TP 2 – Tableaux, chaînes et fonctions avancées

Plusieurs fonctions devront être déposées sur moode, il est donc essentiel :

- De respecter les noms de fonctions indiqués dans le sujet
- De valider tout votre code avec eslint en mode airbnb strict (sans console notamment) avant dépôt. Vous pouvez également utiliser https://validatejavascript.com/ (toujours en mode airbnb).

1. Tableaux

Les tableaux permettent de stocker des valeurs. En Javascript on peut stocker des valeurs de type différent dans le même tableau. Comme dans tous les autres langages la première case d'un tableau est à l'indice 0.

```
var t = [1,4,2,3];
t.length;     // nombre d'éléments dans un tableau -> 4
t.indexOf(4); // position d'un élément donné -> 1
t.pop();      // suppression du dernier élément -> 3 (élément supprimé)
t.push(5);    // ajout à la fin du tableau -> 4 (nouvelle longueur du tableau)
t.reverse();  // inversion en place du tableau -> [5,2,4,1]
t.slice(1,3); // extraction d'un sous tableau (début, [fin]) -> [2,4]
t.sort();     // tri en place d'un tableau -> [1,2,4,5]
t.join("#");  // conversion en chaine de caractère avec séparateur -> 1#4#2#3
```

Exercice 1. écrire les fonctions suivantes avec une boucle for ou une boucle while :

- **lastButOne** : retourne l'avant dernier élément d'un tableau. Si le tableau a moins de deux éléments la fonction doit retourner false.
  lastButOne([1,2,3]) -> 2
  lastButOne([1]) -> false
- **square** : crée un tableau dont les éléments sont le carré des éléments du tableau passé en paramètre.
  square([1,2,3,4]) -> [1,4,9,16]
- **gt10** : crée un tableau en ne gardant que les éléments supérieurs ou égaux à 10 du tableau.
  gt10([1,27,3,42,2]) -> [27,42]
- **sum** : calcule la somme des éléments du tableaux.
  sum([1,2,3]) -> 6

2. Chaînes de caractères

```
var str = "abcdefgh" ;
str.length;               //longueur d'une chaine -> 8
str.charAt(4); str[4];    // caractère à une certaine position -> e
str.indexOf("de");        // position d'un motif (-1 si absent) -> 3
str.replace("de", "xx");  // remplacement d'une sous-chaine -> abcxxfgh
str.substr(2,4);          // sous-chaine (début, longueur) -> cdef
"je suis".split(" ");     // découpage avec séparateur -> ["je", "suis"]
"jesuis".split("") ;      //   si chaine vide : ["j", "e", "s", "u", "i", "s"]
"bonjOUR".toLowerCase(); // mettre en minuscule -> bonjour
"bonjOUR".toUpperCase(); // mettre en majuscule -> BONJOUR
```

Exercice 2. Ecrire les fonctions suivantes :

- **vowel** : compte le nombre de voyelles d'une chaîne (on ne considère pas les caractères accentués).
  vowel("je suis en cours") -> 6
- **palindrome** : teste si une chaine est un palindrome ou pas (se lit dans les deux sens de la même manière).
  palindrome("test") -> false
  palindrome("kayak") -> true

- **uppercase** : met en majuscules la première lettre d'une chaine (et tout le reste en minuscules)
  uppercase("il fait BEAU") -> Il fait beau

3. Fonctions d'ordre supérieur

On appelle fonction d'ordre supérieur toute fonction qui prend en argument une fonction ou qui en retourne une. En javascript, les fonctions sont des variables comme les autres.

```
// appelle la fonction f passée en paramètre avec un attribut incrémenté
function ajoute_un_et_applique(n, f) {
  f(n + 1) ;
}
// incrémente 10 et appelle la fonction qui fait console.log sur le résultat
ajoute_un_et_applique(10, (x) => {console.log(x);}) ;

function ajoute(n) { // ajoute retourne une function
  return (m) => m + n;
}
const ajoute10 = ajoute(10);
ajoute10(1); // -> 11
```

Un exemple très classique est la méthode forEach permet d'appliquer une fonction quelconque à tous les éléments d'un tableau

```
function affiche(x) {
  console.log(x);
}

// applique la fonction affiche à tous els éléments du tableau [1,2,3]
// il faut noter qu'il n'y a pas de () après affiche
[1,2,3].forEach(affiche);

// ou, plus court :
[1,2,3].forEach((x) => console.log(x));
```

Exercice 3. Réécrire les fonctions de l'exercice 1 avec forEach plutôt qu'avec une boucle for ou while.
Exercice 4. Ecrire les fonctions suivantes

- **map** : prend un tableau et une fonction f et retourne un tableau dans lequel chaque élément est l'application de f à l'élément correspondant du tableau. C'est comme pour la fonction square de l'exercice 1 mais avec n'importe quelle fonction.
  map([1,2,3,4], carre) -> [1,4,9,16], si function carre(x) {return x*x ;}
  map([1,2,3,4], plusUn) -> [2,3,4,5] si function plusUn(x) {return x+1 ;}
- **filter** : prend un tableau et une fonction f et ne garde que les éléments pour lequel f est vraie. Inspirez-vous de la fonction gt10 de l'exercice 1.
  filter([1,2,3,4], pair) -> [2,4], si function pair (x) {return x%2==0 ;}
  filter([1,2,3,4], sup3) -> [3,4] si function sup3(x) {return x>=3 ;}
- **reduce** : prend un tableau et calcule récursivement une valeur en combinant l'élément courant et la réduction de la fin du tableau. Inspirez-vous de la fonction sum de l'exercice 1.
  reduce([1,2,3,4], (cur,accu)=>cur+accu, 0) -> 10 // le 0 est la valeur initiale

Les méthode map, filter et reduce existent déjà pour l'objet array en javascript. Allez voir la documentation pour bien les utiliser (en particulier reduce ne prend pas toujours une valeur initiale comme indiqué précédemment) :

```
const tab1 = [1, 4, 9, 16];
const tab2 = tab1.map((x) => x * 2);
console.log(tab2);
// [2, 8, 18, 32]
```

Exercice 5. Réécrire les fonctions square, gt10, sum, vowel avec map, filter ou reduce.

4. Expressions régulières

Une expression régulière permet de savoir si une chaîne contient un motif particulier. On peut bien entendu chercher un mot particulier mais aussi utiliser des jokers, chercher des motifs qui se répetent plusieurs fois, etc.

La syntaxe des expression régulières est assez stricte

```
Par défaut c'est du texte
  /xyz/ : ok si la chaine contient xyz
[] pour indiquer un ensemble de caractères autorisés
  /[xyz]/ : ok si la chaine contient x, y ou z
  /[3-6]/ : ok si la chaine contient un chiffre entre 3 et 6
  /[d-y]/ : idem pour des lettres
  /[10-20/ : ne permet pas de dire qu'on veut des choses entre 10 et 20 !
| correspond à un ou logique
  /abc|xyz/ : ok si la chaine contient abc ou xyz
^ et $ pour indiquer le début et la fin de la chaîne
  /^[0-9]/ : ok si la chaine commence par un chiffre
  /[0-9]$/ : ok si la chaine termine par un chiffre
?, *, + et {} pour indiquer des répétitions
  /(xyz)?/ : xyz apparait 0 ou 1 fois (optionnel)
  /(xyz)*/ : xyz peut apparaitre autant de fois que voulu (y compris 0)
  /(xyz)+/ : xyz doit apparaitre au moins une fois
  /(xyz){3,5} : xyz doit apparaitre entre 3 et 5 fois
```

```
// utilisation d'expression régulière (objet RegExp)
var re = /^0[0-9]{9}$/;
console.log(re.test("0606060606")); // -> true
console.log(re.test("1000")); // -> false
```

Exercice 6. Ecrire les fonctions suivantes **:**

- **isCodePostal** pour reconnaître un code postal contenant 5 chiffres quelconques.
- **isCodePostalFr** pour reconnaitre une chaine contenant un code postal mais en faisant en sorte que les deux premiers chiffres soient entre 01 et 95.
- **isTelephone** pour reconnaitre un numéro de téléphone à 10 chiffres (le premier chiffre est un 0, le suivant un chiffre entre 1 et 7, les 8 autres sont quelconques). On permet que les groupes de deux chiffres soit séparés par rien, un espace, un point ou un tiret (par exemple 01.12.23.34.45 est valide).
- **isEmail** qui teste si une chaine de caractère est une adresse email correcte (on supposera qu'une adresse est composée d'une suite de lettres et de chiffres, puis d'une @, puis d'une suite de lettres contenant un .)
  isEmail("test@test.com") -> true
  isEmail("test@test") -> false (pas de point)
  isEmail("testtest.com") -> false (pas d'@)

5. Exercices supplémentaires

A chaque fois que possible utilisez les fonctions map, filter et reduce.

Exercice 7. Ecrire les fonctions suivantes :

- **miroir** : vérifie si un tableau est symétrique
  miroir([1,2,1,3]) -> false
  miroir ([1,2,1]) -> true
  miroir ([1,2,2,1]) -> true
- **afficheRec** : affiche tous les éléments d'un tableau de nombres un par un. Si le tableau contient des sous-tableaux alors ils doivent être affichés aussi. On ne traitera que le cas de tableau de nombres (éventuellement imbriqués)
  affiche([1,2,3]) -> 1 2 3
  affiche([1,2,[3,4, [5,6,7], 8], 9]) -> 1 2 3 4 5 6 7 8 9

- **majuscules** : transformer une chaine en mettant en majuscule la première lettre de chaque mot.
  majuscules("il fait beau") -> Il Fait Beau
- **alpha** : retourne une chaine avec toutes les lettres dans l'ordre alphabétique (on supprimera ou pas les espaces ce qui peut se faire avec la méthode trim).
  alpha("je suis en cours") -> ceeijnorsssuu
- **anagramme** : teste si deux chaines sont anagrammes l'une de l'autre.
  anagramme("chien", "niche") -> true
  anagramme("chien", "maison") -> false
- **caesar** : code une chaîne de caractères avec le code de caesar qui consiste à décaler chaque lettre d'un certain nombre de positions.
  caesar('je suis en cours', 2) -> 'lg uwku go epwtu'
  caesar('je suis en cours', -1) -> 'id rthr dm bntqr'
- **vigenere** : le code se fait sur le même principe mais au lieu de décaler toutes les lettres de la même manière, on utilise une autre chaîne qui indique les décalages (a=0, b=1, c=2, etc.)
  vigenere('je suis en cours', 'a') = caesar(1)
  vigenere('je suis en cours', 'ab') = 'kg twju fp dqvtt' (les lettres sont décalées alternativement de 1=a et 2=b)