Mbengue.M

# TYPESCRIPT

# 2 PLAN

1. Introduction
2. Les types TypeScript
3. TypeScript et ES6
4. Classes et Objets

# 3 PLAN

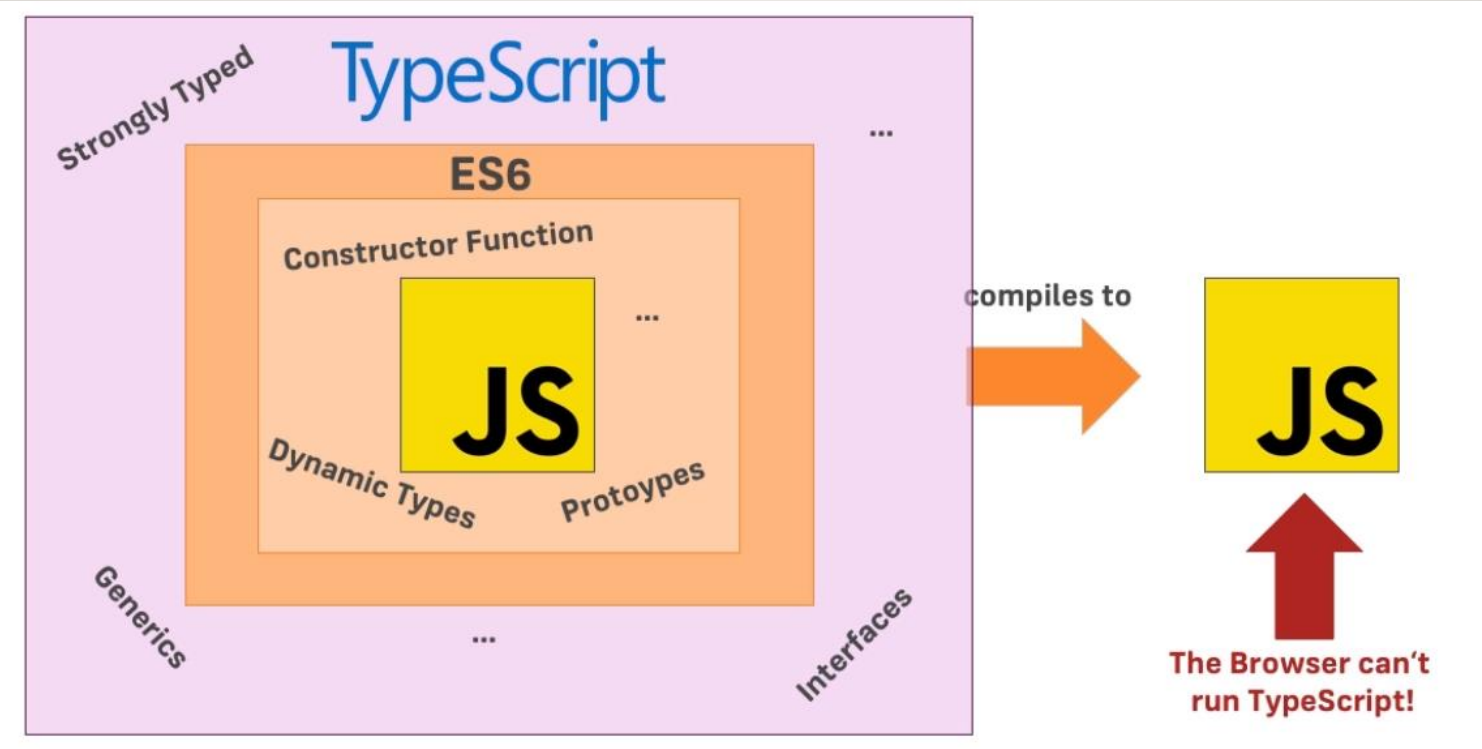
1. Introduction
2. Les types TypeScript
3. TypeScript et ES6
4. Classes et Objets

# 4 INTRODUCTION

- Javascript à typage fort
- Orienté objet
- Forte ressemblance à C# et JAVA
- Maintenance plus simple dans un projet web
- Complète les points faibles de Javascript

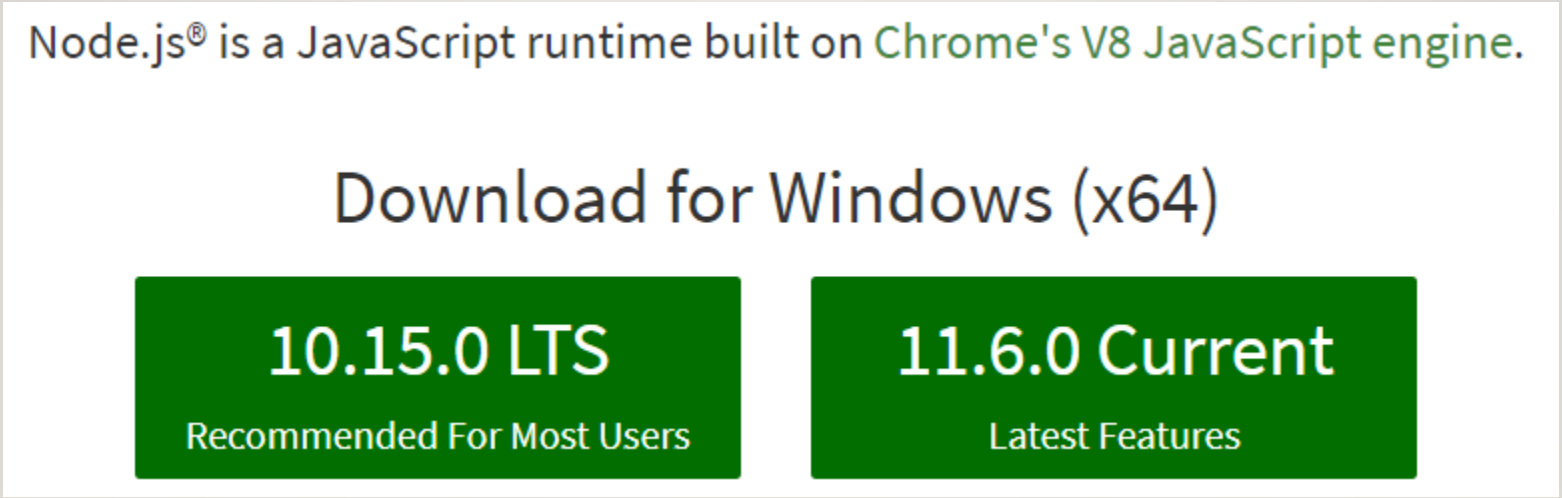
# 5 INTRODUCTION

- Contrairement à Javascript, il ne tourne pas via le navigateur



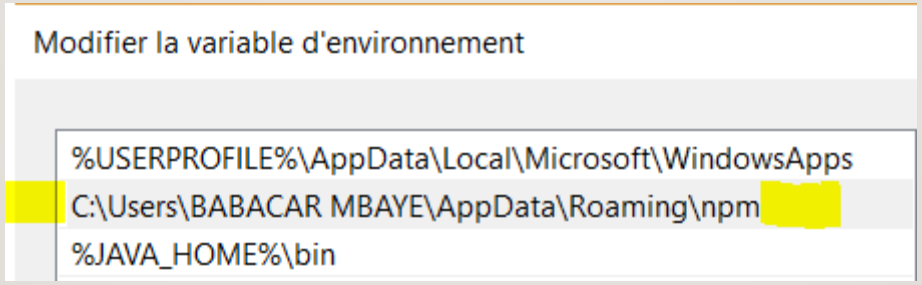
# 6 INTRODUCTION

- Installation



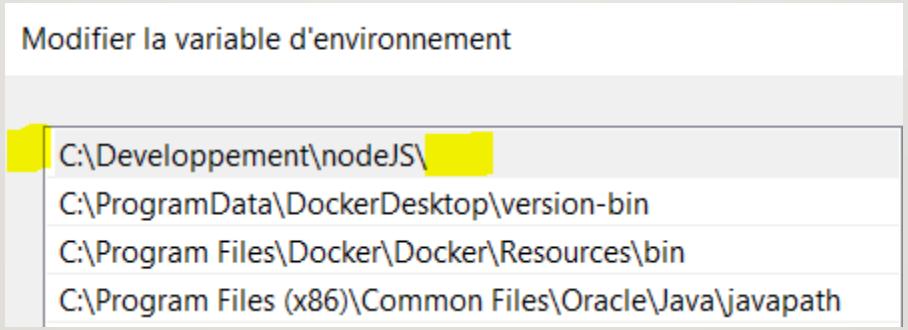
# 7 INTRODUCTION

- Installation



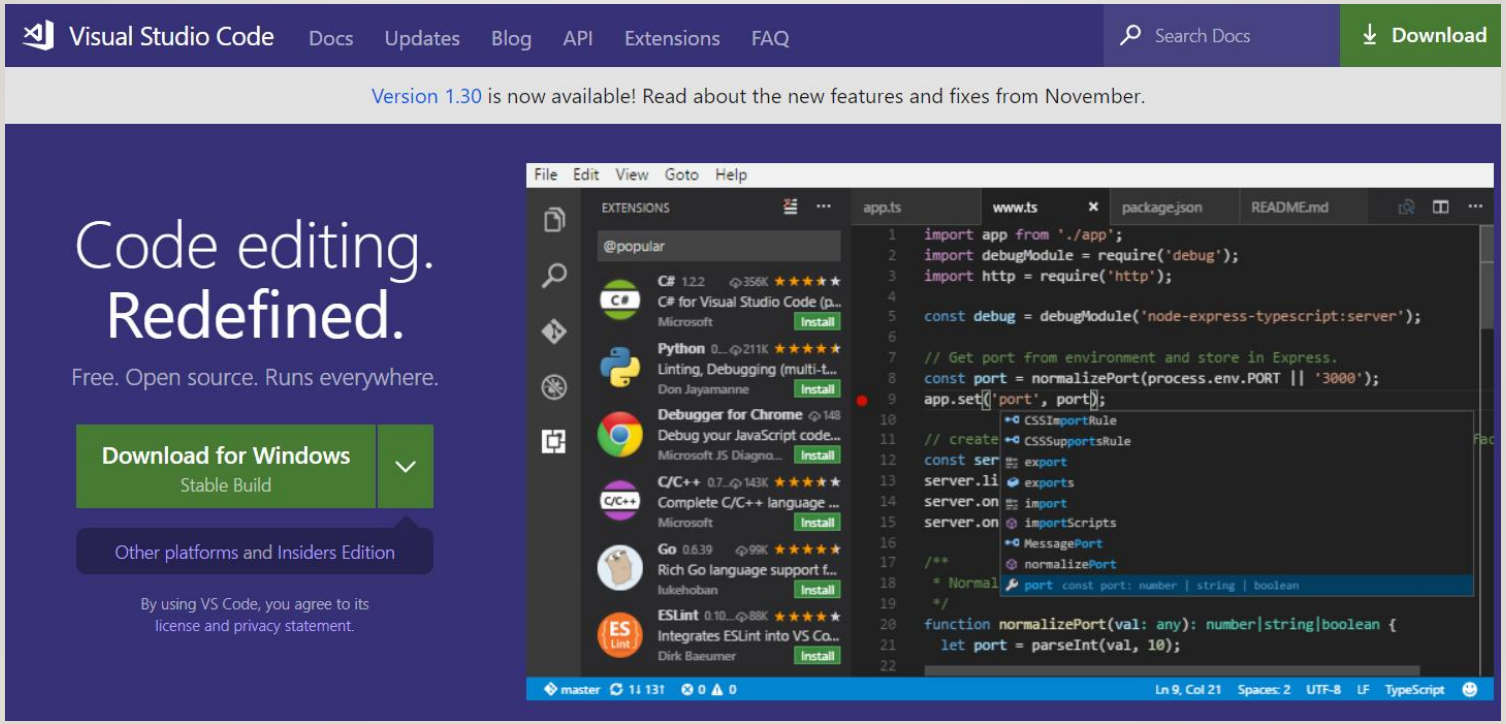
# 8 INTRODUCTION

- Installation



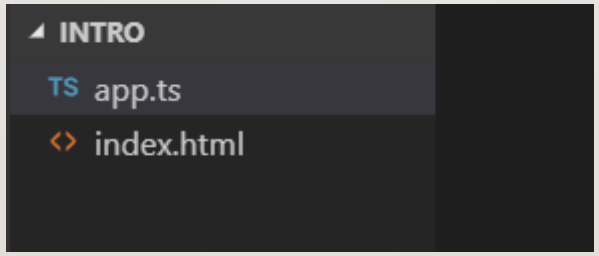
# 9 INTRODUCTION

- IDE et stratégie de développement



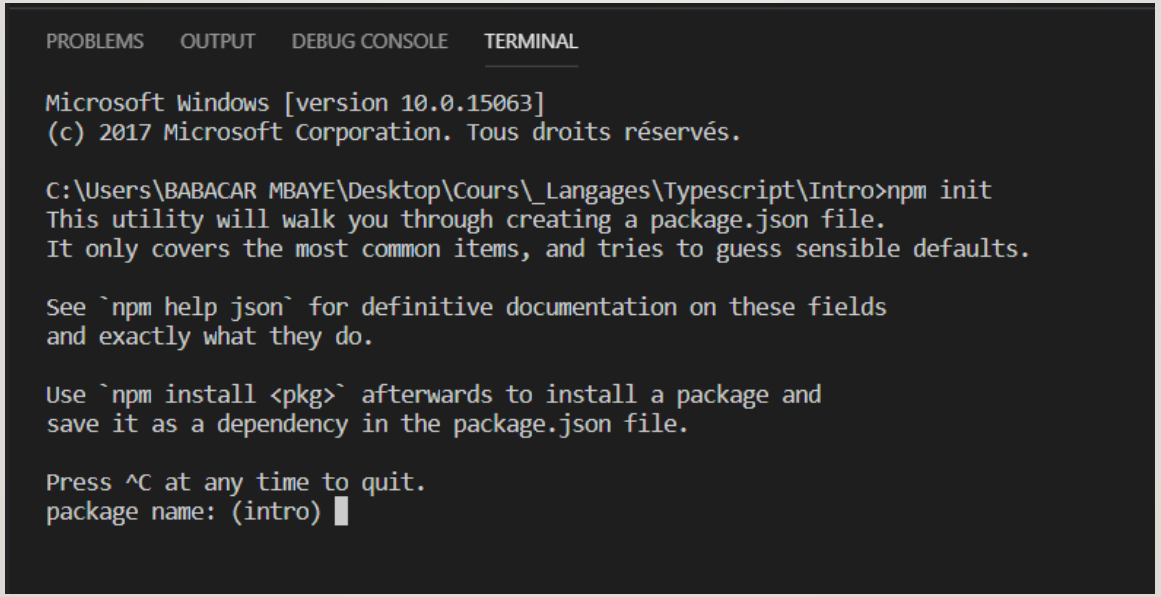
# 10 INTRODUCTION

- Créer un répertoire « intro » contenant 2 fichiers
  - index.html
  - app.ts



# 11 INTRODUCTION

- Run « ***npm init*** » dans le dossier pour la gestion sous le package npm



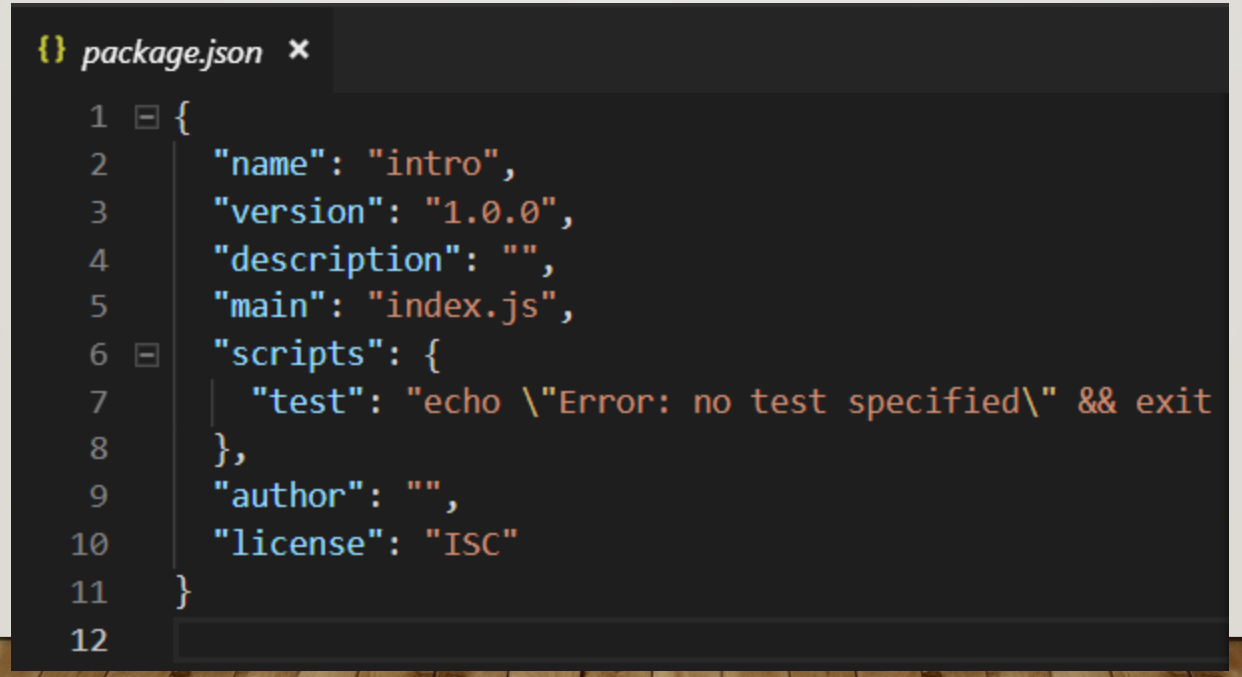
# 12 INTRODUCTION

- Appuyer sur « ***Entrée*** » en validant toutes les demandes par défaut

```
About to write to C:\Users\BABACAR MBAYE\Desktop\Cours\_Langages\Typescript\Intro\package.json:

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


Is this OK? (yes)

C:\Users\BABACAR MBAYE\Desktop\Cours\_Langages\Typescript\Intro>
```

# 13 INTRODUCTION

- Création d'un nouveau fichier : **package.json**
- **Encadré :** Toutes les dépendances utilisées dans le projet



# 14 INTRODUCTION

- Installer le serveur : **lite-server** (serveur très léger qui va hébergé le fichier html)
- Run ***« npm install lite-server --save-dev »***

```
C:\Users\BABACAR MBAYE\Desktop\Cours\_Langages\Typescript\Intro>
C:\Users\BABACAR MBAYE\Desktop\Cours\_Langages\Typescript\Intro>npm install lite-server --save-dev
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN intro@1.0.0 No description
npm WARN intro@1.0.0 No repository field.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin",

+ lite-server@2.4.0
added 347 packages from 274 contributors and audited 2607 packages in 67.251s
found 0 vulnerabilities


C:\Users\BABACAR MBAYE\Desktop\Cours\_Langages\Typescript\Intro>
```

# 15 INTRODUCTION

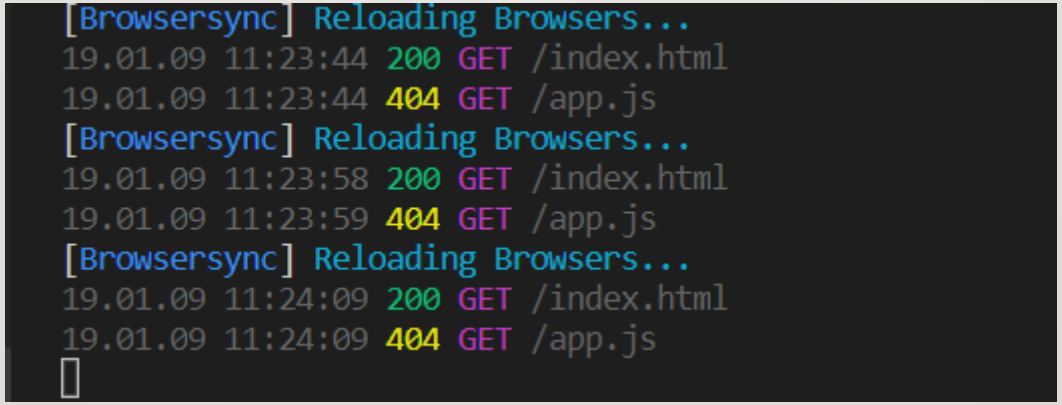
- On voit une nouvelle entrée contenant la version du serveur dans les ***dépendances dev***
- Ajouter dans la rubrique script : un démarrage du serveur via **« start : lite-server »**

# 16 INTRODUCTION

- Run **« npm start »** et votre navigateur s'ouvre !
- N'hésitez pas à faire un save dans votre IDE et/ou le redémarrer en cas d'erreur

# 17 INTRODUCTION

- Les nouvelles modifications du fichier **index.html** seront automatiquement pris en compte lors de la sauvegarde. Faire un essai en ajoutant des infos dans le body



# 18 INTRODUCTION

- Pour la compilation de **app.ts** en **app.js** je pourrai faire un run de « ***tsc app.ts*** »
- Cependant, je peux aussi faire traquer ce dossier par TypeScript, ce qui me permettrai de seulement faire un run de « ***tsc*** » afin que tout les fichiers avec pour extension « **.ts** » se compile en « **.js** »
- Pour cela … (next)

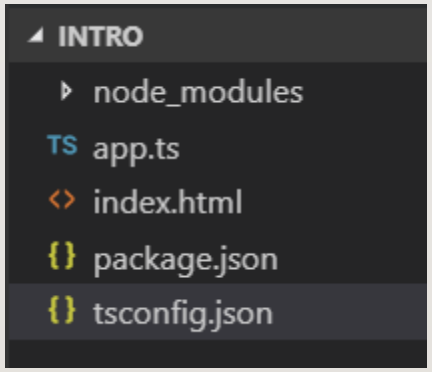
# 19 INTRODUCTION

- Run « **npm install –g typescript** » puis
- Run « **tsc --init** » ***sur un nouvel onglet de terminal*** ( laisser le serveur tranquille ☺ ) pour l'initialiser en tant que projet TypeScript.
- Un nouveau fichier nommé : ***tsconfig.json*** est crée. Nous y reviendrons plus tard !
- Ce dossier est maintenant sous le contrôle de **npm** mais aussi de **TypeScript**

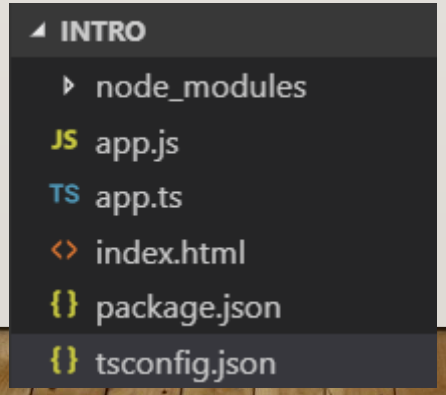
```
C:\Users\BABACAR MBAYE\Desktop\Cours\_Langages\Typescript\Intro>tsc --init
message TS6071: Successfully created a tsconfig.json file.
```

# 20 INTRODUCTION

- Nous avons l'arborescence :

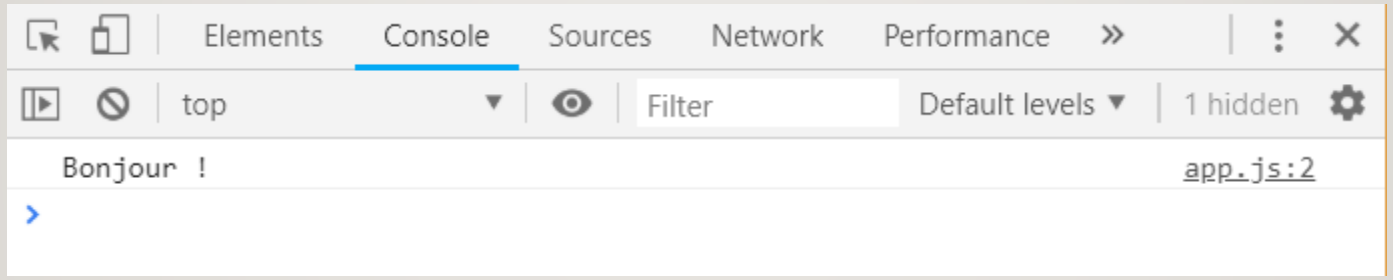



- Run « ***tsc*** » toujours sur le second terminal ensuite observer !



# 21 INTRODUCTION

- Maintenant, faire un refresh du navigateur et voir l'onglet console :



# 22 PLAN

1. Introduction
2. Les types TypeScript
3. TypeScript et ES6
4. Classes et Objets

# 23 LES TYPES TYPESCRIPT

```
1   console.log("Bonjour !");
2   let myName = "pierre";
3
4   //erreur tsc si réaffectation sur un autre type
5   myName = 10;
6   console.log(myName);
7
8   let myName_1;
9   myName_1 = "paul";
10
11  // redéfinition possible
12  myName_1 = "david";
13  console.log(myName_1);
```

# 24 LES TYPES TYPESCRIPT

```
3   // number
4   let myAge = 27;
5
6   // TYPE number peut prendre double, float, etc...
7   myAge = 27.12;
8   myAge = 2003003300303000303003003030030033;
9   console.log(myAge);
10
11  // erreur tsc
12  myAge = 'Titi';
13
14  // boolean
15  let isClear = true;
16
17  // erreur tsc
18  isClear = 1;
```

# 25 LES TYPES TYPESCRIPT

```
1   // TYPES
2
3   // number
4   let myRealAge;
5   myRealAge = 27;
6
7   // tsc ok car le type n'a pas été défini lors de la
8   // déclaration
9   myRealAge = "vingt-sept";
10
11  // affiche 'vingt-sept'
12  console.log(myRealAge);
```

- Si on rajoute : any pour la variable myRealAge : OK car même effet que le vide déclaratif

```
4   let myRealAge : any;
```

# 26 LES TYPES TYPESCRIPT

- Array

```
1   // TYPES
2
3   // Tableau
4   let sport  = ["football", "basket-ball", "handball"];
5
6   // get indice 1
7   console.log(sport[1]);
8   // get type de tableau : Object (An array is an object)
9   console.log(typeof sport);
10
11  // Erreur tsc
12  // Le type ne peut plus etre changé une fois initialisé
13  sport = [1, 10, 1000];
```

# 27 LES TYPES TYPESCRIPT

- Que dois-je ajouter pour le dernier slide marche !

```
3   // Tableau
4   let sport: any[]  = ["football", "basket-ball", "handball"];
```

- Si je mets ceci : Erreur tsc : car il faut quand même un type : tableau

```
11  // Erreur tsc
12  // Le type ne peut plus etre changé une fois initialisé
13  sport = [1, 10, 1000];
14  sport = 20;
```

# 28 LES TYPES TYPESCRIPT

- Les tuples sont des tableaux ayant une limite de nombre d'éléments
- L'ordre et le type sont importants

```
1  // TYPES
2
3  // Tableau
4  let sport: [string, number, number, boolean] = ["abc", 99, 12, false];
5  console.log(sport);
```

# 29 LES TYPES TYPESCRIPT

- Enum :

```
3   // enum
4
5   enum Couleur {
6       gris,
7       bleu,
8       jaune,
9       vert
10  }
11
12  let maCouleur : Couleur = Couleur.jaune;
13  console.log(maCouleur);
```

- Fixer la valeur :

```
5   enum Couleur {
6       gris,
7       bleu = 100,
8       jaune,
9       vert
10  }
```

# 30 LES TYPES TYPESCRIPT

- Any :

```
3   // Any
4
5   let car  : any  = "Renault";
6   console.log(car);
7
8   car  = [307, 308];
9   console.log(car);
```

- A utiliser lorsque l'on ne sait pas encore quel type utiliser dans le programme

# 31 LES TYPES TYPESCRIPT

- Fichier js crée
- Tout le code TypeScript a été compilé pour donner du Javascript pure ou les types spécifiés disparaissent.

# 32 LES TYPES TYPESCRIPT

- Fonctions 1/2

```typescript
// TYPES

// Fonctions
let myName: string = 'Paul';

function direNom() : string {
    return myName;
}

console.log(direNom());

// fonctions sans arugments

function direBonjour() : void {
    console.log("Bonjour");
}

direBonjour();
```

# 33 LES TYPES TYPESCRIPT

- Fonctions 2/2

```
3   // Fonctions avec 2 arguments et retournant un type : number
4   function multiplier (valeur1, valeur2) : number {
5       return valeur1 * valeur2;
6   }
7   // Pas d'erreur tsc car type implicit : any
8   console.log(multiplier(2, "Max"));
9
10  // correction
11  function multiplierBis (valeur1 : number, valeur2 : number) : number {
12      return valeur1 * valeur2;
13  }
14  // Erreur tsc car type explicit non respecté
15  console.log(multiplierBis(2, "Max"));
16  console.log(multiplierBis(2, 5));  // tsc OK : resultat : 10
17
18
19
```

# 34 LES TYPES TYPESCRIPT

- Objets et types

```
4   // Objet et type
5
6   // pas de nom de variable différent, ordre non important,
7   // Le modele (moule)
8   let personne : {nom: string, prenom: string, age: number} = {
9
10      //L'objet réel (objet)
11      age: 27,
12      nom: "Petit",
13      prenom: "Pierre",
14
15  };
16
```

# 35 LES TYPES TYPESCRIPT

- Objets complexes

```
5
6   // complex est un objet avec 2 propriétés : data et output
7   // data  : tableau d'entiers
8   // output : fonction avec un argument boolean et retournant un tableau d'entiers
9   let complex : {data: number[], output: (isSomething:boolean) => number[]} = {
10
11      data: [100, 3.99, 10],
12      output : function (isSomething:boolean) : number[] {
13          return this.data;
14      }
15  };
```

# 36 LES TYPES TYPESCRIPT

- Objets complexes

```
6   // Nouveau ALIAS de type nommé  : Complex
7   type Complex = {data: number[], output: (isSomething:boolean) => number[]}
8
9   let complex : Complex = {
10
11      data: [100, 3.99, 10],
12      output : function (isSomething:boolean) : number[] {
13          return this.data;
14      }
15  };
16
17  // Dans le futur, si je dois modifier sur plusieurs objets impactés,
18  // Le changement ne se fera qu'a un endroit
```

# 37 LES TYPES TYPESCRIPT

- Types multiples

```
5   // Variable qui est de type nombre (ou) boolean uniquement
6   // Alternatif à l'utilisation du type libre : any
7
8   let maVariable : number | boolean  = 30
9   // Not good
10  maVariable = "27";
11  // Good
12  maVariable = true;
13
```

# 38 LES TYPES TYPESCRIPT

- Vérification de type

```
5   let valeurFinale = 100;
6   if(typeof valeurFinale == "number"){
7       console.log("Ceci est un type nombre")
8   }else {
9       console.log("Type inconnu !");
10  }
11
12
```

# 39 TP 1

- Ajouter les types pour être le plus explicit possible avec ce bout de code puis compiler pour voir le résultat

```
let bankAccount = {
    money: 2000,
    deposit(value) {
        this.money += value;
    }
};

let myself = {
    name: "Max",
    bankAccount: bankAccount,
    hobbies: ["Sports", "Cooking"]
};

myself.bankAccount.deposit(3000);

console.log(myself);
```

# 40 PLAN

1. Introduction
2. Les types TypeScript
3. TypeScript et ES6
4. Classes et Objets

# 41 TYPESCRIPT ET ES6

- Let & Const

```
5   // La valeur peut changer
6   let variable = "test";
7   console.log(variable);
8   variable = "unAutreTest";
9   console.log(variable);
10
11  // Une constante
12  const niveau = 10;
13  console.log(niveau);
14  niveau = 9;
15
```

# 42 TYPESCRIPT ET ES6

- Block scope (portée variable)

```
3   let variable = "test";
4
5   // Block scope
6   function reset (){
7       let variable = null;
8       console.log(variable);
9   }
10
11  reset();
12  console.log(variable);
```

# 43 TYPESCRIPT ET ES6

- Fonctions Arrow

```
3   // Fonctions traditionnelles
4   const ajouterNombre = function(number1 : number, number2 : number) : number{
5       return number1 + number2;
6   }
7   console.log(ajouterNombre(10, 4));
8
9   // Fonctions arrow
10  const multiplierNombre = (number1 : number, number2 : number) => number1 * number2
11  console.log(multiplierNombre(10,4));
12
```

# 44 TYPESCRIPT ET ES6

- Fonctions Arrow

```
3   // Fonctions sans arguments
4   let saluer = () => console.log("Bonjour");
5   saluer();
6
7   // Fonctions avec 1 argument : Les ( ) sont alors facultatives
8   let saluerAmis = (ami:string) => console.log(ami);
9   saluerAmis("Paul");
10
```

# 45 TYPESCRIPT ET ES6

- Fonctions et paramètres par défaut

```
3   // Fonctions sans arguments
4   const countNum = (start: number) : void => {
5       while (start > 0){
6           start --;
7       }
8       console.log("Done : ", start, ".");
9   };
10
11  // erreur tsc car cette fonction a besoin d'un argument
12  countNum();
13
14  // Ajoutons un paramètre par défaut
15  // remplacer la première ligne par :
16  const countNum = (start: number = 10) : void => {
```

# 46 TYPESCRIPT ET ES6

- Opérateur « spread »
- Permet de transformer le tableau d'entiers comme étant une liste de valeurs entiers pour ***Math.max() : pas besoin de faire une boucle***

```
2   // SPREAD
3   const numbers = [1, 10, -40, 100];
4
5   // liste d'entiers
6   console.log(Math.max(33, 6, -30, 40));
7   // Erreur tsc : ne prend pas de tableau
8   console.log(Math.max(numbers));
9   // Use spread operator pour le transformer en une liste d'entiers
10  console.log(Math.max(...numbers));
```

# 47 TYPESCRIPT ET ES6

- Opérateur « rest »

```
1
2   // On part de ca et on veut (plus bas)....
3   function construireTableau(arg1: number, arg2: number) {
4       return [arg1, arg2];
5   }
6   console.log(construireTableau(10,20));
7
8   // Rendre flexible les arguments d'une méthode passés en paramètres
9   function construireTableauBis(arg1: string, ...args: number[]) {
10      return arg1 + args;
11  }
12  console.log(construireTableauBis("Moi", 10 , 20, 30, 40, 50, 60));
13
14  // Dans le corps :spread
15  // Dans les arguments de méthodes : rest
16
```

# 48 TYPESCRIPT ET ES6

- Tableau déstructuré

```
2   // Traditionnel
3   const activités  = ["Cuisiner", "Sport", "Danser", "Marcher"];
4   const act1 = activités[0];
5   const act2 = activités[1];
6   const act3 = activités[2];
7   const act4 = activités[3];
8   console.log(act1, act2, act3, act4);
9
10  // Destructuring arrays
11  const activités1  = ["Cuisiner", "Sport", "Danser", "Marcher"];
12  const [activite1, activite2, activite3, activite4] = activités1;
13  console.log(activite1, activite2, activite3, activite4);
14
15  // Gain : factorisation
```

# 49 TYPESCRIPT ET ES6

- Objet déstructuré

```
2   // Traditionnel
3   const personne =
4       {nom : "Petit", prenom : "Pierre", adresse : "Paris", age : 25};
5   const nom = personne.nom;
6   const prenom = personne.prenom;
7   const adresse = personne.adresse;
8   const age = personne.age;
9   console.log(nom, prenom, adresse, age);
10
11  // Destructuring object
12  const etudiant =
13      {nom : "Petit", prenom : "Paul", adresse : "Paris", age : 28};
14  const {nom:monNom, prenom:monPrenom, adresse:monAdresse, age:monAge} = etudiant;
15  console.log(monNom, monPrenom, monAdresse, monAge);
16
17
18
```

# 50 PLAN

1. Introduction
2. Les types TypeScript
3. TypeScript et ES6
4. Classes et Objet

# 51 CLASSES ET OBJETS

- Classes et objets

```
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no,
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Apprendre le TypeScript</title>
    <script src="classes.js"></script>
/head>
```

# 52 CLASSES ET OBJETS

- Héritage

```
class Etudiant extends Person {
    name = "David";
}

// Erreur tsc : si je laisse a vide les arguments sachant
// que c'est le constructeur de la classe mère
const unEtudiant = new Etudiant("Paul", "Pl");
console.log(unEtudiant);
```

# 53 CLASSES ET OBJETS

- Constructeurs et héritage

```
class Prof extends Person {

    //J'ai crée mon propre constructeur qui s'aide aussi
    //du constructeur de la classe mère
    constructor(username: string){
        super("Jacques", username);
    }
}

const prof1 = new Prof("Pr");
console.log(prof1);
```

# 54 CLASSES ET OBJETS

- Héritage
- Comment faire pour ajouter à l'objet **prof1**, un âge de 40 ans ?
- propriété : type : **privé** : non accessible | propriété : âge : **protected** : accessible

```
class Prof extends Person {

    //J'ai crée mon propre constructeur qui s'aide aussi
    //du constructeur de la classe mère
    constructor(username: string){
        super("Jacques", username);
        this.age = 40;
    }
}

const prof1 = new Prof("Pr");
console.log(prof1);
```

# 55 CLASSES ET OBJETS

- Méthodes et propriétés **static**

- Exemple :

```
class Utilitaires {

    static PI: number = 3.14;
    static calculCirconfer(diametre : number ) : number {
        return this.PI*diametre;
    }
}

console.log( 2 * Utilitaires.PI);
console.log(Utilitaires.calculCirconfer(8));
```

# 56 CLASSES ET OBJETS

- Classes abstraites (cannot be instanciated)

- Exemple :

```
abstract class Projet {

    nomProjet : string = "Default";
    budget : number = 1000;

    abstract changeName(nom : string) : void;

}

class ProjetInformatique extends Projet {

    changeName(nom: string): void {
        this.nomProjet = nom;
    }
}

let projet1  = new ProjetInformatique();
console.log(projet1);
projet1.changeName("Projet Big King");
console.log(projet1);
```

# 57 CLASSES ET OBJETS

- Constructeurs privés et singleton
- Exemple :

```
class  OnlyOne {
    private static instance :  OnlyOne;
    private constructor (public name : string) {

    }

    static getInstance(){
        if(!OnlyOne.instance){
            OnlyOne.instance = new OnlyOne("The only one");
        }
        return OnlyOne.instance;
    }
}

// erreur
//let mauvaisinstance = new OnlyOne("Only one");
let boninstance = OnlyOne.getInstance();
console.log(boninstance);
```