



Le Web

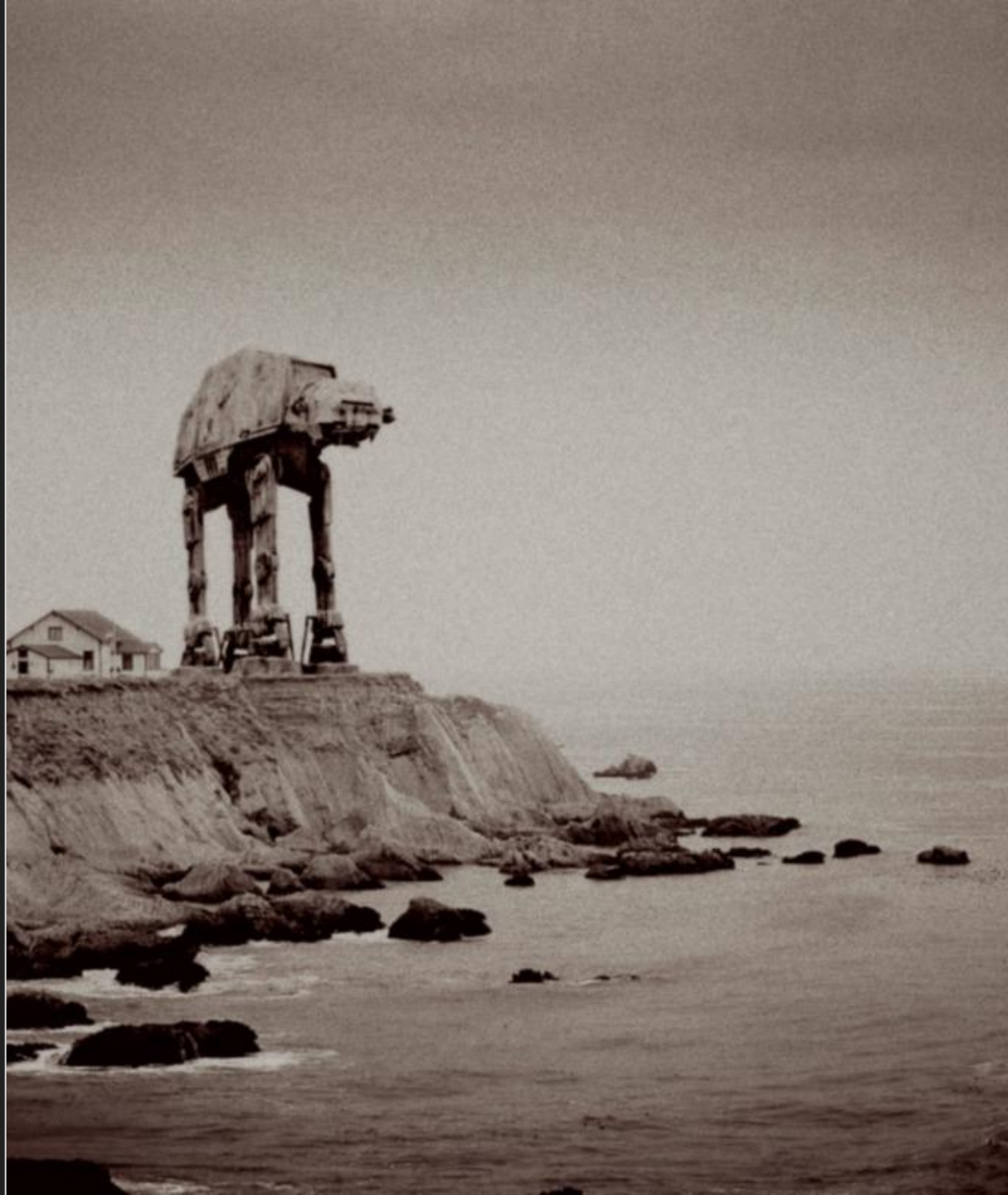
HTML, CSS, Angular 2

Xavier MARIN

@XavMarin

<https://github.com/Giwi>

CTO chez @qaobee



Le Web

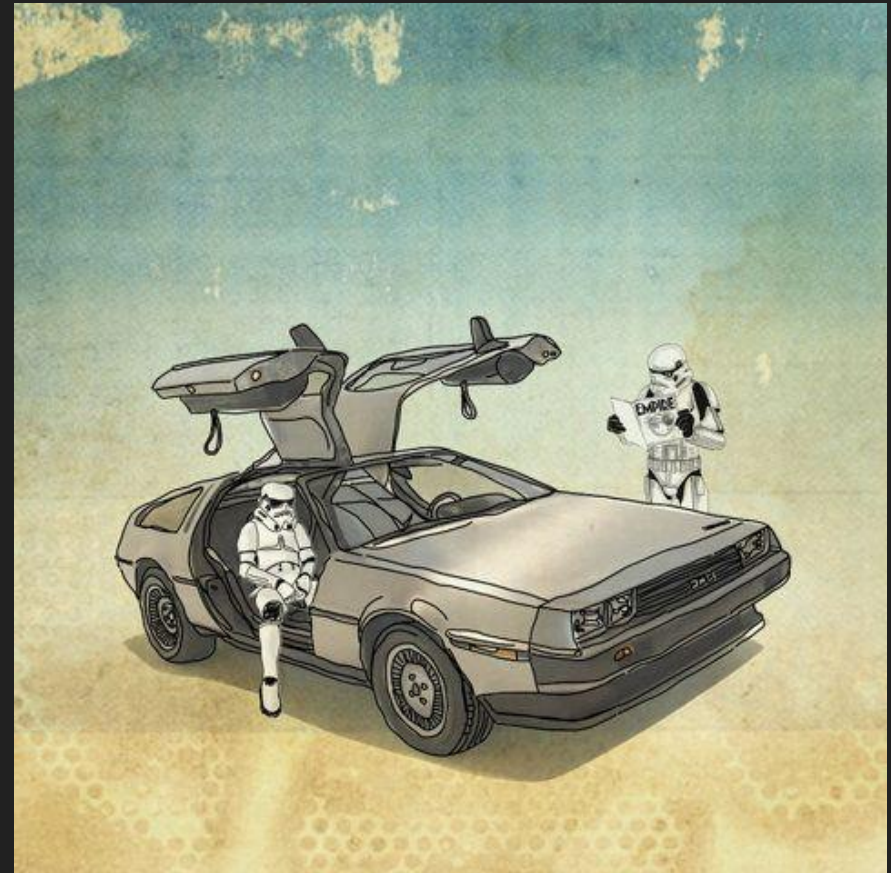
- Un peu d'histoire
- Le Web
- HTML
- CSS
- Angular 2



Un peu d'histoire

La préhistoire

- -3000 : première représentation binaire
- -500 : l'abaque et le boulier
- 1580 : les premiers logarithmes
- 1642 : première machine à calculer de Pascal
- 1728 : métier à tisser avec des cartes perforées
- 1792 : le télégraphe optique
- 1838 : le télégraphe électrique
- 1867 : la machine à écrire
- 1889 : calculatrice de bureau
- 1943 : MARK1 (3 opérations /secondes)



Un peu d'histoire

Les débuts

- 1951 : notion de compilateur
- 1958 : le COBOL
- 1960 : premier multi-tâches et premier micro-ordinateur
- 1963 : la souris
- 1969 : Arpanet (4 nœuds) et premier microprocesseur, UNIX
- 1970 : le C
- 1971 : Arpanet (23 nœuds)
- 1981 : le PC, MSDOS
- 1984 : 1000 nœuds sur Internet
- 1987 : 10000 nœuds sur Internet



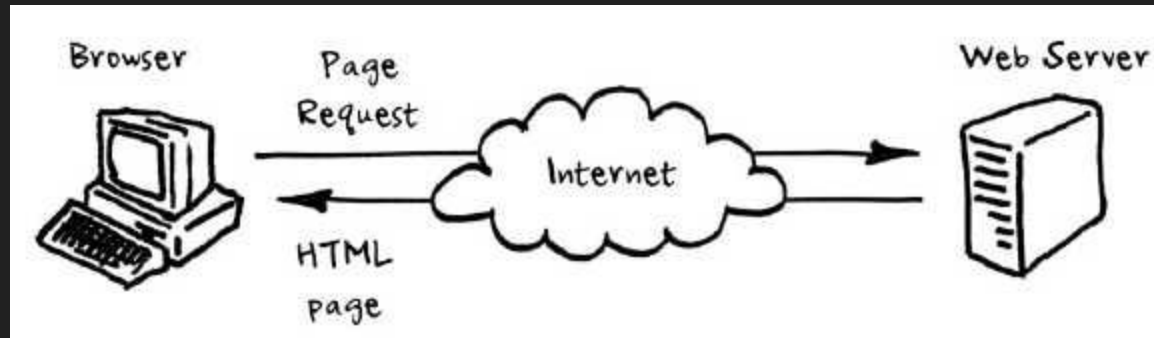
Un peu d'histoire

L'essor

- 1991 : création du protocole HTTP, Linux
- 1992 : 1 million d'ordinateurs sur Internet
- 1993 : premier navigateur Internet
- 1995 : Java
- 1996 : 10 millions d'ordinateurs sur Internet



HTTP



HTTP

- Un client (navigateur ou machine) émet une requête HTTP
- Un serveur est à l'écoute de la requête et l'analyse
- Le serveur envoie au client une réponse HTTP

`<scheme>://<user>:<password>@<domain>/<path>?param1=value1¶m2=value2#<fragment>`

`http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html`

`ftp://gilbert:toto75@ftp.nsa.gov.us/etc/passwd`

`https://www.google.fr/search?q=star+wars&client=ubuntu&hs=cKq&channel=fs&source=lnms&tbm=isch&sa=X&ved=0CAgQ_AUoAmoVChMI7PLn1dn8yAIVA7oUCh1VFw8E&biw=1366&bih=639#channel=fs&tbm=isch&q=death+star`

Une réponse HTTP

Une réponse contient des en-têtes et éventuellement un corps.

```
Status Code: 200 OK
Cache-Control: max-age=21600
Content-Length: 11464
Content-Type: text/html; charset=iso-8859-1
Date: Fri, 06 Nov 2015 22:22:52 GMT
Expires: Sat, 07 Nov 2015 04:22:52 GMT
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
Server: Apache/2
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml'>
<head><title>HTTP/1.1: Request</title></head>
<body><address>part of <a rev='Section' href='rfc2616.html'>Hypertext Transfer Protocol --
HTTP/1.1</a><br />
RFC 2616 Fielding, et al.</address>
<h2><a id='sec5'>5</a> Request</h2>
<p>A request message from a client to a server includes, within the first line of that message,
the method to be applied to the resource, the identifier of the resource, and the protocol
version in use.</p> ...
```

Les status :

- 1xx
 - purement informatif
- 2xx
 - ok, tout va bien navette
- 3xx
 - souvent une redirection
- 4xx
 - ce ne sont pas les droïds que vous cherchez
- 5xx
 - le serveur a un souci

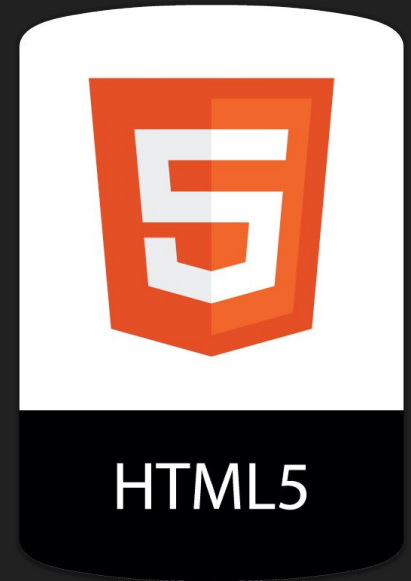
Editeur HTML

- N'importe quel éditeur de texte (sauf Word ;))
- Atom : <https://atom.io/>
- Visual Studio Code : <https://code.visualstudio.com/>
- Sublime text : <https://www.sublimetext.com/>
- Dreamweaver : <http://www.adobe.com/fr/products/dreamweaver.html>
- IntelliJ/Webstorm : <https://www.jetbrains.com/webstorm/>
- Eclipse : <http://www.eclipse.org/>
- CofeeCup, NVU, ...



HTML

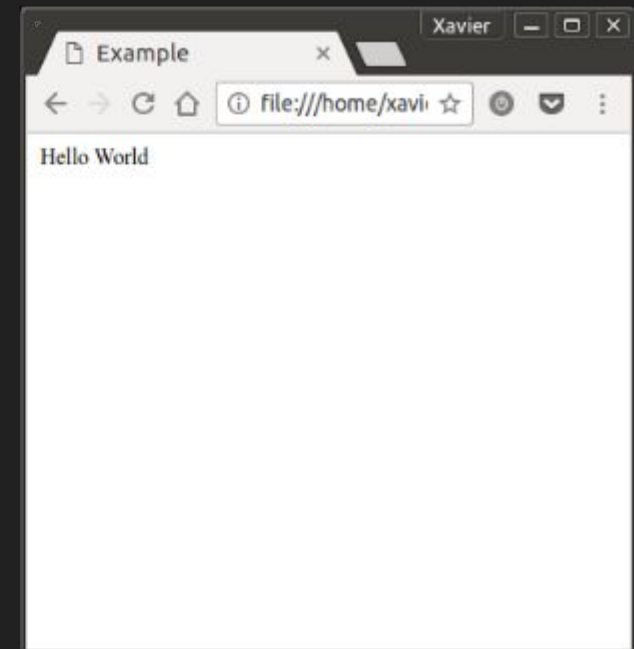
- Langage de description balisé dérivé de XML
 - Insensible à la casse !!!!
- Structure l'affichage de composants graphiques
 - Textes
 - Champs de saisie
 - Média
 - ...
- Interprété par un navigateur
- Normé par le W3C
- HTML = **H**yper **T**ext **M**arkup **L**anguage



Structure

- `<html>`
 - Tous les documents ont cette balise
- `<head>`
 - Contient des méta données et/ou des imports
- `<body>`
 - Le corps du document
- `<title>`
 - Titre de la page

```
er/tmp - Atom
demo.html
1 <html>
2   <head>
3     <title>Example</title>
4   </head>
5   <body>
6     Hello World
7   </body>
8 </html>
9
```

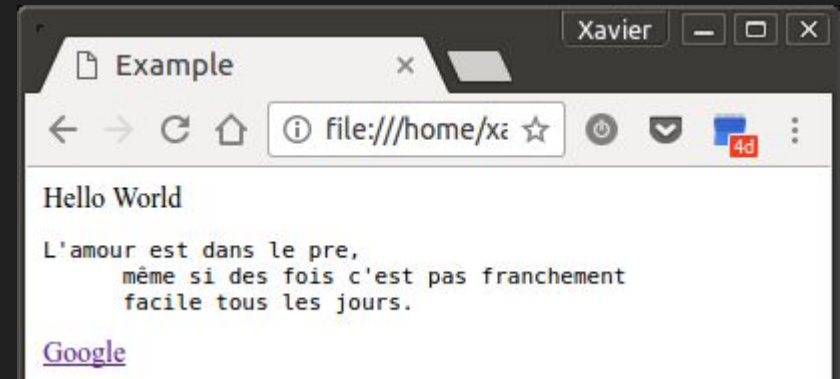


Les balises

`<balise attribut1 attribut2= « valeur » >Contenu</balise>`

```
Atom - demo.html
1 <html>
2   <head>
3     <title>Example</title>
4   </head>
5   <body>
6     Hello World
7     <!-- Ceci n'est pas un commentaire -->
8     <pre width=50>L'amour est dans le pre,
9       même si des fois c'est pas franchement
10      facile tous les jours.</pre>
11     <a href="http://www.google.com">Google</a>
12   </body>
13 </html>
14
```

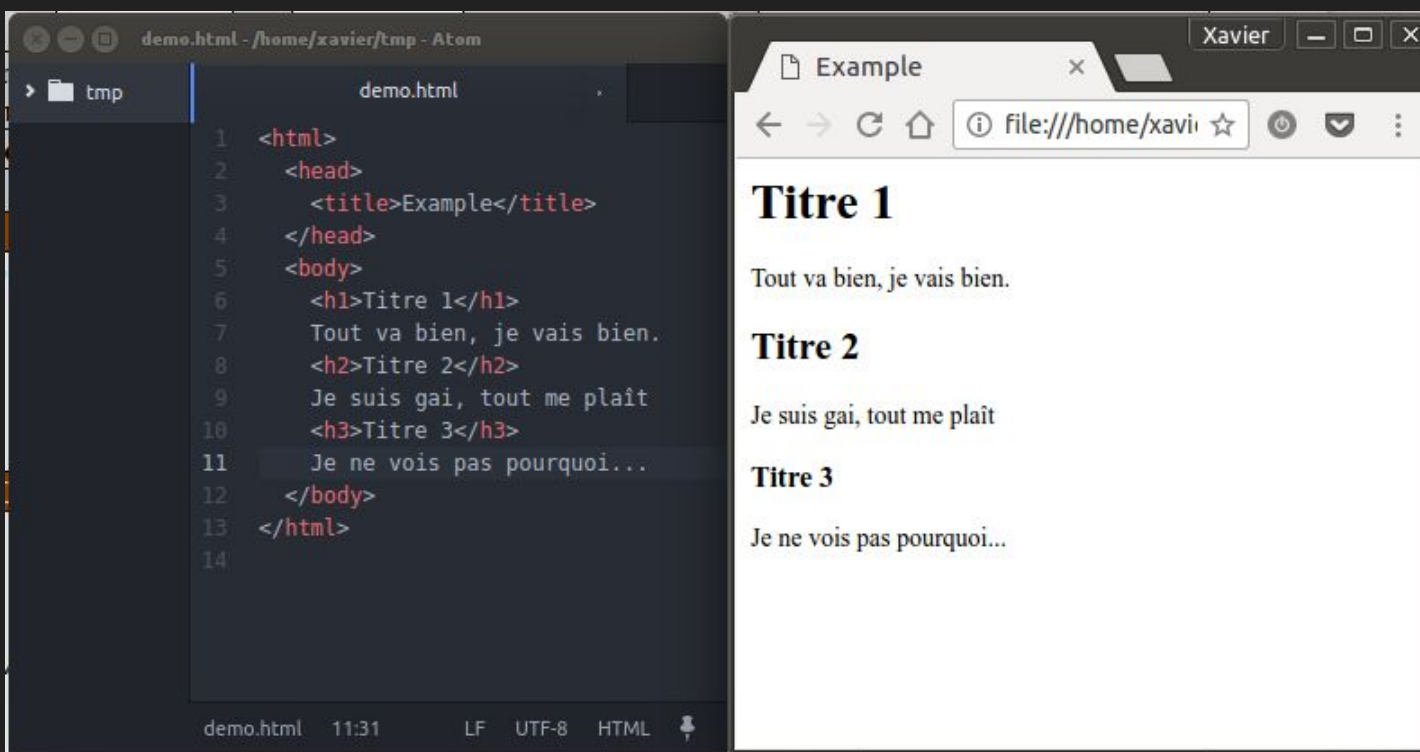
demo.html 8:43 LF UTF-8 HTML



Divisions

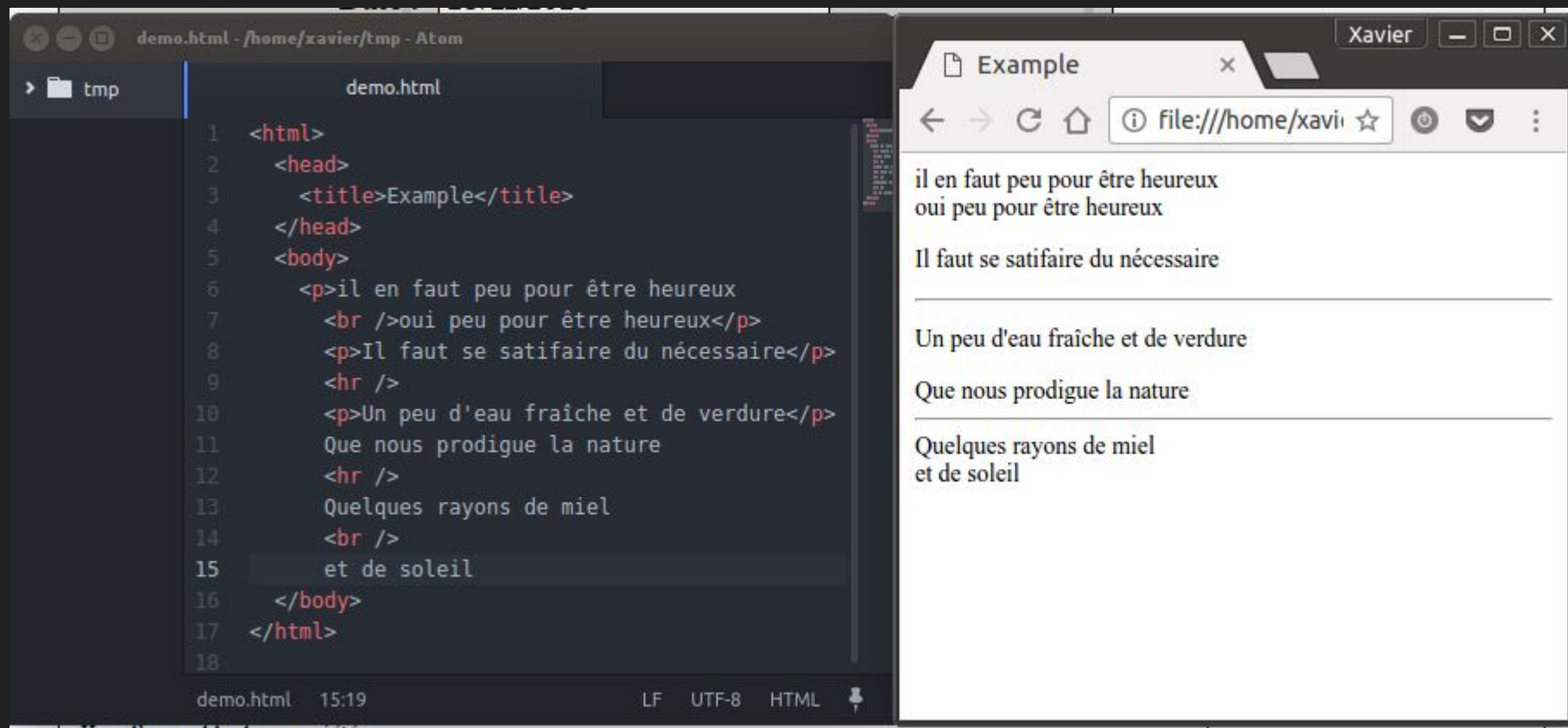
`<h1>` `<h2>` `<h3>` `<h4>` ...

`<div>` ``



Division

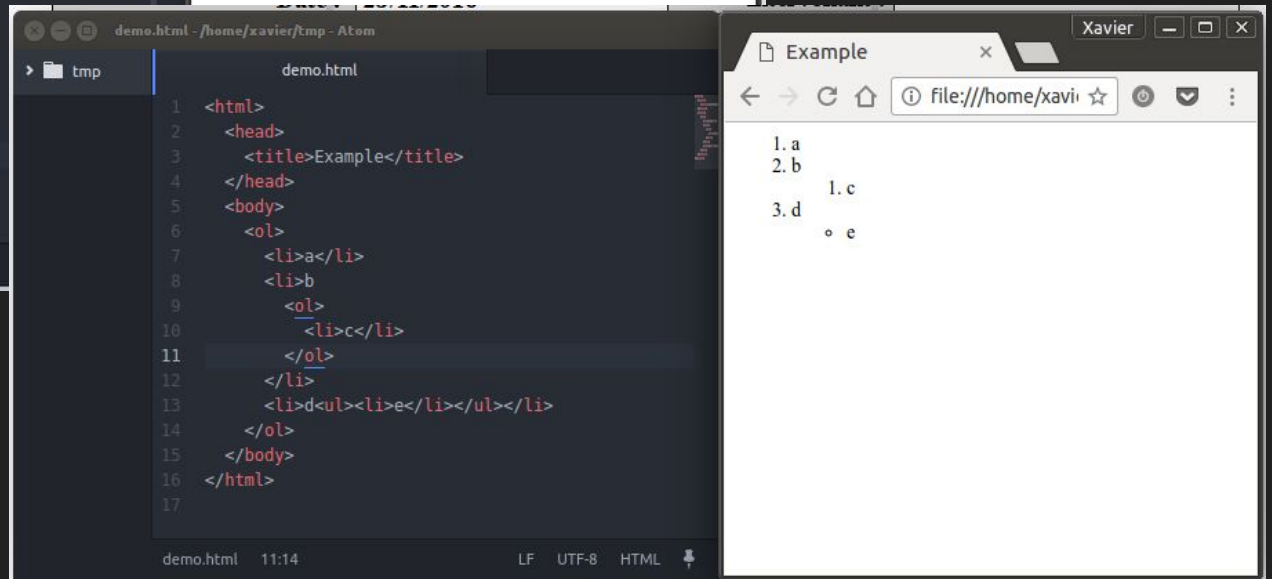
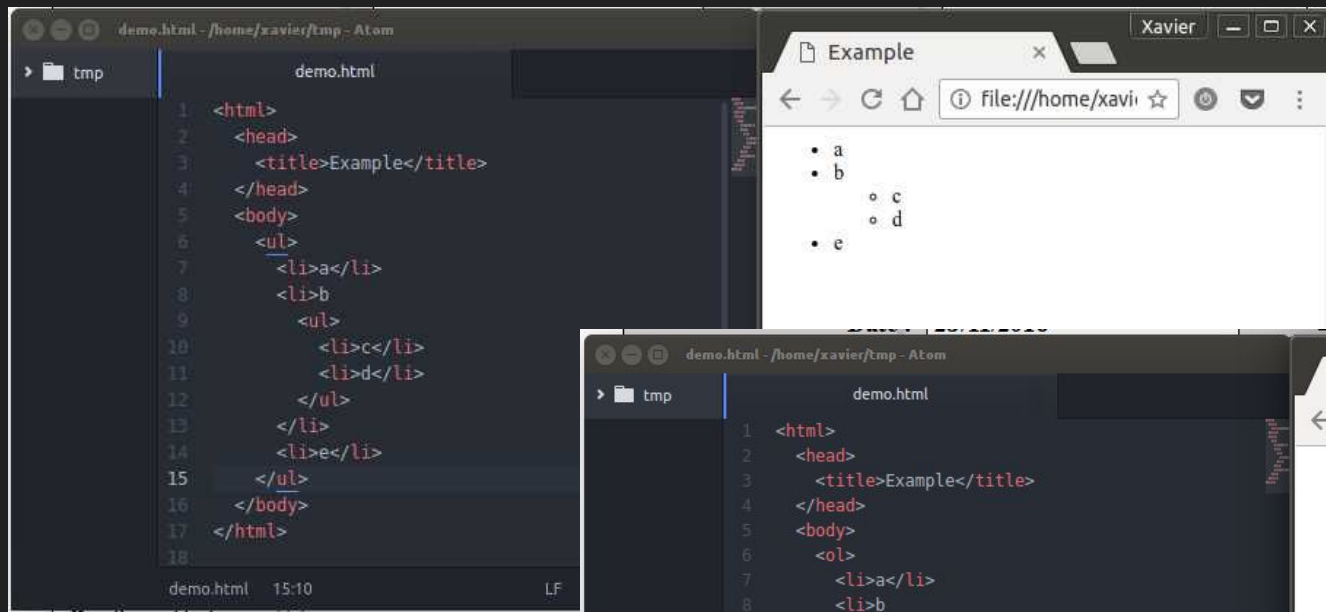
 <p> <hr>



Listes

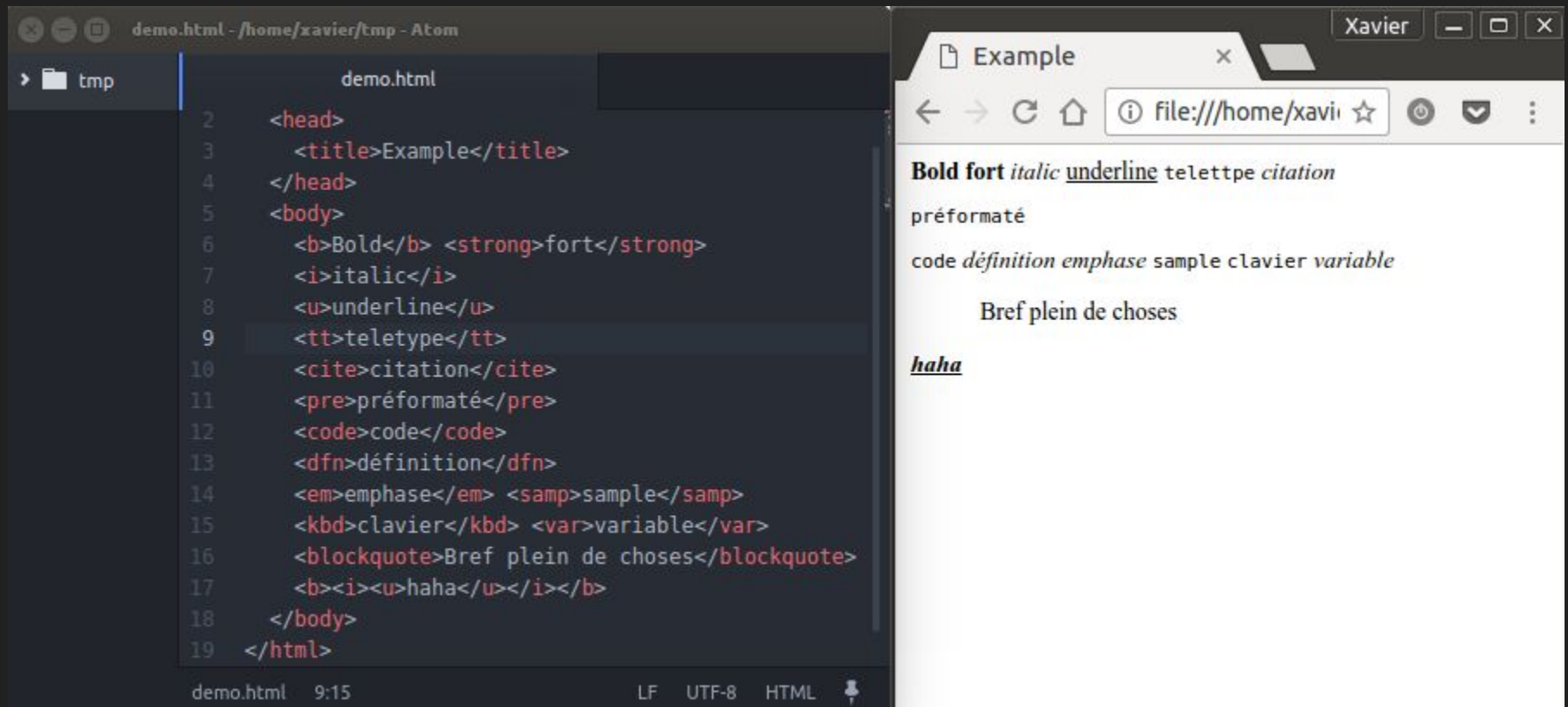
`<ul type= « square|circle|disc »> - `

`<ol type= « 1|A|a|I|i »> - `



Styles

 <i> <u> <tt> <cite> <pre> <code> <dfn> ...



The image shows a side-by-side comparison of an HTML document in its source code and its rendered output. On the left, the Atom editor displays the source code for 'demo.html'. The code uses various HTML tags to format text: **bold** (b), *italic* (i), underline (u), teletype (tt), citation (cite), preformatted (pre), code (code), definition (dfn), emphasis (em), sample (samp), keyboard (kbd), variable (var), and a blockquote. On the right, a web browser window titled 'Example' shows the rendered result. The text is formatted according to the HTML tags: 'Bold fort' is bold, 'italic' is italic, 'underline' is underlined, 'teletype' is in a monospaced font, 'citation' is italicized, 'préformaté' is in a monospaced font, 'code' is in a monospaced font, 'définition' is italicized, 'sample' is in a monospaced font, 'clavier' is in a monospaced font, 'variable' is italicized, 'Bref plein de choses' is enclosed in a blockquote, and 'haha' is bold, italicized, and underlined.

```
demo.html - /home/xavier/tmp - Atom
demo.html
2 <head>
3   <title>Example</title>
4 </head>
5 <body>
6   <b>Bold</b> <strong>fort</strong>
7   <i>italic</i>
8   <u>underline</u>
9   <tt>teletype</tt>
10  <cite>citation</cite>
11  <pre>préformaté</pre>
12  <code>code</code>
13  <dfn>définition</dfn>
14  <em>emphasis</em> <samp>sample</samp>
15  <kbd>clavier</kbd> <var>variable</var>
16  <blockquote>Bref plein de choses</blockquote>
17  <b><i><u>haha</u></i></b>
18 </body>
19 </html>
```

Example
file:///home/xavi...
Bold fort italic underline teletype citation
préformaté
code définition emphase sample clavier variable
Bref plein de choses
haha

Liens

- `Libellé`
- `Google`
- `lien 1`
- ``
- `email`

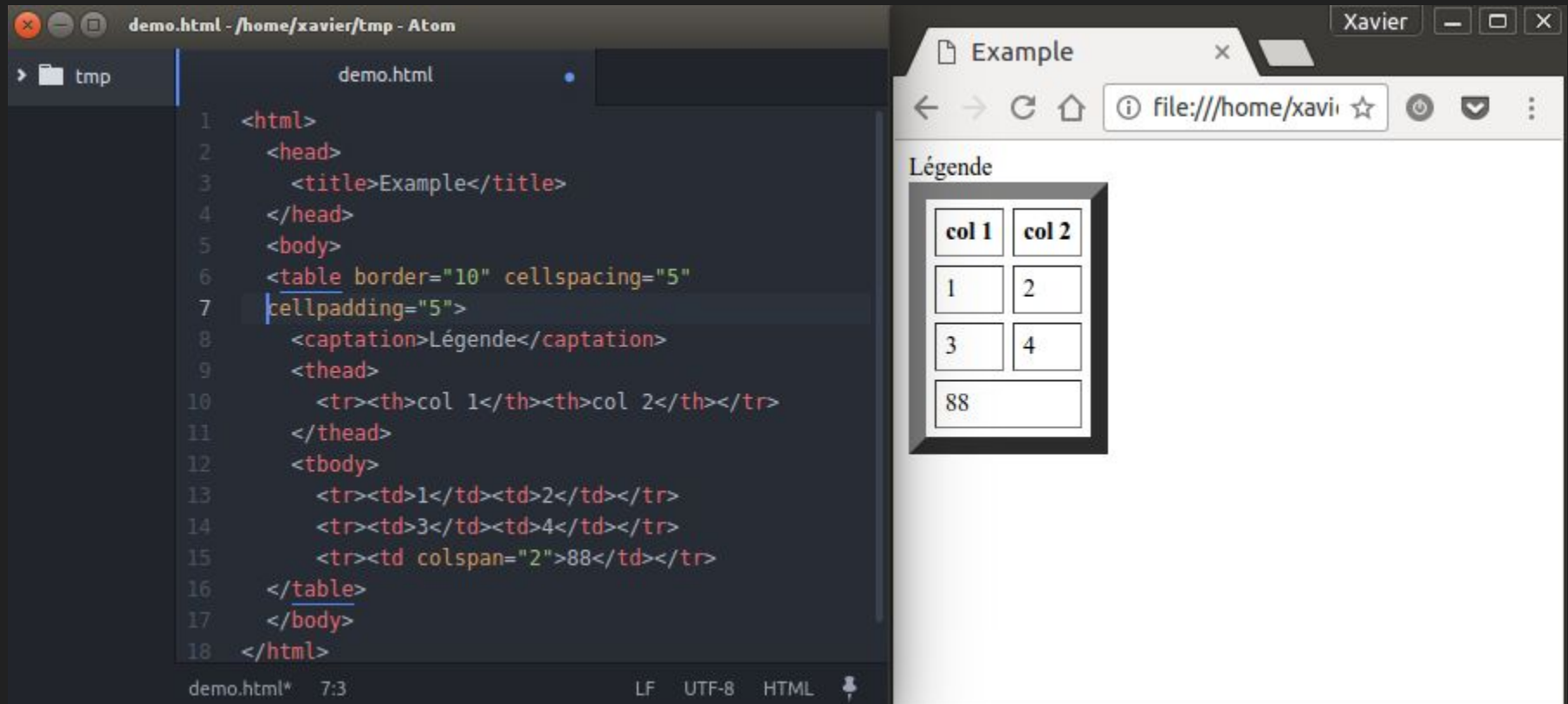
Médias

- ``
- ``
- ``
- ``

- `<video src=«video.mp4» controls loop poster=«moi.jpg» preload=«auto|none|metadata» autoplay />`
- `<video>`
 - `<source src=«lowres.mp4» media=«handheld»></source>`
 - `<source src=«highres.mp4»></source>``</video>`
- `<audio src= «du_hast.mp3» />`

Tableaux

- `<table border cellpadding cellspacing>`
- `<td valign align colspan rowspan nowrap>`



The image shows a side-by-side comparison of an HTML file in an editor and its rendered output in a browser.

Left Panel (Atom Editor): The file `demo.html` is open. The code is as follows:

```
1 <html>
2   <head>
3     <title>Example</title>
4   </head>
5   <body>
6     <table border="10" cellspacing="5"
7       cellpadding="5">
8       <caption>Légende</caption>
9       <thead>
10        <tr><th>col 1</th><th>col 2</th></tr>
11      </thead>
12      <tbody>
13        <tr><td>1</td><td>2</td></tr>
14        <tr><td>3</td><td>4</td></tr>
15        <tr><td colspan="2">88</td></tr>
16      </tbody>
17    </table>
18  </body>
19 </html>
```

Right Panel (Web Browser): The browser displays the rendered page. The title is "Example". The table is titled "Légende" and has the following structure:

col 1	col 2
1	2
3	4
88	

Formulaires

The image displays two examples of HTML forms and their rendered output. Each example consists of an Atom editor window on the left and a browser window on the right.

Top Example:

The Atom editor shows the following HTML code:

```
1 <html>
2 <head>
3   <title>Example</title>
4 </head>
5 <body>
6   <form name="monForm"
7     action="http://www.google.fr"
8     method="POST" target="_blank">
9     <input type="text" value="toto" />
10    <input type="date" />
11    <input type="number" value="5" />
12    <input type="tel" value="06" />
13    <button type="submit">Envoyer</button>
14  </form>
15 </body>
16 </html>
17
```

The browser window shows the rendered form with the following fields and values:

- Text input: toto
- Date input: 01/01/2016
- Number input: 5
- Tel input: 06
- Submit button: Envoyer

Bottom Example:

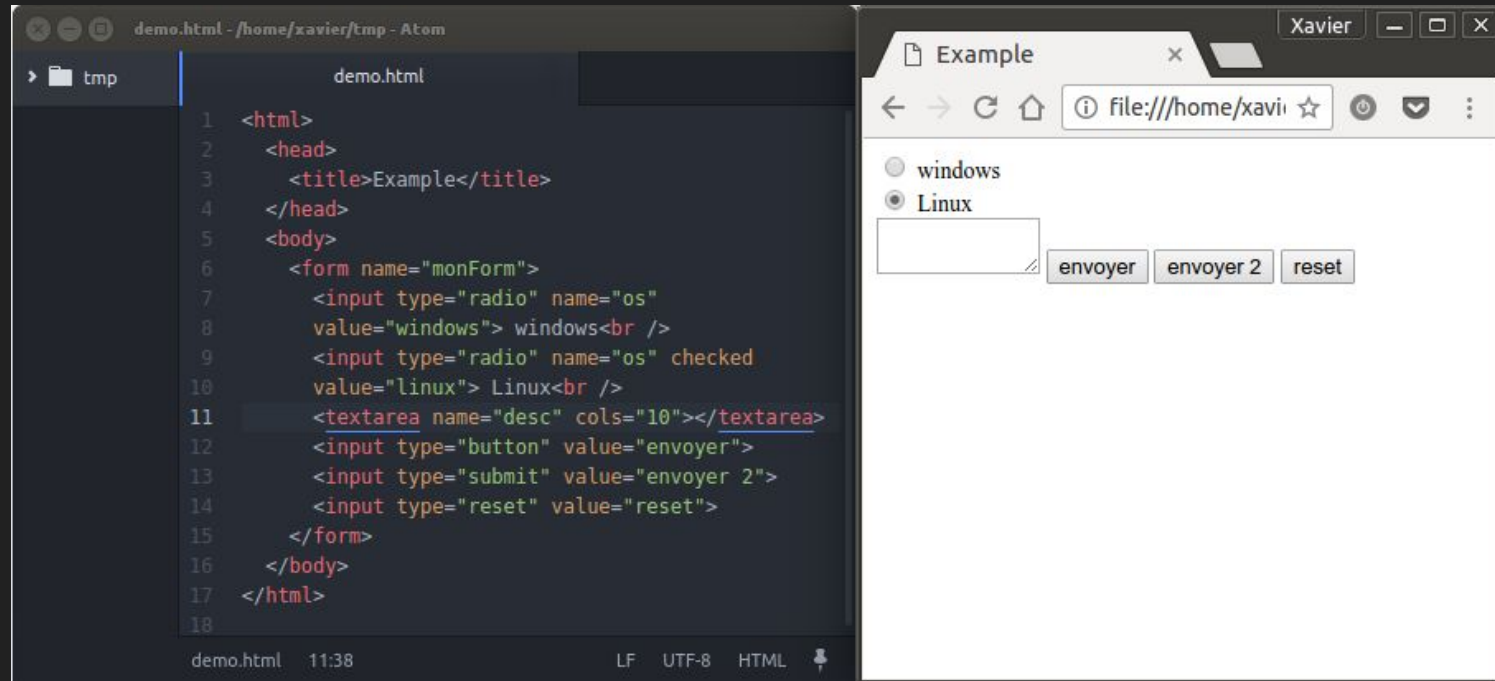
The Atom editor shows the following HTML code:

```
1 <html>
2 <head>
3   <title>Example</title>
4 </head>
5 <body>
6   <form name="monForm"
7     action="http://www.google.fr"
8     method="POST" target="_blank">
9     <select multiple size="2" name="num">
10      <option value="1">un</option>
11      <option value="2" selected="">deux</option>
12    </select><br />
13    <input type="checkbox" name="check" checked
14      value="linux"> OS<br />
15  </form>
16 </body>
17 </html>
18
```

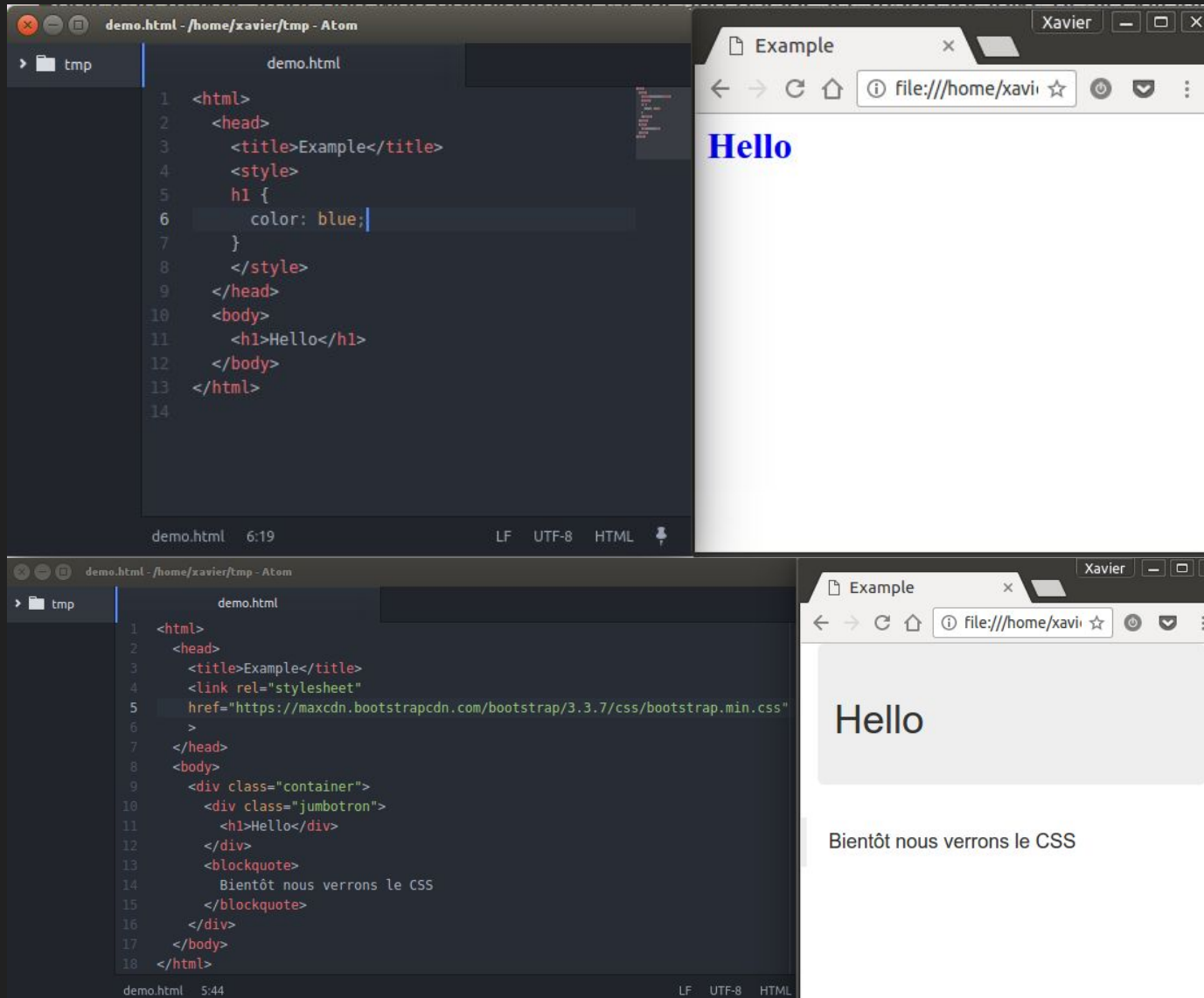
The browser window shows the rendered form with the following fields and values:

- Multiple select: un, deux (deux is selected)
- Checkbox: OS (checked)

Formulaires



Styles



CSS

```
body {  
  background-color: lightblue;  
}  
h1 {  
  color: white;  
  text-align: center;  
}  
p {  
  font-family: verdana;  
  font-size: 20px;  
}
```

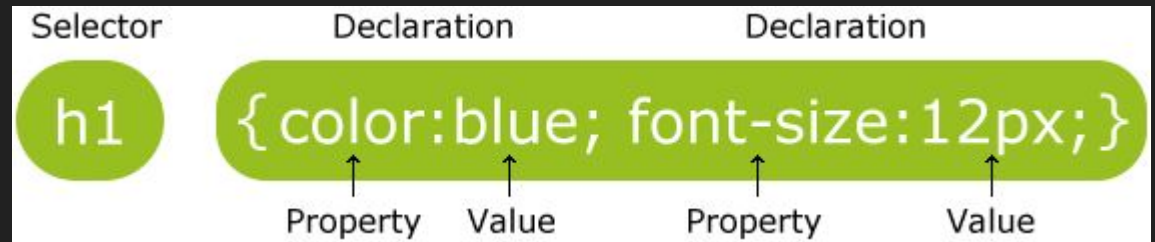
CSS = **C**ascading **S**tyle **S**heets

Met en page, formate, contrôle et anime un ensemble de pages web

Peut être externalisé dans un fichier css



Syntaxe



- Les sélecteurs :
 - `h1` : nom de balise
 - `.toto` : nom de classe : `<div class="toto">blah</div>`
 - `#toto` : id de balise : `<div id="toto">blah</div>`
- Cumulatif : `h1.toto#toto { ... }`
- Groupement : `h1, p, div : { ... }`
- Trois façons de les utiliser
 - Feuille de style externe
 - `<link rel="stylesheet" type="text/css" href="mystyle.css">`
 - Feuille de style interne :
 - `<style>`
`body {`
`background-color: linen;`
`}`
`</style>`
 - Inline
 - `<h1 style="color:blue;margin-left:30px;">This is a heading.</h1>`

Valeurs

- Couleurs
 - Nom : `red`
 - RGB : `rgb(255, 0, 0)`
 - RGB + alpha : `rgba(255, 0, 0, 1)`
 - Hex : `#ff0000`
- Dimensions
 - Pixels : `18px`
 - Relatif : `50%`
 - Taille de font : `1em = 16px`
 - Viewport : `100vh 50vw`
 - Calculé : `calc(100vh - 50px)`

Quelques propriétés

- Background
 - background-color, background-image, background-repeat, background-attachment, background-position
 - `background: #ffffff url("img_tree.png") no-repeat right top;`
- Bordures
 - border-style, border-width, border-color, border-radius
 - border-[top | right | bottom | left]-[color | width | radius | style]
 - outline-style, outline-width, outline-color, outline-offset
 - outline-[top | right | bottom | left]-[color | width | offset | style]
 - `border: 5px solid red; outline: 5px dotted red;`
- Marges
 - margin, margin-[top | right | bottom | left]
 - `margin: auto 150px 100px 80px;`
- Padding
 - padding, padding-[top | right | bottom | left]
 - `padding: 50px 30px 50px 80px;`
- height / width

Quelques propriétés

- Texte

- color
- text-align: center | left | right | justify
- text-decoration: none | overline | line-through | underline
- text-transform : uppercase | lowercase | capitalize
- text-indent : 50px
- letter-spacing : 15px
- line-height : 0.5em
- direction : rtl
- word-spacing : -5px
- vertical-align
- text-shadow

- Fonts

- font-family: "Times New Roman", Times, serif
- font-style: italic
- font-size
- font-weight: bold
- font-variant: small-caps



Pseudo classes et pseudo éléments

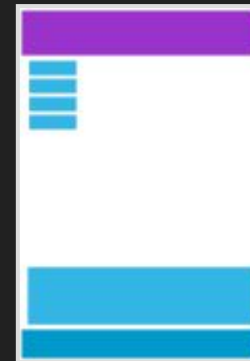
- `a:link` : un lien normal non visité
 - `a:visited` : un lien visité
 - `a:hover` : un lien survolé par la souris
 - `a:active` : un lien au moment où l'on a cliqué dessus
-
- `p::first-line { color: #ff0000; font-variant: small-caps; }`
 - `p::first-letter { color: #ff0045; font-variant: small-caps; }`
 - `h1::before { content: url(smiley.gif); }`
 - `h1::after { content: url(smiley.gif); }`
 - ...

Sélecteurs

- `a[target] { background-color: yellow; }`
- `a[target="_blank"] { background-color: yellow; }`
- `[class|="top"] { background: yellow; }`
- `[class^="top"] { background: yellow; }`
- `[class$="test"] { background: yellow; }`
- `[class*="te"] { background: yellow; }`
- `div > p` : le p dont le parent est une div
- `div p` : tous les p contenus dans une div
- `div + p` : le p placé immédiatement après une div
- `div, p` : tous les div et tous les p
- `div~p` : les p précédés d'une div
- ...

RWD : Responsive Web Design

- Feuille de style qui s'adapte à la taille de l'écran
 - Ordinateur, tablette, téléphone
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`



RWD : Media queries et Grid system

```
/* For desktop: */  
.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}  
  
@media only screen and (max-width: 768px) {  
  /* For mobile phones: */  
  [class*="col-"] {  
    width: 100%;  
  }  
}
```

```
@media only screen and (orientation: landscape) {  
  body {  
    background-color: lightblue;  
  }  
}
```



RWD : frameworks

Twitter Bootstrap : <http://getbootstrap.com/>

Foundation : <http://foundation.zurb.com/>

Skeleton : <http://www.getskeleton.com/>

MaterializeCSS : <http://materializecss.com/>



Angular 2



Angular 2 - les présentations

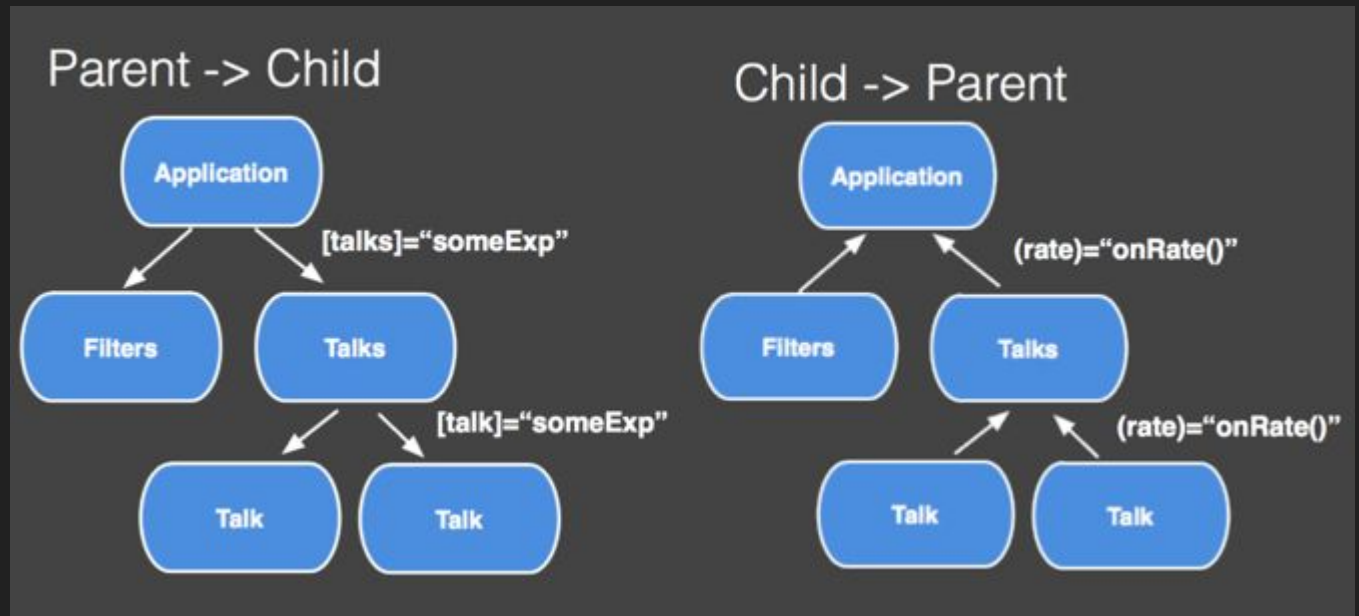
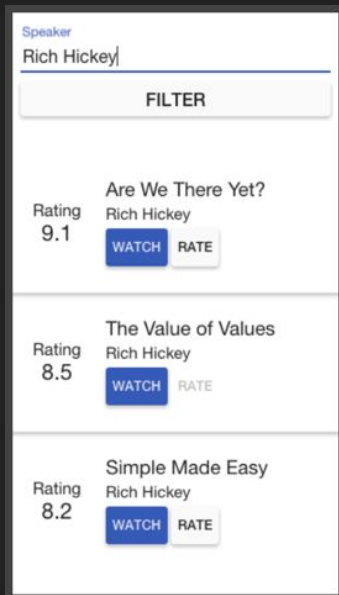
- 2009 - Création de la version originale par Misko Hevery
- 2010 - Misko rejoint Google qui soutient officiellement AngularJS
- 2013 - Explosion de la notoriété
- 2016 - Sortie d'Angular 2
- Framework Javascript pour faire des SPA
- Plusieurs langages : ES5, ES6, TypeScript et Dart
- Contient routeur, requêteur HTTP, gestion des formulaires, i18n, ...
- Modulaire : découpé en sous paquets, nos apps sont découpées en composants et modules
- 5 fois plus rapide que Angular 1
- Tout est composant

Angular 2 : une plateforme

- La vocation d'Angular 2 est de devenir une plateforme pour le développement d'applications web et mobiles :
 - Le noyau de la librairie a été scindé en plusieurs composants logiques, et devient donc plus modulaire. (On n'installe que ce dont on a besoin.)
 - L'outillage a été amélioré, avec des outils comme TypeScript, Angular CLI, Augura...
- Il devient possible d'exécuter Angular facilement dans plusieurs environnements (sur le serveur avec Angular Universal, sur mobile avec Angular Mobile Toolkit ou NativeScript, etc.).
 - <https://universal.angular.io/>
 - <https://mobile.angular.io/>

Angular 2 - composant

- Autonome : savent comment s'afficher et interagir avec leur hôte
- API publique clairement définie
- Peuvent appeler des services externes via la DI
- Réutilisabilité accrue



TypeScript

- Langage créé par Microsoft en 2012, open-source, qui **transpile** vers JavaScript.
- Surensemble d'ES6 (aka ES2015). **Tout JavaScript est donc du TypeScript valide.**
- Principales caractéristiques : types, interfaces, classes, décorateurs, modules, fonctions fléchées, templates chaîne.
- Supporté par de nombreuses librairies JavaScript tierce-partie.
- Supporté par plusieurs IDE : WebStorm/IntelliJ Idea, Visual Studio Code, Sublime Text, etc.
- Langage le plus populaire pour Angular 2. En train de s'imposer comme le langage officiel.

Variables TypeScript

- `var isDone: boolean = false;`
- `var height: number = 6;`
- `var name: string = "dave";`
- `var myList:number[] = [1, 2, 3];` `// option #1`
- `var myList:Array<number> = [1, 2, 3];` `// option #2`
- `var changeMe: any = 4;`
- `changeMe = "I'm a string now";`
- `var myList:any[] = [3, true, "pizza"];`

Fonctions TypeScript

- void return type:

- ```
function myLogger(msg?:string): void {
 console.log("My custom logger: "+msg);
}
```

- Generics:

- ```
function identity<T>(arg: T): T {  
    return arg;  
}  
  
// output has 'string' type (explicit/inferred):  
var output = identity<string>("Dave");  
var output = identity("Dave");
```

- any return type : Désactive le typage

- ```
function selectSomething(x): any {
 if (typeof x == "object") {
 return 10;
 } else {
 return "abc";
 }
}
```

# Interfaces TypeScript

```
interface User {
 name: string;
 weight?: number; // optional
}

function displayUser(user: User) {
 console.log(user.name);
}
```

ça marche même sans “weight” dans l’interface

```
var aUser = {name: "John Smith", weight:200};
displayUser(aUser);
```

# Classes TypeScript

```
class User {
 fname: string;
 lname: string;

 constructor(fname:string, lname:string) {
 this.fname = fname;
 this.lname = lname;
 }

 fullname():string {
 return this.fname+" "+this.lname;
 }
}
```



# Classes TypeScript

Seulement un seul constructeur par classe  
On peut implémenter plusieurs interfaces

```
class User {
 fname: string;
 lname: string;

 constructor(fname:string, lname:string) {
 this.fname = fname;
 this.lname = lname;
 }

 fullname():string {
 return this.fname+" "+this.lname;
 }
}
```

```
var u1:User, u2:User;
u1 = new User("Jane", "Smith");
u2 = new User("John", "Jones");
console.log("user1 = "+u1.fullname());
console.log("user2 = "+u2.fullname());
```

# Angular 2 et TypeScript

- Symbole `@` pour les annotations/decorators
- `@Component` (`{selector, template, ... }`)
- Une classe typique est `AppComponent` dans `app.component.ts`
- Un module dans `app.module.ts`
- Bootstrap du composant racine dans `main.ts`
- Utilisation d'Angular CLI pour générer un squelette d'application :
  - `[sudo] npm install -g angular-cli`
- Créer une application angular 2
  - `ng new myapp`
  - `cd myapp`
  - `ng serve`
- Extension Chrome DevTools : Angular Augry : <https://augury.angular.io/>
- Package Manager : npm
- Module Loader : SystemJS
- Transpiler : Traceur / TypeScript
- Build Tool : Broccoli

# Angular 2 : index.html

```
<head>
 <script src="node_modules/core-js/client/shim.min.js"></script>
 <script src="node_modules/zone.js/dist/zone.js"></script>
 <script src="node_modules/reflect-metadata/Reflect.js"></script>
 <script src="node_modules/systemjs/dist/system.src.js"></script>
 <!-- Configure SystemJS -->
 <script src="systemjs.config.js"></script>
<script>
 System.import('app').catch(function(err){ console.error(err); });
</script>
</head>
<!-- Display the application -->
<body>
 <my-app>Loading...</my-app>
</body>
```

# Angular 2

## main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

## app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
@NgModule({
 imports: [BrowserModule],
 declarations: [AppComponent],
 bootstrap: [AppComponent]
})
export class AppModule { }
```

# Angular 2

app.component.ts

```
import {Component} from '@angular/core';
@Component({
 selector: 'my-app', // located in index.html
 template: `<div>Hello from Angular 2 <button (click)="clickedMe(event)">Press
Me</button></div>`
})
export class AppComponent {
 clickedMe(event) {
 // do stuff here
 console.log("you clicked me");
 event.preventDefault();
 }
}
```

# Angular 2 vs Angular 1

Angular 1	Angular 2
Framework	Plateforme
JavaScript	TypeScript
Pattern Modèle-Vue-*	Pattern Composant
Liaison de données principalement Bldirectionnelle	Liaison de données principalement UNIdirectionnelle
Scope	Bye bye le scope
Injection de dépendance : plusieurs syntaxes possibles	Injection de dépendance : syntaxe unique.
API complexe	API simplifiée
Rendering normal	Rendering 5 fois plus rapide
Plusieurs “bonnes pratiques” concurrentes par la communauté	Bonnes pratiques officielles : <a href="https://angular.io/styleguide">https://angular.io/styleguide</a>

# 1. Bootstraper Angular

- NG1: directive ng-app (bootstrap automatique).
- NG2: bootstrap via code avec la fonction bootstrap()

## Angular 1

```
<html ng-app="app">
```

## Angular 2

```
import { bootstrap } from 'angular2/platform/browser';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

## 2. Des contrôleurs aux composants

- NG1: `angular.controller()`
- NG2: Classe avec décorateur `@Component`

### Angular 1

```
<body ng-controller="StoryController as vm">
 <h3>{{vm.story.name}}</h3>
 <h3 ng-bind="vm.story.name"></h3>
</body>

(function () {
 angular
 .module('app')
 .controller('StoryController', StoryController);

 function StoryController() {
 var vm = this;
 vm.story = { id: 100, name: 'The Force Awakens' };
 }
})();
```

### Angular 2

```
<my-story></my-story>

import { Component } from 'angular2/core';

@Component({
 selector: 'my-story',
 template: '<h3>{{story.name}}</h3>'
})
export class StoryComponent {
 story = { id: 100, name: 'The Force Awakens' };
}
```



### 3. Directives structurelles

- NG1: Beaucoup de directives structurelles. Ici, ng-repeat et ng-if.
- NG2: Seules quelques directives conservées (comme \*ngFor et \*ngIf). Points importants : notation camelcase, étoile \* devant nom de la directive (signale une directive structurelle), syntaxe let vehicle of vehicles (of et non pas in).

#### Angular 1

```

 <li ng-repeat="vehicle in vm.vehicles">
 {{vehicle.name}}

<div ng-if="vm.vehicles.length">
 <h3>You have {{vm.vehicles.length}} vehicles</h3>
</div>
```

#### Angular 2

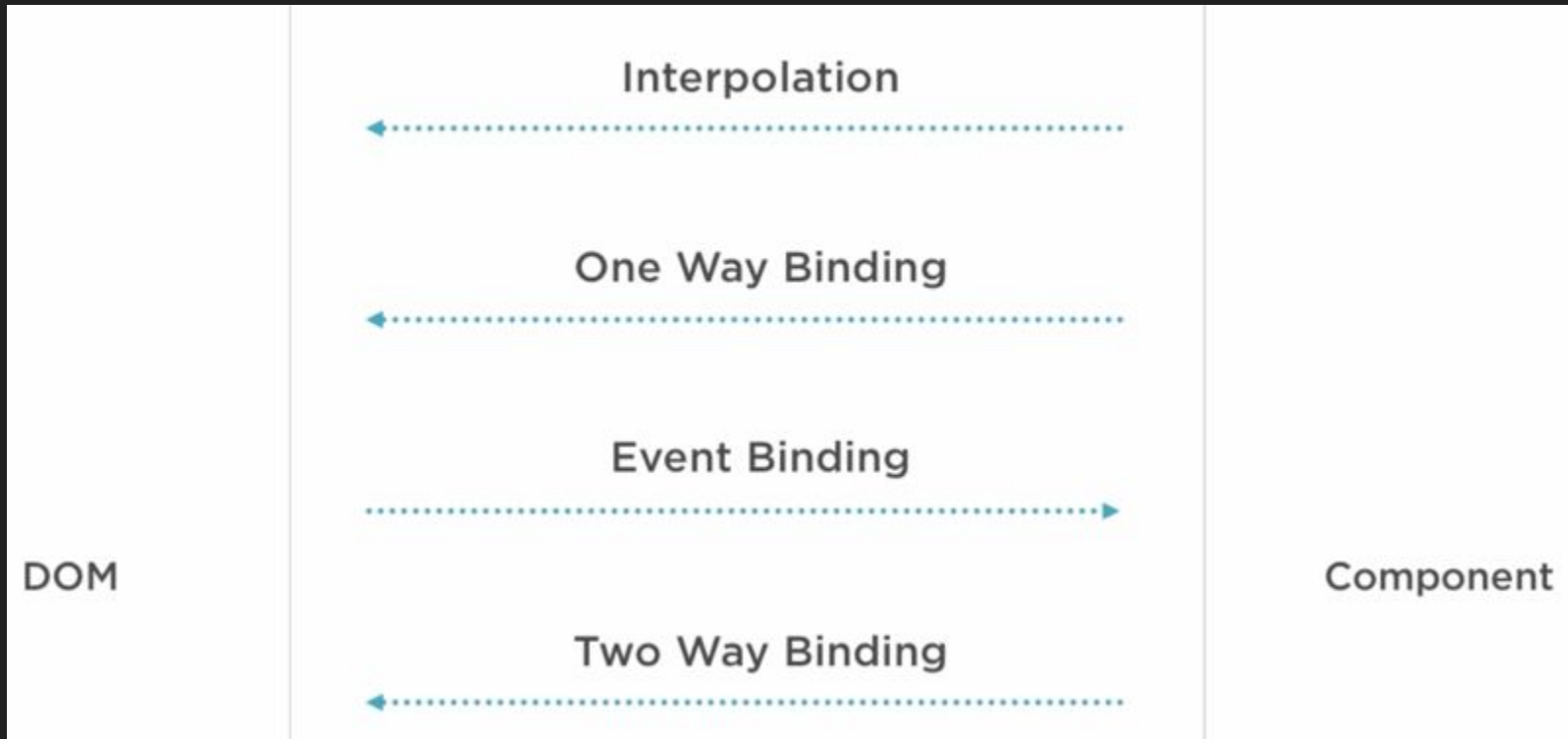
```

 <li *ngFor="#vehicle of vehicles">
 {{vehicle.name}}

<div *ngIf="vehicles.length">
 <h3>You have {{vehicles.length}} vehicles</h3>
</div>
```

## 4. Liaison de données

- Permet de synchroniser les données entre les composants et le DOM (aka la vue).



## 4. Liaison de données

- Interpolation



- Binding de propriété (unidirectionnel)



## 4. Liaison de données

- Binding d'événement
  - NG1: ng-click, ng-blur... — Directives custom Angular.
  - NG2: (click), (blur) — Fini les directives custom, on utilise les événements natifs d'un HTMLInputElement entre parenthèses.

Angular 1	Angular 2
ng-click="saveVehicle(vehicle)"	(click)="saveVehicle(vehicle)"
ng-focus="log('focus')"	(focus)="log('focus')"
ng-blur="log('blur')"	(blur)="log('blur')"
ng-keyup="checkValue()"	(keyup)="checkValue()"

## 4. Liaison de données

- Liaison de données bidirectionnelle (champ de formulaire)



## 5. Moins de directives

- NG1: ng-style, ng-src, ng-href...
- NG2: Plus de 40 directives NG1 ont disparu dans NG2 !

### Angular 1

```
<div ng-style=
 "vm.story ?
 {visibility: 'visible'}
 : {visibility: 'hidden'}">

 <a ng-href="{{vm.link}}">
 {{vm.story}}

</div>
```

### Angular 2

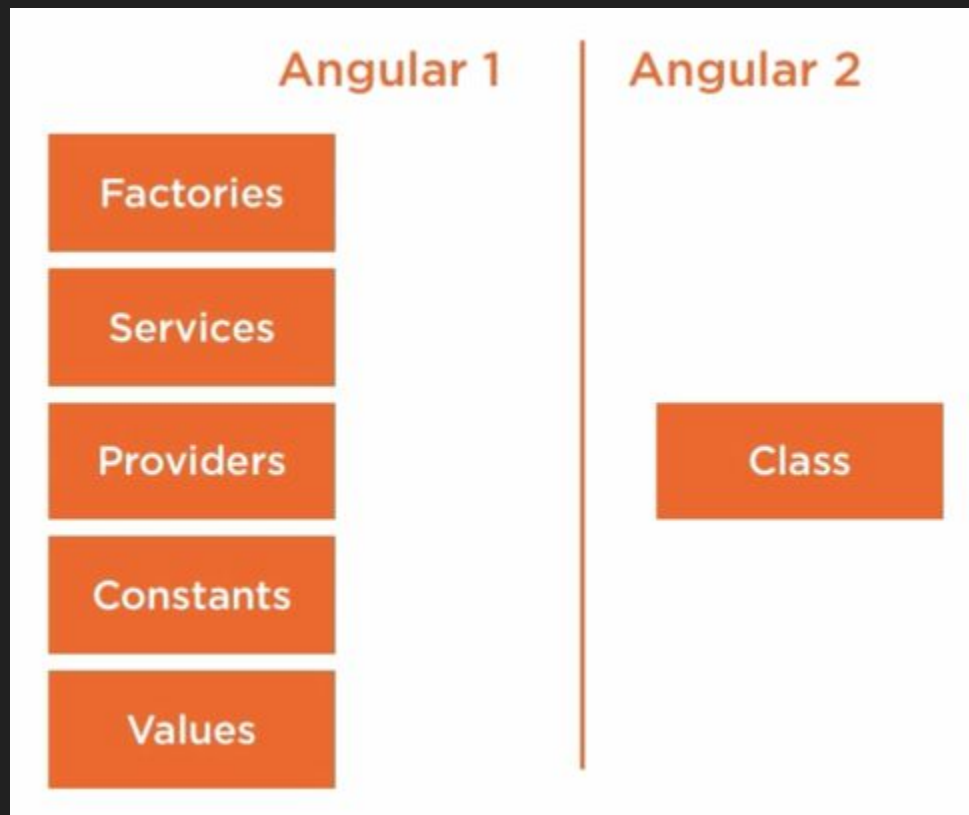
```
<div [style.visibility]=
 "story ? 'visible' : 'hidden'">

 <a [href]="link">{{story}}

</div>
```

## 6. Services et DI (1/2)

- NG1: Les données ou fonctionnalités partagées utilisent des factories, des services, des providers...
- NG2: Un seul concept subsiste : une classe TypeScript.



## 6. Services et DI (2/2)

- NG1: Ici, `angular.service()`, mais pourrait être `angular.factory()`, `angular.provider()`...
- NG2: Simple classe avec le décorateur `@Injectable`.

### Angular 1

```
angular
 .module('app')
 .service('VehicleService', VehicleService);

function VehicleService() {
 this.getVehicles = function () {
 return [
 { id: 1, name: 'X-Wing Fighter' },
 { id: 2, name: 'Tie Fighter' },
 { id: 3, name: 'Y-Wing Fighter' }
];
 }
}
```

### Angular 2

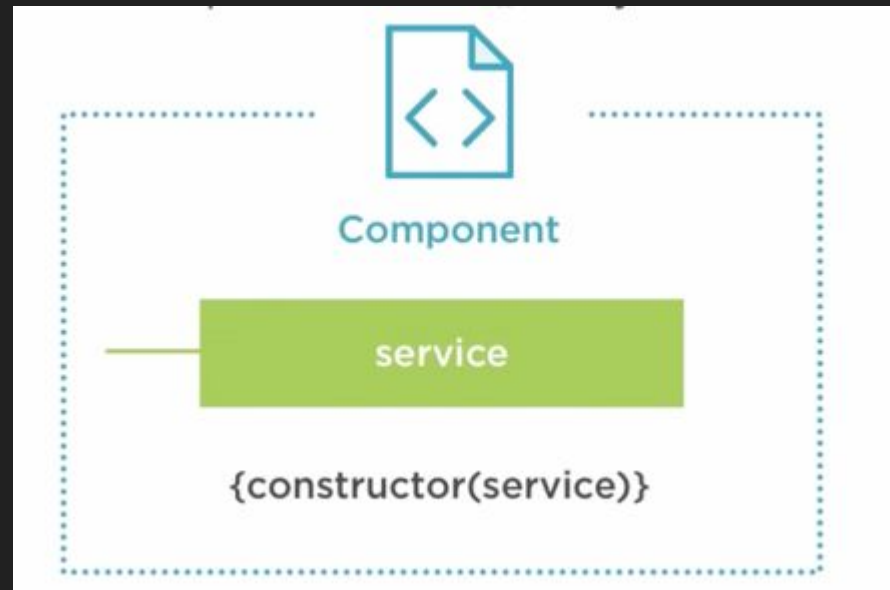
```
import {Injectable} from 'angular2/core';

@Injectable()
export class VehicleService {
 getVehicles = () => [
 { id: 1, name: 'X-Wing Fighter' },
 { id: 2, name: 'Tie Fighter' },
 { id: 3, name: 'Y-Wing Fighter' }
];
}
```



## 7. Injection de dépendance

- Dans Angular, les services contiennent toute la logique applicative. Exemple : service qui récupère les données du serveur via un appel HTTP.
- Lorsqu'un composant a besoin d'utiliser un service, il utilise l'injection de dépendance (DI).
- La DI se fait en deux temps :
  - Déclaration
  - Injection



## 7. Injection de dépendance : Déclaration

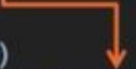
- NG1: La déclaration se fait avec `angular.service()` et une chaîne de caractères qui identifie le service.
- NG2: Pas de chaîne de caractères, on déclare le service dans la propriété `providers` du composant qui l'utilise

### Registration

#### Angular 1

```
angular
 .module('app')
 .service('VehicleService', VehicleService);

function VehicleService() {
 this.getVehicles = function () {
 return [
 { id: 1, name: 'X-Wing Fighter' },
 { id: 2, name: 'Tie Fighter' },
 { id: 3, name: 'Y-Wing Fighter' }
];
 };
}
```



#### Angular 2

### Registration

```
import { VehicleService } from './vehicle.service';

@Component({
 selector: 'my-vehicles',
 templateUrl: 'app/vehicles.component.html',
 providers: [VehicleService]
})
export class VehiclesComponent {
 constructor(
 private _vehicleService: VehicleService) { }
 vehicles = this._vehicleService.getVehicles();
}
```



## 7. Injection de dépendance : Injection

- NG1: Propriété \$.inject qui matche les arguments passés à la fonction factory du contrôleur.
- NG2: On passe le service au constructor du composant (ou plutôt, on type un param du constructor).

### Angular 1

```
angular
 .module('app')
 .controller('VehiclesController', VehiclesController);

VehiclesController.$inject = ['VehicleService'];
function VehiclesController(VehicleService) {
 var vm = this;
 vm.title = 'Services';
 vm.vehicles = VehicleService.getVehicles();
}
```



Injection

### Angular 2

```
import { VehicleService } from './vehicle.service';

@Component({
 selector: 'my-vehicles',
 templateUrl: 'app/vehicles.component.html',
 providers: [VehicleService]
})
export class VehiclesComponent {
 constructor(
 private _vehicleService: VehicleService) { }
 vehicles = this._vehicleService.getVehicles();
}
```



Injection

# Librairies de composants UI

- ng-bootstrap (<https://github.com/ng-bootstrap/core>) - Ré-écriture en Angular 2 des composants UI de Bootstrap CSS (v4).
- Angular Material (<https://material.angular.io/>) - Librairie de composants UI développés par Google spécifiquement pour Angular 2. Actuellement en early alpha, mais développement assez actif.
- PrimeNG (<http://www.primefaces.org/primeng/>) - Collection de composants UI pour Angular 2 par les créateurs de PrimeFaces (une librairie populaire utilisée avec le framework JavaServer Faces).
- • Polymer (<https://www.polymer-project.org/>) - Librairie de “Web Components” extensibles par Google. L’intégration avec Angular 2 est réputée simple.
- NG-Lightning (<http://ng-lightning.github.io/ng-lightning/>) - Librairie de composants et directives Angular 2 écrits directement en TypeScript sur la base du framework CSS Lightning Design System

Une question?

