

Cours JavaScript

Part 2

PLAN

JS

Les Variables

Array & Object

Correction des TP

Arrow Function

TP

Conclusion

/!\ Warning /!\

JS

Repl.it => éviter d'utiliser **prompt**

Failed tests

✓ testGetTriangleType

No message from your teacher

Expected 'Equilatéral' to be 'équilatéral'.

▶ More info

Repl.it => Les tests unitaire teste plus de cas que ceux de l'énoncé

Les Variables



Les Variables

Contraintes

Pas de caractères spéciaux (sauf \$ et _)

Pas de chiffre en premier caractère

Pas d'espace

Pas de mot réservé

Les Variables

Exemple non exhaustif des caractères interdits

```
1 let break;  
2 let case;  
3 let catch;  
4 let continue;  
5 let export;  
6 let import;  
7 let interface;  
8 let public;  
9 let super;  
10  
11 let 1WrongName;  
12 let [wronName];  
13 let wrong name;  
14 let wrong!name;  
15 let wrong?name;  
16 let wrong:name;  
17 let wrong,name;  
18 let wrong.name;  
19 let wrong/name;  
20 let wrong$name;  
21 let wrong*name;
```

```
1 let wrongfname;  
2 let wrong꜀name;  
3 let wrong`name;  
4 let wrong^name;  
5 let wrong|name;  
6 let wrong\name;  
7 let wrong&name;  
8 let wrong~name;  
9 let wrong#name;  
10 let wrong{name;  
11 let wrong[name;  
12 let wrong@name;  
13 let wrong)name;  
14 let wrong°name;  
15 let wrong=name;  
16 let wrong%name;  
17 let wrong<name;  
18 let wrong>name;  
19 let wrong`name;
```

Les Variables

Exemple caractères valides

```
1 let abc;  
2 let $__aBc$_9000;  
3 let _abc_$1337$;  
4 let привет;  
5 let صباح الخير;  
6 let 你好;  
7  
8  
9 let jÉRÉmy; // Eviter les accents /!\
```

Les Variables

On peut déclarer les variables séparé par une virgule

```
1 let animals, wild, pets = "Dog";  
2 const COMPUTER, SIZE = 23, SCREEN;
```


Array & Object

Déclaration et initialisation d'un tableau

```
1 const fruits = ['Apple', 'Banana'];
```

Afficher la taille d'un tableau

(Permet de connaître le nombre d'élément d'un tableau)

```
1 const fruits = ['Apple', 'Banana'];  
2 console.log(fruits.length); // => 2
```

MDN => Votre référence du Dev JS

Propriétés

`Array.length`

`Array.prototype`

`Array.prototype[[@unscopables]]`

Méthodes

`Array.from()`

`Array.isArray()`

 `Array.observe()`

`Array.of()`

`Array.prototype.concat()`

`Array.prototype.copyWithin()`

`Array.prototype.entries()`

`Array.prototype.every()`

`Array.prototype.fill()`

`Array.prototype.filter()`

`Array.prototype.find()`

`Array.prototype.findIndex()`

`Array.prototype.flat()`

Accéder (via son index) à un élément du tableau

```
1 | var first = fruits[0];  
2 | // Apple  
3 |  
4 | var last = fruits[fruits.length - 1];  
5 | // Banana
```

Boucler sur un tableau

```
1 | fruits.forEach(function(item, index, array) {  
2 |     console.log(item, index);  
3 | });  
4 | // Apple 0  
5 | // Banana 1
```

Ajouter à la fin du tableau

```
1 | var newLength = fruits.push('Orange');  
2 | // ["Apple", "Banana", "Orange"]
```

Array

Ajouter un élément à la fin

```
1 const fruits = ['Apple', 'Banana'];  
2  
3 fruits.push('Orange'); // ["Apple", "Banana", "Orange"]
```

Supprimer un élément à la fin

```
1 const fruits = ['Apple', 'Banana'];  
2 const removedFruit = fruits.pop();  
3 console.log(fruits, removedFruit); // ["Apple"] "Banana"
```

Supprimer un élément au début

```
1 const fruits = ['Apple', 'Banana'];  
2 const removedFruit = fruits.shift();  
3 console.log(fruits, removedFruit); // ["Banana"] "Apple"
```

Array

Ajouter un élément au début

```
1 const fruits = ['Apple', 'Banana'];  
2  
3 fruits.unshift('Orange'); // ["Orange", "Apple", "Banana"]
```

Trouver l'index d'un élément

```
1 const fruits = ['Apple', 'Banana'];  
2  
3 fruits.indexOf('Banana'); // 1
```

Filtrer les éléments

```
1 const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];  
2  
3 words.filter(word => word.length > 6);  
4 // ["exuberant", "destruction", "present"]
```

Array

Supprimer un élément suivant son index

```
1 const fruits = ['Apple', 'Banana'];  
2  
3 fruits.splice(0, 1); // supprime 1 élément à la position 0
```

Supprimer deux éléments

```
1 const vegetables = ['Cabbage', 'Turnip', 'Radish', 'Carrot']  
2 const pos = 1, number = 2;  
3 const removedItems = vegetables.splice(pos, number);  
4 // number définit le nombre d'éléments à supprimer,  
5 // à partir de la position pos  
6  
7 console.log(vegetables); // ["Cabbage", "Carrot"]  
8 // (le tableau d'origine est changé)  
9  
10 console.log(removedItems); // ["Turnip", "Radish"]  
11 // (splice retourne la liste des éléments supprimés)
```

Array

Les tableau on beaucoup d'autres fonctions natives, allez sur [MDN](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array) pour voir toutes les autres fonctions

Object

Déclaration et initialisation d'un objet

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };
```

Vérifie si l'objet est modifiable

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };  
5  
6 console.log(Object.isFrozen(fruit)); // false
```

Object

Objets standards

Object

Propriétés

Object.prototype

⚠️🗑️ Object.prototype.__count__

⚠️🗑️ Object.prototype.__noSuchMethod__

⚠️🗑️ Object.prototype.__parent__

🗨️ Object.prototype.__proto__

Object.prototype.constructor

Méthodes

Object.assign()

Object.create()

Object.defineProperties()

Object.defineProperty()

Object.entries()

Object.freeze()

Object.fromEntries()

🗑️ Object.getNotififier()

Object.getOwnPropertyDescriptor()

Object.getOwnPropertyDescriptors()

Object.getOwnPropertyNames()

Object.getOwnPropertySymbols()

```
// Initialisateur d'objet ou littéral { [ paireNomValeur1[, paireNomValeur2[,
...paireNomValeurN] ] ] }
```

```
// Appelé comme un constructeur
new Object([valeur])
```

Paramètres [🔗](#)

paireNomValeur1, paireNomValeur2, ... paireNomValeurN

Paires de noms (chaînes) et de valeurs (toutes valeurs) où le nom est séparé de la valeur par deux points (:).

valeur

Toute valeur.

Description [🔗](#)

Le constructeur `Object` crée un wrapper d'objet pour la valeur donnée. Si la valeur est `null` ou `undefined`, il créera et retournera un objet vide, sinon, il retournera un objet du Type qui correspond à la valeur donnée. Si la valeur est déjà un objet, le constructeur retournera cette valeur.

Lorsqu'il n'est pas appelé dans un contexte constructeur, `Object` se comporte de façon identique à `new Object()`.

Object

Accéder à une propriété

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };  
5  
6 console.log(fruit.price); // 1.5  
7 console.log(fruit.name); // Apple
```

Vérifie si l'objet est modifiable

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };  
5  
6 console.log(Object.isFrozen(fruit)); // false
```

Object

Modifier une propriété

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };  
5  
6 fruit.name = 'Banana';  
7 console.log(fruit.name); // Banana
```

Ajouter une propriété

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };  
5  
6 fruit.weight = 68;  
7 console.log(fruit); // { name: 'Apple', price: 1.5, weight: 68}
```

Object

Supprimer une propriété

```
1 const fruit = {  
2   name: 'Apple',  
3   price: 1.5  
4 };  
5  
6 delete fruit.price;  
7 console.log(fruit); // { name: ' Apple' }
```

Correction des TP

**Les TP ci-dessous sont sélectionnés
(uniquement sur les JS-Facile-B2-*) pour
la correction en live**

Correction des TP

```
1 function square(number) {  
2     // Implémenter le code nécessaire permettant de réaliser toutes les conditions de l'énoncé  
3     number = prompt("Entrez un chiffre");  
4     if (number<0) {  
5         number = prompt("Veuillez entrer un nombre POSITIF");  
6         let carre = number*number;  
7         return("Le carré de", number, "est", carre);  
8     }  
9     else {  
10        let carre = number*number;  
11        return("Le carré de", number, "est", carre);  
12    }  
13 }  
14 square();
```

Correction des TP

```
1 function say(firstName) {  
2     // Ajouter le code necessaire ici pour remplir les conditions de l'énoncé  
3     let name = firstName ;  
4     let noName = "toi" ;  
5     let OneFor = "un pour" ;  
6     let OneForMe = "un pour moi" ;  
7  
8     if(firstName === undefined){  
9  
10         return OneFor+ " " + noName + ", " + OneForMe;  
11     }  
12  
13     else{  
14  
15         return OneFor+ " " +name+ ", " +OneForMe;  
16     }  
17 }
```

Correction des TP

```
1 function say (firstName) {  
2  
3   if(firstName === "Stella") {  
4     console.log("Un pour", firstName, ", un pour moi");  
5   }  
6   else if(firstName === "Jean") {  
7     console.log("Un pour", firstName, ", un pour moi");  
8   }  
9   else {  
10    console.log("Un pour toi, un pour moi");  
11  }  
12 }  
13 say();  
14 console.log(say("Stella"));  
15 console.log(say("Jean"));  
16 console.log(say());
```

Correction des TP

```
1 let cote1 = prompt('Choisir votre premier cote');
2 let cote2 = prompt('Choisir votre deuxieme cote');
3 let cote3 = prompt('Choisir votre troisieme cote');
4 let coteResult = parseInt(cote1) + parseInt(cote3);
5 // faudra m'expliquer pourquoi le prompt recupère en 'object'...
6 // ça oblige a faire une vieille ligne parseInt
7
8 getTriangleType(cote1, cote2, cote3);
9
10 function getTriangleType(cote1, cote2, cote3) {
11     if (coteResult < cote2) {
12         return "ce triangle est impossible car " + cote1 + " + " + cote2 + " < " + cote3 + " !";
13     }
14     else if (cote1 == cote2 && cote2 == cote3) {
15         return "équilatéral";
16     }
17     else if (cote1 == cote2 || cote2 == cote3 || cote1 == cote3) {
18         return "isocèle";
19     }
20     else if (cote1 != cote2,cote3 || cote2 != cote1,cote3 || cote3 != cote1,cote2) {
21         return "scalène";
22     }
23 }
```


Correction des TP

```
1 function getTriangleType(cote1, cote2, cote3) {
2     // Implémenter le code necessaire permettant de réaliser toutes les conditions de l'énoncé
3
4     if (cote1 == cote2 & cote1 == cote3 & cote2 == cote3) {
5         let phrase = + cote1 + "," + cote2 + "," + cote3 + " est un triangle équilatéral";
6         return phrase;
7     }
8
9     else if (cote1 == cote2 || cote1 == cote3 || cote2 == cote3) {
10        phrase = + cote1 + "," + cote2 + "," + cote3 + " est un triangle isocèle";
11        return phrase;
12    }
13
14    else if (cote1 != cote2 || cote1 != cote3 || cote2 != cote3) {
15        if(cote1 + cote3 < cote2) {
16            phrase = + cote1 + "," + cote2 + "," + cote3 + " est un triangle impossible";
17            return phrase;
18        }
19        else {
20            phrase = + cote1 + "," + cote2 + "," + cote3 + " est un triangle scalène";
21            return phrase;
22        }
23    }
24 }
25 console.log (getTriangleType(2, 3, 2));
```

Correction des TP


```
1 function getTriangleType(cote1, cote2, cote3) {
2     // Implémenter le code nécessaire permettant de réaliser toutes les conditions de l'énoncé
3
4     if (cote1 + cote3 < cote2) {
5         let phrase = "impossible"
6         return phrase;
7     }
8     else if (cote1 == cote2 & cote1 == cote3 & cote2 == cote3) {
9         let phrase = "équilatéral";
10        return phrase;
11    }
12
13    else if ((cote1 == cote2) || (cote2 == cote3) || (cote1 == cote3)) {
14        let phrase = "isocèle"
15        return phrase;
16    }
17
18
19
20
21    else if (cote1 != cote2 != cote3 && cote1 + cote3 >= cote2) {
22        let phrase = "scalène"
23        return phrase;
24    }
25 }
```

Correction des TP

```
1 function convertToRomanNumber (number) {
2     // Implémenter le code nécessaire permettant de réaliser toutes les conditions de l'énoncé
3     let t_unit() = [ " ", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX" ];
4     let t_diz() = [ " ", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC" ];
5     let t_cent() = [ " ", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM" ];
6     let t_mille() = [ " ", "M", "MM", "MMM", "MMMM" ];
7
8
9     //Je n'arrive pas à développer ma pensée je voudrai décomposer mon nombre entre unités/dizaines/...
10    //et ensuite que les unités aillent chercher dans t_unit et pareil pour le reste pour au final tout col
11 }
```

Correction des TP

JS

 repl.it

[back to assignment](#) [run code](#)

```
1 function convertToRomanNumber (number) {  
2   // Implémenter le code nécessaire permettant de réaliser toutes les conditions de l'énoncé  
3   return "ca me saoul, TOZ!!"  
4 }
```

maurin margail
Sep 30, 2019 12:20 am

my repls teacher student notifications

Due: Oct 02, 2019 08:30 pm

JS-B2-Facile-4

L'objectif de cet exercice est de créer une fonction qui traduit un nombre en chiffres Romains.

Par exemple pour nombre = 12, la fonction doit renvoyer XII.

On utilisera les lettres i, v, x, l, c, d, m en majuscule pour construire les chiffres Romains.

Voici les règles de construction d'un chiffre Romain :

- Les nombres 1, 2 et 3 s'écrivent respectivement I, II et III
- Le nombre 5 s'écrit V
- Le nombre 10 s'écrit X
- Le nombre 50 s'écrit L
- Le nombre 100 s'écrit C
- Le nombre 500 s'écrit D
- Le nombre 1000 s'écrit M

Correction des TP

```
1 function convertToAcronym (sentence) {
2     // Implémenter le code nécessaire permettant de réaliser toutes les conditions de l'énoncé
3     let acronym = "";
4     let upper = sentence.toUpperCase();
5     let phrase = upper.split(" ");
6
7     phrase.forEach(function(item, array) {
8         let remove = item.replace(' ', '');
9         acronym += remove.charAt(0);
10    });
11    return acronym;
12 }
13 console.log(convertToAcronym("JavaScript est vraiment top"));
14 console.log(convertToAcronym("toute une phrase en minuscule"));
15 console.log(convertToAcronym("TU PEUX AUSSI TOUT ECRIRE EN MAJUSCULE"));
16 console.log(convertToAcronym("On PeUT AusSi MixEr LeS DEUx"));
17 console.log(convertToAcronym("Dépêche toi j'ai rendez-vous"));
18 console.log(convertToAcronym("Les \"guillemets\" vont faire planter ton code"));
```

Correction des TP

```
1 function convertToAcronym (sentence) {
2     // Implémenter le code necessaire permettant de réaliser toutes les conditions de l'énoncé
3     //[3][0] (troisième mot première lettre)
4     let tab = [];
5     let tab2 = [];
6     tab = sentence.split(' ');
7     for(let i = 0; i<tab.length; i++){
8         tab2.push(tab[i][0].toUpperCase());
9     }
10    return tab2;
11 }
12
13 console.log(convertToAcronym("JavaScript est vraiment top").join(''));
14 console.log(convertToAcronym("une phrase en minuscule").join(''));
15 console.log(convertToAcronym("TU PEUX AUSSI TOUT ECRIRE EN MAJUSCULE").join(''));
16 console.log(convertToAcronym("On PeUT AusSi MixEr LeS DEUx").join(''));
17 console.log(convertToAcronym("Dépêche toi j'ai rendez-vous").join(''));
18 console.log(convertToAcronym(" Les 'guillemets' vont faire planter ton code").join(''));
```

Correction des TP

```
1 function convertToAcronym () {
2
3   let sentence = ["JavaScript est vraiment top"
4     , "toute une phrase en minuscule"
5     , "TU PEUX AUSSI TOUT ECRIRE EN MAJUSCULE"
6     , "On PeUT AusSi MixEr LeS DEUX"
7     , "Dépêche toi j'ai rendrez-vous"
8     , 'Les "guillemets" vont faire planter ton code'];
9
10  let acronym = '';
11
12
13  for(let j = 0; j < sentence.length; j++){
14    splitted = sentence[j].toString().split(" ");
15
16    for(let i = 0; i < splitted.length; i++){
17      if(splitted[i].charAt(0) !== ' '){
18        acronym += splitted[i].charAt(0).toUpperCase();
19      }else{
20        acronym += splitted[i].charAt(1).toUpperCase();
21      }
22    }
23    acronym += "\n"
24  }
25
26
27  return acronym;
```

Arrow Function

Contrairement aux fonctions classiques :

Permet d'avoir une syntaxe plus courte

Ne redéfinit pas la valeur de **this**

Ne redéfinit pas la valeur de **argument**

Ne redéfinit pas la valeur de **super**

Ne redéfinit pas la valeur de **new.target**

Arrow Function

Fonction anonyme

```
1  const MATERIALS = [  
2    'Hydrogen',  
3    'Helium',  
4    'Lithium',  
5    'Beryllium'  
6  ];  
7  
8  const QUANTITY = 15;  
9  
10 // function (paramètre1, paramètre 2, ...) {  
11 //   corp de ma fonction  
12 //   return result;  
13 // }  
14  
15 console.log(  
16   materials.map(  
17     function (MATERIALS, QUANTITY) {  
18       return MATERIALS.length + QUANTITY;  
19     }  
20   )  
21 );
```

Arrow function

```
1  const MATERIALS = [  
2    'Hydrogen',  
3    'Helium',  
4    'Lithium',  
5    'Beryllium'  
6  ];  
7  
8  const QUANTITY = 15;  
9  
10 // (paramètre1, paramètre 2, ...) => {  
11 //   corp de ma fonction  
12 //   return result;  
13 // }  
14  
15 console.log(  
16   materials.map(  
17     (MATERIALS, QUANTITY) => {  
18       return MATERIALS.length + QUANTITY;  
19     }  
20   )  
21 );
```

Arrow Function

Arrow function compact

```
1 console.log(  
2   materials.map(  
3     (MATERIALS, QUANTITY) => {  
4       return MATERIALS.length + QUANTITY;  
5     }  
6   )  
7 );
```

Arrow function compact +

```
1 const MATERIALS = [  
2   'Hydrogen',  
3   'Helium',  
4   'Lithium',  
5   'Beryllium'  
6 ];  
7  
8 const QUANTITY = 15;  
9  
10 // (paramètre1, paramètre 2, ...) => result;  
11 // Ici je n'ai plus besoin du mot réservé 'return' pour renvoyer le résultat  
12  
13 console.log(MATERIALS.map((MATERIALS, QUANTITY) => MATERIALS.length + QUANTITY));
```

Arrow Function

Arrow function compact +

```
1 console.log(MATERIALS.map((MATERIALS, QUANTITY) => MATERIALS.length + QUANTITY));
```

Arrow function compact ++

```
1 const MATERIALS = [  
2   'Hydrogen',  
3   'Helium',  
4   'Lithium',  
5   'Beryllium'  
6 ];  
7  
8 // paramètre1 => result;  
9 // Ici je n'ai plus qu'un seul paramètre, plus besoin de parenthèses  
10  
11 console.log(MATERIALS.map(MATERIALS => MATERIALS.length));
```

TP

JS

JS-B2-Moyen*

A faire pour le prochain cours

Conclusion

Nous avons vu :

Les variables

Array & Object

Arrow function

MindMap time !!!