

Mémento

TypeScript

Version 1.0 (créé le 23/10/2024, modifié le 23/10/2024)



TS (TypeScript) est un langage de programmation côté client développé par l'entreprise Microsoft. C'est une surcouche de JavaScript. Le TypeScript est transpilé en JavaScript, qui est typé, orienté objet et avec des classes ayant plus de fonctionnalités.

Cette technologie est la propriété de Microsoft.



Loric Informatique

Table des matières

1. Prise en main.....	5
1.1. Outils nécessaires	5
1.2. Initialiser un projet TS.....	5
1.3. Transpiler le projet TS en JS.....	6
1.4. Premier code « Hello World! ».....	6
2. Bases	6
2.1. Syntaxe	6
2.2. Les variables	6
2.2.1. Types de variables.....	6
2.2.2. Opérations sur les variables	7
2.3. Commentaires.....	9
2.4. Les tableaux.....	9
2.5. Les objets	10
2.6. Conditions.....	10
2.6.1. Opérateurs de comparaison.....	10
2.6.2. Tests de conditions	11
2.7. Boucles	12
3. Les fonctions	13
3.1. Créer une fonction.....	13
3.2. Retourner une valeur de la fonction	13
3.3. Faire un appel à la fonction.....	13
3.4. Créer une fonction anonyme	14
3.4.1. Manière simple.....	14
3.4.1. Syntaxe fléchée.....	14
4. Les classes.....	14

4.1. Visibilités.....	14
4.2. Les classes de base.....	15
4.2.1. Créer une classe	15
4.2.2. Définir la classe dans un objet.....	15
4.2.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	15
4.2.4. Créer une méthode (dans une classe)	15
4.2.5. Créer une méthode statique (dans une classe)	16
4.2.6. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable dans une classe.....	16
4.2.7. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable en dehors d'une classe.....	16
4.2.8. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe.....	16
4.2.9. Créer un getter et un setter	16
4.3. Les classes internes.....	17
4.4. L'héritage	17
4.4.1. Créer une classe héritée d'une autre classe.....	18
4.4.2. Définir la classe dans un objet.....	18
4.4.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	18
4.5. Les classes abstraites.....	19
4.5.1. Créer une classe abstraite	19
4.5.2. Créer une méthode abstraite	19
4.5.3. Créer une classe héritant d'une classe abstraite	20
4.5.4. Implémenter une méthode abstraite (dans la classe fille).....	20
4.5.5. Définir la classe dans un objet	20
4.6. Les interfaces.....	21
4.6.1. Créer une interface.....	21

4.6.2. Définir une interface dans un objet.....	21
4.6.3. Créer une méthode abstraite	21
4.6.4. Créer une classe implémentant une ou plusieurs interfaces.....	22
4.6.5. Implémenter une méthode abstraite (dans la classe fille).....	22
4.6.6. Définir la classe dans un objet	22
4.7. Les classes anonymes.....	23
4.7.1. Manière simple	23
4.7.2. Expression lambda (ne fonctionne que si la classe abstraite ne contient qu'une seule méthode abstraite)	23
4.8. Les collections	24
4.8.1. Créer une collection	24
4.8.2. Définir la collection dans un objet.....	24
5. Le stockage local	25
5.1. Cookies.....	25
5.2. Session Storage.....	25
5.3. Local Storage	25
6. Les instructions.....	27
6.1. Instructions de bases	27
6.2. Math	27
6.3. JSON	27
6.4. Exécution asynchrone	28
7. Les importations et exportations	29
7.1. Exporter une classe (fonctionne aussi avec les fonctions et autres)....	29
7.2. Importer une classe exportée dans un autre fichier	29

1. Prise en main

1.1. Outils nécessaires

- Navigateur Internet à jour (ex : Firefox ou Chrome)
- Un logiciel de codage (Visual Studio Code (recommandé) ou Notepad++)
- Connaissances en HTML, CSS et JS
- Valideur JS : <https://jshint.com/> (faire précéder son code par `// jshint browser:true, eqeqeq:true, undef:true, devel:true, esversion: 6`)



Le memento HTML est disponible en ligne sur le site :

https://loricaudin.github.io/loric-informatique/mementos/html/memento_html.html



Le memento CSS est disponible en ligne sur le site :

https://loricaudin.github.io/loric-informatique/mementos/css/memento_css.html



Le memento JS est disponible en ligne sur le site :

https://loricaudin.github.io/loric-informatique/mementos/js/memento_js.html

1.2. Initialiser un projet TS

Installer TypeScript avec la commande : `npm install -g typescript`

Créer un dossier typescript et y faire la commande : `tsc --init`

Si la commande ci-dessus et la commande ci-dessous ne fonctionnent pas ou sont introuvables, essayer d'ajouter « npx » devant chaque commande « tsc ».

1.3. Transpiler le projet TS en JS

Commande : tsc

1.4. Premier code « Hello World! »

```
alert("Hello World!");
```

2. Bases

2.1. Syntaxe

```
instruction1;  
instruction2;  
instruction3;
```

2.2. Les variables

2.2.1. Types de variables

Fonction pour connaître le type : `typeof(variable)`

2.2.1.1. Types de base

Type	Description	Fonction pour le convertir
int ou number	Nombre entier	<code>parseInt(variable)</code> ou <code>parseInt(variable, base)</code>
float ou number	Nombre décimal (ex : 0.1)	<code>parseFloat(variable)</code>

string	Chaîne de caractères entre ' ' ou " " (caractère \n : retour à la ligne ; \t : tabulation ; \b : retour en arrière ; \f : nouvelle page)	<i>variable.toString()</i>
--------	---	----------------------------

2.2.1.2. Autres types

Type	Description
boolean	Valeur pouvant être true ou false
object	Autres types (tableau (ou liste), null...)

Il existe 2 types "nulle" : null qui est une valeur vide, et undefined qui est un élément inexistant.

2.2.2. Opérations sur les variables

Instruction	Description
1 + 2	Renvoie 3
3 - 1	Renvoie 2
6 * 4	Renvoie 24
5 / 2	Renvoie 2.5
Math.floor(5 / 2)	Renvoie 2 (le quotient sans décimal)
Math.trunc(5 / 2)	Renvoie 2 (le quotient sans décimal), utile si Math.floor() échoue (trop grand nombre)
5 % 2	Renvoie 1 (le reste de la division)
let <i>variable</i> : <i>type</i> = <i>valeur</i> ;	Déclare une nouvelle variable dont la portée est limitée à celle du bloc dans lequel elle est déclarée

<code>var variable: type = valeur;</code>	Déclare une nouvelle variable globale ou locale à une fonction (sans distinction des blocs utilisés dans la fonction)
<code>var a?: type;</code>	Déclare la variable <i>a</i> sans affecter de valeur (contient null)
<code>a!</code>	Ignorer les possibles valeurs null et undefined
<code>var a: unknown;</code>	Déclare la variable <i>a</i> sans affecter de valeur (contient null) et sans déclaré de type (le type est choisi automatiquement)
<code>var b: type1 type2 = valeur;</code>	Déclare une nouvelle variable pouvant être de plusieurs types
<code>var c: valeur1 valeur2... = valeur1;</code>	Déclarer une variable ne pouvant contenir qu'une des valeurs énumérées
<code>a = 5;</code>	Affecte 5 à une variable
<code>a = a + 3;</code>	Ajoute 3 à une variable
<code>a += 3;</code>	Ajoute 3 à une variable
<code>a++;</code>	Ajoute 1 à une variable
<code>b = a++;</code>	Équivalent à : <code>b = a; a = a + 1;</code>
<code>b = ++a;</code>	Équivalent à : <code>a = a + 1; b = a;</code>
<code>const PI = 3.14;</code>	Crée une variable constante (non modifiable)
<code>static variable: type = valeur;</code>	Déclare une nouvelle variable qui n'est jamais détruite (si déjà exécuté, cette instruction est ignorée)
<code>readonly variable: type = valeur;</code>	Crée variable avec une seule initialisation (utile dans les interfaces pour créer des constantes)
<code>let texte: string = 'chaîne';</code>	Affecte une chaîne de caractères à une variable (les chaînes peuvent s'additionner avec l'opérateur +)

<code>let texte: string = "chaîne";</code>	Affecte une chaîne de caractères à une variable (les chaînes peuvent s'additionner avec l'opérateur +)
--	--

2.3. Commentaires

```
//Commentaire tenant sur une ligne

/*
Commentaire pouvant être sur une ou plusieurs lignes
*/
```

2.4. Les tableaux

Instruction	Description
<code>var tableau: type[];</code>	Crée un tableau vide
<code>var tableau: Array<type>;</code>	Crée un tableau vide de type Array
<code>var tableau: any[] = ["Loric", "Informatique", 69];</code>	Crée une liste avec des valeurs (le type de valeurs n'a pas d'importance) (any est déconseillé)
<code>tableau[i]</code>	Renvoie la valeur du tableau à la position <i>i</i> (l'indice de la première valeur est 0), et permet aussi l'écriture d'une autre valeur
<code>tableau.push(valeur);</code>	Ajoute une valeur dans le tableau
<code>tableau.shift();</code>	Enlève et renvoie la première valeur du tableau
<code>tableau.pop();</code>	Enlève et renvoie la dernière valeur du tableau
<code>tableau.splice(indice, n);</code>	Enlève <i>n</i> valeurs à partir de l'indice spécifié
<code>tableau.length</code>	Renvoie la longueur du tableau
<code>tableau.includes(valeur);</code>	Recherche si la valeur est présente dans le tableau
<code>tableau.indexOf(valeur);</code>	Renvoie la position de la valeur recherchée ou -1 si elle n'est pas trouvée

<code>tableau = chaine.split(separateur);</code>	Couper une chaîne de caractères en précisant le séparateur et renvoie un tableau
<code>chaine = tableau.join(separateur);</code>	Convertir un tableau en chaîne de caractères en séparant par un séparateur

2.5. Les objets

Instruction	Description
<code>obj = {};</code>	Crée un objet vide
<code>obj = {a: "Loric", b: 2004};</code>	Crée un objet avec des valeurs (le type de valeurs n'a pas d'importance)
<code>obj["a"];</code> ou <code>obj.a;</code>	Récupère la valeur contenue dans une clé
<code>obj["c"] = 3;</code> ou <code>obj.c = 3;</code>	Ajoute 3 à l'objet
<code>delete obj.c;</code>	Supprime une clé de l'objet

2.6. Conditions

Une condition renvoie true si elle est respectée et false sinon

2.6.1. Opérateurs de comparaison

Condition	Description de ce que vérifie la condition
<code>a == b</code>	<i>a</i> égal à <i>b</i> (seulement en contenu : 1="1")
<code>a === b</code>	<i>a</i> égal à <i>b</i> (en type et en contenu : 1≠"1")
<code>a < b</code>	<i>a</i> strictement inférieur à <i>b</i>
<code>a > b</code>	<i>a</i> strictement supérieur à <i>b</i>

<code>a <= b</code>	<code>a</code> supérieur ou égal à <code>b</code>
<code>a != b</code>	<code>a</code> n'est pas égal à <code>b</code> (seulement en contenu : <code>1!="1"</code>)
<code>a !== b</code>	<code>a</code> n'est pas égal à <code>b</code> (en type et en contenu : <code>1≠"1"</code>)
<code>a in b</code>	<code>a</code> est présent dans <code>b</code> (qui peut être un tableau)
<code> </code>	À mettre entre deux conditions, permet d'avoir une des deux conditions qui doit être vraie
<code>&&</code>	À mettre entre deux conditions, permet d'avoir deux conditions qui doivent être vraies
<code>!condition</code>	Ne doit pas respecter la condition

2.6.2. Tests de conditions

Instruction	Description
<pre>if (condition1) { instruction1; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i>
<pre>if (condition1) { instruction1; } else { instruction2; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i> , sinon, on exécute <i>instruction2</i>
<pre>if (condition1) { instruction1; } else if (condition2) { instruction2; } else { instruction3; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i> , sinon, si <i>condition2</i> est vraie, on exécute <i>instruction2</i> , sinon, on exécute <i>instruction3</i>
<pre>switch (nombre) { case a: instruction1; break; case b: case c: instruction2; break; default:</pre>	Si <i>nombre</i> == <i>a</i> , alors on exécute <i>instruction1</i> , sinon, si <i>nombre</i> == <i>b</i> ou <i>c</i> , on exécute <i>instruction2</i> , sinon, on exécute <i>instruction3</i>

<code>instruction3;</code> <code>}</code>	
<code>a = (condition1) ? 1 : 0;</code>	Si <i>condition1</i> est vraie, <i>a</i> prend la valeur 1, sinon 0.
<code>try {</code> <code>instruction1;</code> <code>} catch(erreur) {</code> <code>instruction2;</code> <code>}</code>	Si <i>instruction1</i> provoque une erreur, on exécute <i>instruction2</i> (si <code>throw message;</code> est placé dans <i>instruction1</i> , alors <i>instruction2</i> est exécuté avec <i>erreur</i> = <i>message</i>)

2.7. Boucles

Instruction	Description
<code>for (let i = d; i < f; i++) {</code> <code>instruction1;</code> <code>}</code>	On répète $f-d$ fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris
<code>for (let i = d; i < f; i+=p) {</code> <code>instruction1;</code> <code>}</code>	On répète $(f-d)/p$ fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris avec pour pas égal à <i>p</i>
<code>for (let i in tableau) {</code> <code>instruction1;</code> <code>}</code>	On parcourt le tableau (ou une chaîne de caractères) pour <i>i</i> allant de 0 à la longueur de tableau
<code>for (let elt of tableau) {</code> <code>instruction1;</code> <code>}</code>	On parcourt le tableau (ou une chaîne de caractères) pour <i>elt</i> prenant toutes les valeurs du tableau
<code>while (condition) {</code> <code>instruction1;</code> <code>}</code>	On répète jusqu'à ce que la condition soit fausse (peut ne pas être répété)
<code>do {</code> <code>instruction1;</code> <code>} while(condition);</code>	On répète jusqu'à ce que la condition soit fausse (est forcément répété une fois)
<code>break;</code>	Permet de sortir d'une boucle sans la terminer (à éviter si possible)

3. Les fonctions

Les fonctions peuvent être situées dans le même fichier, mais doivent être définies avant d'être appelées.

3.1. Créer une fonction

```
function maFonction(variable1: type1, variable2: type2...): type0 {  
    instructions;  
}
```

Si aucune valeur n'est renvoyée, le type de retour est égal à `void`

3.2. Retourner une valeur de la fonction

```
return variable;
```

3.3. Faire un appel à la fonction

```
variable = maFonction(valeur1, valeur2...);
```

ou (s'il n'y a pas de variable de retour)

```
maFonction(valeur1, valeur2...);
```

Remarque : il est possible de faire une surcharge de fonctions, c'est-à-dire qu'il est possible de créer deux fonctions identiques avec des paramètres de types différents, ce qui permet à l'interpréteur de choisir la fonction correspondant au type de variables saisies. Idem pour les constructeurs des classes.

Note : Si une valeur n'a pas de valeur renseignée, alors elle prend comme valeur `undefined`, il est cependant possible d'indiquer avec un « ? » après le nom de la variable que le paramètre est facultatif.

3.4. Créer une fonction anonyme

3.4.1. Manière simple

```
function(variable1: type1, variable2: type2...): type0 {  
    instructions;  
}
```

3.4.1. Syntaxe fléchée

```
(variable1: type1, variable2: type2...): type0 => {  
    instructions;  
}
```

Une fonction anonyme peut être affectée comme une variable. Celle-ci devient donc une fonction.

4. Les classes

4.1. Visibilités

- `public` : Peut être appelé en dehors de la classe
- `private` : Ne peut être appelé qu'au sein de la classe (pour des valeurs, il est préférable de créer des getter et des setter pour accéder à la valeur depuis l'extérieur)
- `protected` : Ne peut être appelé qu'au sein de la classe et dans les classes héritées

4.2. Les classes de base

4.2.1. Créer une classe

```
public class MaClasse {  
  
    visibilite variable1: type1;  
    visibilite variable2: type2;  
  
    static variableStatique3: type3 = valeur3;  
  
    constructor(mVariable1: type1, mVariable2: type2...) {  
        this.variable1 = mVariable1;  
        this.variable2 = mVariable2;  
    }  
}
```

Note : Il est possible de surcharger le constructeur et de créer une variable statique qui reste la même valeur pour tous les objets.

4.2.2. Définir la classe dans un objet

```
var monObjet: MaClasse = new MaClasse(valeur1, valeur2...);
```

4.2.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

```
toString(): string {  
    return "message";  
}
```

4.2.4. Créer une méthode (dans une classe)

```
visibilite maMethode(variable1: type1, variable2: type2...): type0 {  
    instructions;  
}
```

4.2.5. Créer une méthode statique (dans une classe)

```
visibilite static maMethodeStatique(variable1: type1, variable2:
type2...): type0 {
    instructions;
}
```

4.2.6. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable dans une classe

```
this.maMethode(valeur1, valeur2...)
this.variable1 = valeur1;
```

4.2.7. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable en dehors d'une classe

```
monObjet.maMethode(valeur1, valeur2...)
monObjet.variable1 = valeur1;
```

Il est préférable d'utiliser des méthodes (appelées getter et setter) pour modifier les variables non statiques

4.2.8. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe

```
MaClasse.maMethodeStatique(valeur1, valeur2...)
MaClasse.variableStatique3 = valeur3;
```

4.2.9. Créer un getter et un setter

```
get variable1(): type {
    return this.#variable1;
}
```



```
set variable1(value: type) {  
    this.#variable1 = value;  
}
```

Les getters et les setters peuvent avoir le même nom. Il est cependant préférable de nommer les variables avec au début un #. Les getters et les setters sont par suite appelés comme-ci c'était des variables.

4.3. Les classes internes

Les classes internes sont des classes présentes dans une autre classe.

```
public class MaClasse {  
  
    public class MaClasseInterne {  
        ...  
    }  
  
}
```

Il est préférable de mettre la classe interne interne en privée ou en protégée. Si la classe interne est publique, elle est accessible avec `MaClasse.MaClasseInterne`, mais dans ce cas, il est déconseillé d'utiliser une classe interne.

4.4. L'héritage

L'héritage permet à des classes filles de reprendre les mêmes caractéristiques que leur classe mère, et d'ajouter des nouveaux attributs et/ou méthodes qui leur sont propres.

Il est possible de faire un dépassement d'une méthode de la classe mère, c'est-à-dire de créer une méthode du même nom qu'une méthode de la classe mère pour la remplacer.

Une classe ne peut hériter que d'une seule classe.

4.4.1. Créer une classe héritée d'une autre classe

```
class MaClasseHeritee extends MaClasse {  
  
    visibilite variable4: type4;  
    visibilite variable5: type5;  
  
    static variableStatique6: type6 = valeur6;  
  
    constructor(mVariable1: type1, mVariable2: type2..., mVariable4:  
type4, mVariable5: type5...) {  
        super(mVariable1, mVariable2...);  
        this.variable4 = mVariable4;  
        this.variable5 = mVariable5;  
    }  
}
```

La méthode `super()` permet d'appeler le constructeur de la classe initiale.

4.4.2. Définir la classe dans un objet

```
monObjet: MaClasse = new MaClasseHeritee(valeur1, valeur2..., valeur4,  
valeur5...);
```

ou

```
monObjet: MaClasseHeritee = new MaClasseHeritee(valeur1, valeur2...,  
valeur4, valeur5...);
```

4.4.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

```
toString(): string {  
    return (super.toString() + "message");  
}
```

4.5. Les classes abstraites

Les classes abstraites sont des classes qui ne peuvent pas être instanciées et qui sont utilisées pour définir une structure de classe de base pour les classes dérivées.

Les classes abstraites sont déclarées avec le mot-clé `abstract`. Elles peuvent contenir des méthodes abstraites, qui sont des méthodes déclarées sans corps. Les méthodes abstraites sont utilisées pour définir une interface pour les classes dérivées.

Les classes dérivées doivent implémenter toutes les méthodes abstraites de la classe abstraite parente.

4.5.1. Créer une classe abstraite

```
public abstract class MaClasseAbstraite {  
  
    visibilite variable1: type1;  
    visibilite variable2: type2;  
  
    static variableStatique3: type3 = valeur3;  
  
    constructor(mVariable1: type1, mVariable2: type2...) {  
        this.variable1 = mVariable1;  
        this.variable2 = mVariable2;  
    }  
}
```

4.5.2. Créer une méthode abstraite

```
visibilite abstract maMethodeAbstraite(variable1: type1, variable2:  
type2...): type0;
```

Toutes les méthodes qui ne possèdent pas de mot-clé `abstract` ne doivent pas être obligatoirement implémentées par les classes filles.

4.5.3. Créer une classe héritant d'une classe abstraite

```
public class MaClasseHeritee extends MaClasseAbstraite {  
  
    visibilite variable4: type4;  
    visibilite variable5: type5;  
  
    static variableStatique6: type6 = valeur6;  
  
    constructor(mVariable1: type1, mVariable2: type2..., mVariable4:  
type4, mVariable5: type5...) {  
        super(mVariable1, mVariable2...);  
        this.variable4 = mVariable4;  
        this.variable5 = mVariable5;  
    }  
}
```

4.5.4. Implémenter une méthode abstraite (dans la classe fille)

```
visibilite maMethodeAbstraite(variable1: type1, variable2: type2...):  
type0 {  
    instructions;  
}
```

4.5.5. Définir la classe dans un objet

```
monObjet: MaClasseAbstraite = new MaClasseHeritee(valeur1, valeur2...,  
valeur4, valeur5...);
```

ou

```
monObjet: MaClasseHeritee = new MaClasseHeritee(valeur1, valeur2...,  
valeur4, valeur5...);
```

Il est possible de définir directement la classe abstraite dans un objet, sans faire d'héritage. Voir les classes anonymes.

4.6. Les interfaces

Les interfaces sont des classes abstraites qui permettent de définir un ensemble de méthodes sans les implémenter. Elles sont très utiles pour réduire la dépendance entre classes. Les classes peuvent implémenter plusieurs interfaces et doivent fournir une implémentation pour chacune des méthodes annoncées.

4.6.1. Créer une interface

```
public interface MonInterface {  
  
    variable1: type1;  
    variable2: type2;  
  
    ...  
}
```

4.6.2. Définir une interface dans un objet

```
monObjet: MonInterface = {variable1: valeur1, variable2: valeur2...};
```

4.6.3. Créer une méthode abstraite

```
visibilite maMethodeAbstraite(variable1: type1, variable2: type2...):  
type0;
```

Toutes les méthodes n'ont pas besoin de mot-clé `abstract` car toutes les méthodes d'une interface doivent être obligatoirement implémentées par les classes filles.

4.6.4. Créer une classe implémentant une ou plusieurs interfaces

```
public class MaClasseHeritee extends MaClasseAbstraite implements
MonInterface1, MonInterface2... {

    visibilite variable4: type4;
    visibilite variable5: type5;

    static variableStatique6: type6 = valeur6;

    constructor(mVariable1: type1, mVariable2: type2..., mVariable4:
type4, mVariable5: type5...) {
        super(mVariable1, mVariable2...);
        this.variable4 = mVariable4;
        this.variable5 = mVariable5;
    }
}
```

4.6.5. Implémenter une méthode abstraite (dans la classe fille)

```
visibilite maMethodeAbstraite(variable1: type1, variable2: type2...):
type0 {
    instructions;
}
```

4.6.6. Définir la classe dans un objet

```
MonInterface monObjet = new MaClasseAbstraite(valeur1, valeur2...,
valeur4, valeur5...);
```

ou

```
MaClasseAbstraite monObjet = new MaClasseAbstraite(valeur1,
valeur2...);
```

ou

```
MaClasse monObjet = new MaClasseAbstraite(valeur1, valeur2...,
valeur4, valeur5...);
```

Il est possible de définir directement l'interface dans un objet, sans faire d'implémentation (uniquement s'il n'y a qu'une seule interface). Voir les classes anonymes.

4.7. Les classes anonymes

Les classes anonymes sont généralement des classes abstraites ou des interfaces définies directement dans un objet sans avoir à passer par une classe héritant d'une classe abstraite (ça peut également être une classe simple, mais c'est déconseillé).

Chaque classe anonyme doit implémenter obligatoirement toutes les méthodes abstraites entre { }. Elle peut également contenir des méthodes non obligatoires à implémenter. Cela peut être utile si le code ne peut être exécuté que par un seul objet.

4.7.1. Manière simple

```
MaClasseAbstraite monObjet = new MaClasseAbstraite(valeur1,
valeur2...) {
    ...
};
```

4.7.2. Expression lambda (ne fonctionne que si la classe abstraite ne contient qu'une seule méthode abstraite)

```
MaClasseAbstraite monObjet = (valeur1, valeur2...) => {
    instructions;
};
```

Pas besoin d'entrer le nom de la méthode abstraite. Seules les paramètres sont renseignés.

4.8. Les collections

Les collections sont des classes qui peuvent avoir un ou plusieurs types de données personnalisables.

4.8.1. Créer une collection

```
public class MaCollection<T1, T2...> {  
  
    visibilite type1 variable1: type1;  
    visibilite type2 variable2: type2;  
    visibilite variable3: T1;  
    visibilite variable4: T2;  
  
    public static type3 variableStatique3 = valeur;  
  
    visibilite MaClasse(mVariable1: type1, mVariable2: type2...,  
mVariable3: T1, mVariable4: T2...) {  
        this.variable1 = mVariable1;  
        this.variable2 = mVariable2;  
        this.variable3 = mVariable3;  
        this.variable4 = mVariable4;  
    }  
}
```

4.8.2. Définir la collection dans un objet

```
MaCollection<TypeObjet1, TypeObjet2...> monObjet = new  
MaCollection<>(valeur1, valeur2..., valeur4, valeur5...);
```


5. Le stockage local

5.1. Cookies

Cookies permet de stocker des variables localement sur l'ordinateur, accessible sur toutes les fenêtres, avec ou sans date d'expiration et avec communication avec un serveur. Sa capacité maximale est de 4 Ko par variables. Les cookies sont créés depuis le serveur web.

Instruction	Description
<code>Document.cookie</code>	Récupérer la liste des cookies

5.2. Session Storage

Session Storage permet de stocker des variables localement sur l'ordinateur, expirant à la fermeture de l'onglet et sans communication avec un serveur. Sa capacité maximale est de 5 Mo par variables.

Instruction	Description
<code>sessionStorage.setItem("clé", "valeur");</code>	Créer une variable avec une valeur affectée
<code>variable = sessionStorage.getItem("clé");</code>	Récupérer le contenu d'une variable
<code>sessionStorage.removeItem("clé");</code>	Supprimer une variable
<code>sessionStorage.clear();</code>	Supprimer toutes les variables

5.3. Local Storage

Local Storage permet de stocker des variables localement sur l'ordinateur, accessible sur toutes les fenêtres, sans date d'expiration et sans communication avec un serveur. Sa capacité maximale est de 10 Mo par variables.

Instruction	Description
-------------	-------------

<code>window.localStorage</code>	Récupérer toutes les variables stockées dans l'ordinateur localement
<code>localStorage.setItem("clé", "valeur");</code>	Créer une variable avec une valeur affectée
<code>variable = localStorage.getItem("clé");</code>	Récupérer le contenu d'une variable
<code>localStorage.removeItem("clé");</code>	Supprimer une variable
<code>localStorage.clear();</code>	Supprimer toutes les variables

6. Les instructions

6.1. Instructions de bases

Instruction	Description
<code>console.log("texte");</code>	Affiche un texte dans la console avec un retour à la ligne
<code>console.log(variable);</code> <code>console.log("Valeur : " + variable);</code>	Affiche une variable dans la console
<code>alert(message);</code>	Affiche un message dans une boîte de dialogue
<code>variable = prompt("Entrer une valeur :");</code>	Demande une valeur dans une boîte de dialogue avec le retour dans une variable
<code>variable = confirm("Voulez-vous confirmer ?");</code>	Demande une confirmation dans une boîte de dialogue, renvoyée dans la variable sous forme d'un booléen

6.2. Math

Instruction	Description
<code>Math.random();</code>	Créer un nombre aléatoire entre 0 compris et 1 exclu
<code>Math.floor(Math.random() * n);</code>	Créer un nombre entier aléatoire entre 0 et n compris (si fonction importé ci-dessus)

6.3. JSON

Instruction	Description
<code>monObjet = JSON.parse(objJSON);</code>	Convertir une chaîne JSON en objet
<code>objJSON = JSON.stringify(monObjet);</code>	Convertir un objet en chaîne JSON

6.4. Exécution asynchrone

Instruction	Description
<code>setTimeout(<i>maFonction</i>, <i>temps</i>);</code>	Exécuter en asynchrone une fonction après un certain nombre de millisecondes paramétré
<code>let <i>idInterval</i> = setInterval(<i>maFonction</i>, <i>temps</i>);</code>	Exécuter en asynchrone une fonction de manière répétitive
<code>clearInterval(<i>idInterval</i>);</code>	Arrêter la répétition de la fonction

7. Les importations et exportations

Les importations et exportations ne peuvent être effectuées uniquement si l'appel du script dans le HTML soit sous la forme : `<script type="module" src="script.js"></script>`

7.1. Exporter une classe (fonctionne aussi avec les fonctions et autres)

```
export { MaClasse1, MaClasse2... };  
ou  
export default MaClasse1;
```

Il est possible de mettre le mot-clé `export` devant `class` pour exporter une classe.

7.2. Importer une classe exportée dans un autre fichier

```
import { MaClasse1, MaClasse2... } from "../fichier";  
ou (si c'est une exportation par défaut) :  
import MaClasse1 from "../fichier";
```

Si c'est une exportation par défaut, il est possible de renommer l'objet.