ES6 - TypeScript Cours

Naby Daouda Diakite

Présentation des participants

- Formateur
- Stagiaires
- Déroulement de la formation (*combinaison des parties théoriques et pratiques*)
- Echange sur des aspects transverses de nos métiers
- Partage des expériences

Programme

I. ECMAScript (ES6)
II. TypeScript et JavaScript
III. Mise en place de l'environnement
IV. Syntaxes de base
V. Types
VI. Variables
VII. Operateurs
VIII. Décisions et Boucles
IX. Nombres
X. Chaînes de caractère
XI. Collections
XII. Fonctions
XIII. Interfaces
XIV. Classes
XV. Généricité
XVI. Namespace
XVII. Modules
XVIII. Ambient

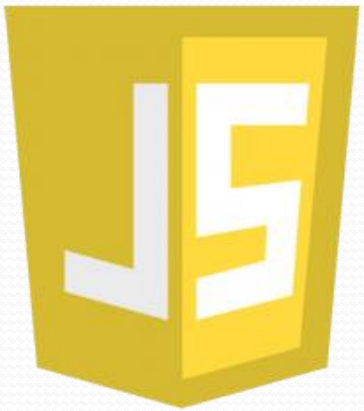
I. ECMAScript (ES6)

- I.I. JavaScript
- I.II. Spécifications JavaScript : ECMAScript 6
- I.III. Evolutions JavaScript

I. ECMAScript (ES6)

I.I. JavaScript

- JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives mais aussi pour les serveurs avec l'utilisation de Node.js



I. ECMAScript (ES6)

I.II. Spécifications JavaScript : ECMAScript 6 (1)

- JavaScript est standardisé par Ecma International — une association européenne de standardisation des systèmes d'information et de communication

- ECMA étant historiquement un acronyme pour *European Computer Manufacturers Association*, qui délivre un langage de programmation standardisé, international appelé **ECMAScript**.

I. ECMAScript (ES6)

I.II. Spécifications JavaScript : ECMAScript 6 (2)

- Le standard ECMA-262 est également approuvé par l'ISO (*International Organization for Standardization*) sous ISO-16262. La spécification peut également être trouvée sur le site web d'Ecma International.

- La spécification ECMAScript ne décrit pas le *Document Object Model* (DOM) qui est standardisé par le World Wide Web Consortium (W3C) et le WHATWG (Web Hypertext Application Technology Working Group).

- Le DOM définit la façon dont les documents HTML sont exposés aux scripts.

I. ECMAScript (ES6)

I.II. Spécifications JavaScript : ECMAScript 6

- ECMAScript 2015 (sixième édition) est aussi appelé **ES6 ou ECMAScript 6**

- Les versions actuelles des principaux navigateurs web supportent ECMAScript 5.1 et ECMAScript 2015 (aussi appelé ES6) mais certaines anciennes versions n'implémentent que ECMAScript 5.

- La sixième édition majeure d'ECMAScript a été officiellement approuvée et publiée en tant que standard le 17 juin 2015 par l'assemblée générale ECMA.
  - ***Depuis cette édition, les éditions ECMAScript sont publiées à un rythme annuel.***

I. ECMAScript (ES6)

I.III. Evolutions JavaScript (1)

- Les dernières spécifications sont :
  - ES7 (Juin 2016)
    - Array includes
    - Opérateur exponentiel
  - ES8 (Juin 2017)
    - String padding
    - Fonctions aysnchrones

I. ECMAScript (ES6)

I.III. Evolutions JavaScript (2)

- Exemple nouveauté ES7 : **Array includes**

```
const monTableau = [1,2,3,4,5];

if(monTableau.indexOf(3) !== -1 ) {

    //3 est dans le tableau

}
```

```
const monTableau = [1,2,3,4,5];

if(monTableau.includes(3)) {

//3 est dans le tableau

}
```

I. ECMAScript (ES6)

I.III. Evolutions JavaScript (3)

- Exemple nouveauté ES8 : **String padding**

Une sorte de formatage de chaînes de caractères au sein du script.

En utilisant les méthodes **padStart**() ou **padEnd**(), on peut ajouter des espaces ou caractères spécifiques, selon les paramètres, au début ou à la fin d'une chaîne de caractères.

```
'machaine'.padStart(20);
// "        machaine"


'machaine'.padEnd(20, '*');
// "machaine********"
```

II. TypeScript et JavaScript

- II.I. TypeScript
- II.II. Lien entre TypeScript et JavaScript
- II.III. Exemple de différences

II. TypeScript et JavaScript

II.I. TypeScript (1)

- **TypeScript** est un langage de programmation libre et open source développé par Microsoft (Octobre 2012) qui a pour but d'améliorer et de sécuriser la production de code JavaScript.

- Il a été co-créé par Anders Hejlsberg, principal inventeur du langage C#

II. TypeScript et JavaScript

II.I. TypeScript (2)

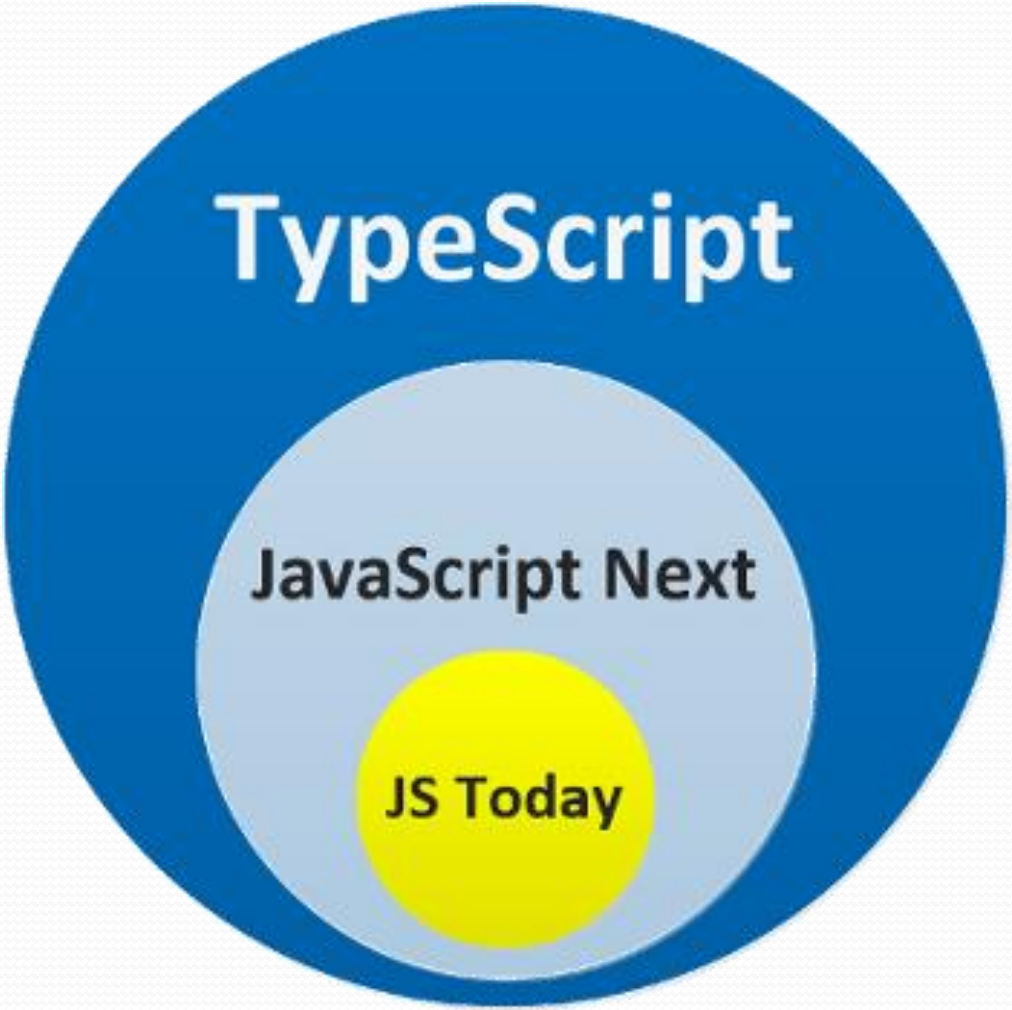
- TypeScript permet un typage statique optionnel des variables et des fonctions, la création de classes et d'interfaces, l'import de modules, tout en conservant l'approche non-contraignante de JavaScript.

- **Il supporte la spécification ECMAScript 6.**

- ***AngularJS*** *: framework JS très populaire est écrit en TypeScript*

II. TypeScript et JavaScript

II.II. Lien entre TypeScript et JavaScript (1)



II. TypeScript et JavaScript

II.II. Lien entre TypeScript et JavaScript (2)

- C'est un **sur-ensemble de JavaScript**
  - *Tout code JavaScript correct peut être utilisé avec TypeScript*

- Le code TypeScript est compilé en JavaScript, pouvant ainsi être interprété par n'importe quel navigateur web ou moteur JavaScript.

II. TypeScript et JavaScript

II.III. Exemple de différences

- **Typage fort**

```
function add(num1 ,num2) {
        return num1 + num2
}

add(1,2) // 3
add(1 , “Hello”) // 1Hello ➔ ne devrait pas fonctionner
```

```
function add(num1: number
,num2:number) {
        //num1 & num2 should be only and only of type "number"
        return num1 + num2
}
```

- ***Plein d’autres nouveautés à voir dans la suite du cours ..***

III. Mise en place de l'environnement

- III.I. Environnement local
- III.II. Node
- III.III. Environnement de développement (IDE)
- III.IV. Visual Studio Code

III. Mise en place de l'environnement

III.I. Environnement local

- TypeScript est une solution open source. Il peut être lancé sur tous les navigateurs et systèmes d'exploitation.

- Editeur de texte
  - Permet d'écrire le code TypeScript
  - Exemple : Notepad++ / vim / vi / Emacs..

- Compilateur TypeScript `Pour être plus précis, on utilise le terme transpilation`
  - Transforme le code TypeScript (fichier .ts ou .d.ts) en code JavaScript (fichier .js)

III. Mise en place de l'environnement

III.II. node

- Node : est un environnement d'exécution côté serveur pour JavaScript
- Il utilise le moteur JavaScript Google v8
- Lien de téléchargement : http://nodejs.org
  - *Télécharger la version 8.9.3 LTS* Si possible toujours travailler avec du LTS
- Il permet d'installer TypeScript via la commande :
  - npm install -g typescript
  - tsc –v ➔ // Version 2.6.2

III. Mise en place de l'environnement

III.III. Environnement de développement (IDE)

- Editeur de vode (IDE)
  - Permet d'écrire le code TypeScript (fonctionnalités facilitant l'écriture de code, le lancement, le déploiement, etc..)
  - Exemple : Visual Studio Code / Intellij IDEA / Eclipse ..

III. Mise en place de l'environnement

III.IV. Visual Studio Code

- IDE Open Source de Miscrosoft
  - Lien de téléchargement : https://code.visualstudio.com/

IV. Syntaxes de base

- IV.I. Compilation et Exécution
- IV.II. Identifiants
- IV.III. Mots clés
- IV.IV. Commentaires
- IV.V. TypeScript et Programmation Orientée Objet (POO)

IV. Syntaxes de base

IV.I. Compilation et Exécution (1)

- Exemples de code source (.ts) et complié (.js)

```
// Fichier compilation.ts

var message:string="Hello World"

console.log(message)
```

```
// Fichier compilation.js
//Generated by typescript 1.8.10

var message = "Hello World";

console.log(message);
```

IV. Syntaxes de base

IV.I. Compilation et Exécution (2)

- **Etapes de compilation d'un fichier**
  - Sauvegarder le fichier « compilation.ts »
  - Se positionner sur le nom d'un fichier et cliquer sur le bouton droit, puis choisir « Open in Terminal »
  - Lancer la compilation via la commande
    - **tsc compilation.ts**
  - Vérifier le contenu et lancer le fichier généré « compilation.js »
    - **node .\compilation.js**
- **Plusieurs fichiers peuvent être compilés à la fois**
  - Exemple : **tsc file1.ts, file2.ts, file3.ts**

IV. Syntaxes de base

IV.I. Compilation et Exécution (3)

- **Contexte de compilation**
  - Permet de définir les fichiers à analyser afin de déterminer ceux valides et invalides.
  - Permet également de définir les configurations pour la compilation.
  - Il faut définir un fichier « tsconfig.json » à la racine du projet.
  - Lien vers toutes les options et leur valeur par défaut : https://www.typescriptlang.org/docs/handbook/compiler-options.html
  - Commandes
    - Lancement simple : **tsc**
    - Lancement en mode suivi des changements : **tsc --watch**

IV. Syntaxes de base

IV.I. Compilation et Exécution (4)

- **Contexte de compilation**

```
{
    "include":["./src"],                    // liste des fichiers à analyser
    "exclude":["./folder/**/*.spec.ts"] , // liste des fichiers à ne pas analyser
    "compilerOptions": {
        "target":  "es6",                   // version cible ecmascript
        " sourceMap":  « true",             // génération de fichier .map permettant de
                                            // faciliter le debug
        "declaration": false,               // génération de fichier de déclaration .d.ts
        "noImplicitAny":   false,           // désactiver l'inférence du type any (déclaration des
        "removeComments":  true             // retirer tous les commentaires ,  variables)
        "outFile":   "dist/output.js"       // fichier de sortie regroupant tous les
                                            // fichiers
    }
}
```

IV. Syntaxes de base

IV.II. Identifiants (1)

- Les identifiants sont les noms des éléments du programme (variables, fonctions, etc..).

- Leur définition doit suivre certaines règles.
  - *Les identifiants peuvent inclure des caractères et des chiffres mais ne doit pas commencer par un chiffre*
  - *Les identifiants ne peuvent pas inclure les caractères spéciaux sauf le « _ » et le « $ »*
  - *Les identifiants ne peuvent pas être des mots clés*
  - *Les identifiants doivent être uniques dans un domaine*
  - *Les identifiants sont sensibles à la casse*
  - *Les identifiants ne peuvent pas contenir d'espaces*

IV. Syntaxes de base

IV.II. Identifiants (2)

- Quelques exemples

Identifiants valides	Identifiants invalides
firstName	var
first_name	first name
num1	first-name
$result	1number
.....	.....

IV. Syntaxes de base

IV.III. Mots clés (1) `= mots réservés`

- Quelques exemples

break	**var**
if	switch
string	extends
return	instanceof
public	this
.....	.....

IV. Syntaxes de base

IV.III. Mots clés (2)

- Quelques remarques
  - Les espaces, tabulations et les sauts de ligne sont ignorés par TypeScript
  - TypeScript est sensible à la casse : différentiation de la minuscule et de la majuscule
  - Le « ; » est optionnel à la fin d'une instruction

IV. Syntaxes de base

IV.IV. Commentaires

- Deux types de commentaire

```
//this is single line comment

/* This is a  Multi-line comment */
```

IV. Syntaxes de base

IV.V. TypeScript et Programmation Orientée Objet (POO) (1)

- TypeScript est un langage orienté objet `==> Javascript ne l'est pas`

- Il se base sur l'ensemble des principes de la POO
  - Classe
    - Attribut(s)
    - Méthode(s)
  - Héritage
  - Polymorphisme
  - Objet / Instance

IV. Syntaxes de base

IV.V. TypeScript et Programmation Orientée Objet (POO) (2)

- Exemple

```
class Greeting {
        greet():void     {
                console.log("Hello World!!!")
        }
}

var obj = new Greeting();

obj.greet(); // ➔ Hello World!!!
```

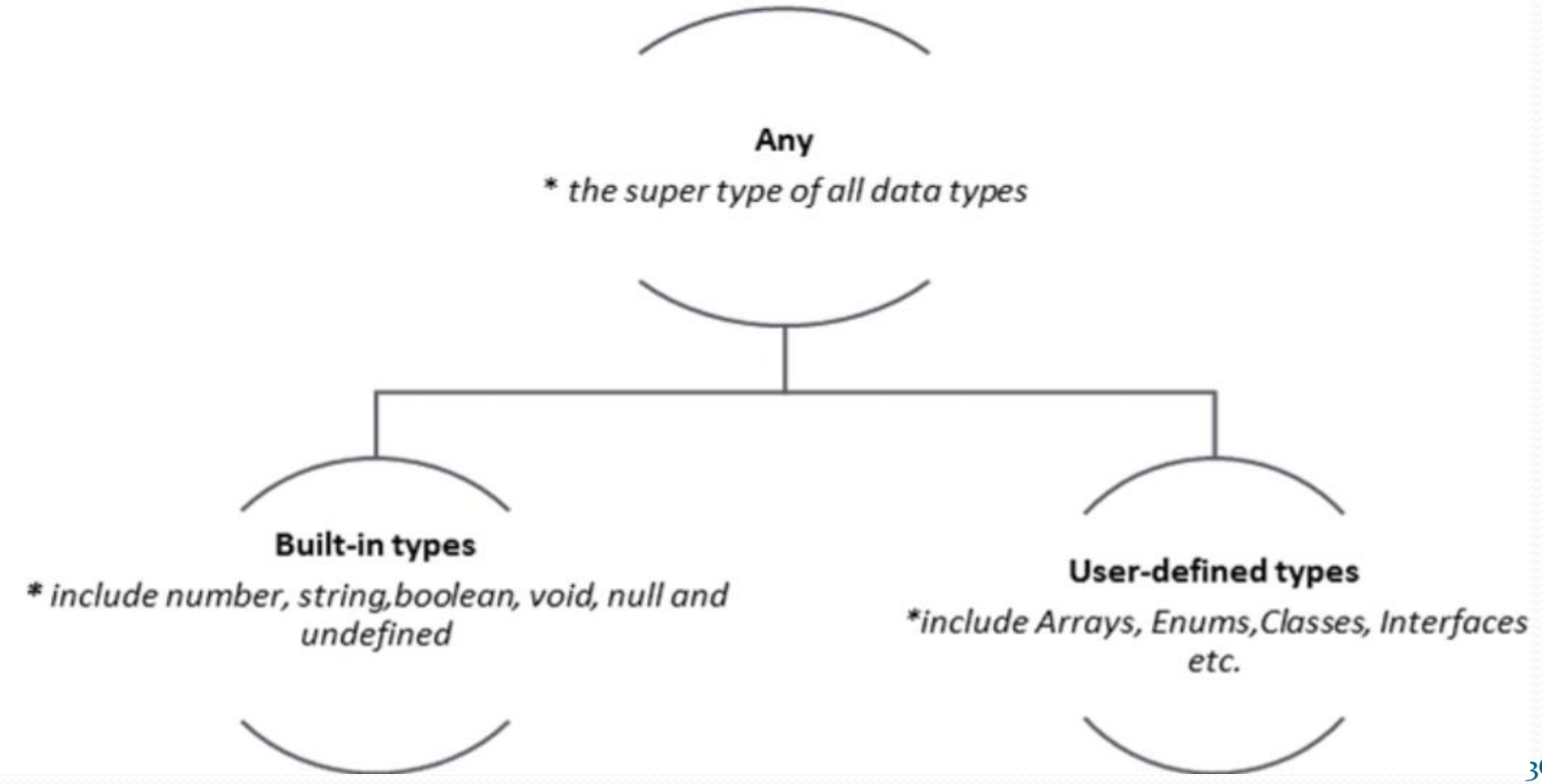
V. Types

- V.I. Type « any »
- V.II. Types natifs
- V.III. Types personnalisés

V. Types

V.I. Type « any »

- Super type de tous les types, peut être utilisé pour stocker toutes les valeurs possibles



V. Types

V.II. Types natifs (1)

- Il y en a plusieurs
  - number
    - Représente les nombres : entiers, décimaux, etc..
  - string
    - Représente les chaînes de caractère
  - boolean
    - Représente les valeurs logiques : true / false

V. Types

V.II. Types natifs (2)

- Il y en a plusieurs
  - void
    - Représente l'absence de type de retour pour les fonctions n'ayant pas de valeur de retour
  - null
    - Représente une absence volontaire de valeur
  - undefined
    - Représente la valeur par défaut des variables non initialisés

V. Types

V.II. Types personnalisés

- Ce sont ceux crées par les développeurs
  - *Classe*
  - *Interface*
  - *Enum* = Enumération permet de ne choisir que les variables présentes dans la déclaration ex: enumGenre {Homme, Femme}; ne peut prendre que les valeurs homme ou femme mais pas masculin ou féminin
  - *Tableaux d'objets*
  - *Etc..*

VI. Variables

- VI.I. Déclaration
- VI.II. Assertion de type
- VI.III. Inférence
- VI.IV. Portée
- VI.V. Constante

VI. Variables

VI.I. Déclaration

- Une variable est un élément permettant de stocker des éléments en mémoire dans une application

Exemples de déclaration	Commentaire
var name:string=”mary”	
var name:string;	Initialisation à la valeur « undefined »
var name=”mary”	Inférence du type vers le type « chaîne de caractère »
var name;	Initialisation à la valeur « undefined », avec un type « any »

VI. Variables

VI.II. Assertion de type

- TypeScript permet de forcer le type d'une variable pour la stocker dans un autre type, cela est appelé « ***Assertion de type*** »

- Exemple

```
var str='1';

var str2:number = <number><any> str; //str is now of type number

console.log(str2); // Résultat affiché 1
```

VI. Variables

VI.III. Inférence

- « **L'inférence de type** » dans TypeScript est le comportement par lequel le compilateur détermine le type d'une variable qui n'en a pas au travers de la 1ere assignation de valeur à cette variable.

- Exemple

```
var num=2;    // data type inferred as  number

console.log("value of num "+num);

num="12";  // error TS2011: Cannot convert 'string' to 'number'.

console.log(num);
```

VI. Variables

VI.IV. Portée (1)

- La portée définie l'espace dans lequel la variable est définie. La disponibilité de la variable dans une application dépend de sa portée.

- Il y a plusieurs types de portée :
  - Portée Global
  - Portée Classe
  - Portée Locale

VI. Variables

VI.IV. Portée (2)

```
var global_num=12     //global variable

class Numbers {
        num_val=13;     //class variable
        static sval=10;  //static field

        storeNum():void     {
                var local_num=14;   //local variable
        }
}

console.log("Global num: "+ global_num) //global variable
console.log(Numbers.sval) //static field
var obj= new Numbers();
console.log("Global num: "+ obj.num_val) //class variable
console.log("Global num: "+ local_num) //local variable ➔ error
```

VI. Variables

VI.V. Constante (1)

- Une constante est un élément utilisé pour stocker une valeur qui ne changera pas au cours du temps. Ne peut pas être utilisé comme attribut d'une classe.

- Exemple

```
// Low     readability
if (x > 10){ }

// Better!
const maxRows = 10;

if (x > maxRows){ }
```

VII. Operateurs

- VII.I. Description
- VII.II. Opérateurs arithmétiques
- VII.III. Opérateurs relationnels
- VII.IV. Opérateurs logiques
- VII.V. Opérateurs d'assignation
- VII.VI. Opérateurs particuliers

VII. Operateurs

VII.I. Description (1)

- Les opérateurs sont des éléments permettant d'effectuer des actions sur les données et entre les données.

- Exemples
  - Addition : +

VII. Operateurs

VII.II. Opérateurs arithmétiques

- Les principaux
- Exemple (a vaut 10 et b vaut 5)

Operateur	Fonction	Exemple
+	Addition	a + b = 15
-	Soustraction	a - b = 5
*	Multiplication	a * b = 50
/	Division	a / b = 2
%	Modulo `division reste`	a % b = ~~2~~ 0
++	Incrémentation	a++ / ++a = 11
--	Décrémentation	a-- / --a = 9

VII. Operateurs

VII.III. Opérateurs relationnels

- Les principaux
- Exemple (a vaut 10 et b vaut 20)

Operateur	Fonction	Exemple
>	Supérieur	a > b vaut false
<	Inférieur	a < b vaut true
>=	Supérieur ou égal	a >= b vaut false
<=	Inférieur ou égal	a <= b vaut true
==	Egalité	a + b == c vaut true
!=	Différence	a != b vaut true

VII. Operateurs

VII.IV. Opérateurs logiques

- Les principaux
- Exemple (a vaut 10 et b vaut 20)

Operateur	Fonction	Exemple
&&	Et	(a > 10 && b > 10) vaut false
||	Ou	(a > 10 || b > 10) vaut false
!	Négation	!(a > 10 && b > 10) vaut true

VII. Operateurs

VII.V. Opérateurs d'assignation

- Les principaux
- Exemple (a vaut 10 et b vaut 5)

Operateur	Fonction	Exemple
=	Assignation	a = 22 après a vaut 22
+	Addition	a += b après a vaut 15
-	Soustraction	après a vaut 5
*	Multiplication	après a vaut 50
/	Division	après a vaut 2

VII. Operateurs

VII.VI. Opérateurs particuliers (1)

- Operateur de négation
  - Exemple
    - a = -2
- Operateur de concaténation
  - Exemple
    - var msg:string = "hello"+"world"
- Opérateur ternaire : Test ? expr1 : expr2
  - Exemple
    - var num:number=-2
    - var result = num > 0 ?"positive":"non-positive"

```
le résultat de cette valeur
(fonction ou opération) doit être
un booléen (true ou false)
```

VII. Operateurs

VII.VI. Opérateurs particuliers (2)

- Operateur typeof
  - Exemple
    - var num=12
    - console.log(typeof num); //output: number

- Operateur instanceof
  - Exemple
    - var obj= new Numbers();
    - console.log(obj instanceof Numbers); //output: true

VIII. Décisions et Boucles

- VIII.I. Déclaration avec If Else
- VIII.II. Déclaration Switch
- VIII.III. Boucle avec While
- VIII.IV. Boucle avec For
- VIII.V. Boucle avec Do While
- VIII.VI. Déclaration Break
- VIII.VII. Déclaration Continue
- VIII.VIII. Boucle infinie

VIII. Décisions et Boucles

VIII.I. Déclaration avec If Else

- Exemple

```
if(boolean_expression1)  {

        // statement(s) will execute if the Boolean expression1 is true

} else if (boolean_expression2) {

        // statement(s) will execute if the Boolean expression2 is false

} else {

        //statements if both expression1 and expression2 result to false

}
```

VIII. Décisions et Boucles

VIII.II. Déclaration Switch

- Exemple

```
switch(variable_expression){
        case constant_expr1: {
                //statements;   break;
        }
        case constant_expr2: {
                //statements;   break;
        }
        default: {
                //statements;   break;
        }
}
```

VIII. Décisions et Boucles

VIII.III. Boucle avec While

- Exemple

```
while(condition) {

        // statements if the condition is true

}
```

VIII. Décisions et Boucles

VIII.IV. Boucle avec For

- Exemple

```
for (initial_count_value; termination-condition; step) {
        //statements ➔ Iteration sur un index local
 }
```

```
for (var key in list) {
        //statements ➔ Iteration sur les index des éléments du tableau
}
```

```
for (var value of list) {
        //statements ➔ Iteration sur les valeurs des éléments du tableau
}
```

VIII. Décisions et Boucles

VIII.V. Boucle avec Do While

- Exemple

```
do {

        //statements

} while(condition)
```

VIII. Décisions et Boucles

VIII.VI. Déclaration Break

- Permet de sortir d'une boucle

```
permet de sortir de toutes les boucles même si il y en a 15
```

- Exemple
  - break;

VIII. Décisions et Boucles

VIII.VII. Déclaration Continue

- Permet de passer à l'itération suivante d'une boucle

```
sort juste de la boucle où il est
```

- Exemple
  - continue;

```
ex :
for(....){            le continue permet de sortir du if sans faire les instructions suivante
     if (...){        mais de relancer la boucle for
     continue
      }
instruction
instruction
}
```

VIII. Décisions et Boucles

VIII.VIII. Boucle infinie

- Exemple

```
for(;;) {
        //statements
}
```

```
while(true) {
        //statements
}
```

```
do {
        //statements
} while(true);
```

IX. Nombres

- IX.I. Description
- IX.II. Méthodes

IX. Nombres

IX.I. Description (1)

- Permet de stocker des nombres (entiers, décimaux, etc..)

- Exemple
  - 2
  - 3,15

IX. Nombres

IX.II. Méthodes (1)

- toExponential() : affiche le nombre dans le format exponentiel
  - var num1=1225.30
  - var val= num1.toExponential();
  - console.log(val) // ➔ Output : 1.2253e+3

- toFixed() : affiche un certain nombre de digits après la virgule
  - var num3=177.234
  - num3.toFixed() is 177
  - num3.toFixed(2) is 177.23
  - num3.toFixed(6) is 177.234000

IX. Nombres

IX.II. Méthodes (2)

- toString() : affiche la chaîne de caractère correspondant au nombre
  - var num = new Number(177.1234);
  - console.log(num.toString()); ➔ Output : 177.1234
- valueOf() : affiche la valeur stockée dans le nombre
  - var num = new Number(10);
  - console.log(num.valueOf()); ➔ Output : 10
- Etc..

X. Chaînes de caractère

- X.I. Description

- X.II. Méthodes

X. Chaînes de caractère

X.I. Description (1)

- Permet de stocker des chaînes de caractère

- Exemple
  - Toto
  - Louis est de la partie

- Pour avoir le nombre de caractères, exécuter : maChaine.length

X. Chaînes de caractère

X.II. Méthodes (1)

- charAt() : affiche le caractère situé à la position en paramètre
  - var str = new String("This is string");
  - str.charAt(0) is:T
  - str.charAt(1) is:h

- concat() : affiche une concaténation de chaînes
  - var str1 = new String("This is string one");
  - var str2 = new String("This is string two");
  - var str3 = str1.concat(str2);
  - console.log("str1 + str2 : "+str3) ➔ Output : str1 + str2 : This is string oneThis is string two

X. Chaînes de caractère

X.II. Méthodes (2)

- indexOf() : retourne la position de la 1ere occurrence du paramètre ou -1 si aucune occurrence
  - var str1 = new String("This is string one");
  - var index = str1.indexOf("string");
  - console.log("indexOf found String :" + index); ➔ Output : 8
  - var index = str1.indexOf("one");
  - console.log("indexOf found String :" + index); ➔ Output : 15

- split() : subdivise la chaîne de caractère en plusieurs sous chaînes de caractères à partir d'une chaîne de caractère donnée
  - var str = "Apples are round, and apples are juicy.";
  - var splitted = str.split(" ", 3);
  - console.log(splitted) ➔ Output : ['Apples', 'are', 'round,']

condition de séparation, ici l'espace, ça pourrait être une lettre

- Etc..

XI. Collections

- XI.I. Tableaux : Déclaration et Initialisation
- XI.II. Tableaux : Méthodes
- XI.III. Tableaux multidimensionnels
- XI.IV. Concepts de « Spread » et de « Déstructuration »
- XI.V. Collection « Set »
- XI.VI. Collection « Map »

XI. Collections

XI.I. Tableaux : Déclaration et Initialisation

- Déclaration
  - var array_name[:datatype]; //declaration
  - array_name=[val1,val2,valn..] //initialization

- Initialisation
  - var alphas:string[];
  - alphas=["1","2","3","4"]
  - console.log(alphas[0]);
  - console.log(alphas[1]);

XI. Collections

XI.II. Tableaux : Méthodes (1)

- concat() : crée un nouveau tableau avec la concaténation de plusieurs tableaux
  - var alpha = ["a", "b", "c"];
  - var numeric = [1, 2, 3];
  - var alphaNumeric = alpha.concat(numeric);
  - console.log("alphaNumeric : " + alphaNumeric); ➔ Output : alphaNumeric : a,b,c,1,2,3

- filter() : crée un nouveau tableau en gardant uniquement les éléments respectant les conditions de la fonction passée
  - function isBigEnough(element, index, array) {
    return (element >= 10);
    }
  - var passed = [12, 5, 8, 130, 44].filter(isBigEnough);
  - console.log("Test Value : " + passed); ➔ Output : Test Value :12,130,44

XI. Collections

XI.II. Tableaux : Méthodes (2)

- indexOf() : retourne la position de la 1ere occurrence du paramètre ou -1 si aucune occurrence
  - var index = [12, 5, 8, 130, 44].indexOf(8);
  - console.log("index is : " + index); ➔ Output : index is : 2

- join() : concatène tous les éléments du tableau pour créer un chaîne de caractère
  - var arr = new Array("First","Second","Third");
  - var str = arr.join(", ");
  - console.log("str : " + str); ➔ Output : str : First, Second, Third

- Etc..

XI. Collections

XI.III. Tableaux multidimensionnels

- Un élément d'un tableau peut être un autre tableau, on parle de tableaux multidimensionnels
  - var arr_name:datatype[][]=[[val1,val2,val3],[v1,v2,v3]]

- Exemple
  - var multi:number[][]=[[1,2,3],[23,24,25]]
  - console.log(multi[0][1]) ➔ Output : 2
  - console.log(multi[1][1]) ➔ Output : 24

XI. Collections

XI.IV. Concepts de « Spread » et « Déstructuration » (1)

- Le concept de « spread » permet de transformer un tableau ou un objet en plusieurs sous-éléments.

- Le concept de « Déstructuration » est un exemple usage du concept de « Spread », la spécificité est que l'objectif est de stocker ces sous-éléments dans plusieurs variables.

XI. Collections

XI.IV. Concepts de « Spread » et « Destructuration » (2)

- Exemples
  - Tableau
    - Spread : appel de fonction
      - function foo(x, y, z) { }
      - var args = [0, 1, 2];
      - foo(...args);
    - Spread : assignation de tableaux
      - var list = [1, 2];
      - list = [...list, 3, 4];
      - console.log(list); // [1,2,3,4]
    - Déstructuration
      - var [x, y, ...remaining] = [1, 2, 3, 4];
      - console.log(x, y, remaining); // 1, 2, [3,4]

XI. Collections

XI.IV. Concepts de « Spread » et « Destructuration » (3)

- Exemples
  - Objet
    - Spread : ajout de propriété
      - const point2D = {x: 1, y: 2};
      - /** Create a new object by using all the point2D props along with z */
      - const point3D = {...point2D, z: 3};
    - Déstructuration
      - var rect = { x:0, y: 10, width: 15, height: 20};
      - // Destructuring assignment
      - var { x, y, width, height} = rect;
      - console.log(x, y, width, height); // **0,10,15,20**

XI. Collections

XI.V. Collection « Set » (1)

- Permet de stocker les éléments d'une collection.

- A la différence d'un tableau ou d'une liste, les éléments d'une collection de type « Set » sont uniques.

```
Jeu de données : 1,2,1,5 à placer dans :
- Tableau : [1,2,1,5]
- Set : [1,2,5]
```

XI. Collections

XI.V. Collection « Set » (2)

- Exemples

```
var myArray:string[] = ["Hello", "World of Array“, "Hello", "World of Array"];
var mySet = new Set(myArray);

for(let item of mySet) {
        console.log(item); // "Hello", "World of Array“ (2 elements seulement)
}
```

XI. Collections

XI.VI. Collection « Map » (1)

- Permet de stocker les données sous la forme clé / valeur.

- Cela permet un accès très rapidement à chaque élément sans avoir à parcourir tout l'ensemble comme pour d'autres collections.

```
Notion de marque pages
```

XI. Collections

XI.VI. Collection « Map » (2)

- Exemples

```
var myMap:Map<string, string> = new Map([["myArray", "Hello World of
Array"], ["mySet", "Hello World of Set"]]);

for(let item of myMap) {
        console.log(item);
}
```

XII. Fonctions

- XII.I. Déclaration
- XII.II. Appel
- XII.III. Fonctions paramétrées
- XII.IV. Paramètres optionnels
- XII.V. Paramètres « Rest »
- XII. VI. Paramètres par défaut
- XII.VII. Lambda

XII. Fonctions

XII.I. Déclaration

- Description de la fonction

```
function function_name():return_type {

        //statements

        return value;

}
```

```
function greet():string {

        console.log(“Hello World“);

        return "Hello World“;
}
```

XII. Fonctions

XII.II. Appel

- Consiste à exécuter le code d'une fonction dans une instruction

```
function caller() {

        var msg=greet(); //function greet() invoked
        console.log(msg) ;

}

//invoke function
caller();
```

XII. Fonctions

XII.III. Fonctions paramétrées

- Une fonction peut avoir des paramètres d'appel

```
function func_name( param1 [:datatype], ( param2 [:datatype]) {

}
```

```
function test_param(n1:number, s1:string) {

        console.log(n1);
        console.log(s1);

}

test_param(123, "this is a string");
```

XII. Fonctions

XII.IV. Paramètres optionnels

- Les paramètres d'une fonction peuvent être optionnels

```
function function_name (param1[:type], param2[:type], param3?[:type]) {

}
```

```
function disp_details(id:number, name:string, mail_id?:string) {
        console.log("ID:", id);
        console.log("Name",name);
        if(mail_id!=undefined)
                console.log("Email Id",mail_id);
}

disp_details(123,"John");
disp_details(111,"mary","mary@xyz.com");
```

XII. Fonctions

XII.V. Paramètres « Rest »

- Les paramètres « rest » permettent de ne pas définir le nombre de paramètre en entrée d'une fonction.

```
function addNumbers(...nums:number[]) {
        var i;
        var sum:number=0;

        for(i=0;i<nums.length;i++)    {
                sum=sum+nums[i];
        }

        console.log("sum of the numbers",sum);
}

addNumbers(1,2,3);
addNumbers(10,10,10,10,10);
```

XII. Fonctions

XII.VI. Paramètres par défaut

- Les paramètres d'une fonction peuvent avoir une valeur par défaut

```
function function_name(param1[:type],param2[:type]=default_value) {

}
```

```
function calculate_discount(price:number,rate:number=0.50) {
        var discount= price * rate;
        console.log("Discount Amount: ",discount);
}

calculate_discount(1000);
calculate_discount(1000,0.30);
```

XII. Fonctions

XII.VII. Lambda (1)

- Les fonctions lambdas sont des fonctions concises qui permettent de définir de simples opérations et des fonctions anonymes.

- La structure est :
  - Paramètres : les éléments en entrée
  - Symbole « => » : opérateur de lancement ou opérateur lambda
  - Instructions : les étapes de la fonction

XII. Fonctions

XII.VII. Lambda (2)

- **Expression lambda**
  - Modèle

```
( [param1, parma2,…param n] )=>statement;
```

  - Exemple

```
var foo=(x:number)=>10+x; Ne pas mettre le type de la fonction
console.log(foo(100))  //outputs 110
```

XII. Fonctions

XII.VII. Lambda (3)

- **Instruction lambda**
  - Modèle

```
( [param1, parma2,...param n] )=> {
        //code block
}
```

  - Exemple

```
var foo=(x:number)=> {
    x=10+x;
    console.log(x);
}

foo(100)
```

ES6 - TypeScript Cours

Naby Daouda Diakite

XIII. Interfaces

- XIII.I. Déclaration et Appel
- XIII.II. Union de types
- XIII.III. Interfaces et Tableaux
- XIII.IV. Héritage

XIII. Interfaces

XIII.I. Déclaration et Utilisation (1)

- Une interface est un contrat que doit suivre une entité ou une classe dérivée.

- Une interface peut comprendre des propriétés et des méthodes

XIII. Interfaces

XIII.I. Déclaration et Utilisation (2)

- Exemple

```
interface IPerson {
        firstName:string,
        lastName:string,
        sayHi: ()=>string
}

var customer:IPerson={
        firstName:"Tom",
        lastName:"Hanks",
        sayHi: ():string =>{return "Hi there"}
}

console.log(customer.firstName);
console.log(customer.lastName);
console.log(customer.sayHi());
```

XIII. Interfaces

XIII.II. Union de types

- Permet d'associer deux types à une propriété.

```
interface RunOptions {
        program:string;
        commandline:string[] | string;
}

//commandline as string var
options:RunOptions={program:"test1",commandline:"Hello"};
console.log(options.commandline)

//commandline as a string array
options={program:"test1",commandline:["Hello","World"]};
console.log(options.commandline[0]);
console.log(options.commandline[1]);
```

XIII. Interfaces

XIII.III. Interfaces et Tableaux

- Permet de définir le type de l'index et de la valeur pour étendre les tableaux

```
interface namelist {
        [index:number]:string
}

var list2:namelist=["John", 1, "Bran"]  //Error. 1 is not type string

interface ages {
        [index:string]:number
}

var agelist:ages;
agelist["John"]=15  // Ok
agelist[2]="nine"  // Error
```

XIII. Interfaces

XIII.IV. Héritage (1)

- Le concept « Héritage multiple » s'applique aux interfaces, il permet de mettre en commun des éléments métiers

```
interface Person {
        age:number
}

interface Musician extends Person {
        instrument:string
}

var drummer=<Musician>{};
drummer.age=27;
drummer.instrument="Drums“;

console.log("Age:  "+ drummer.age);
console.log("Instrument:  "+ drummer.instrument);
```

XIV. Classes

- XIV.I. Déclaration
- XIV.II. Instance
- XIV.III. Attributs et fonctions
- XIV.IV. Héritage
- XIV. V. Mot clé « static »
- XIV.VI. Opérateur « instanceof »
- XIV.VII. Classes et interfaces

XIV. Classes

XIV.I. Déclaration (1)

- Une classe permet de définir les éléments du monde réel.

- Elle comporte des propriétés, des constructeurs et des fonctions.

XIV. Classes

XIV.I. Déclaration (2)

- Exemple

```
class Car {
        //field
        engine:string;

        //constructor
        constructor(engine:string)  {
                this.engine=engine
        }

        //function  disp():void {
              console.log("Engine is  :   "+this.engine)
        }
}
```

XIV. Classes

XIV.II. Instance

- Exemple

```
var object_name= new class_name([ arguments ]);

var obj= new Car("Engine1");
```

XIV. Classes

XIV.III. Attributs et fonctions

- Exemple

```
//accessing an attribute
obj.field_name;

//accessing a function
obj.function_name();
```

XIV. Classes

XIV.IV. Héritage

- Le concept « Héritage multiple » s'applique aux classes, il permet de mettre en commun des éléments métiers

```
class Shape {
        Area:number;

        constructor(a:number) {
                this.Area=a;
        }
}

class Circle extends Shape{
        disp():void  {
                console.log("Area of the circle:  "+this.Area);
        }
}

var obj=new Circle(223);
obj.disp()
```

XIV. Classes

XIV.V. Mot clé « static »

- Le mot clé « static » permet de rendre dépendant une propriété de toute instance

```
class StaticMem {
        static num:number;

        static disp():void {
                console.log("The value of num is"+ StaticMem.num);
        }
}

StaticMem.num=12;  // initialize the static variable
StaticMem.disp();  // invoke the static method
```

XIV. Classes

XIV.VI. Opérateur « instanceof »

- Permet de savoir si une instance est d'un type donné.

```
class Person{ }

var obj=new Person();
var isPerson=obj instanceof Person;

console.log(" obj is an instance of Person " + isPerson);
```

XIV. Classes

XIV.VII. Classes et interfaces (1)

- En implémentant une interface, une classe doit respecter le contrat de l'interface, en définissant tous les éléments de l'interface (propriétés et méthodes).

XIV. Classes

XIV.VII. Classes et interfaces (2)

- Exemple

```
interface ILoan {
        interest:number
}

class AgriLoan implements ILoan {
    interest:number ;
    rebate:number;

    constructor(interest:number,rebate:number) {
        this.interest=interest;
        this.rebate=rebate;
    }
}

var obj:ILoan = new AgriLoan(10,1);
console.log("Interest is : "+obj.interest+" Rebate is : "+obj.rebate )
```

XV. Généricité

- XV.I. Principe

- XV.II. Mise en place

XV. Généricité ==> Spécialisation

XV.I. Principe (1)

- Permet de mettre en commun des traitements, cela permet de :
  - Réduire la duplication de code
  - Réduire la base de code donc les potentiels bugs
  - Faciliter les évolutions

- Le principe est d'utiliser un type générique T au lieu d'un type connu.

XV. Généricité

XV.II. Mise en place (1)

- **Méthode générique**

```
function identity<T>(arg: T): T {
        return arg;
}

let outputString = identity<string>("myString"); // type of output will be 'string'

let outputNumber = identity<number>(22);

console.log("Result of Generic identity : " + identity("erqgg")); // erqgg
console.log("Result of Generic outputString : " + outputString); // myString
console.log("Result of Generic identity : " + identity(11)); // 11
console.log("Result of Generic outputNumber : " + outputNumber); // 22
```

XV. Généricité

XV.III. Mise en place (2)

- **Classe générique**

```
class Person {
        lastName: string;
        firstName: string;

        constructor(lastName:string, firstName:string) {
                this.lastName = lastName;
                this.firstName = firstName;
        }
}
```

XV. Généricité

XV.III. Mise en place (3)

- **Classe générique**

```
class Soldier extends Person {
        target: number;

        constructor(lastName:string, firstName:string, target: number) {
                super(lastName, firstName);
                this.target = target;
        }
}
```

XV. Généricité

XV.III. Mise en place (4)

- **Classe générique**

```
class Doctor extends Person {
        grade: string;

        constructor(lastName:string, firstName:string, grade: string) {
                super(lastName, firstName);
                this.grade = grade;
        }
}
```

XV. Généricité

XV.III. Mise en place (6)

- **Classe générique**

```
class GenericIdentityManager<T extends Person> {
        identifyPersonalInformation(t:  T): void {
                console.log(t.firstName + " " + t.lastName + " has been identified has a citizen");
        }
}

class SoldierIdentityManager extends GenericIdentityManager<Soldier> {
}

var soldierIdentityManager:SoldierIdentityManager  = new SoldierIdentityManager();

soldierIdentityManager.identifyPersonalInformation(new  Soldier("Legrand", "Louis", 22));

// soldierIdentityManager.identifyPersonalInformation(new  Doctor("Dub", "Max", A++"));
```

XVI. Namespace

- XVI.I. Principe
- XVI.II. Mise en place

XVI. Namespace

XVI.I. Principe

- Permet de **regrouper de façon logique du code homogène** afin de construire des blocs homogènes.

- Cela permet d'éviter les **collisions de variable dans la portée globale** même pour des éléments ayant le même nom.

```
namespace SomeNameSpaceName {
        export interface ISomeInterfaceName {
                ……
        }
        export class SomeClassName {
                ……
        }
}
```

- Un **espace de nom peut en contenir un autre**, dans ce cas pour les référencer, il faut séparer le nom par un « . »

XVI. Namespace

XVI.II. Mise en place (1)

- Exemple

```
namespace SomeNameSpaceName {
    export interface ISomeInterfaceName {
        helloWorld(name: string):void;
    }

    export class SomeClassName implements ISomeInterfaceName {
        constructor() {

        }

        helloWorld(name: string):void {
                console.log("Helloworld : " + name);
        }
    }
}
```

XVI. Namespace

XVI.II. Mise en place (2)

- Exemple

```
// Référencement d'un namespace provenant d'un autre fichier
/// <reference path="generic.ts" />

var namespace:SomeNameSpaceName.ISomeInterfaceName = new
SomeNameSpaceName.SomeClassName();

namespace.helloWorld("Naby");
```

XVII. Modules

- XVII.I. Historique
- XVII.II. Description

XVII. Modules

XVII.II. Historique

- Historiquement, il y avait deux types de modules
  - *Module interne : maintenant cela correspond aux « namespace » (depuis ES6)*
  - *Module externe : maintenant cela est appelé « module »*

XVII. Modules

XVII.II. Description

- Le concept de « module » permet de construire des blocs applicatifs homogènes au sein d'un gros projet afin de mieux gérer les problématiques métiers.
  - *En TypeScript (depuis ES6), tout fichier contenant comme 1ere instruction un « import » ou « export » est considéré comme un module.*
  - *Les relations entre les modules s'effectuent via les « import » et « export »,*

XVIII. Ambient

- XVIII.I. Description

XVIII. Ambient

XVIII.I. Description

- « Ambient » est une technique permettant d'indiquer au compilateur que le code d'une librairie sera fourni ultérieurement, tout en permettant d'accéder aux fonctionnalités de cette librairie à travers une interface TypeScript.
  - *Cela passe par la mise en place de fichiers de définitions (.d.ts)*

ES6 - TypeScript Cours

Naby Daouda Diakite

Merci pour votre attention !!