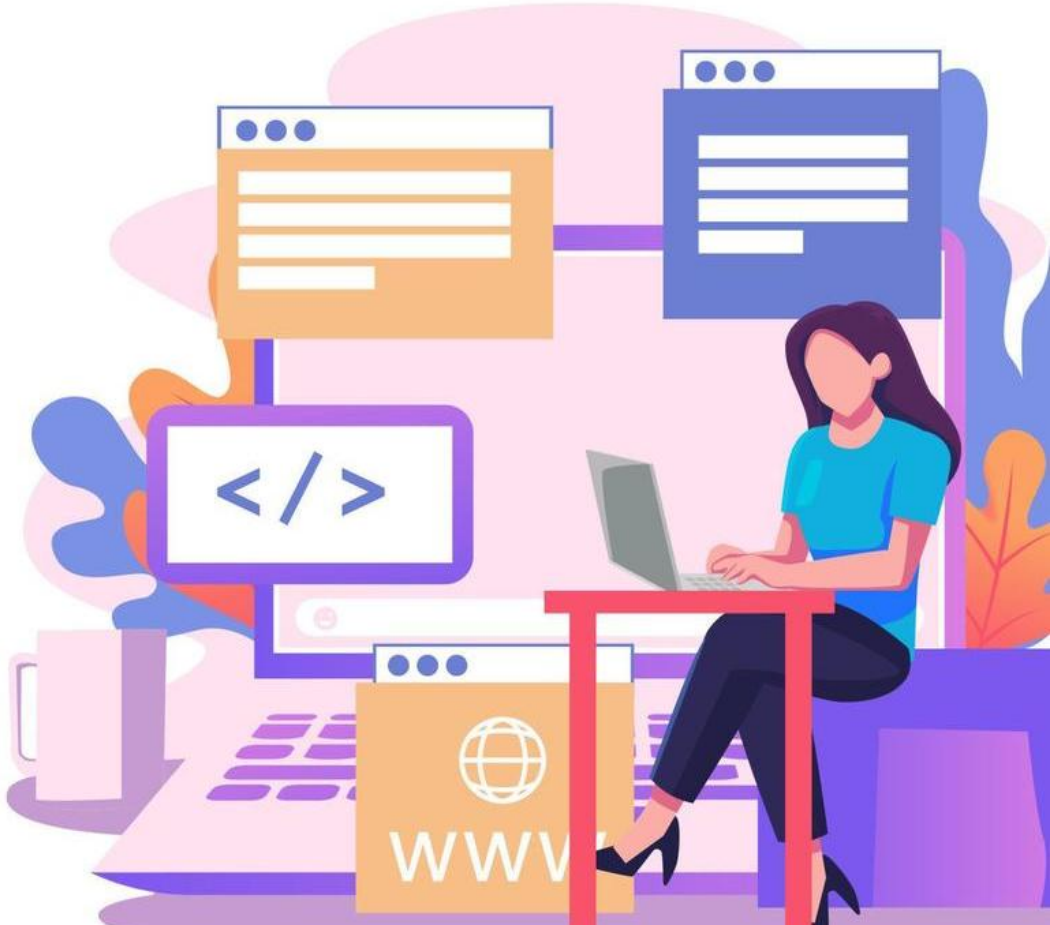
# DÉVELOPPEMENT D'APPLICATIONS WEB

FRONTEND | BACKEND | BASES DE DONNÉES

## TypeScript

& NodeJS




charles.meunier@ube.fr

# I JavaScript

## Quelques rappels

# JavaScript ?

- langage de programmation **interprété**
- utilise le **typage dynamique**
- prend en charge les paradigmes **impératif**, **fonctionnel** et **orienté objet** *(partiellement)*
- **seul** langage de programmation **nativement supporté** par tous les navigateurs

# Variables & Constantes

## var

```
function test(value)
{
    if(value === 0)
    {
        var data = 5;
    }

    console.log(data); // ok
}
```

## let

```
function test(value)
{
    if(value === 0)
    {
        let data = 42;
    }

    console.log(data); // error
}
```

## const

```
function test(value)
{
    if(value === 0)
    {
        const data = 42;
        data = 7: // error
    }

    console.log(data); // error
}
```

À ÉVITER

FORTEMENT RECOMMANDÉ

# Tableaux

```
//Déclaration d'un tableau
const array = [ 1, '2', "Bonjour", [1, 2, 3] ];

//Récupération du 2ème élément du tableau
const secondValue = array[1];

//Récupération de la taille du tableau
const arraySize = array.length;

//Ajout d'un élément au tableau
array.push(42);

//Suppression du 4ème élément
array.splice(3, 1);

//Duplication d'un tableau (spread operator)
const array2 = [...array];
```

> **array** est une constante mais peut tout de même être modifié.
>
> **Pourquoi ?**

# Fonctions

## fonction

```
function equalZero(value)
{
    return value === 0;
}
```

## fonction anonyme

```
const two = [1, 2, 3].find(
    function(item)
    {
        return item === 2
    }
);
```

Attention aux utilisations avec `this`

## fonction fléchée

```
const two = [1, 2, 3].find(
    item => item === 2
);
```

A privilégier aux fonctions anonymes... sauf dans quelques cas

# Objets

- Déclaration et manipulation

```
const john = {
    name: "John",
    age: 34,
    phoneNumber: "0606060606"
}

console.log(john.name)
```

> Même structure qu'un fichier JSON
> *(JavaScript Object Notation)*

- Destructuration

```
const { name } = john

console.log(name)
```

# Classes

- Déclaration et manipulation

```
class User {
    #name
    #age
    #phoneNumber

    constructor(name, age, phoneNumber)
    {
        this.#name = name
        this.#age = age
        this.#phoneNumber = phoneNumber
    }

    toString()
    {
        return `${this.#name} (${this.#age}) - ${this.#phoneNumber}`
    }
}

const john = new User("John", 34, "0606060606")

console.log(`${john}`)
```

# = Attributs privés

# Fonctionnement asynchrone & Promesses

- Promesses

```
function loadPage(url)
{
    fetch(url).then((response) => {
        response.text().then((html) => {
            console.log(html)
        }
    }
}
```

- Chaînage des promesses

```
function loadPage(url)
{
    fetch(url)
        .then(response => response.text())
        .then(html => console.log(html))
        .catch(error => console.error(error))
}
```

# async / await

```
async function loadPage(url)
{
    const response = await fetch(url)
    const html = await response.text()
    console.log(html)
}
```

# Types natifs

| TYPE | DESCRIPTION | EXEMPLES | PASSAGE PAR |
|---|---|---|---|
| `string` | Chaine de caractères | `"L'improbable"`<br>`'Il a dit "Oui !" à ma proposition'`<br>`` `Mon nom est ${name}` `` | **recopie** |
| `number` | Tous les nombres jusqu'à $2^{53}$ - 1 | `42.6`<br>`-Infinity`<br>`NaN` | |
| `boolean` | Valeur booléenne | `true / false` | |
| `bigint` | Très grand nombre | `123123123123123123123123123123123n` | |
| `undefined` | Lorsqu'aucune variable/constante n'a pas été initialisée. | `let value;  //value = undefined` | |
| `null` | Représente une absence de valeur (mais ce n'est pas `undefined`) | `const item = document.querySelector(".not-in-page")`<br>`//item = null` | |
| `symbol` | Pour créer des valeurs uniques... mais pas que. | `const helloSymbol = Symbol("HelloWorld")` | |
| `object` | Tout le reste (tableaux, fonctions, classes, ...) | `const array = ['A', 'B', 'C']`<br>`console.log(typeof array)  // "object"` | **référence** |

# NodeJS

Exécuter du JS hors navigateur

# NodeJS ?

NodeJS est un environnement permettant d'**exécuter du code JavaScript en dehors d'un navigateur**.

Pour cela, il combine le **moteur JavaScript V8** et des **API de bas niveau** pour pouvoir interagir avec le système de fichiers, les processus et le réseau.

Comme le JavaScript est exécuté en dehors d'un navigateur, **il n'y a pas de DOM à manipuler**.

# Installer NodeJS

- Depuis le site officiel de NodeJS

  Suivez les instructions disponibles sur https://nodejs.org en fonction de votre système d'exploitation.

  Pour connaître la version de NodeJS installée sur votre poste :

```
# Indique la version de NodeJS installée
node -v
```

- Via NVM (NodeJS Version Manager)

  Suivez les instructions disponibles sur https://nvmnode.com en fonction de votre système d'exploitation.

  Puis utilisez les commandes suivantes pour installer la version de NodeJs souhaitée :

```
# Lister les versions de NodeJS disponibles
nvm ls-remote

# Installer une version spécifique de NodeJS
nvm install [node-version]

# Utiliser une version de NodeJS parmi celles installées
nvm use [node-version]
```

# Premier programme

Pour exécuter un programme JavaScript avec NodeJS, il vous faut :

- Un programme :

```
// index.js
console.log("Hello World !");
```

- NodeJS installé sur le poste

La commande suivante permet alors d'exécuter le script index.js :

```
// Exécuter une seule fois le programme
node index.js

// Relancer le programme automatiquement si des changements sont détectés sur les fichiers JS
// (nécessite d'installer le paquet nodemon)
nodemon index.js
```

# NPM

Lorsqu'il est installé, NodeJS est accompagné du Node Package Manager (NPM). A la façon de PIP pour Python, NPM centralise les bibliothèques qui peuvent être utilisées dans les applications NodeJS.

```
# Connaître la version de NPM installée
npm -v

# Installer une dépendance nécessaire à l'exécution du programme en production
npm install [package-name]

# Installer une dépendance nécessaire uniquement en développement
npm install -D [package-name]

# Désinstaller une dépendance
npm remove [package-name]
```

Le fichier `package.json` contient, entre autres, la liste des dépendances nécessaires à un projet NodeJS.

# node_modules

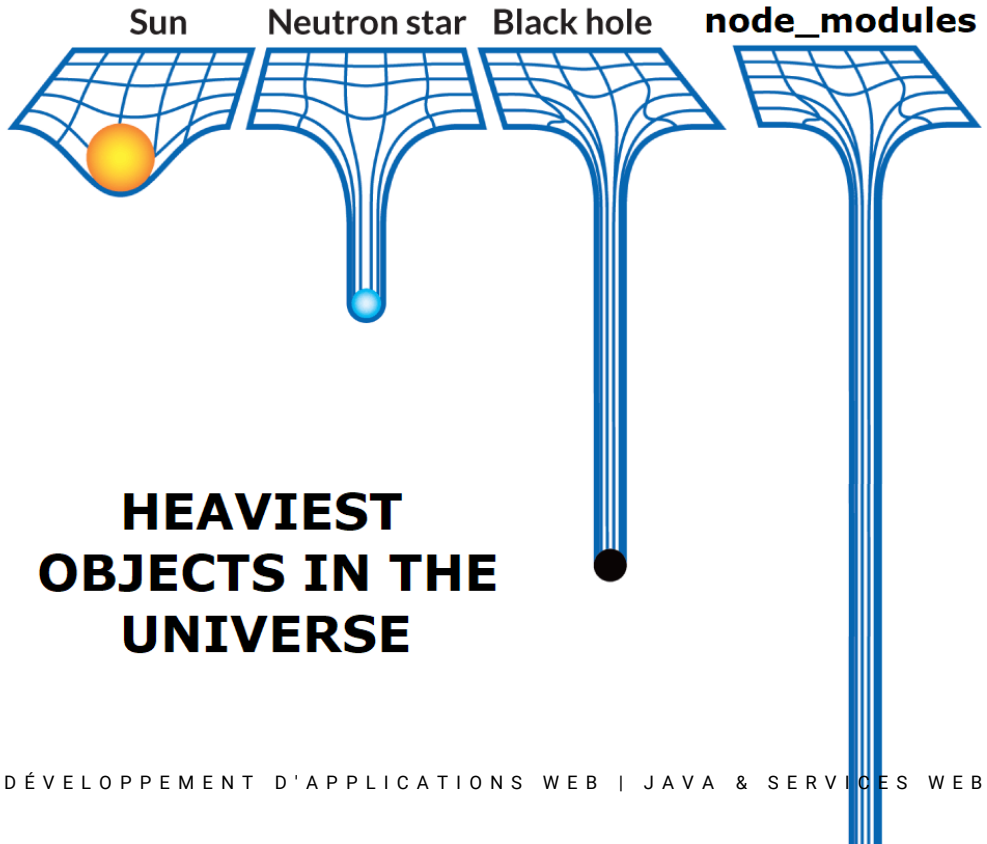
Les paquets NPM d'un projet sont installés dans le dossier `node_modules`.

Pour installer les dépendances requises par un projet :

```
npm i
```

Le dossier `node_modules` sera alors entièrement recréé.

**! Important !**
Le dossier `node_modules` ne doit jamais être remonté sur le dépôt du projet. Il doit toujours être ajouté au fichier `.gitignore`.



# TypeScript

## Introduction

# TypeScript ?

- sur-ensemble à JavaScript
- utilise le **typage statique**
- prend en charge les paradigmes **impératif**, **fonctionnel** et **orienté objet**
- **transpilé en JavaScript** pour être exécuté par un navigateur

# Typage statique

Le type d'une variable est définit à sa création et ne peut plus changer ensuite

```
let text: string
text = "Hello World !"      // OK
text = 5                    // Erreur
```

Le type d'une variable peut être déterminé par inférence de type

```
let entier = 5
entier = 42                 // OK
entier = "Test"             // Erreur
```

# Type personnalisé

Il est possible de créer ses propres types

```
type Grade = number | null

let grade: Grade = 17       // OK
grade = null                // OK
grade = "7"                 // Erreur
```

Il est possible de définir des types complexes

```
type Person = {
    firstname: string,
    lastname: string
}
```

```
const john: Person = {
    firstname: "John",
    lastname: "Doe"
}                   // OK
```

```
const donald: Person = {
    lastname: "Trump"
}                   // Erreur
```

Il est possible de combiner des types

```
type User = Person & {
    login: string,
    password: string
}
```

# Classes

```
class Coordinates {
    private x: number
    private y: number

    constructor(x: number = 0, y: number = 0)
    {
        this.x = x
        this.y = y
    }

    move(vector: Coordinates): void
    {
        this.x += vector.x
        this.y += vector.y
    }
}

const origin = new Coordinates()
const point = new Coordinates(134, 47)
```

Les modificateurs d'accès `public`, `private` et `protected` sont pris en charge

Le type des paramètres des fonctions et méthodes doivent être indiqués

Les paramètres des fonctions peuvent prendre une valeur par défaut

Le type de retour d'une fonction ou d'une méthode doit être précisé

# Classes abstraites

```
class abstract Shape {
    private color: Color
    private position: Position

    constructor()
    {
        this.color = {r: 0, g: 0, b: 0}
        this.position = {x: 0, y: 0}
    }

    draw(): void
}

const shape = new Shape()                    // Erreur
```

Une classe doit être définie `abstract` si au moins l'une de ces méthodes n'est pas définie (pas de code)

Une classe abstraite ne peut pas être instanciée

```
class Square extends Shape {
    constructor() {
        super()
    }

    draw(): void
    {
        // code of the function
    }
}

const square = new Square()                  // OK
```

Une classe ne peut hériter que d'une seule classe mère (**pas d'héritage multiple**)

Seule un classe fille qui redéfinit toutes les méthodes abstraites de la classe mère peut être instanciée.

# Interface

Une interface est une classe abstraite qui **ne contient que des méthodes abstraites** et qui **ne possède pas d'attribut**.

Elle représente un contrat qui garantit que toutes classe qui implémentera l'interface proposera bien l'ensemble des méthodes décrites dans l'interface.

```
interface Drawable {
    draw(): void
}

class Square implements Serializable {
    draw(): void {
        // code of the function
    }
}
```

Une classe peut implémenter **plusieurs** interfaces.

# Transpiler un projet TypeScript

TypeScript doit être installé comme dépendance de développement de votre projet :

```
npm install -D typescript
```

Puis, il est nécessaire d'initialiser TypeScript :

```
npx tsc --init
```

Enfin, il est possible de transpiler le projet :

```
npx tsc
```

L'option `Watch` permet transpiler les fichiers TypeScript à chaque enregistrement :

```
npx tsc -w
```