

NodeJS

Yoni FIRROLONI



CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**



Qu'est ce que c'est ?

- Runtime asynchrone pour le JS
- Permet la programmation concurrentielle
- Partage le même langage que le navigateur
- Aussi intéressant en backend qu'en frontend.
- Langage qui fonctionne par référence
- Défini par les normes ECMAScript : <https://tc39.es/ecma262/>
- Rien a voir avec Java






Une technologie web dynamique

- Permet de faire des sites dynamiques bien plus simplement
- Écosystème très riche avec NPM et une communauté active
- Plus rapide à mettre en oeuvre, plus “agile” que ses contreparties statiques
- Facile à scaler
- Asynchrone
- Adapté aux bases de données graph et document
- Permet aussi de faire des clients lourds et des applications Android/IOS
- Sérialisation et désérialisation JSON natives

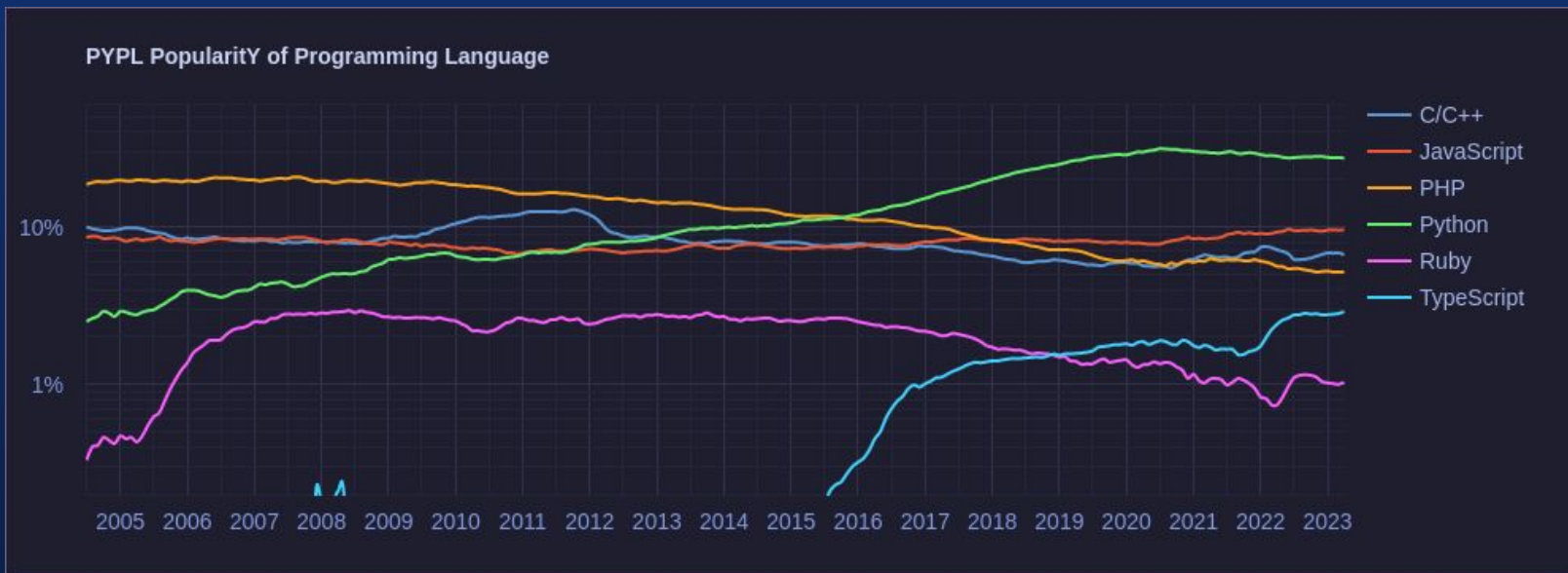




Quelque détails techniques

- JS est un langage interprété
 - Typage dynamique (attribué au runtime)
 - Plusieurs implémentations différentes(V8, JSCore, SpiderMonkey...)
 - A été grandement étendu via des compilateurs JIT : (Babel, TS...)
- 

Une technologie qui a le vent en poupe



Environnement Nodejs

- NPM (Node Package Manager)/yarn
- Express.js
- Typescript
- ESLint
- Webpack



Package*.json

Package.json

Permet de noter les dépendances et leurs contraintes de versions, les scripts de lancement, et autres.

Package-lock.json

Générer automatiquement (ne pas modifier !)

Permet de garder la version exacte de chaque dépendance

```
{
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "eject": "expo eject",
    "test": "jest"
  },
  "dependencies": {
    "expo": "^35.0.0",
    "jest": "^24.9.0",
    "react": "16.8.3",
    "react-dom": "16.8.3",
    "react-native": "https://github.com/expo/react-native/archive/s",
    "react-native-web": "^0.11.7"
  },
  "devDependencies": {
    "babel-preset-expo": "^7.0.0"
  },
  "private": true
}
```

Architecture JS courante



1. PGSQL / MongoDB



1. REST API
2. *NestJS, Express.js*



1. Frontend
2. *React,*
3. *Vue,*
4. *Angular*

Installation de NodeJS

Mac, WSL et linux :

nvm : <https://github.com/nvm-sh/nvm>

NVM permet de gérer de multiples installations nodeJS sur une même machine en évitant tout conflit.

```
nvm install --lts
```

```
nvm use --lts
```

Windows :

<https://docs.npmjs.com/downloading-and-installing-node-js-and-npm>

<https://nodejs.org/en/download/>

Installez la LTS !

Node Package Manager

package.json

- license
- version
- description
- point d'entrée
- auteur
- contributeurs
- license
- **scripts**
- packages
- devPackages
- Repository

package-lock.json

- généré automatiquement
- Garde un historique des versions des dépendances
- versionné
- Ne pas modifier

<https://docs.npmjs.com/cli/v7/configuring-npm/package-json>


Les bases



Créer un projet

\$ npm init

Par défaut, le fichier point d'entrée est index.js



```
> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.


See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (playground)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/vagahbond/courses/cours-nodejs/playground/package.json:

{
  "name": "playground",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes) yes
```





Créer un script dans package.json

```
{  
  "name": "playground",  
  "version": "1.0.0",  
  "description": "Un projet qui déchire",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1",  
    "start": "node index.js" // `npm run start` permet de lancer le programme  
  },  
  "author": "quelqu'un",  
  "license": "ISC"  
}
```





Hello world

On print dans la console avec :

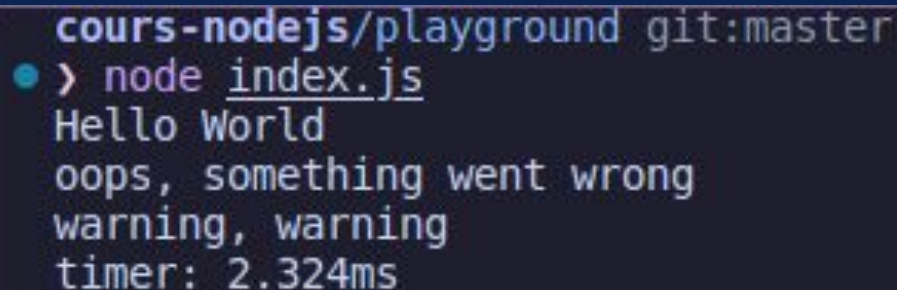
```
console.time("timer"); // This starts a timer
```

```
console.log("Hello World"); // This is a STDOUT log
```

```
console.error("oops, something went wrong"); // This is a STDERR  
log
```

```
console.warn("warning, warning"); // This is a STDERR log
```

```
console.timeEnd("timer"); // This ends the timer
```



```
cours-nodejs/playground git:master  
• > node index.js  
Hello World  
oops, something went wrong  
warning, warning  
timer: 2.324ms
```





Déclarer une variable: var

```
var x = 1;
```

```
if (x === 1) {  
  var x = 2;
```

```
  console.log(x);  
  // Expected output: 2  
}
```

```
console.log(x);  
// Expected output: 2
```

Var Déclare une variable qui est disponible soit dans sa fonction, soit de façon globale, ce qui fait qu'on peut lui affecter une valeur dans un scope enfant en le déclarant à nouveau.

Je déconseille vivement son utilisation.





Déclarer une variable: const

```
const x = 1;

if (x === 1) {
  const x = 2;

  console.log(x); // Expected output: 2
}

console.log(x); // Expected output: 1

x = 5; // TypeError: Assignment to constant variable.
```

Const permet de déclarer une constante. Elle appartient à un bloc, et ne peut pas être réaffectée.





Déclarer une variable: let

```
let x = 1;
```

```
if (x === 1) {  
  let x = 2;
```

```
  console.log(x); // Expected output: 2  
}
```

```
console.log(x); // Expected output: 1
```

```
x = 5;
```

```
console.log(x); // Expected output: 5
```

let permet de créer une variable qui n'est pas immuable. Elle appartient à son bloc, comme const.





Condition: if

```
// if, else if, else  
if (condition) {  
    // do something  
} else if (condition) {  
    // do something else  
} else {  
    // do something else  
}
```





Condition: switch case

// switch case example

switch (expression) {

case value1:

// code block

break;

case value2:

// code block

break;

default:

// code block

}





Condition: ternaire

//this is how you can make a ternary condition in javascript

```
const age = 18;
```

```
const drink = age >= 18 ? 'wine' : 'water';
```





Condition: Opérateurs logique

`&&` - AND operator - les deux côtés doivent être vrais

`||` - OR operator - un seul côté doit être vrai

`!` - NOT operator - Inverse la valeur accolée

`===` - strict equality operator - Les deux côtés doivent partager leur type et leur valeur

`==` - loose equality operator - Les deux côtés doivent avoir la même valeur

`!==` - strict inequality operator - inverse de `===`

`!=` - loose inequality operator - inverse de `==`

`<` - less than operator - inférieur à

`>` - greater than operator - supérieur à

`<=` - less than or equal to operator - left side needs to be less than or equal to right side

`>=` - greater than or equal to operator - left side needs to be greater than or equal to right side





Boucle: for

```
for (var i = 0; i < 10; i++) {  
    console.log(i);  
}
```





Boucle: while

```
let i = 0;  
  
while (i < 10) {  
    console.log(i);  
    i++;  
}
```





Fonction: normale

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

```
console.log(add(1, 2)); // 3
```





Fonction: lambda

// Il est possible de faire des fonctions anonymes et de les stocker dans des variables

```
const add = (a, b) => a + b;
```

```
console.log(add(1, 2));
```





Types: Array

// Créer un array

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

// Ajouter un élément à la fin

```
numbers.push(10);
```

// Ajouter un élément au début

```
numbers.unshift(0);
```

// Retirer un élément à la fin

```
numbers.pop();
```

// Retirer un élément au début

```
numbers.shift();
```

// Trouver l'index d'un élément

```
numbers.indexOf(5);
```

// Retirer le 6ème élément

```
numbers.splice(5, 1);
```

// Inverser

```
numbers.reverse();
```

// Obtenir la longueur d'un array

```
array.length;
```

// Transformer un array en string

```
numbers.join();
```

// trier l'array

```
numbers.sort();
```





Types: Array

// forEach parcourt la liste

```
tab.forEach((item) => {  
  console.log(item);  
});
```

// map retourne un nouveau tableau dérivé du premier

```
const tab2 = tab.map((item) => {  
  return item * 2;  
}  
);
```

// filter retourne un tableau sans les éléments qui ne répondent pas au prédicat

```
const tab3 = tab.filter((item) => {  
  return item > 5;  
}  
);
```



// find retourne le premier élément qui répond au prédicat, ou alors `null`

```
const value = tab.find((item) => {  
  return item > 5;  
}
```

```
);
```

// every indique si tous les éléments répondent au prédicat

```
const isValid = tab.every((item) => {  
  return item > 5;  
}
```

```
);
```

// some Indique si un des éléments répond au prédicat

```
const hasValue = tab.some((item) => {  
  return item > 5;  
}
```

```
);
```

// reduce Exécute la fonction sur tous les éléments pour n'en renvoyer qu'un.

```
const sum = tab.reduce((total, item) => {  
  return total + item;  
}
```

```
);
```



Types:Objects

```
// créer un objet
var obj = {
  name: 'John',
  age: 30,
  city: 'New York'
};
// Accéder à ses propriétés
console.log(obj.name); // John
// Ajouter une nouvelle propriété
obj.country = 'USA';
// Supprimer une propriété
delete obj.city;
// Vérifier qu'une propriété existe
console.log('age' in obj); // true
```

```
// Ajouter une méthode à un objet
var obj2 = {
  name: 'John',
  age: 30,
  city: 'New York',
  print: function() {
    console.log(this.name + ' is ' + this.age + '
years old');
  }
};
// appeler la méthode de l'objet
obj2.print(); // John is 30 years old

// itérer au travers des clés-valeurs
for (var key in obj2) {
  console.log(key + ': ' + obj2[key]);
}

// Autre façon d'itérer au travers des clé-valeurs
Object.keys(obj2).forEach(function(key) {
  console.log(key + ': ' + obj2[key]);
});
```



Types:Strings

// single quotes

```
var singleQuote = 'single quote';
```

// double quotes

```
var doubleQuote = "double quote";
```

// Interpolation avec le backtick

```
var interpolation = `interpolation ${singleQuote}`;
```

// obtenir la longueur de la string

```
var length = interpolation.length;
```

// Mettre en majuscule (il y a aussi toLowerCase())

```
var upperCase = interpolation.toUpperCase();
```

*// Séparer en plusieurs string en fonction
du caractère en paramètre*

```
var split = interpolation.split(' ');
```

// Remplacer du contenu dans la chaîne, avec REGEX

```
var replace = interpolation.replace('interpolation',  
'replace');
```

// Savoir si la string en contient une autre

```
var includes =  
interpolation.includes('interpolation');
```

// Obtenir une liste des correspondances au REGEX

```
var match = interpolation.match(/interpolation/);
```

*// Trouver la première substring qui correspond au
REGEX*

```
var search = interpolation.search(/interpolation/);
```

// Concatène

```
var concat = interpolation.concat(' concat');
```



Types:Classes

```
class Person {  
  
    constructor(name, age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`Hello, my name is ${this.name} and  
I am ${this.age}`);  
    }  
}  
  
const person1 = new Person('John', 33);  
const person2 = new Person('Sara', 28);  
  
console.log(person1.name, person2.name);
```

```
class Customer extends Person {  
    constructor(name, age, balance) {  
        super(name, age);  
        this.balance = balance;  
    }  
  
    #personalInfo() { // Privée  
        return `${this.name} owes ${this.balance}.00`;  
    }  
  
    info() { // publique  
        return `${this.name} owes ${this.balance}.00`;  
    }  
}  
  
const customer1 = new Customer('Kevin', 32, 300);
```





Imports/exports

// Exporter une déclaration principale

```
export default {  
  name: 'John Doe',  
  age: 30  
}
```

// Importer une déclaration "default"

```
import person from './person.js'
```

// exporter une déclaration

```
export const person = {  
  name: 'John Doe',  
  age: 30  
}
```

// Importer une déclaration secondaire

```
import { person } from './person.js'
```

