

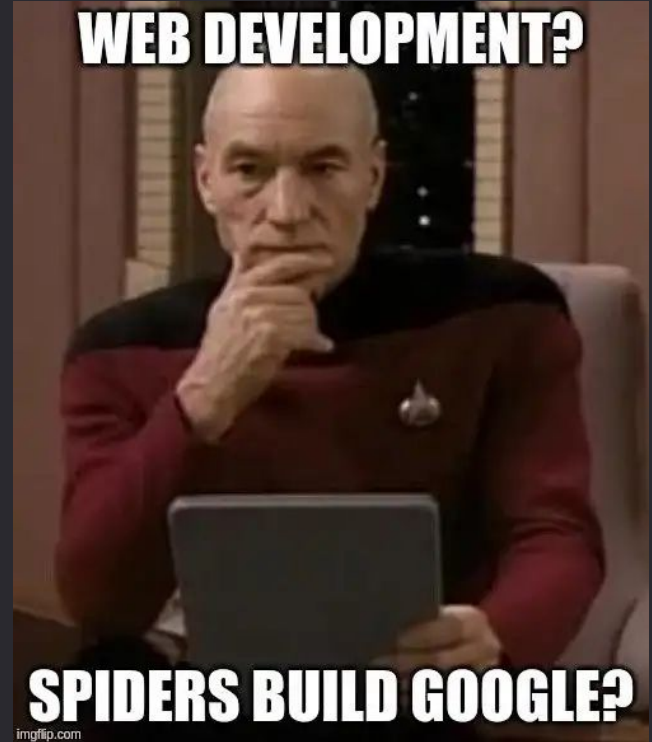
Module développement WEB



Introduction



- Présentation
- Objectifs du module
- Déroulement et critères des évaluations





Introduction



- Guyot Cédric : Développeur C/C++/C#
- Expertise en Web et C++
- Passionné par l'informatique et le JV
- **Contact** : guyot.ced@gmail.com
- **Ressources** : <https://github.com/ProfesseurPoire/L3WebInfo>





Introduction



Sujets abordées :

- Web
- HTML
- CSS
- Javascript
- VueJS
- NodeJS
- RGPD

Objectifs des TP

Appliquer les enseignements vus en cours

Projet

Création de votre choix, en groupe ou seul.





Introduction::Évaluations



- Épreuve écrite : 25%
- TP : 1 à 5 : 25%
- Projets : 50%

Si la note du projet est supérieur à la note des 3 évaluations, elle seule sera utilisée.





Introduction::Évaluations écrite



- Elle aura lieu le 13 février de 10h à 11h30 en salle INFO 2.
- Elle portera principalement sur des définitions vu en cours.
- Partie QCM





Introduction::Travaux pratiques

- Les TP 1 à 5 devront être rendus avant le 2 avril minuit.
- Les TP et projets devront être rendus sur le dropbox suivant :
<https://www.dropbox.com/scl/fo/0r7w2gqi8mxt180u4i1d9/h?dl=0&rlkey=w1k081n4uza1dbwuzfou4u45i>
- Convention de nommage : TP{x}/nom_prenom/
- Pour les projets, trigramme des membres du projet + fichier readme avec nom/prénom des participants.
- Les TP appliquent les concepts vus en cours.
- Les TP se présentent sous la forme d'une liste de spécifications techniques à mettre en œuvre.



Introduction::Projets



- Seul ou en groupe (jusqu'à 3)
- Réussir un projet web qui combine les techniques vues en cours
- A me rendre à l'adresse guyot.ced@gmail.com avant le 13 février minuit.
- Critères de notation
 - Partie client (HTML, JS, VueJS, CSS)
 - Partie server (nodeJS)
 - Respecter une convention de codage





Web



Qu'est ce que le web?

Le web émerge de la combinaison des technologies suivantes :

- Un langage de balisage utilisé pour représenter des page internet : HTML
- Un protocole pour échanger des documents : HTTP
- Un serveur HTTP pour émettre ces documents (httpd)
- Un client HTTP pour afficher ces documents
- Adresses Web



Web::Adresses



Aussi connu sous le nom d'URL

- URL : Uniform Resource Locator.
- Inventé en 1994
- Mieux connu sous sa désignation française : **adresse réticulaire**.

Les noms de domaines des URL sont convertis en adresses IP par les serveurs DNS





Web::Protocole HTTP



HTTP : Hypertext transfer protocol

Protocole de communication utilisé pour transférer des documents entre un client et un serveur.

Requêtes HTTP :

Une requête demande au serveur d'effectuer une action.



Web::Protocole HTTP::Requêtes



Requête GET

- Utilisé pour demander des données au serveur.
- N'impacte pas les données sur le serveur.
- Aucune modification des données du serveur.
- Les arguments de la requête GET sont visible dans l'URL

Exemple :

`/test/demo.php?name1="value1"`



Web::Protocole HTTP::Requêtes



Requête POST

- Envoie des données au serveur.
- Ces données sont utilisées par le serveur pour mettre à jour ou créer une ressource.
- Généralement envoyé par des formulaires.

Exemple :

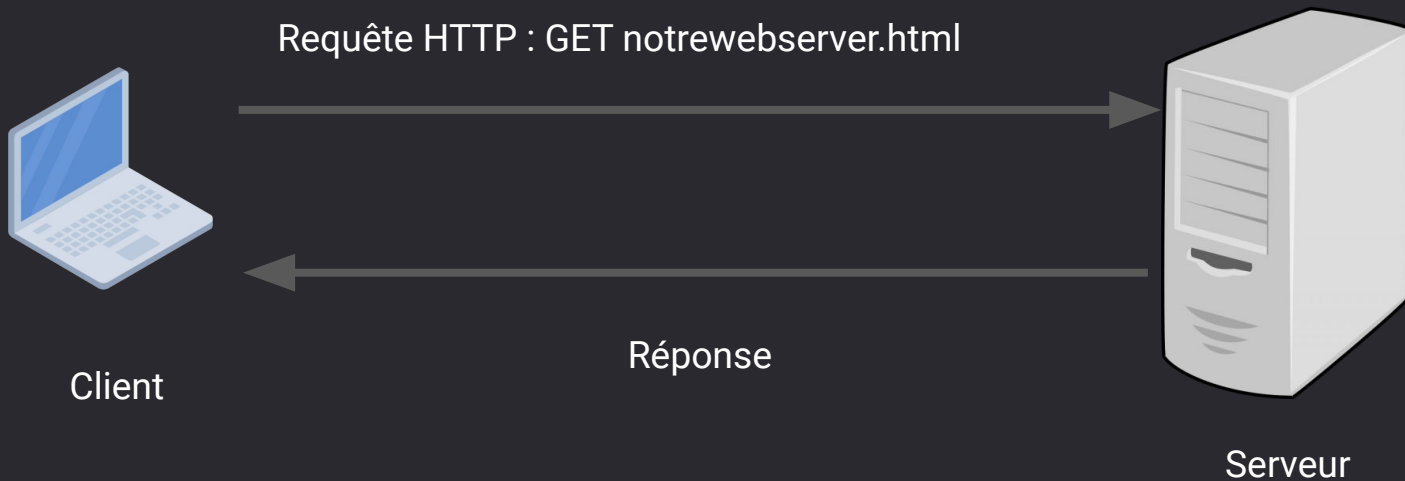
POST /members?id=1234 HTTP/1.1

host : www.example.com

```
{"name": "me"}
```



Web::Exemple



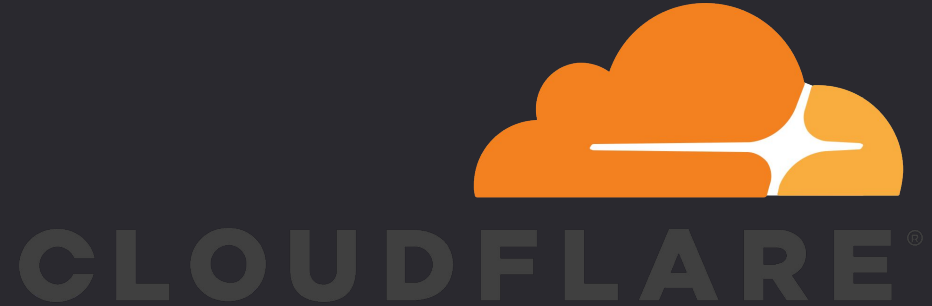


Web::Serveurs



- Contiennent des documents ou services à partager (images, pages HTML, son, vidéos etc)
- Écoutent les requêtes HTTP entrantes (généralement sur le port 8080).
- Répondent aux requêtes HTTP et partagent les documents.

Web::Principaux serveurs



NGINX





Web::Clients

Navigateurs internet





Web::Clients



Autres application utilisant le web :





Objectif TP et Cours



Notre super bibliothèque

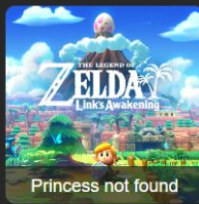
Filtres :

Players

- ☒ Multijoueur
- ☒ Solo

Tri :

Alphabétique ▼



Princess not found



Le chapelier fou



Dab Simulator



League of wood



Overwatch 1.2



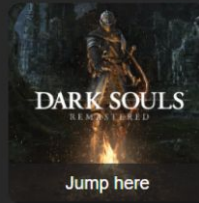
Le ouicheur 4



Chill/20



Tomber dehors 3



Jump here



HTML



Hyper Text Markup Language

- Langage de **balisage**
- Définit des pages web
- Version 5.2
- Composé **d'éléments**
- Les **balises** servent à représenter ces éléments
- Un élément est composé **d'attributs**





HTML



- Un élément est un composant d'un document HTML.
- Un élément est représenté par une balise.

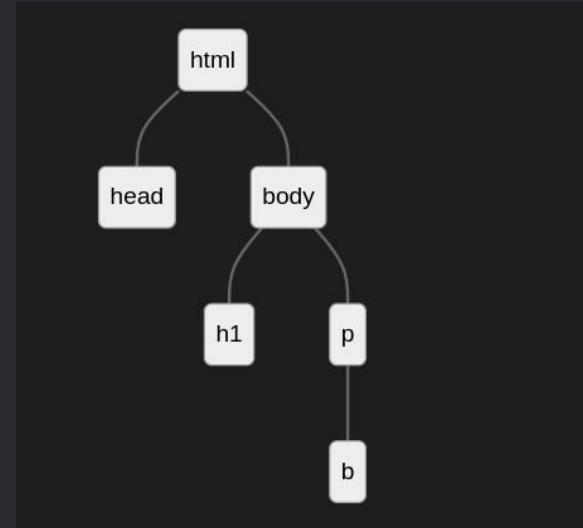


HTML

- Une page HTML peut être représentée comme un arbre dont les nœuds seraient les éléments

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4    </head>
5    <body>
6      <h1>Titre</h1>
7      <p>
8        <b>Hello</b>there
9      </p>
10   </body>
11 </html>
```

Document HTML



Représentation en arbre



HTML::Éléments à retenir



- `<p>` : paragraphe
- `` : bold
- `<h1~h9>` : heading : titre
- `` : ajoute une image
- `<input>`
 - Utilisé avec certains attribut : button, checkbox,
- `<select>` : Permet de créer une combobox
 - On utilise `<option>` pour ajouter des éléments
- `<div>` : Permet de structurer le document
- `<ul / li>` : liste non ordonnée



HTML::Attributs



Un élément HTML possède une **liste d'attributs**.

Un **attribut** ajoute des informations complémentaires à un élément.

Syntaxe :

Valeur de l'attribut

`<p id="first"> text </p>`

nom de l'attribut



HTML::Attributs à retenir

- **id** : Donne un identifiant unique à un élément (important pour le sélectionner plus tard)
- **class** : donne une “classe” à un élément (permettra également d’identifier les éléments d’une classe commune plus tard)



HTML::Structure d'un fichier



Un fichier **HTML** est découpé en plusieurs parties :

- **Doctype**
 - Précise le type de document. Obligatoire.
 - Dans notre cas on utilisera toujours `<!DOCTYPE html>`
- **<head>**
 - Contient des informations meta sur la page
- **<body>**
 - Contient le cœur de la page.

HTML::Exemple d'application



Notre super bibliothèque

Filtres :

Multijoueur ☐ Solo ☐ Action ☐ Aventure ☐

- Zeldo : A lonk to the future
- Méga Rioma
- Dab Simulator
- La ligue des légendaires
- Par dessus la montre 3
- Le ouicheur 4
- La vallée des étoiles
- Tomber dehors édition nucléaire
- Les sombres âmes
- L'anneau des vieux
- L'art de la guerre
- Les monstres de la poche



CSS



- Avant d'aller plus loin, profitons de ces rares images d'un développeur en pleine édition de CSS

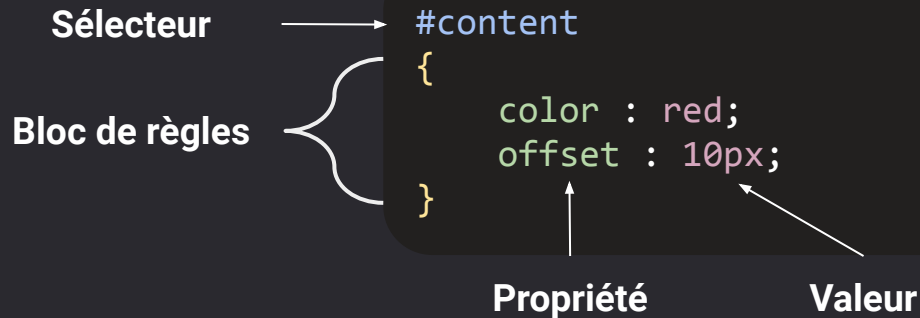




CSS

Cascading Style Sheet

- Modifie l'apparence d'une page HTML
- Sépare la partie **présentation** de la partie **modèle**
- Composé d'un ensemble de **blocs de règles** associés à un **sélecteur**





CSS::Sélecteurs

Le **sélecteur** désigne les éléments de la page **HTML** qui seront affectés par le bloc de règle qui lui est associé.

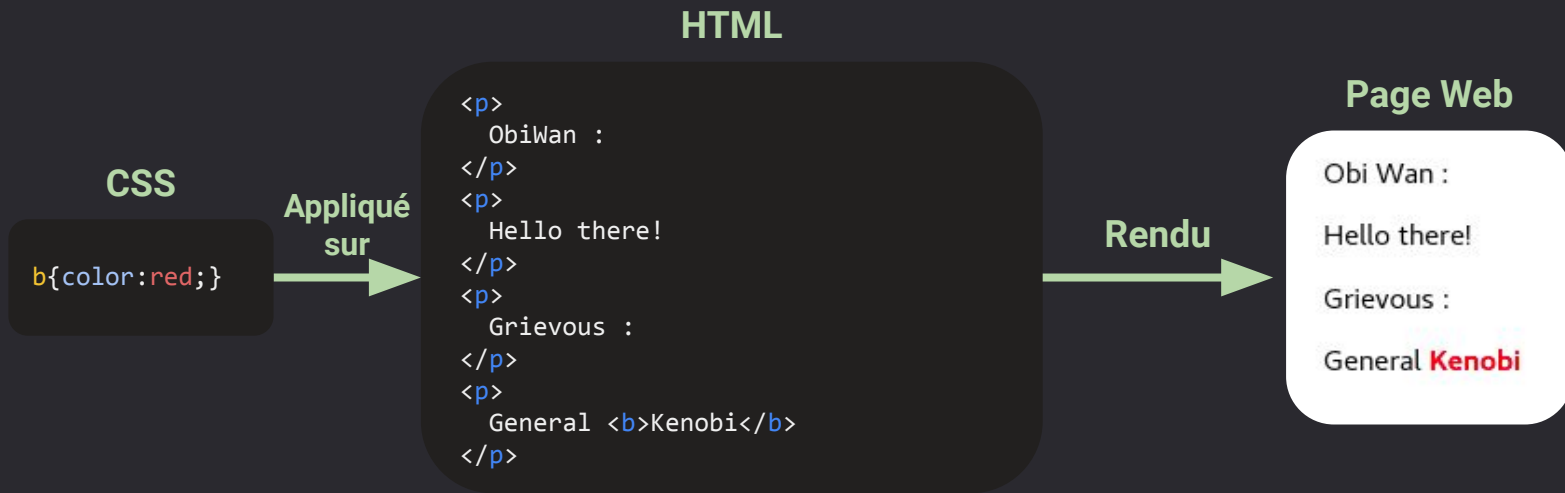
Il existe plusieurs type de sélecteurs

- Sélecteur **d'élément**
- Sélecteur **d'identifiant**
- Sélecteur de **classe**
- Sélecteur de **groupe**
- Sélecteur **universel**
- Sélecteur de **Pseudo-Classes**



CSS::Sélecteurs d'éléments

Le **sélecteur d'éléments** sélectionne tous les **éléments** d'un type (ici les éléments de type **b**) et leur applique le bloc de règle associé.





CSS::Sélecteurs d'identifiant

Le **sélecteur d'identifiant** débute par le caractère # et sélectionne l'élément dont la valeur de l'**attribut id** correspond à celle indiquée dans le sélecteur.

L'élément sélectionné se verra ensuite appliquer le bloc de règle associé au sélecteur.



CSS::Sélecteurs de classe

Le **sélecteur de classe** débute par le caractère . et sélectionne les éléments dont la valeur de l'**attribut class** correspond à celle indiquée dans le sélecteur.

Les éléments sélectionnés se verront ensuite appliquer le bloc de règle associé au sélecteur.



CSS::Sélecteur de groupe

Le **sélecteur de groupe** permet de regrouper plusieurs sélecteurs et d'éviter les répétitions.

Le bloc de règle sera appliqué à tous les éléments du groupe.





CSS::Sélecteur universel

Le **sélecteur universel** * permet de sélectionner tous les éléments de la page.

Le bloc de règle sera appliqué à tous les éléments de la page.





CSS::Sélecteur de pseudo-classes

Le **sélecteur de pseudo-classes** se combine à un des sélecteurs précédents et permet d'appliquer une règle lorsqu'un élément se trouve dans un état particulier. Par exemple, la pseudo classe hover s'applique lorsque la souris survole l'élément ciblé.

CSS

```
#jedi:hover  
{  
  color:aqua;  
}
```

Appliqué
sur

HTML

```
<p id="jedi" class="name">  
  ObiWan :  
</p>  
<p class="line">  
  Hello there!  
</p>  
<p class="name">  
  Grievous :  
</p>  
<p class="line">  
  General <b>Kenobi</b>  
</p>
```

Rendu

Page Web

Obi Wan :
Hello there!
Grievous :
General Kenobi



CSS::Sélecteurs de pseudo-classes

Exemple de Pseudo classes :

- **hover** : Se déclenche lors d'un survol de l'élément.
- **checked** : Se déclenche lorsque l'élément est "checked" (cf checkboxes).
- **focus** : Se déclenche lorsqu'un élément a le focus.
- **first-child** : Premier élément d'un groupe de frères.
- **active** : Se déclenche après activation (généralement après un clic)
- **empty** : Se déclenche lorsqu'un élément ne possède pas d'enfants.



CSS::Propriétés

Un bloc de règle est constitué d'un ensemble de couple {propriété;valeur}.

Certaines valeurs, numériques notamment, ont un type.

- px : distance en pixel
- % : distance relative à la taille de la page
- em : Relatif à la taille de la fonte. 2em = 2 fois la taille de la fonte
- cm : En cm, dépend du DPI de l'écran. (Déconseillé pour du web)

Exemple d'utilisation :

```
#jedi:hover  
{  
  margin:10px 2em 3cm 23%;  
}
```



CSS::Propriétés



Propriétés communes

- color : couleur du texte
- background-color : couleur de fond
- font-weight : taille du texte
- border-size : taille de la bordure :
- margin : Distance entre la bordure et le conteneur de niveau supérieur
- padding : Distance entre le contenu et la bordure



CSS::Intégration



On peut intégrer une feuille CSS de 2 manières :

- En ajoutant une balise `<style>` et en écrivant directement les règles à l'intérieur de cette balise
- En utilisant la balise `link` dans la partie `<head>`
 - `<link rel="stylesheet" href="style.css">`



Web::Javascript



Langage de programmation employé pour la création de pages web dynamiques et interactives.

- Code interprété côté client (navigateur).
- Langage orienté objet (plus précisément objet à prototype)



Web::Javascript::Intérêt



Le Javascript possède des fonctionnalités qui lui permettent d'interagir avec les éléments d'une page web, d'en modifier dynamiquement les propriétés et de réagir à divers types d'événements (click, mousemove etc).

- Possible d'ajouter ou de supprimer des éléments
- Possibilité d'ajouter, de supprimer ou de modifier des attributs
- Possibilité de modifier des règles css
- etc

Javascript::Types primitifs

- **Strings** : chaîne de caractère
- **Number** : nombre flottant
- **BigInt** : gros entier
- **Boolean** : faux/vrai
- **Undefined** : type automatiquement assigné à une variable uniquement déclaré
- **null** : Variable marquant l'absence de valeur
- **symbol** : hors scope, mais ça existe

Javascript::Objet

- Un objet une variable qui contient des propriétés et des méthodes (fonction associé à un objet).
- Comment définir un objet en javascript :

```
var pokemon = {type="electric", name="voltoutou", attackSpe=2000000 };
```

- On peut ensuite accéder aux propriétés de l'objet ainsi :

```
console.log(pokemon.type); //Affiche electric dans la console
```



Javascript::Objet

- Il est possible d'ajouter des propriétés ou des méthodes après la création de l'objet.

```
var pokemon = {type="electric", name="voltoutou", attackSpe=2000000 };  
pokemon["defense"]=2000000; // Ajoute la propriété défense à l'objet en cours
```

- Un objet javascript contient une liste de propriétés accessibles et modifiables à l'aide d'un identifiant.

Javascript::Tableaux

- Un tableau est une variable qui peut enregistrer plusieurs valeurs :

```
var bestPokemons = ["voltoutou","moumouton"];
```

- Pour accéder à un élément, on utilise son index avec l'opérateur []:

```
console.log(bestPokemons[0]); // Affiche voltoutou dans la console
```

- Les tableaux sont des objets, ils ont des méthodes associés :
 - `length()` : retourne le nombre d'élément dans le tableau
 - `sort()` : trie le tableau
 - etc

Javascript::Conditions

- Code exécuté en fonction du résultat de la condition
- If

```
if(condition==true)
{
    return a+b;
}
```

- else

```
if(condition==true)
{
    return a+b;
}
else
{
    return b;
}
```



Javascript::Conditions::if

- else if

```
if(condition==true)
{
    return a+b;
}
else if( condition2==true)
{
    return a;
}
```

- switch

```
switch(value)
{
    case "first":
        console.log("firstCase");
        break;
    case "second":
        console.log("secondCase");
        break;
    default:
        console.log("defaultCase");
}
```




Javascript::Fonctions

- Une fonction est une suite d'instructions.
- Identifié par un nom.
- Une fonction peut éventuellement “retourner” une valeur.
- Une fonction peut éventuellement prendre des valeurs en paramètres.

Javascript::Fonctions::Syntaxe

- Exemple de fonction

```
function sum(a,b)
{
    return a+b;
}
```

- Exemple d'invocation de fonction :

```
sum(10,12); // La fonction retournera la valeur 22
```

Javascript::Boucles

- Une boucle while permet de répéter un ensemble d'instructions aussi longtemps que la condition entre parenthèse est vrai

```
while(a<10)
{
    a = a+b;
}
```

Javascript::Boucles::for

- La boucle for permet de répéter un jeu d'instruction un n fois.
- Syntaxe facilitant l'initialisation d'un compteur.

```
for(let i = 0; i<10; i++)  
{  
    a = a+b;  
}
```

Javascript::Boucles::for in

- La boucle for in permet de répéter un ensemble d'instructions pour chaque élément d'un tableau.

```
for(let a in array)
{
    console.log(a);
}
```



Javascript::ArrowFunction

- Une “ArrowFunction” est une autre manière de créer des fonctions
- La fonction est enregistrée dans un objet réutilisable.
- Aussi appelé lambda dans d'autres langages.

```
var sum = (a, b) =>
{
    return a + b;
};
```

```
console.log(sum(1,2)); // Affiche 3 dans la console
```

Javascript::Variables

- Une variable est un conteneur qui enregistre des données (ou valeurs)

```
var cake = 10;  
console.log(cake); // affiche la valeur de cake dans la console, qui est 10
```

Il existe 3 façons de déclarer une variable en JS :

- var
- let
- const

Javascript::Variables::var

- Portée de la déclaration : function

```
function baking()
{
    var cake = 10;
    {
        var cookies = 20;
    }
    console.log(cake);    // 10
    console.log(cookies); // 20
}

function cooking()
{
    console.log(cake); // undefined
}
```


Javascript::Variables::var

- Une variable déclarée par var peut être “hoisted”.
- Hoisted signifie que l’interpréteur décide de déclarer la variable au début du scope dans lequel il se trouve

Exemple :

```
function baking()
{
    cake = 10;
    console.log(cake); // 10. Valide car la variable cake est hoisted
    var cake;          // C'est à dire que cake est en réalité déclaré au début de
                        // la fonction
}
```

Javascript::Variables::var

- Une variable déclaré par var peut être réassignée

```
function cooking()  
{  
    var cake = 10;  
    cake     = 20;  
}
```

- Une variable var peut être redéclaré

```
function cooking()  
{  
    var cake = 10;  
    var cake = 20; // pas d'erreur  
}
```

Javascript::Variables::let

- Portée de la déclaration : block

```
function baking()  
{  
  let cake = 10;  
  {  
    let cookies = 20;  
  }  
  console.log(cake);           // 10  
  console.log(cookies);       // undefined  
}
```





Javascript::Variables::let

- Peut être réassignée

```
function baking()  
{  
  let cake = 10;  
  cake = 20;  
  console.log(cake) // 20  
}
```

- Non sujet au hoisting

```
function baking()  
{  
  cake = 10; // Erreur, cake n'est pas encore défini  
  console.log(cake) // 20  
  let cake;  
}
```

Javascript::Variables::const

- Portée de la déclaration : block

```
function baking()  
{  
    const cake = 10;  
    {  
        const cookies = 20;  
    }  
    console.log(cake);           // 10  
    console.log(cookies);       // undefined  
}
```

Javascript::Variables::const

- La référence ne peut être réassignée

```
function baking()  
{  
    const cake = 10;  
    cake = 20;    // Erreur  
    console.log(cake)    // 20  
}
```

- Non sujet au hoisting

```
function baking()  
{  
    cake = 10;    // Erreur, cake n'est pas encore défini  
    console.log(cake) // 20  
    const cake;  
}
```

Javascript::Variables::Récap

VAR vs LET vs CONST

	var	let	const
Stored in Global Scope	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can Be Reassigned?	✓	✓	✗
Can Be Redeclared?	✓	✗	✗
Can Be Hoisted?	✓	✗	✗



Javascript::Window

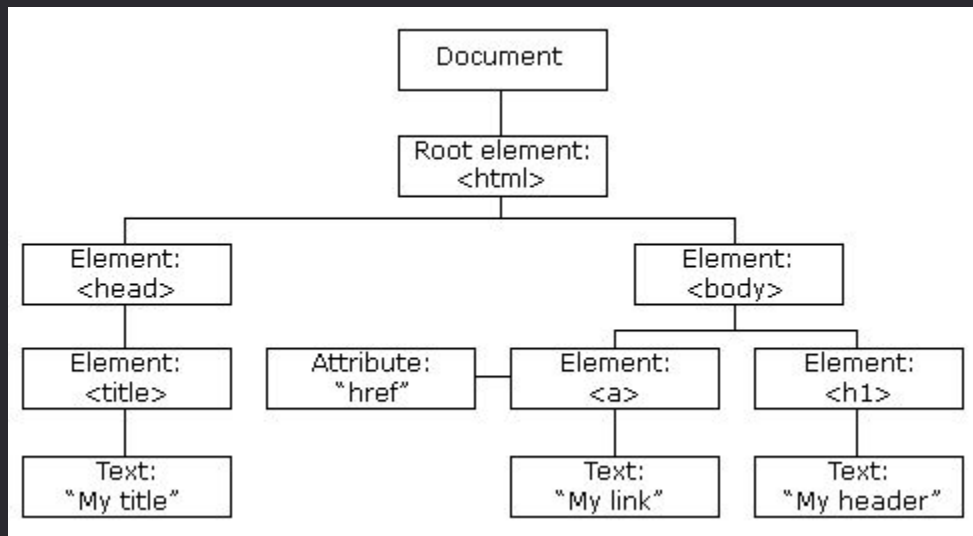


- L'objet window représente une fenêtre ouverte à l'intérieur d'un navigateur
- Accessible dans toute balise script
- Contient un ensemble de propriétés et de fonctions concernant la fenêtre et le navigateur.
 - screen : Contient des informations concernant l'écran (dimension, encodage couleurs)
 - confirm() : Affiche une fenêtre de validation
 - alert() : Affiche une fenêtre d'alerte

Javascript::DOM

DOM : Document Object Model

Interface permettant d'accéder aux éléments d'une page HTML





Javascript::DOM



- L'objet document est une propriété de l'objet windows
- Il est possible d'accéder directement au document sans écrire window.document.
- Racine du document HTML
- Représente la page HTML dans son entièreté
- Fonctions utiles :
 - document.getElementById("id");
 - Permet de récupérer un élément de la page HTML en fonction de son attribut ID
 - document.getElementsByTagName("tagName");
 - Permet de récupérer les éléments du type (tag) donné en paramètre



Javascript::Événements

- Un événement est un signal émis par un document HTML lorsque certaines conditions sont remplies.
- Par exemple, lorsque l'utilisateur clic sur un bouton, un événement "click" est lancé.
- Il est possible de lier un événement à une fonction afin d'y réagir.



Javascript::Événements

- `addEventListener()`
- Permet d'ajouter une fonction à un événement

Exemple :

```
function onLoad()  
{  
    console.log("Document loaded");  
}  
  
document.addEventListener("DOMContentLoaded", onLoad);
```



Javascript::Événements

Type d'événements :

- change : lorsqu'un élément est modifié
- click : lorsqu'on click sur un élément
- DOMContentLoaded : une fois que le document est chargé
- mouseenter : lorsque la souris "entre" dans un élément
- mouseleave : lorsque la souris "sort" d'un élément

Javascript::Intégration

Il est possible d'intégrer du code dans une page de plusieurs manières

- En ajoutant le code directement dans la balise script à l'intérieur de notre document HTML

```
<script>  
console.log("hi");  
</script>
```

- En référençant un fichier contenant le code javascript

```
<script src="monJs.js"></script>
```



Framework



- Qu'est ce qu'un framework?
 - Un framework est une solution logicielle qui fournit au développeur un cadre générique pour réaliser certaines applications.
 - Exemple de Framework :
 - C++ : Qt : Framework de création d'interfaces.
 - C# : Monogame : Framework de création de jeu vidéo.



Framework web

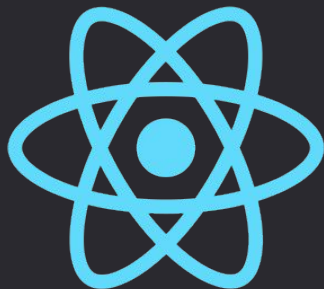


- Qu'est ce qu'un framework Web?
 - Solution logicielle pour faciliter la création d'applications Web.
 - Sécurité.
 - Données : Database access, data mapping.
 - Architecture : force la séparation entre les données et leur présentation.

Principaux Frameworks Web



Angular



React JS

VueJS

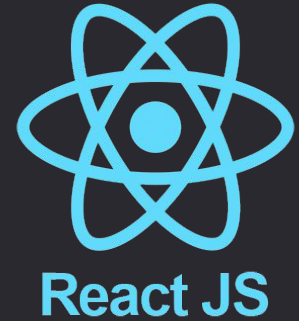




React



- OpenSource
- Maintenu par Meta (facebook)
- Framework le plus utilisé
- Première version : 2013
- Javascript



Pour rappel, meta,
c'est aussi ça



Angular

- OpenSource
- Maintenu par Google
- Première version : 2016 (2010 pour l'ancienne version AngularJS)
- Typescript





VueJS

- OpenSource
- Maintenu par la communauté
- Première version : 2014
- Javascript
- Léger
- Utilise des noms d'animés en guise de version (actuellement à la version "The quintessential QUintuplets")





VueJS

Versions [\[edit \]](#)

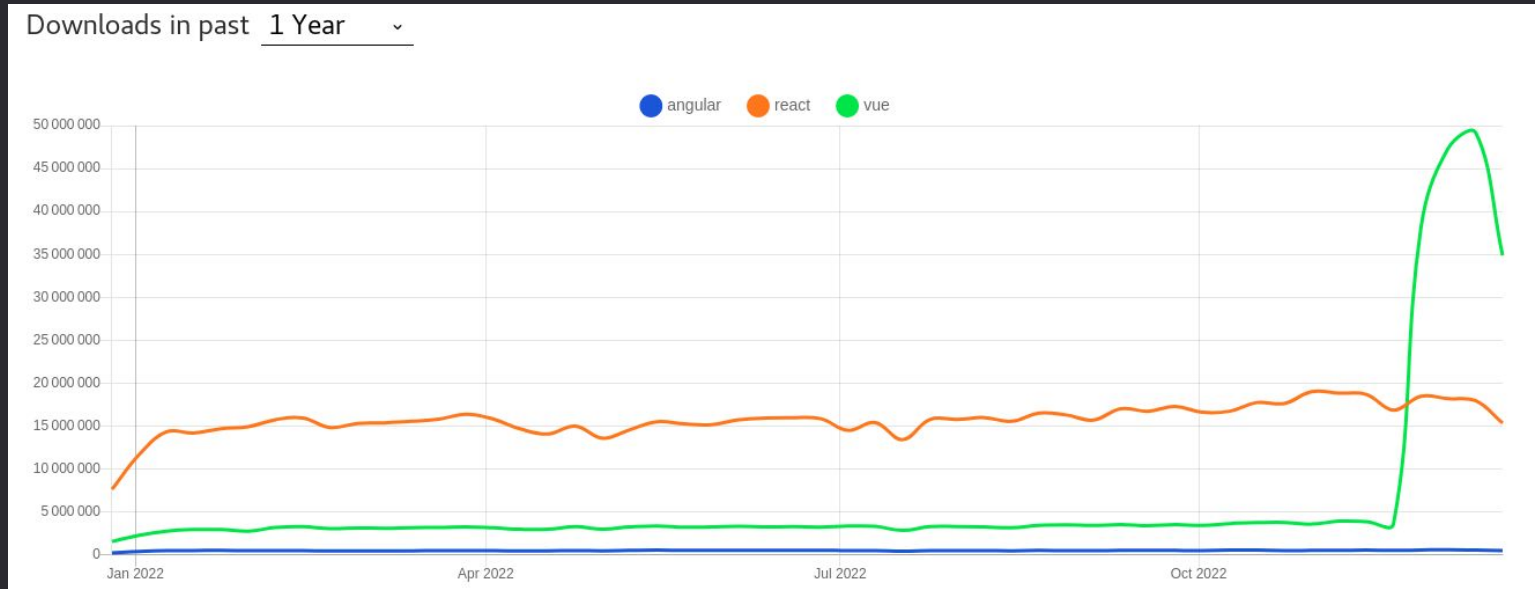
Version	Release date	Title	End of LTS	End of Life
3.2	August 5, 2021	Quintessential Quintuplets ^[15]		
3.1	June 7, 2021	Pluto ^[16]		
3.0	September 18, 2020	One Piece ^[17]		
2.7	July 1, 2022	Naruto ^[18]	December 31, 2023	December 31, 2023
2.6	February 4, 2019	Macross ^[19]	March 18, 2022	September 18, 2023
2.5	October 13, 2017	Level E ^[20]		
2.4	July 13, 2017	Kill la Kill ^[21]		
2.3	April 27, 2017	JoJo's Bizarre Adventure ^[22]		
2.2	February 26, 2017	Initial D ^[23]		
2.1	November 22, 2016	Hunter X Hunter ^[24]		
2.0	September 30, 2016	Ghost in the Shell ^[25]		
1.0	October 27, 2015	Evangelion ^[26]		
0.12	June 12, 2015	Dragon Ball ^[27]		
0.11	November 7, 2014	Cowboy Bebop ^[28]		
0.10	March 23, 2014	Blade Runner ^[29]		
0.9	February 25, 2014	Animatrix ^[30]		
0.8	January 27, 2014	__ ^[31]		
0.7	December 24, 2013	__ ^[32]		
0.6	December 8, 2013	VueJS ^[33]		



Utilisation



Téléchargements des Frameworks :





Choix : VueJS



Pourquoi utiliser VueJS?

- Développement plus simple
- Déploiement plus simple
- Plus récent
- Plus en phase avec les valeurs de l'open Source (Non géré par une méga corporation)
- En phase ascendante.



Installation



VueJS peut être utilisé de 2 manières :

- La manière dite “standard” : Utiliser npm pour créer un nouveau projet
- Balise script qui récupère le framework sur un CDN (content delivery network)

○ `<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>`



Outils



- Visual Studio Code
 - Extensions :
 - Live Server : Lance un serveur local avec notre application.
 - Vue Language Features : Support pour VueJS.
- Navigateur Web
 - Outils de développement.



Objectif

Le retour de la vengeance :

Notre super bibliothèque

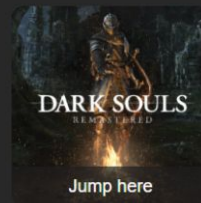
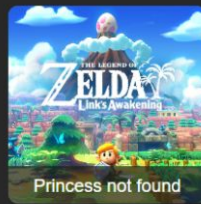
Filtres :

Players

- ☒ Multijoueur
- ☒ Solo

Tri :

Alphabétique ▼





Initialisation



- Première étape :
 - Créer un nouveau fichier html.
 - Repartir d'une page HTML vide.

```
<!doctype html>
<html>
  <head></head>
  <body>
  </body>
</html>
```



Initialisation



- Ajouter la balise pour intégrer VueJS

```
<!doctype html>
<html>
  <head></head>
  <body>
    <script src="https://unpkg.com/vue@3/dist/vue.global.js">
    </script>
  </body>
</html>
```



Initialisation



- Pour utiliser VueJS :
 - Créer une instance de l'objet App
 - `let app = Vue.createApp({});`
 - “Monter” notre objet à un élément.
 - `app.mount("#app");`



Initialisation



```
<!doctype html>
<html>
  <head></head>
  <body>
    <div id="app">Hello There</div>
    <script src="https://unpkg.com/vue@3/dist/vue.global.js">
    </script>
    <script>
      let app = Vue.createApp({});
      app.mount("#app");
    </script>
  </body>
</html>
```



Data Function



- Fonction data :
 - Retourne un objet.
 - L'Objet contient les données de l'application.

```
<script>  
  let app = Vue.createApp({  
    data: function(){  
      return {dialog:"Hello There"}  
    }  
  });  
  app.mount("#app");  
</script>
```



Double Mustache {{}}



- Permet d'utiliser une variable de l'objet data.
- La {{}} doit être dans un élément HTML sur lequel on a monté notre instance de VueJS.

```
<div id="app">{{dialog}}</div>
```




Directives::v-model

- v-model
 - Permet de lier le contenu d'un élément à une variable.
 - Two way Data Binding.

```
<div id="app">  
  {{dialog}}  
  <input v-model="dialog"/>  
</div>
```



Directives::v-if



- v-if
 - Prends un booléen en paramètre.
 - Conditionne l'affichage d'un élément HTML.
 - L'élément n'existe pas dans le DOM.

```
<div id="app">  
  <p v-if="isVisible"> {{dialog}} </p>  
</div>
```



Directives::v-show



- v-show
 - Prends un booléen en paramètre.
 - Conditionne l'affichage d'un élément HTML.
 - L'élément existe pas dans le DOM.
 - Passe la propriété css display à none.

```
<div id="app">  
  <p v-show="firstCondition"> {{dialog}} </p>  
</div>
```



Directives::v-else

- v-else
 - Associé à une instruction v-if (ou v-else-if).
 - Affiche un élément dans le cas où la condition du if est **false**.
 - L'élément n'existe pas dans le DOM.

```
<div id="app">  
  <p v-if="firstCondition"> If {{dialog}} </p>  
  <p v-else> Else {{dialog}} </p>  
  <input v-model="dialog"/>  
</div>
```



Directives::v-else-if

- v-else-if
 - Associé à une instruction v-if.
 - Affiche un élément dans le cas où la condition du if est **false** et si la condition else if est **true**.
 - L'élément n'existe pas dans le DOM.

```
<div id="app">  
  <p v-if="firstCondition"> If {{dialog}} </p>  
  <p v-else-if="secondCondition"> Else if {{dialog}} </p>  
  <p v-else> Else {{dialog}} </p>  
</div>
```



Directives::v-cloak

- v-cloak
 - Masque un élément tant que la page n'est pas chargée.
 - Règle css définit comment l'élément cloaked est affiché pendant le chargement :

```
<style>
  [v-cloak]{display:none}
</style>
<div id="app" v-cloak>
  <p v-if="firstCondition"> If {{dialog}} </p>
</div>
```



Directives::v-for

- v-for
 - Permet de répéter un élément x fois
 - Utile pour afficher une liste d'éléments :

```
let app = Vue.create App({  
  data: function(){  
    return {items:["Hello", "There"]}  
  });  
...  
<ul>  
  <li v-for="item in items">{{item}}</li>  
</ul>
```



Directives::v-on



- v-on
 - Exécute une fonction callback lorsqu'un événement est déclenché.
 - Exemple d'événement : click

```
<div id="app" v-cloak>  
  <p v-if="firstCondition"> If {{dialog}} </p>  
  <button v-on:click="firstCondition= !firstCondition;">  
    Show Text  
  </button>  
</div>
```




Directives::v-on

- Raccourcis pour v-on : @
- Possibilité d'invoquer une fonction définie dans l'instance de l'objet App :
- Le mot clé `this` permet d'accéder aux variables à l'intérieur de notre objet App



Directives::v-on



```
<script>
  let app = Vue.createApp({
    data: function() {},
    methods: {
      toggleText() {
        this.firstCondition = !this.firstCondition;
      }
    }
  });
</script>
<div id="app" v-cloak>
  <p v-if="firstCondition"> If {{dialog}} </p>
  <button @click="toggleText">
    Show Text
  </button>
</div>
```



Directives::v-on



- Autres événements:
 - keyup
 - keydown
- keyUp et keydown prennent un “modifieur”
 - Contient la touche sur laquelle on souhaite réagir
- Exemple d'un événement keydown :

```
<div id="app" v-cloak>  
  <input @keydown.enter:"showText(greeting+'!')" v-model="dialog"/>  
</div>
```



Directives::v-on



- Sur les modifieurs :
 - Existent aussi pour la souris :
 - @click.right : réagit à un clic sur le bouton droit de la souris.
 - Modifient les événements :
 - stop: stop la propagation d'un événement dans le DOM
 - capture : déclenche l'événement pendant la phase de capture.



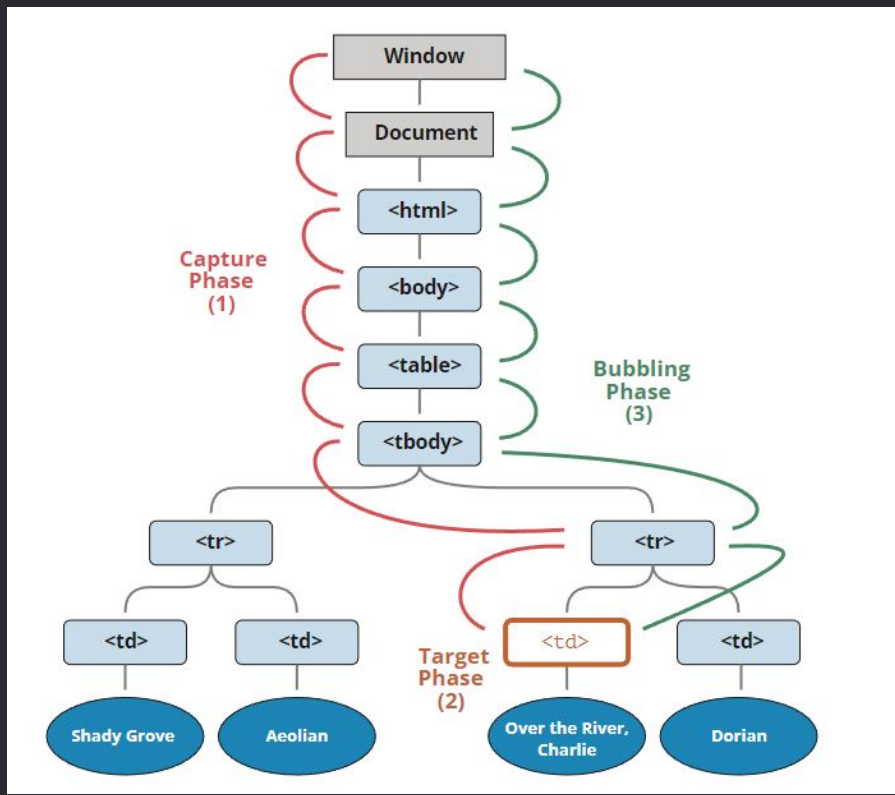
Directives::v-on



- self : l'événement se lance uniquement s'il est sur l'élément ciblé.
- Il est possible d'enchaîner les événements :
 - @click.self.right : l'événement ne se déclenche que sur l'élément cible et lorsque le bouton droit de la souris est enfoncé.



Directives::v-on::Event Bubbling





Directives::v-on::Event Bubbling

- La phase de capture n'est généralement pas utilisée
 - Sauf si spécifié par le modifier capture.
- Phase de capture :
 - Les événements descendent de la racine jusqu'à la cible
- Cible atteinte.
- Phase de "bubbling" :
 - les événements remontent l'arbre de la cible jusqu'à la racine.



Directives::v-on::Event Bubbling

- Concrètement, si vous cliquez sur td et ajoutez un événement @click au niveau de l'élément tr, il sera lui aussi exécuté (sauf si on utilise stop).



Components



- VueJS permet la création de components.
- Un component est réutilisable.
- Un component est indépendant.
- Un composant contient
 - template : Partie présentation de notre composant.
 - data : les données/variables du component.
 - methods : les fonctions du component.



Components



Comment créer un nouveau component avec VueJS :

- Utiliser la fonction component de l'objet app :
- Premier paramètre : nom du component
- Deuxième argument : Objet qui contient Données/méthodes (similaire à App)

```
let newComponent= app.component("newComponent",  
{  
  data:function(){ return { filter:true }},  
  template:'<p> Mon component </p>'  
}
```



Components



Une fois le component définit, il suffit de créer un élément avec le nom donné en premier argument à la fonction component :

```
<div id="app" v-cloak>  
  <newComponent></newComponent>  
</div>
```

Components::Props

Props, pour propriétés :

- Il est possible de définir des propriétés personnalisées à nos component.
- Propriété props de l'objet component :

```
let newComponent= app.component("newComponent",  
{  
  data:function(){ return { filter:true }},  
  props:["title"],  
  template:'<p> {{title}}</p>'  
}
```



Components::Props

Et dans la partie HTML :

```
<div id="app" v-cloak>  
  <newComponent title="Hi"></newComponent>  
</div>
```



Components::slot



Il est possible de récupérer le contenu à l'intérieur de notre custom component à l'aide de la balise `<slot/>`

```
<div id="app" v-cloak>
  <new_component title="Hi"> This is my content </new_component>
</div>
```

...

```
let newComponent= app.component("new_component",
{
  data:function(){ return { filter:true }},
  props:["title"],
  template:'<p> {{title}} <slot/> </p>'
}
```

Components::Events

Communication entre components : Via des events.

- Création d'une fonction qui émet un événement :
- Premier paramètre : nom de l'événement.
- Deuxième paramètre : liste de paramètres à envoyer.

```
let newComponent= app.component("new_component",  
{  
  template:'<p> {{title}} <slot/> </p>',  
  methods: {  
    sendMessage(){ this.$emit('message_sent',"Hi");  
  }  
}
```



Components::Events



Comment réagir à cet événement?

- Depuis un component parent :

```
let parentComponent = app.component("parent",  
{  
  template: '<new_component  
@message_sent="onMessage"><new_component>',  
  methods: {  
    onMessage(arg){ console.log("hi")};  
  }  
})
```