TP 2 – Tableaux, chaînes et fonctions avancées

1. Linter

Pour tous les exercices, vous utiliserez le linter eslint et les règles de airbnb associées.

2. 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 -> 4
t.indexOf(4); // position d'un élément -> 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

- **lastButOne** : retourne l'avant dernier élément d'un tableau ou false si le tableau a moins de deux éléments
- **square** : crée un tableau dont les éléments sont le carré 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 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

3. 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
  vowel("je suis en cours") -> 6
- **palindrome** : teste si une chaine est un palindrome ou pas
  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

4. 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 passée en paramètre avec un attribut incrémenté
function ajouteUnEtApplique(n, f) {
  f(n + 1) ;
}
ajouteUnEtApplique (10, x => {console.log(x);}) ;

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

Exercice 3. 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. Inspirez-vous de la fonction carre faite plus tôt.
  map([1,2,3,4], square) -> [1,4,9,16], si function carre(x) {return x*x ;}
  map([1,2,3,4], plus_un) -> [2,3,4,5] si function plus_un(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 sup10 faite plus tôt.
  filter([1,2,3,4], pair) -> [2,4], si function pair (x) {return x%2==0 ;}
  filter([1,2,3,4], superieur_a_trois) -> [3,4] si function superieur_a_trois(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 somme.
  reduce([1,2,3,4], (cur,accu)=>cur+accu, 0) -> 10 // le 0 est la valeur initiale
- Réécrire les fonctions carre, sup10, somme, voyelles avec map, filter ou reduce.

Les fonctions map, filter et reduce existent déjà pour l'objet array en javascript.

1. Expressions régulières

Expression régulières : principes de base. On met la chaine à chercher entre deux /

- /xyz/ : ok si la chaine contient xyz
- /[xyz]/ : ok si la chaine contient x, y ou z
- /[3-6]/ : ok si la chaine contient un chiffre entre 3 et 6 (marche aussi avec des lettres [d-y])
- /abc|xyz/ : ok si la chaine contient abc ou xyz
- /^[0-9]/ : ok si la chaine commence par un chiffre
- /[0-9]$/ : ok si la chaine termine par un chiffre
- /(xyz)?/ : xyz est 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

Exercice 4. Ecrire des expressions régulières pour :

- Reconnaitre une chaine contenant un code postal (5 chiffres).
- Reconnaitre une chaine contenant un code postal mais en faisant en sorte que les deux premiers chiffres soient entre 01 et 95.
- Reconnaitre exactement un numéro de téléphone (le premier chiffre est un 0, le suivant un chiffre entre 1 et 7, les 8 autres sont quelconques). On peut ajouter un espace, un point ou un tiret entre des groupes de deux chiffres (par exemple 01.12.23.34.45)

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

```
console.log(re.test("1000")); // -> false
```

2. Exercice supplémentaire pour le TP

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

Exercice 5. 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.
  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
- **isEmail** : 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'@)
- Reprenez l'exercice syracuse3 du TP1 et améliorez son efficacité en utilisant un tableau. L'idée est de ne jamais recalculer une valeur déjà calculée.