Document : Javascript ES

Fait le : 19/09/2017

Ressources : Youtube

Acteur : Youssef Zaghir

Table des matières

[**Différence entre GruntJS et GulpJS** 2](#_Toc489911540)

[**GulpJS Api :** 3](#_Toc489911541)

[**gulp.task** 3](#_Toc489911542)

[**gulp.src** 4](#_Toc489911543)

[**gulp.dest** 4](#_Toc489911544)

[**gulp.watch** 5](#_Toc489911545)

[**Ce que vous avez besoin pour utiliser GulJS** 5](#_Toc489911546)

**Différence entre const , let , var**

**function** *setWidth*(){  
 **var** width = 100 ;  
 **console**.log(width);  
}  
*setWidth*();  
  
**var *age*** = 100 ;  
**if**(***age***>12){  
 **var *dogYears*** = ***age*** \* 7 ;  
 *// console.log('you are ${docYears} dog years old!');*

**console**.log(**'you are %s dog years old!'** , dogYears);  
}

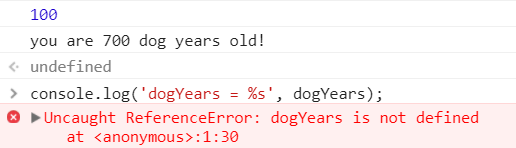


* **Scope** ou la portée de la  variable width est limité entre les deux accolades de la fonction, la variable ***dogYears*** est defini dans un bloque de test ***if***  si le test est ok la variable sera créer et son scope sera dans toute l’application, or dans le programme on veut seulement utiliser la variable dans le bloc de test d’où l’utilisation de ***let*** et ***const*** dans le blocScope



**function** *setWidth*(){  
 **var** width = 100 ;  
 **console**.log(width);  
}  
*setWidth*();  
  
**var *age*** = 100 ;  
**if**(***age***>12){  
 **let *dogYears*** = ***age*** \* 7 ; *// meme resultat avec const*  
 *// console.log('you are ${docYears} dog years old!');*

**console**.log(**'you are %s dog years old!'** , dogYears);  
}



* **La portée de la variable *dogYears*** doc est limité dans le bloc test if (blocScope) quand on déclare la fonction avec l’option let ou const

**const *key*** = **'123456'**;  
**let *points*** = 50;  
**var *winner*** = **false** ;  
  
**if**(***points*** > 40) {  
 **console**.log(**'it ran1'**);  
 **var *winner*** = **true** ;  
}

**const *key*** = **'123456'**;  
**let *points*** = 50;  
**let *winner*** = **false** ;  
  
**if**(***points*** > 40) {  
 **console**.log(**'it ran1'**);  
 **let *winner*** = **true** ;  
}

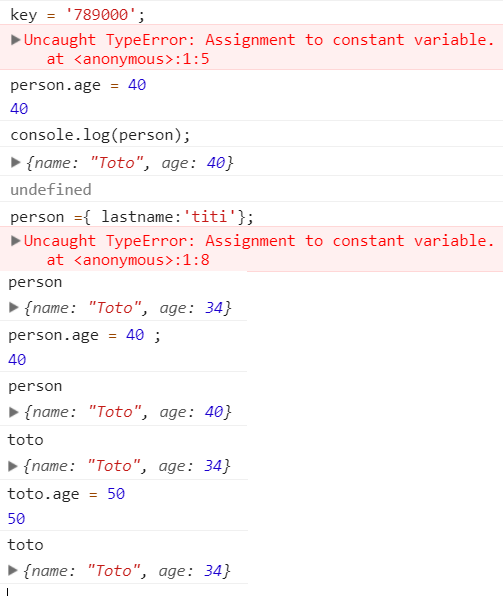
**Bloc scope**

* La variable est défini deux fois mais avec let option, la deuxième délation est propre au bloc test if , avec une nouvelle référence , donc elle n’impacte pas la première variable déclarée
* La déclaration de la variable winner avec l’option var change le scope de l’application en général , elle ne créer pas une nouvelle référence , elle change seulement sa valeur

**let**

**const *key*** = **'123456'**;  
**const *person*** ={  
 **name**:**'Toto'**,  
 **age**:34  
}  
**const *toto*** = Object.freeze(***person***);

* On ne peut pas changé la valeur d’une const
* On ne peut pas redéfinie un objet s’il est de type const mais en peu changé ses paramètres
* Il est possible de bloquer tout changement sur un objet par l’option Object.freeze(***obj***);



**const**





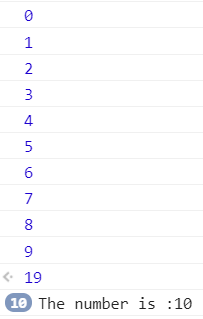
**Fonction autonome en ES6**

(**function**(){  
 **const** name = **'toto'**;  
 **console**.log(***name***);  
})()

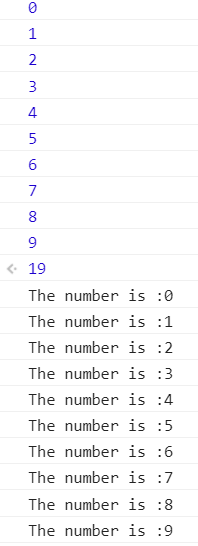
{  
 **const *name*** = **'toto'**;  
 **console**.log(***name***);  
}

ES6

**for**(**var *i*** = 0 ; ***i*** < 10 ; ***i***++){  
 **console**.log(***i***);  
 setTimeout(**function**(){  
 **console**.log(**'The number is :'**+***i***);  
 }, 1000);  
}



**for**(**let *i*** = 0 ; ***i*** < 10 ; ***i***++){  
 **console**.log(***i***);  
 setTimeout(**function**(){  
 **console**.log(**'The number is :'**+***i***);  
 }, 1000);  
}



* Règles de portée Les variables déclarées avec let appartiennent à la portée du bloc dans lequel elles sont définies et indirectement aux portées des blocs de ce bloc. D'une certaine façon let fonctionne comme var, la seule différence dans cette analogie est que let fonctionne avec les portées de bloc et var avec les portées des fonctions

Le mot-clé let permet de lier des variables localement dans la portée des boucles for. Contrairement au mot-clé var qui lui rend les variables visibles depuis l'ensemble de la fonction qui contient la boucle.

**Zone morte temporaire (*Temporal Dead Zone* / TDZ)  et les erreurs liées à let**

Lorsqu'on redéclare une même variable au sein d'une même portée de bloc, cela entraîne une exception [**SyntaxError**](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/SyntaxError).

**if** (x) {  
 **let *toto***;  
 **let *toto***; *// SyntaxError*}

Si on redéclare une variable pour le corps d'une fonction, cela ne pose aucun problème :

**function** *faire\_quelque\_chose*() {  
 **let** toto;  
 **let** toto; *// Cela fonctionne.*}

Avec ECMAScript 2015 (ES6), let **remontera** la déclaration variable au début de la portée (au début du bloc) mais pas l'initialisation. Si on fait référence à une variable dans un bloc avant la déclaration de celle-ci avec let, cela entraînera une exception [**ReferenceError**](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/ReferenceError). En effet, la variable est placée dans une « zone morte temporaire » entre le début du bloc et le moment où la déclaration est traitée. Autrement dit, la déclaration est bien remontée mais la variable ne peut pas être utilisée tant que l'affectation (qui n'est pas remontée) n'a pas été effectuée.

**function** *faire\_quelque\_chose*() {  
 **console**.log(truc); *// undefined* **console**.log(toto); *// ReferenceError* **let** toto = 2;  
 **var** truc = 1;  
}

Il est possible d'obtenir des erreurs au sein de l'instruction **Instructions/switch**. En effet, il y a un seul bloc implicite pour cette instruction.

**switch** (x) {  
 **case** 0:  
 **let *toto***;  
 **break**;  
  
 **case** 1:  
 **let *toto***; *// SyntaxError for redeclaration.* **break**;  
}

Par contre, si on ajoute une instruction de bloc dans la clause case, cela créera une nouvelle portée et empêchera l'erreur :

let x = 1;

switch(x) {

case 0: {

let toto;

break;

}

case 1: {

let toto;

break;

}

}

**Autre exemple lié à la zone morte temporaire et aux portées lexicales**

Dans l'exemple qui suit, dans l'expression toto + 55, l'identifiant toto fait référence à la variable du bloc courant et non à celle qui est déclarée au dessus et qui a la valeur 33. Dans l'instruction let toto = (toto + 55); l'instruction est bien remontée mais l'endroit où on utilise toto (dans le fragment (toto + 55)) est toujours dans la zone morte temporaire car toto n'a pas encore été affecté.

**function** *test*(){  
 **var** toto = 33;  
 **if** (**true**) {  
 **let** toto = (toto + 55); *// ReferenceError: can't access lexical declaration `toto` before initialization* }  
}  
*test*();

Si on utilise let avec un nom de variable qui est le même que celui de l'argument passé à la fonction, on aura une erreur due à la confusion des portées :

**function** *go*(n) {  
 **for** (**let** n of n.**a**){ *// ReferenceError: can't access lexical declaration `n' before initialization* **console**.log(n);  
 }  
}  
*go*({**a**:[1, 2, 3]});

**Bonne pratique :**

* Utiliser **const** par défaut
* Utiliser **let** seulement si vous avez besoin de relier ou attacher vos objets (faire un binding), pour des variables local dans une portée réduite (small scope)
* Il ne faut pas utiliser **var** dans ES6 , ou l’utilisée seulement pour des variables qui seront utilisées dans l’ensemble de l’applications (scope , global)

**Array.Map()**

La méthode map() crée un nouveau tableau composé des images des éléments du tableau courant par une fonction donnée en argument.

**Syntaxe**

*var nouveauTableau = arr*.map(*callback*[, *thisArg*])

callback

La fonction qui est utilisée pour créer un élément du nouveau tableau. Elle utilise trois arguments :

valeur Courante

La valeur de l'élément du tableau à traiter.

index

L'index de l'élément qui est traité par la fonction.

tableau

Le tableau sur lequel on a appelé la méthode map.

thisArg Facultatif

La valeur à utiliser pour this lors de l'exécution de callback. La valeur par défaut est l'objet global d e l'environnement (Window pour un navigateur).

Valeur de retour

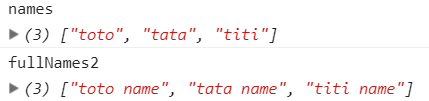
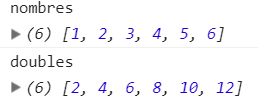
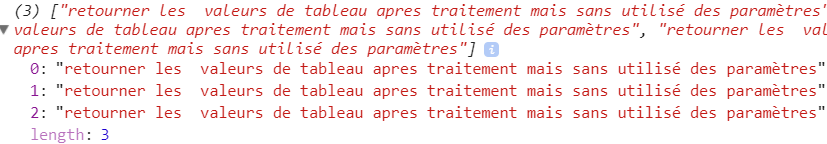
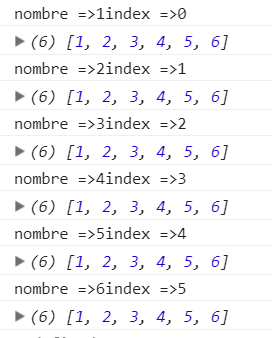
Un nouveau tableau composé des images de la fonction de rappel.

**const *names*** =[**'toto'** , **'tata'** , **'titi'**];  
**const *fullNames*** = ***names***.map(**function**(name){  
 **return** name+**' name'** ;  
});  
*// explicite return Es6***const *fullNames2*** = ***names***.map(**name** => **name**+**' name'** );

**const *fullNames3*** = ***names***.map(()=>**'retourner les valeurs de tableau apres traitement mais sans utilisé des paramètres'**)

**const *nombres*** = [1,2,3,4,5,6];  
**const *doubles*** = ***nombres***.map((nombre)=>{  
 **return** nombre \* 2 ;  
});

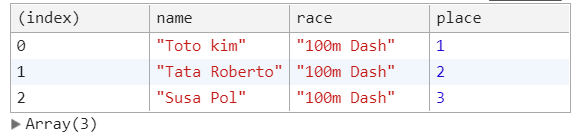
**const *nombres*** = [1,2,3,4,5,6];   
**const *doubles*** = ***nombres***.map((nombre , ***index*** , tableau)=>{  
 **console**.log(**'nombre =>'**+nombre + **'index =>'**+***index*** );  
 **console**.log(tableau) ;  
 **return** nombre \* 2 ;  
});  
  
*// function anonyme en Es6***const *sayMyName*** = (**name**) => {  
 alert(**'Hello '** +**name** +**'!'**);  
}



**Créer un nouvel objet à partir d’un tableau**

On itère le tableau par la fonction map() et on retourne les objets setter la fonction callback de map()

**const *race*** = **'100m Dash'**;  
**const *winners*** =[**'Toto kim'** , **'Tata Roberto'** , **'Susa Pol'**] ;  
**const *win*** = ***winners***.map((***winner*** , ***index***) =>  
 (  
 {  
 **name**:***winner*** ,  
 **race** :***race*** ,  
 **place**: ***index*** + 1}  
 )  
 );  
**console**.table(***win***); // pour afficher le tableau



**filter()**

La méthode **filter()** crée et retourne un nouveau tableau contenant tous les éléments du tableau d'origine qui remplissent une condition déterminée par la fonction callback.

**Syntaxe**

arr.filter(callback);

var nouveauTableau = arr.filter(callback, thisArg);

callback

La fonction de test (ou prédicat) à appliquer à chaque élément du tableau. Cette fonction est appelée avec les arguments suivants :

element

L'élément à traiter

index

Son indice.

array

Le tableau complet

Cette fonction renvoie true — ou une valeur équivalente — si l'élément doit être conservé pour le tableau résultat et false dans le cas contraire.

thisArg Facultatif

Objet à utiliser en tant que this quand la fonction callback est exécutée.

Valeur de retour

Un nouveau tableau contenant les éléments qui respectent la condition du filtre.

**const *ages*** = [70,75,80,10,15,20,25,30,35,40,45,50,55,60,65];  
**const *old*** = ***ages***.filter(***age*** => ***age***>=60) ;  
**console**.log(***old***) ;

**const *fruits*** = [**'pomme'**, **'banane'**, **'raisin'**, **'mangue'**];  
**const *filtreTexte*** = (requete) => {  
 **return *fruits***.filter((el) =>  
 el.toLowerCase().indexOf(requete.toLowerCase()) > -1  
);  
}  
**console**.log(***filtreTexte***(**'an'**)); *// ['banane', 'mangue'];***console**.log(***filtreTexte***(**'m'**)); *// ['pomme', 'mangue'];*

**var *arr*** = [ { **id**: 15 }, { **id**: -1 }, { **id**: 0 }, { **id**: 3 },  
 { **id**: 12.2 },{ },{ **id**: **null** },{ **id**: **NaN** },  
 { **id**: **'undefined'** }  
];  
**var *elementsInvalides*** = 0;  
**function** *filtrerParID*(obj) {  
 *// Si c'est un nombre* **if** (obj.**id** !== **undefined** && **typeof**(obj.**id**) === **'number'** && !isNaN(obj.**id**)) {  
 **return true**;  
 } **else** {  
 ***elementsInvalides***++;  
 **return false**;  
 }  
}  
**var *arrByID*** = ***arr***.filter(*filtrerParID*);  
**console**.log(**'Tableau filtré\n'**, ***arrByID***);  
*// Le tableau filtré est :  
// [{ id: 15 }, { id: -1 }, { id: 0 }, { id: 3 }, { id: 12.2 }]***console**.log(**'Nombre d\'éléments invalides = '**, ***elementsInvalides***);  
*// Nombre d'éléments invalides 4*

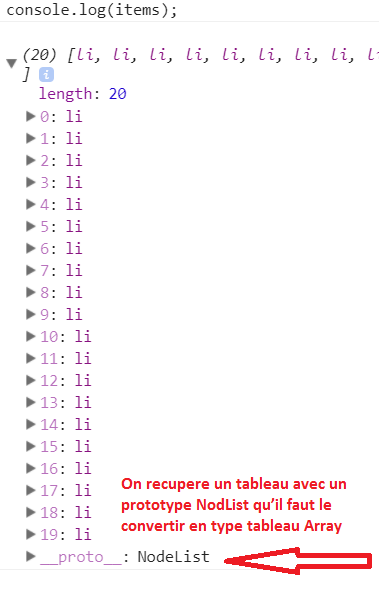


**Paramètres par défaut d’une fonction**

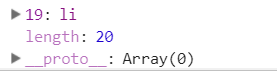
**function** *calculateBill*(total , tax , tip){  
 **return** total + (total\*tax) + (total\*tip) ;  
}  
**console**.log(*calculateBill*(10 , 0.2 , 0.1)); *// === 13***console**.log(*calculateBill*(10); *//=== NaN car tax et tip n ont pas de valeur***function** *calculateBill2*(total , tax , tip){  
 **if**(tax === **undefined** ){  
 tax = 0.2;  
 }  
 **if**(tip === **undefined**){  
 tip = 0.1 ;  
 }  
 **return** total + (total\*tax) + (total\*tip) ;  
}  
  
**function** *calculateBill2*(total , tax , tip){  
 tax = tax || 0.2 ; *// plus simple que la test dans* tip = tip || 0.1 ; *// la fonction calculateBill2* **return** total + (total\*tax) + (total\*tip) ;  
}  
*// en defini les params par defaut dans la signature de la methode dans Es6***function** *calculateBill3*(total , tax = 0.2 , tip = 0.1){  
 **return** total + (total\*tax) + (total\*tip) ;  
}

**Cas d’utilisation de map() ,filter(), querySelector()  pour calculer une durée dans page html:**

<!DOCTYPE html>  
<**html** lang=**"en"**>  
<**head**>  
 <**meta** charset=**"UTF-8"**>  
 <**title**>Title</**title**>  
</**head**>  
<**body**>  
<**ul**>  
 <**li** data-time=**"5:30"**>Flexbox video</**li**>  
 <**li** data-time=**"5:21"**>Flexbox video</**li**>  
 <**li** data-time=**"7:09"**>Redux video</**li**>  
 <**li** data-time=**"6:50"**>Flexbox video</**li**>  
 <**li** data-time=**"11:50"**>Redux video</**li**>  
 <**li** data-time=**"2:50"**>Flexbox video</**li**>  
 <**li** data-time=**"4:50"**>Redux video</**li**>  
 <**li** data-time=**"3:20"**>Flexbox video</**li**>  
 <**li** data-time=**"5:40"**>Redux video</**li**>  
 <**li** data-time=**"8:50"**>Flexbox video</**li**>  
 <**li** data-time=**"5:50"**>Flexbox video</**li**>  
 <**li** data-time=**"2:50"**>Flexbox video</**li**>  
 <**li** data-time=**"1:40"**>Redux video</**li**>  
 <**li** data-time=**"0:24"**>Flexbox video</**li**>  
 <**li** data-time=**"4:50"**>Redux video</**li**>  
 <**li** data-time=**"3:50"**>Flexbox video</**li**>  
 <**li** data-time=**"2:40"**>Redux video</**li**>  
 <**li** data-time=**"8:30"**>Redux video</**li**>  
 <**li** data-time=**"1:20"**>Flexbox video</**li**>  
 <**li** data-time=**"9:10"**>Redux video</**li**>  
</**ul**>  
<**script**>  
 **const *items*** = **document**.querySelectorAll(**'[data-time]'**);  
</**script**>  
</**body**>  
</**html**>



<**script**>  
 *// selectionner tous les elements dans la page  
 // et les convertir en type table* **const *items*** = Array.from(**document**.querySelectorAll(**'[data-time]'**));  
 **console**.log(***items***);  
</**script**>



<**script**>  
 *// selectionner tous les elements dans la page  
 // et les convertir en type table* **const *items*** = Array.from(**document**.querySelectorAll(**'[data-time]'**));  
 *// un filtre pour recuperer les elements qui contiennt le mot ''flexbox* **const *filtered*** = ***items*** .filter( **item** => **item**.**textContent**.**includes**(**'Flexbox'**))  
 *//recuperer la liste des temps en String* .map(**item** =>**item**.**dataset**.time) *// data-time  
 // convertir vers un tableau de seconds* .map(**timeCode** => {  
 **const parts** = **timeCode**.split(**':'**) *// ['5:50'] ==> ['5','50']* .map(**part** => parseFloat(**part**)); *// convetir String to float [5,50]* **return** (parts[0] \* 60) + parts[1];  
 })  
 .reduce((runningTotal , second) => runningTotal + second ,0);  
 **console**.log(***filtered***+**'s'**);  
</**script**>

**Template pour les chaines de caractères (concat …** )

Nouvelle forme pour la concaténation des données dans une string

**const *name*** = **'youssef'**;  
**const *age*** = 34;  
**const *sentenceOldVersion*** = **'Monsieur '**+***name***+**' a '**+***age***+**' !'**;  
*// ici on utilise l'acent pas du simple ou double cotes c’est plus propre***const *sentenceNewVersion*** = **`Monsieur** ${***name***} **a** ${***age***+20} **!`**;  
**console**.log(***sentenceOldVersion***) ;  
**console**.log(***sentenceNewVersion***) ;

Eviter les \ pour écrire une String sur plusieurs lignes

**const *person*** ={  
 **name**:**'youssef'**,  
 **job**:**'Developper'**,  
 **city**:**'Gagny'**,  
 **bio** :**'Wes is really good guy to teatch web development!'**}  
*// utilisation de l'accent permet de ne pas mètre de le \   
// sur chaque retour à la ligne***const *marckup*** =**`<div class="person">  
 <h2>** ${***person***.**name**} **<span class="job">**${***person***.**job**} **</span>  
 </h2>  
 <p class="location" >**${***person***.**city**}**</p>  
 <p class="bio">**${***person***.**bio**}**</p>  
 </div>  
 `**;  
**document**.**body**.**innerHTML**= ***marckup***;  
**console**.log(***marckup***)

Utilisation des boucles à l’intérieur d’une String

**const *chats*** = [  
 {**name**: **'Migalo'** , **age**: 3},  
 {**name**: **'Miouz'** , **age**: 10},  
 {**name**: **'Mouch'** ,**age**: 1}  
];  
*// dans la fonction map on change les {} par ``   
// c'est le changement à faire dans la template String***const *markup*** = **`  
 <ul class="chat">** ${***chats***.map(**chat** =>**`  
 <li>** ${**chat**.**name**} **a** ${**chat**.**age**} **</li>  
 `**).join(**''**)} *// join(‘’) pour suprimer les virgules à la fin* **</ul>  
 `**;  
**document**.**body**.**innerHTML** = ***markup*** ;

Conditionner l’affection d’une String

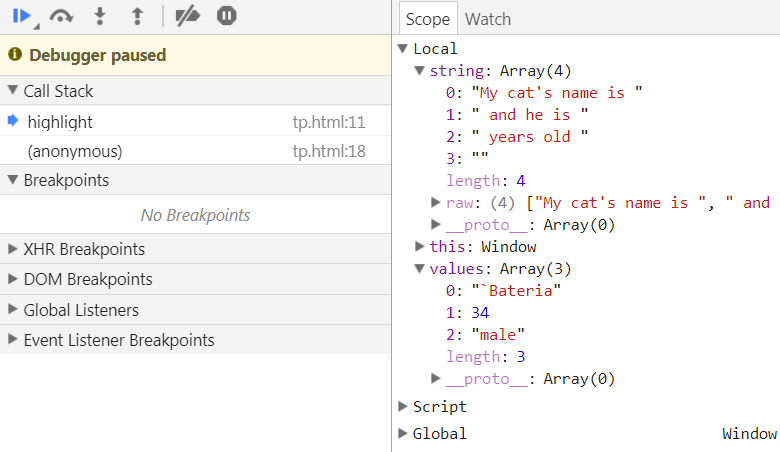
**const *song*** = {  
 **name**: **'Aligator'**,  
 **artist**:**'Armin Vann Burren'**}  
*// ici on test si la propriété featuring existe dans l'objet song  
// si c'est oui on l'affiche si non on insert un espace vide***const *markup*** =**`  
 <div class="song">  
 <p>** ${***song***.**name**} **-** ${***song***.**artist**}  
 ${***song***.featuring ? **`(Featuring** ${***song***.featuring}**)`** : **''**} **</p>  
 </div>  
`**;  
**document**.**body**.**innerHTML** = ***markup*** ;

Utilisation d’une fonction dans le chargement d’une variable

**const *beer*** = {  
 **name**: **'Belgian Wit'**,  
 **brewery**: **'Steam Whistle Brewery'**,  
 **keywords**: [**'pale'**, **'cloudy'**, **'spiced'**, **'crisp'**]  
};  
**function** *renderKeywords*(keywords){  
 **return `  
 <ul>** ${keywords.map(**keyword** => **`<li>**${**keyword**}**</li>`**).join(**''**)} **</ul>  
 `**;  
}  
**const *markup*** =**`  
 <div class="beer">  
 <h2>** ${***beer***.**name**}  **</h2>  
 <p class="beer">** ${***beer***.**brewery**}  
 ${*renderKeywords*(***beer***.**keywords**)} **</p>  
 </div>  
`**;  
**document**.**body**.**innerHTML** = ***markup*** ;

**Récupéré les différentes partie d’une string passé en paramètres d’une fonction**

**function** *highlight*(string , ...values) {  
 **debugger**;  
}  
  
**const *name*** = **'`Bacteria'**;  
**const *age*** = 34 ;  
**const *gender*** = **'male'** ;  
  
**const *sentence*** = *highlight***`My cat's name is** ${name} **and he is** ${age} **years old** ${gender}**`**;  
**console**.log(***sentence***);



<**style**>  
 .**h1**{  
 **background**:**#fff600**;  
 }  
</**style**>

Ajouter le style dans le head

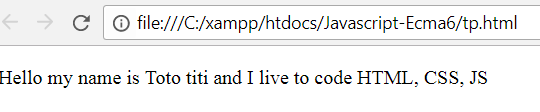
**function** *highlight*(strings , ...values) {  
 **let *str*** = **''**;  
 strings.forEach((string , ***i***)=>{  
*// str += string +(values[i] || '' );* **str** +=**`**${string} **<span contenteditable class="h1">**${values[i] || **''** } **</span>`**;  
 });  
 **return** str ;  
  
 }  
  
 **const *name*** = **'`Bateria'**;  
 **const *age*** = 34 ;  
  
 **const *sentence*** = *highlight***`My cat's name is** ${name} **and he is** ${age} **years old`**;  
 **console**.log(***sentence***);  
 **document**.**body**.**innerHTML** = ***sentence***;

On vérifie si valeurs de i existe si non on ajoute un espace vide

**Contenteditable** permet de changer les valeurs de i dans la page html

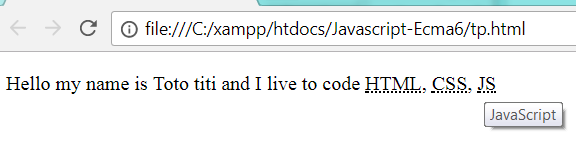
Création du dictionnaire avec les fonctions de String , exercice :

<**body**>  
<**div class="bio"**></**div**>  
<**script**>  
 **const *first*** = **'Toto'**;  
 **const *last*** = **'titi'**;  
 **const *sentence*** = **`Hello my name is** ${first} ${last} **and I live to code** ${**'HTML'**}**,** ${**'CSS'**}**,** ${**'JS'**}**`**;  
  
 **const *bio*** = **document**.querySelector(**'.bio'**);  
 **const *p*** = **document**.createElement(**'p'**);  
 ***p***.**innerHTML** = ***sentence*** ;  
 ***bio***.appendChild(***p***);  
</**script**>  
</**body**>



${**'JS'**} on passe un Constant avec une valeur = JS

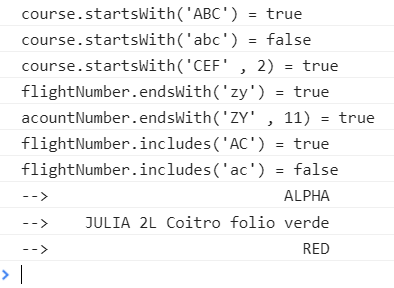
<**body**>  
<**div class="bio"**></**div**>  
<**script**>  
  
 **const *dict*** = {  
 **HTML**:**'Hyper Text Markup Language'**,  
 **CSS**:**'Cascading Style Sheets'**,  
 **JS**:**'JavaScript'** };  
  
 **function** *addAbbreviations* (strings , ...values){  
 **const *abbreviated*** = values.map((value)=>{  
 **if**(dict[value]){  
 **return `<abbr title="**${dict[value]}**">**${value}**</abbr>`** ;  
 }  
 **console**.log(value);  
 **return** value;  
 });  
  
 **return** strings.reduce((***sentence***, string , ***i***)=>{  
 **return `**${sentence}${string}${abbreviated[i] || **''**}**`**;  
 },**''**);  
 }  
  
 **const *first*** = **'Toto'**;  
 **const *last*** = **'titi'**;  
 **const *sentence*** = *addAbbreviations***`Hello my name is** ${first} ${last} **and I live to code** ${**'HTML'**}**,** ${**'CSS'**}**,** ${**'JS'**}**`**;  
  
 **const *bio*** = **document**.querySelector(**'.bio'**);  
 **const *p*** = **document**.createElement(**'p'**);  
 ***p***.**innerHTML** = ***sentence*** ;  
 ***bio***.appendChild(***p***);  
</**script**>  
  
</**body**>



En javascript il n y a pas d’objet , on peut accéder la propriété d’une objet comme un indice d’un tableau dict[value]

Nouvelle fonctions pour les String .starsWith() , .endsWith() , .includes() , .repeat() :

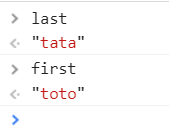
**const *course*** =**'ABCEFGH'**;  
**const *flightNumber*** = **'45-AC2017-zy'**;  
**const *acountNumber*** = **'958631247ZY003'**;  
  
**const *make*** =**'ALPHA'**;  
**const *model*** = **'JULIA 2L Coitro folio verde'**;  
**const *color*** = **'RED'**;  
  
**console**.log(**`course.startsWith('ABC') =** ${course.startsWith(**'ABC'**)}**`**);  
**console**.log(**`course.startsWith('abc') =** ${course.startsWith(**'abc'**)}**`**);  
*// tester la chaine apres les 2 premier caractere***console**.log(**`course.startsWith('CEF' , 2) =** ${course.startsWith(**'CEF'** , 2)}**`**);  
  
**console**.log(**`flightNumber.endsWith('zy') =** ${flightNumber.endsWith(**'zy'**)}**`**);  
*// recupert 11 charactere et test si la fin de la zone recupéré fini pas ZY***console**.log(**`acountNumber.endsWith('ZY' , 11) =** ${acountNumber.endsWith(**'ZY'** , 11)}**`**);  
  
**console**.log(**`flightNumber.includes('AC') =** ${flightNumber.includes(**'AC'**)}**`**);  
**console**.log(**`flightNumber.includes('ac') =** ${flightNumber.includes(**'ac'**)}**`**);  
  
**function** *leftPad*(str, length = 30){  
 **return `-->** ${**' '**.repeat(length - str.length)}${str}**`**;  
}  
  
**console**.log(*leftPad*(***make***));  
**console**.log(*leftPad*(***model***));  
**console**.log(*leftPad*(***color***));



**Affectation par décomposition (Destructuring)**

Affectation par décomposition d'objet

**const *person*** = {  
 **first**:**'toto'**,  
 **last**:**'tata'**,  
 **country**:**'France'**,  
 **city**:**'Paris'**,  
 **twitter**:**'@zy'**};  
*// affectation par décomposition de l'objet  
// il faut utiliser les même noms des propriétés de l'objet***const** { ***first*** ,***last*** ,***country***} = ***person*** ;  
*// si on veut changer le nom de variable et récupérer la valeur de propriété***const** { first:***f*** ,last:***l*** ,country:***c*** } = ***person*** ;  
  
 *// affecter les valeurs par défaut , dans le car ou l'objet n'a pas cette propriété  
 // ou la valeur n'est pas encore renseigné ici la propriété h n'existe pas* **const** { w:***width*** = 600 , h:***height***= 300 } = {**w**: 800} ;



Affectation par décomposition du tableau

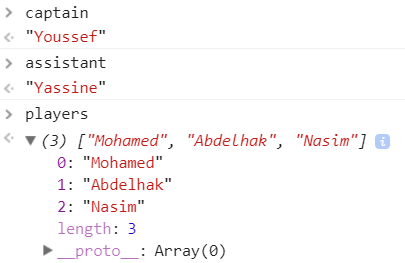
**const *toto***= [**"un"**, **"deux"**, **"trois"**];  
*// sans utiliser la décomposition***const *un*** = ***toto***[0];  
**const *deux*** = ***toto***[1];  
**const *trois*** = ***toto***[2];  
*// en utilisant la décomposition on utilise [ ] pour les tableau***const** [***un***, ***deux***, ***trois***] = ***toto***;

Ignorer un certain nombre de valeurs

**function** *f*() {  
 **return** [1, 2, 3];  
}  
  
**var** [***a***, , ***b***] = *f*();  
**console**.log(**"A vaut "** + ***a*** + **" B vaut "** + ***b***);  
*// en sortir A vaut 1 B vaut 3*

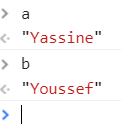
Récupération du reste des éléments d’un tableau

**const *team*** = [**'Youssef'**, **'Yassine'**, **'Mohamed'**,**'Abdelhak'**,**'Nasim'**];  
*// si on utilise seulement la variable players on récupère seulement une seule valeur  
//const [captain , assistant , players] = team;  
// alors on utilise la fonction ... pour récupérer le reste des joueurs***const** [***captain*** , ***assistant*** , ...***players***] =***team*** ;



Swapping Variables par la décomposition

**let *a*** = **'Youssef'** ;  
**let *b*** = **'Yassine'** ;  
*// pour permuter les deux variables il faut une variable tmp***let *tmp*** = ***a*** ;  
***a*** = ***b*** ;  
***b*** = ***tmp*** ;  
*// utilisant la décomposition on a pas besoin de la variable tmp  
// c'est comme la permutation avec l'opérateur XOR*[***a***,***b***] = [***b***,***a***];



Récupération pas la décomposition de la fonction, retour multi, valeur par défaut d’une fonction avec la décomposition

**function** *convertCurrency*(amount){  
 **const** converted = {  
 **USD**: amount \* 0.85 ,  
 **GPB**: amount \* 0.43 ,  
 **AUD**: amount \* 1.25 ,  
 **MEX**: amount \* 14.35  
 }  
 **return** converted;  
};  
*// on peut recuperer les valeurs sans respecter l'ordre  
// car on n'utilise un tableau mais les propriétés de l'objet***const** { ***MEX*** , ***GPB*** , ***USD***} = *convertCurrency*(100);  
**console**.log(***MEX*** , ***GPB*** , ***USD***);



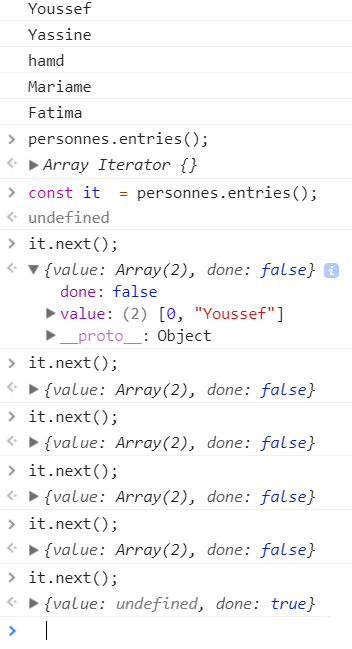
**function** *convertCurrency*(amount){  
 **const** converted = {  
 **USD**: amount \* 0.85 ,  
 **GPB**: amount \* 0.43 ,  
 **AUD**: amount \* 1.25 ,  
 **MEX**: amount \* 14.35  
 }  
 **return** converted;  
};  
  
*// on défini les paramètres par défaut de la fonction   
// on utilise une fonction de décomposition pour matché l'objet recu par la fonction  
// si on envoie pas un objet dans la signature de la méthode on set par défaut un objet vide ==> = {}  
// la fonction de décomposions récupère l'objet vide et elle va utiliser les valeur par défaut quand elle ne les trouves pas les propriétés*

**function** *tipCalc*({ total = 100 , tip = 0.12 , tax = 0.14 } = {} ){  
 **return** total + (tip \* total) + (tax + total);  
 }  
  
 **let *bill*** = *tipCalc*({ **tip** :0.25 , **total** : 100});  
 **console**.log(***bill***);  
  
 *// sans objet* ***bill*** = *tipCalc*();  
 **console**.log(***bill***);

Objet avec les valeurs

Fn de décomposition

En action le for de loop

Utilisation de la boucle loop avec l’option for

**const *personnes*** =

[**'Youssef'**,

**'Yassine'**,

**'hamd'**,

**'Mariame'**,

**'Fatima'**] ;  
**for**(**const *nom*** of ***personnes*** ){  
 **console**.log(***nom***)  
}

La fonction ***entries()*** permette de récupérer un intégrateur à partir d’un tableau

Littérateur it ici à une fonction ***next()*** qui retour un objet avec 2 propriétés

{

done : true ou false qui indique est ce qu’il y des donnée encore ou pas

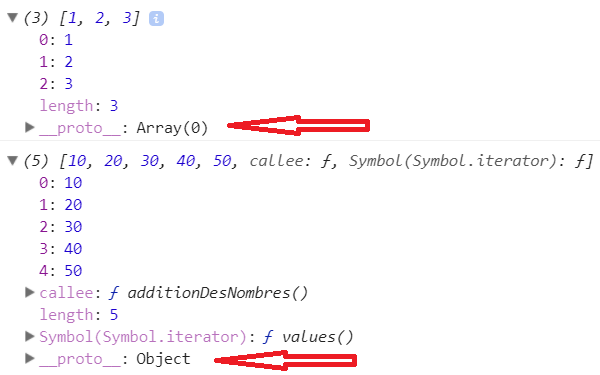
value : Array[ index, value ]

}

Pour faciliter itération des données d’un tableau on va utiliser la notion de décomposition(Destructuring)

**const *personnes*** = [**'Youssef'**, **'Yassine'**, **'hamd'**, **'Mariame'**, **'Fatima'**] ;  
**for**(**const** [***index*** , ***nom***] of ***personnes***.entries() ){  
 **console**.log(**`** ${***nom***} **est en position** ${***index*** +1} **`**);  
}

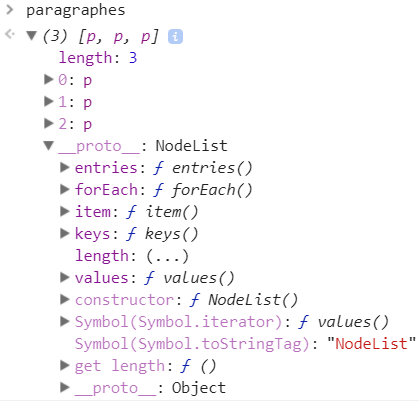
Utilisation de nom réserver ***arguments*** pour récupérer tous les paramètres sans les déclarés dans la signature de la fonction



**function** *additionDesNombres*(){  
 **console**.log([1,2,3]);  
 **console**.log(arguments);  
}  
*additionDesNombres*(10,20,30,40,50);

arguments ne renvoie pas un tableau mais un objet avec une fonction length et Symbol.itretor

**function** *additionDesNombres*(){  
 let total = 0 ;  
 **for**(**const** num of arguments){  
 total += num;  
 }  
 **return** total;  
}  
**console**.log(*additionDesNombres*(10,20,30,40,50));



<**body**>  
  
<**p**>Utilisateur numéro 01</**p**>  
<**p**>Utilisateur numéro 02</**p**>  
<**p**>Utilisateur numéro 03</**p**>  
  
<**script**>  
 **const *paragraphes*** = **document**.querySelectorAll(**'p'**);  
 **for**(**const *par*** of ***paragraphes***){  
 ***par***.addEventListener(**'click'** , **function**(){  
 **console**.log(**this**.**textContent**);  
 })  
 }  
</**script**>  
</**body**>

***addEventListener*** ajoute un écouteur sur chaque paragraphe

convertir une NodeList en tableau Array :

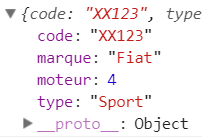
<**body**>  
<**div class="fruit"**>  
 <**p**>Banane</**p**>  
 <**p**>Orange</**p**>  
 <**p**>Pomme</**p**>  
</**div**>  
<**script**>  
 **const *fruits*** = **document**.querySelectorAll(**'.fruit p'**);  
 **console**.log(***fruits***);  
 **const *fruitsArray*** = Array.from(***fruits***);  
 **const *names*** = ***fruitsArray***.map(**fruit**=> **fruit**.**textContent**);  
 **for**(**const *name*** of ***names***){  
 **console**.log(***name***);  
 }  
</**script**>  
</**body**>

<**script**>  
 **const *fruits*** = **document**.querySelectorAll(**'.fruit p'**);  
 **console**.log(***fruits***);  
 *// la fonction de converssion de en tableau Array.from   
 // accepte plus l'objet à convertir , une fonction pour recuperer  
 // les valeurs du tableau par exemple*

*// ArrayFunction(objetSrc , fn)* **const *fruitsArray*** = Array.from(***fruits*** , **fruit**=>{  
 **console**.log(**fruit**.**textContent**);  
 **return** fruit.textContent ;  
 });  
 **console**.log(***fruitsArray***);  
</**script**>

Utilisation de la méthode find() et findIndex() sur un tableau

On peut utiliser ces méthodes pour créer un filtre sur un tableau



**const *cars*** =[  
 {  
 **"code"**:**"XX123"**,  
 **"type"**:**"Sport"**,  
 **"moteur"**:4,  
 **"marque"**:**"Fiat"** },  
 {  
 **"code"**:**"YY123"**,  
 **"type"**:**"Sport"**,  
 **"moteur"**:4,  
 **"marque"**:**"BMW"** },  
 {  
 **"code"**:**"ZZ123"**,  
 **"type"**:**"Sport"**,  
 **"moteur"**:4,  
 **"marque"**:**"Mercedes"** },  
];  
  
**const *code*** = **'XX123'  
const *car*** = ***cars***.find(car => {  
 **if**(car.code=== ***code***){  
 **return true** ;  
 }**else**{  
 **return false** ;  
 }  
 });  
**console**.log(***car***);

*// deuxième façon pour définir le comportement   
// de la méthode find()***const *car*** = ***cars***.find(car => ***car***.code === ***code***);  
**console**.log(***car***);

Pour récupérer l’index de l’objet trouvé on utilise la méthode findIndex()

**const *code*** = **'YY123'  
const *car*** = ***cars***.find(car => ***car***.code === ***code***);  
**const *carIndex*** = ***cars***.findIndex(car => {  
 **if**(car.code === ***code***){  
 **return true** ;  
 } **else**{  
 **return false**;  
 }  
 });  
**console**.log(***carIndex***);



**const *code*** = **'YY123'  
const *car*** = ***cars***.find(car => ***car***.code === ***code***);  
**const *carIndex*** = ***cars***.findIndex(car => ***car***.code === ***code***);  
**console**.log(***carIndex***);

Utilisation de la fonction .some() et .every() d’un tableau :

La méthode .some() permet de vérifier chaque élément d’un tableau si la condition de test est bonne sur un seul élément de tableau , la méthode retourne true.

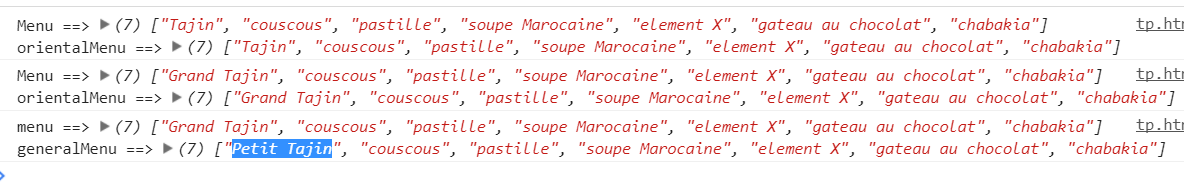
La méthode .every() test aussi les éléments d’un tableau mais elle retourne un true si seulement si tous les éléments du tableau passe le test

**const *ages*** =[34 , 32 , 52 , 18 , 8];  
  
**const *adulPresent*** = ***ages***.some(**age** => **age** >= 18);  
**console**.log(***adulPresent***) ; *// le test = true car il y a un au moins***const *tousAdult*** = ***ages***.every(**age** => **age** >= 19 );  
**console**.log(***tousAdult***) ; *// le test = false car il y a une personne agé de 8 ans*

**Opération de propagation et décomposition :**

Concaténer les valeurs des tableaux

**const *plats*** = [**'Tajin'** , **'couscous'** ,**'pastille'** , **'soupe Marocaine'**] ;  
**const *patisseries*** = [**'gateau au chocolat'** ,**'chabakia'** ] ;  
  
**const *demoPropagation*** = [...**'Tajin'**];  
  
let ***menu*** = [...plats ,**'element X'**, ...***patisseries***] ;  
*// cette affectation copie seulement la référence du tableau menu dans  
// la variable orientalMenu, chaque changement d'un des deux menu  
// va changé les valeurs de l'autre*let ***orientalMenu*** = ***menu***;  
  
**console**.log(**"Menu ==>"** ,***menu*** ,**"orientalMenu ==>"**, ***orientalMenu***) ;  
  
***menu***[0] = **'Grand Tajin'** ;  
**console**.log(**"Menu ==>"** ,***menu*** ,**"orientalMenu ==>"**, ***orientalMenu***) ;  
  
*// pour créer une nouvelle instance de orientalMenu on va utiliser  
// la fonction de propagation [...] qui décompose les éléments du tableau  
// et crée une nouvelle instance*let ***generalMenu*** = [...menu];  
***generalMenu***[0] = **'Petit Tajin'** ;  
**console**.log(**'menu ==>'** , ***menu*** ,**'generalMenu ==>'**, ***generalMenu***) ;



Décomptions dans la signature d’une fonction

**const *plats*** = [**'Tajin'** , **'couscous'** ,**'pastille'** , **'soupe Marocaine'**] ;  
  
**function** *platDuJour*( plat1 , plat2){  
 alert(**`plat du jour est** ${plat1} **et** ${plat2}**`**);  
}  
*platDuJour*(...plats) ;

**Différentes formes pour créer un objet :**

**const *prenom*** =**'Youssef'**;  
**const *nom*** = **'Redal'**;  
**const *age*** = 34 ;  
**const *tel*** = **'07.55.55.55.55'**;  
  
**const *personne*** ={  
 **prenomPer** :***prenom***,  
 **nomPer** :***nom*** ,  
 **agePer** : ***age*** ,  
 **telPer** : ***tel***,  
 **couleur** :[**'Noir'**,**'Blanc'** ]  
};  
**console**.log(**'personne ===> '**,***personne***);  
  
**const *daoPersonne*** ={  
 **dataSource**:**'jdbc/mysql'**,  
 create(per){  
  
 },  
 updatePesonne(per){  
  
 },  
 deletePersonne(id){  
  
 },  
 getListPersonne(){  
  
 }  
};  
**console**.log(**'daoPersonne ===> '**,***daoPersonne***);  
  
**function** *invertColor*(color){  
 **return '#'**+(**'000000'**+(0xFFFFFF ^ parseInt(color.substring(1) ,16)).toString(16)).slice(-6);  
}  
**const *key*** =**'couleurNoir'**;  
**const *value*** = **'#000'**;  
*//créer dynamiquement la propriété couleur de l'objet tShirt  
// on a pas spécifié le nom de la propriété couleur pour l'objet***const *tShirt*** = {  
 [key]: ***value***};  
**console**.log(**'tShirt ===> '** ,***tShirt***);  
  
*// création d'une propriété avec tempateString hihihi !***const *chemise*** ={  
 [key]:***value***,  
 [**`**${key}**Opposite`**] :*invertColor*(***value***)  
}  
**console**.log(**'chemise ==> '** ,***chemise***) ;  
  
**const *keys*** =[**'size'** , **'color'** ,**'weight'**];  
**const *values*** =[**'medium'** ,**'red'** , 100];  
*// un matching des valeurs des tableaux pour créer un objet dynamique***const *pull*** ={  
 [keys.shift()] :***values***.shift(), *// 1er élément des tableaux <==> 1er propriété de l'objet pull* [keys.shift()] :***values***.shift(), *// 2eme* [keys.shift()] :***values***.shift(), *// 3 eme*};  
**console**.log(**'pull ===> '** ,***pull***);

