnimal() {
        let adjective = adjectives[ Mat
        let animal = animals[ Math.flo
        return `${adjective}-${animal}`
      }

    function updateName(name) {
      const titleTag = document.getElementsByTagName("title")[0];
      const headerTag = document.getElementsByTagName("h1")[0];

      titleTag.textContent = "Hi " + name;
      headerTag.textContent = `Hi ${name}`;
    }
    </script>
    <button onclick="updateName(specialAnimal());">Who am I?</button>
  </body>
</html>
```

```
function updateName(name) {
    .....
}

<button onclick="updateName(specialAnimal());">Who am I?</button>
```

Events

Alternative techniques

addEventListener()

```
<script type="application/ecmascript" >
  function doSomething(evt) { … }
</script>
<text onclick="doSomething(evt)" >Hello World!</text>
```

```
<script type="application/ecmascript" >
 function doSomething(evt) { … }
 e=document.getElementById('T');
 e.addEventListener('click', doSomething, false);
</script>
<text id="T" >Hello World!</text>
```

```
<script type="application/ecmascript" >
 function doSomething(evt) { … }
 e=document.getElementById('T');
 e.onclick=doSomething;
</script>
<text id="T" >Hello World!</text>
```

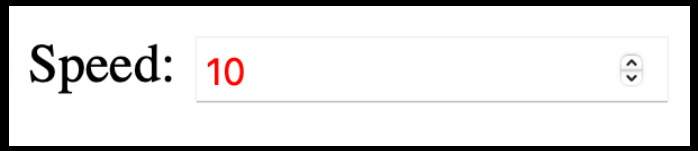
Events

- Mouse Events
  - click, mousedown, mouseup, mouseover, mousemove, mouseout
- Key Events
  - keypress, keyrelease
- Touch events
  - touchstart, touchend, touchleave, touchmove, ...
- Drag events
  - dragstart, dragend, ...
- Network events
  - load, error, abort, progress
- Form events
  - submit, focus ...
- Media events
  - play, pause ...

Event Attributes

https://www.w3schools.com/tags/ref_eventattributes.asp

Input element




```
function foo() {
    const setting = document.getElementById("setting");
    const setting_value = setting.value;
}

<label>Speed
    <input type="number" id="setting" value="10" oninput="foo()">
</label>
```

Value of an **input** element: **value** (not textContent!)

Event when text is entered: **oninput**

Getting property values

```
function foo() {
    const toto = document.getElementById("toto");

    let color = toto.style.color;
    let x = toto.style.left;
    let w = toto.style.width;
}
```

style attribute = **properties** of an element

Correct?

Getting property values

```
function foo() {
    const toto = document.getElementById("toto");

    let color = toto.style.color;
    let x = toto.style.left;
    let w = toto.style.width;
}
```

```
#my-list {
    font-family: Georgia, serif;
    color: blue;
}
```

style attribute = **properties** of an element

What about style sheets and inherited properties ?

Getting property values

```
function foo() {
    const toto = document.getElementById("toto");
    const style = getComputedStyle(toto);

    let color = style.color;
    let x = parseInt(style.left);
    let w = parseInt(style.width);
}
```

- **getComputedStyle**() : **computed** (actual) values of the properties
- parseInt() : converts string into integer

Callback functions

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Functions **given as an argument** to another function

Animations

```
let intervalID;

function startAnim() {
    intervalID = setInterval(foo, 50);

    function foo() {
        // etc.
    }
}

function stopAnim() {
    clearInterval(intervalID);
}
```

Start / stop an animation

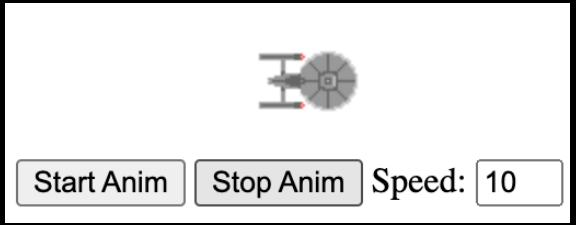
- **setInterval**() : calls **repeatedly**
- **clearInterval**() : cancels

Or:

- **setTimeout**() : calls **once**
- call again at each step

Animations

```
let intervalID;

function startAnim() {
    intervalID = setInterval(foo, 50);
    const toto = document.getElementById("toto");
    toto.style.left  = "0px";

    function foo() {
        let x = toto.style.left;
        x += 10;
        toto.style.left = x;
    }
}
```



left = horizontal position

note the **unit** (px)

Correct?

Animations

```
let intervalID;

function startAnim() {
    intervalID = setInterval(foo, 50);
    const toto = document.getElementById("toto");
    toto.style.left  = "0px";

    function foo() {
        let x = parseInt(toto.style.left);
        x += 10;
        toto.style.left = x + "px";
    }
}
```

parseInt() needed

missing unit

left = horizontal position of **positioned** elements

static *by default*
=> no effect!

position

- **static** (default) : relative to normal flow
- **fixed :** relative to **window**
- **relative :** relative to **normal** position
- **absolute :** relative to **first non static** ancestor
- **sticky** : depends on user's scroll position

Animations

```
let intervalID;

function startAnim() {
    intervalID = setInterval(foo, 50);
    const toto = document.getElementById("toto");
    toto.style.left  = "0px";

    function foo() {
        let x = parseInt(toto.style.left);
        x += 10;
        toto.style.left = x + "px";
    }
}
```

```
#toto {
    position: relative;
}
```

Possible solutions

- *toto* **position** = **relative**
- *toto position* = **absolute**
  if parent not **static**

Animated properties

```
#toto {
    position: absolute;
    color: red;
    left: 0px;
    transition: color 2s, left 4s;
}

#toto:hover {
   color: blue;
   left: 500px;
}

 <h1 id="toto">Hello HTML!</h1>
```

Transitions

- Change property values **smoothly**
- Over a given **duration**

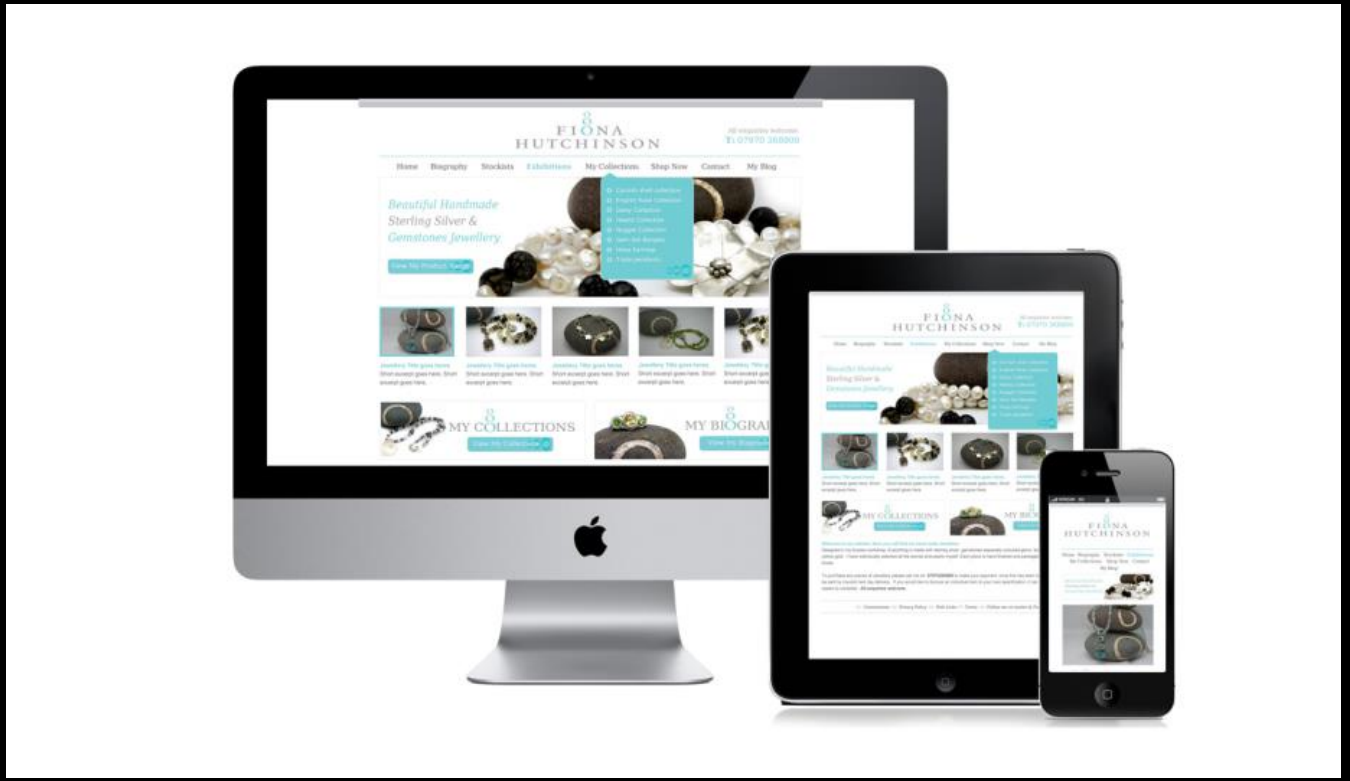
Animated properties

```
@keyframes myanim {
    from {color: red; left: 0px;}
    50% {color: green;}
    to {color: blue; left: 500px;}
}

#toto {
    position: absolute;
    animation: myanim 5s infinite;
}

 <h1 id="toto">Hello HTML!</h1>
```

Keyframes

- Rule that specifies an **animation**
- Change property values according to the specified **steps**

Responsive design



Pages **adapt** to the screen using **CSS Media Queries**

CSS Media queries

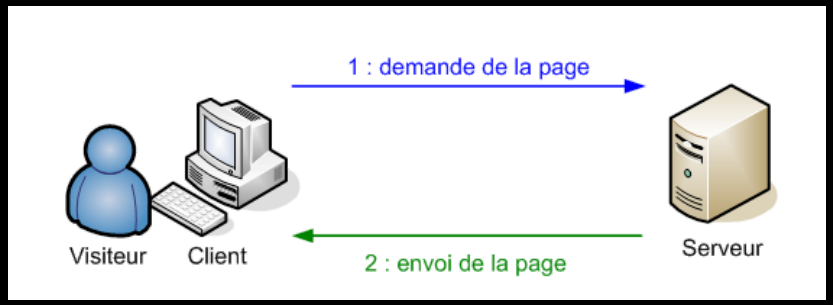
```
/* Set the background color of body to tan */
body {
  background-color: tan;
}

/* On screens that are 992px or less, set the background color to blue */
@media screen and (max-width: 992px) {
  body {
    background-color: blue;
  }
}

/* On screens that are 600px or less, set the background color to olive */
@media screen and (max-width: 600px) {
  body {
    background-color: olive;
  }
}
```

Layout adapts depending on client characteristics:

- screen size
- resolution
- type of device
- etc.
-

HTTP

- **Client / server** protocol
- Download resources identified by an **URL**



UNIFORM RESOURCE LOCATOR

- Initially standardized by IETF (new versions in development by IETF/W3C/WHATWG)

`https`	`://`	`www.example.com`	`:443`	`/path/to/doc`	`?name=foo&town=bar`	`#para`
scheme		hostname	port	path	query	fragment

- scheme:
  - way the resource can be accessed; generally http or https, but also ftp, data, ws, wss, ..
- hostname:
  - domain name of a host (cf. DNS); hostname of a website may start with www., but not rule.
- port:
  - TCP port; defaults: 80 for http and 443 for https
- path:
  - logical path of the document for the server, with/without extensions or with extension corresponding to content generated content (e.g. php)
- query string:
  - optional additional parameters (dynamic documents)
- fragment:
  - optional subpart of the document

HTTP messages

- How data is **exchanged** between **client** and **server**
- Two types: **requests** and **responses**
- Textual information encoded in **ASCII**
- Several HTTP "**methods**"

Message type = Requests or responses

- Request=Method+URL+ProtocolVersion+Header(+data)
- Method
  - GET
  - POST
  - HEAD
  - OPTIONS
  - PUT
  - DELETE
  - TRACE
  - CONNECT
  - PATCH
- Response=ProtocolVersion+Response Code+Header+Resource

GET

- Simplest type of request.
- Possible parameter are sent at the end of a URL, after a '?'
  - Not applicable when there are too many parameters, or when their values are too long (total length < 2000 chars).

- Example:
  - URL in the browser

```
http://www.google.com/search?q=hello
```

  - Corresponding HTTP Request

```
GET /search?q=hello HTTP/1.1
Host: www.google.com
```

POST

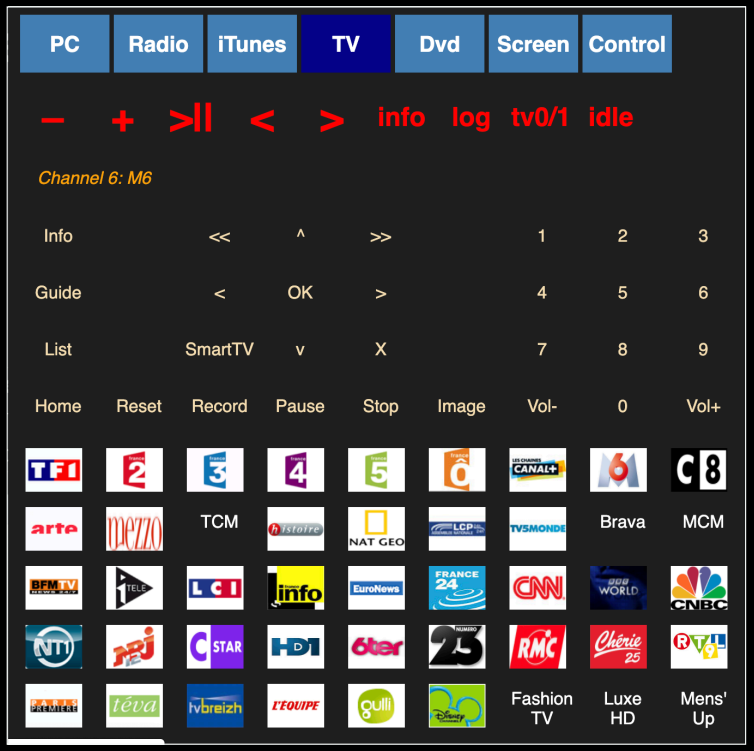
- Method only used for submitting forms.
- Example:

```
POST /php/test.php HTTP/1.1
Host: www.w3.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 100
type=search&title=The+Dictator&format=long&country=US
```

- By default, parameters are sent using: `name1=value1&name2=value2`
  - special characters (accented characters, spaces…) are replaced by codes such as +, %20
  - This way of sending parameters is called `application/x-www-form-urlencoded`.
  - Also used with GET requests in the URL: `http://www.example.org?name1=value1&name2=value2`

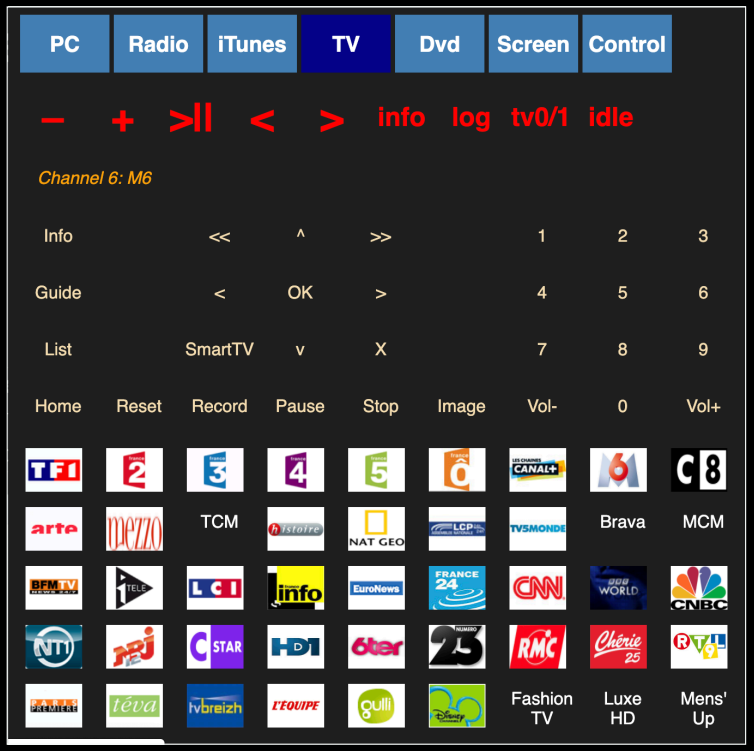
Client / server example

- **Web interface** that controls a **program** that changes TV channels, etc.
- Exchanges data with a **Perl script** running on an **Apache server**



Client / server example

- The **Perl script**:
  - **receives** a command from the Web page
    + **sends** it to the *program*
  - **receives** a response from the *program*
    + **sends** it back to the Web page



Client: HTML + Javascript

javascript

```
function command(cmd) {
    let req = new XMLHttpRequest();
    req.onreadystatechange = function() {
        if (req.readyState == 4 && req.status == 200) {
            let status = document.getElementById("status");
            status.textContent = req.responseText;
        }
    }
    try {
        req.open("GET", "/cgi-bin/myservice.pl?cmd=" + encodeURIComponent(cmd), true);
    }
    catch (e) {
        alert(e);
    }
    req.send();
}
```

this function is called when the response is received from the Perl script

html

```
<div id="status">Status</div>
<button onclick="command('play:TF1')">TF1</button>
<button onclick="command('play:France2')">France 2</button>
<button onclick="command('volume+')">Vol +</button>
...etc...
```

Server: Perl script

myservice.pl

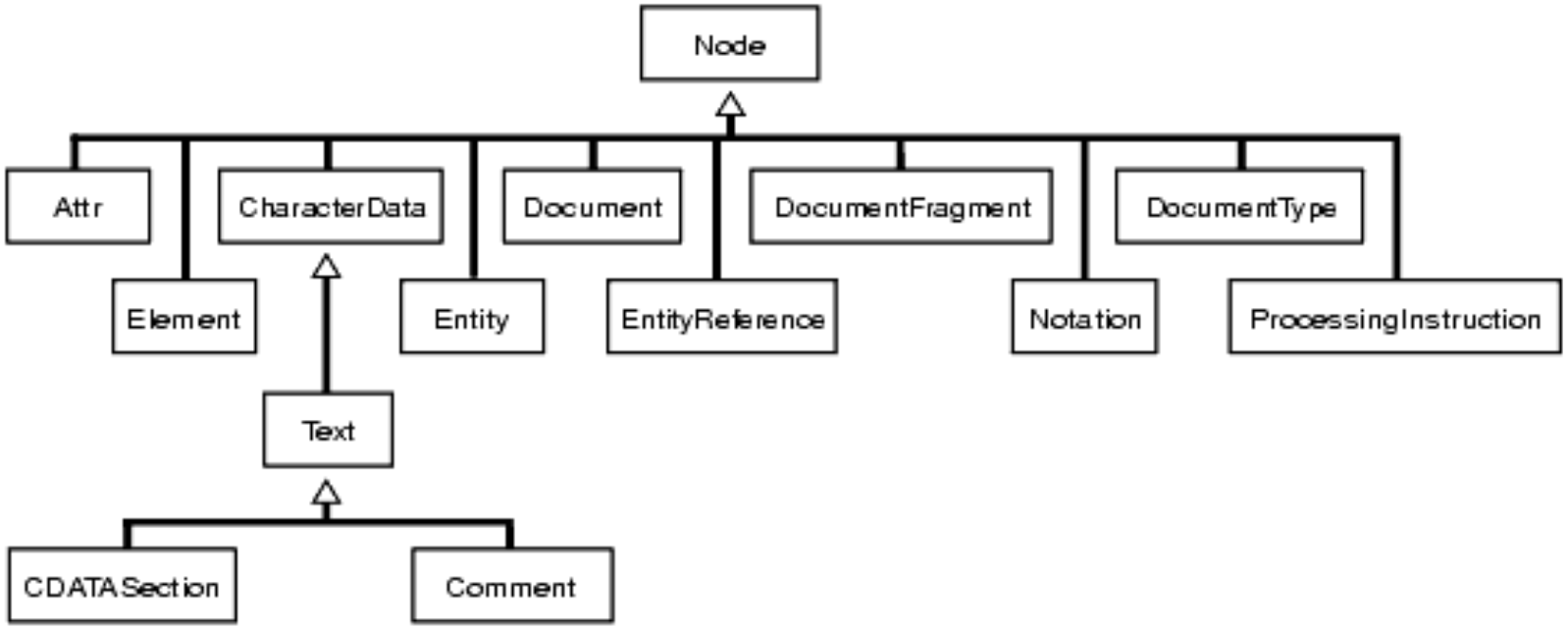
```perl
#!/usr/bin/perl

use strict;
use warnings;
use CGI;
use IO::Socket;

my $query = CGI->new;
my $cmd = $query->param('cmd');
my $socket = IO::Socket::INET->new(Proto    => "tcp",
                                   PeerAddr => "localhost",
                                   PeerPort => 7722)

or die "Failed: $@\n";

print $socket "?$cmd\n";
my $response=<$socket>;

print $query->header;
print $response;
close $socket;
```

```
req.open("GET", "/cgi-bin/myservice.pl?cmd=" + encodeURIComponent(cmd), true);
```

communicates via a socket with the program

DOM (document object model)

DOM INTERFACES HIERARCHY



e Node interface

```
nodeType
parentNode
firstChild
nextChild
firstSibling
hasChildNodes()
hasAttributes()
appendChild()
removeChild()
```

e Document interface

```
documentElement
getElementById()
getElementsByTagName()
querySelector()
createElement()
```

e Element interface

```
innerHTML
getAttribute
setAttribute
removeAttribute
```

Adding an element

The page before

```
<html>
  <body>
    <p id="someid">some new text</p>
  </body>
</html>
```

The JS code

```
var obj = document.getElementById("someid");
obj.innerHTML = "some <span style='color: red;'>other</span> text";
```

The page after

```
<html>
  <body>
    <p id="someid">some <span style="color: red;">other</span> text</p>
  </body>
</html>
```

Adding an element

The page before

```
<html>
  <body>
  </body>
</html>
```

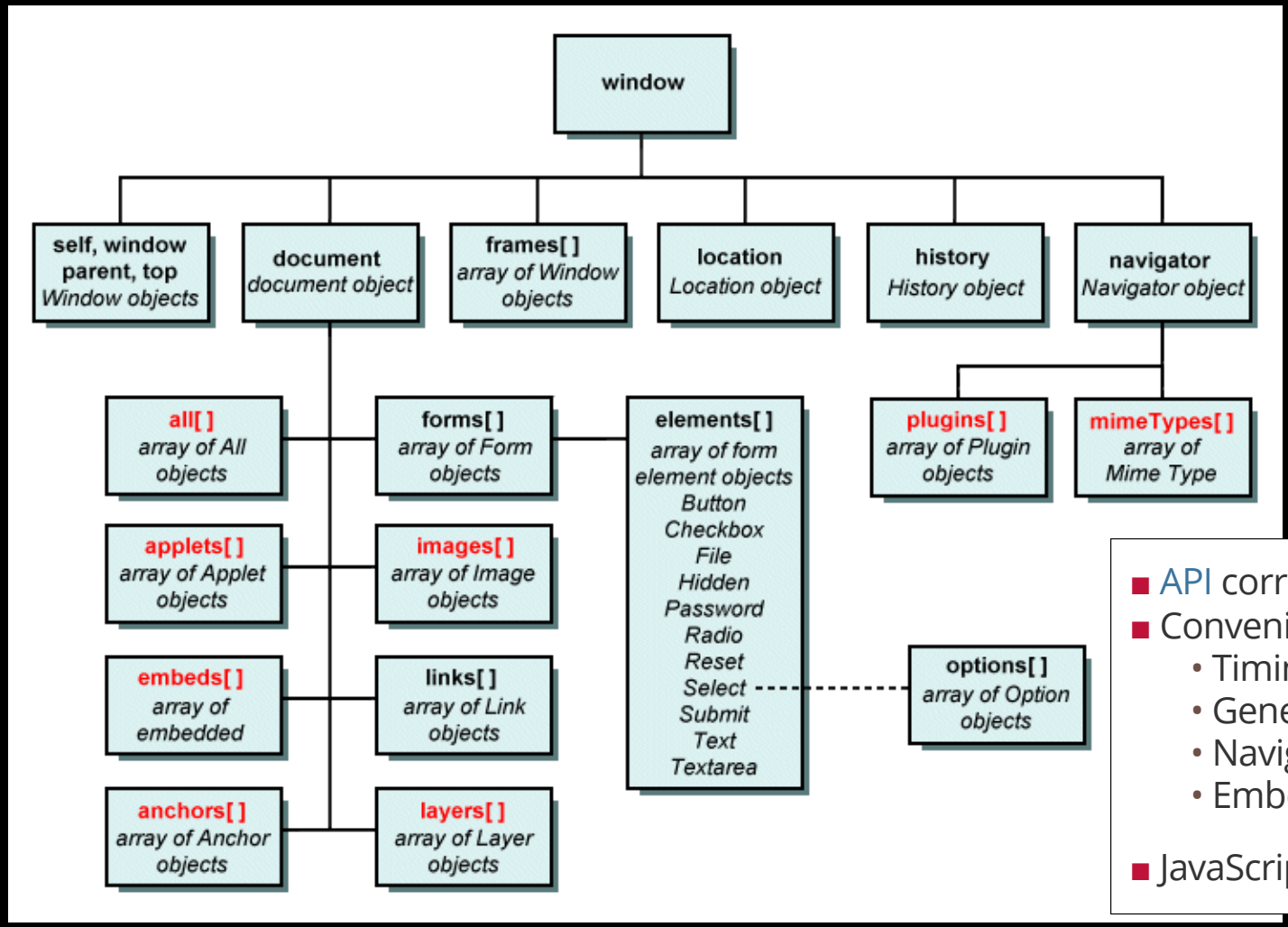
The JS code

```
var obj = document.createElement("p");
obj.textContent="some new text";
var body = document.getElementsByTagName("body")[0];
body.appendChild(obj);
```

The page after

```
<html>
  <body>
    <p>some new text</p>
  </body>
</html>
```

The Window object



- API corresponding to the browser window or tab
- Convenient API for various usages
  - Timing (animations)
  - General events (load, ...)
  - Navigation (history)
  - Embedding (openURL)
- JavaScript `global` object in browser

Tools and code playgrounds

Platforms & libraries

- JQuery
- Angular
- Bootstrap

Code playgrounds

- JS Fiddle: https://jsfiddle.net/
- Code Pen: https://codepen.io/
- CSS Deck: https://cssdeck.com/
- JSBin: http://jsbin.com/?html,output

Links

Lab:
https://perso.telecom-paristech.fr/elc/web

Tutorials and manual pages:
https://www.w3schools.com/css/
https://developer.mozilla.org/

INF203 Courses:
https://perso.telecom-paristech.fr/dufourd/cours/inf203/#/slides