Langage JavaScript

Les fonctions

- function mafonction(argument1, argument2, …) {
  instruction(s);
  }
- Une fonction a pour objectif de regrouper des instructions qui pourront ainsi être exécutées facilement lors de chaque appel.
- Peut accepter un ou plusieurs arguments, placés lors de l'appel entre les parenthèses.
- Retourne éventuellement, en fin de son exécution, une valeur (une seule) avec l'instruction return.
- prompt(), alert() ou confirm() sont des fonctions natives du langage (*enfin, pas tout à fait mais on y reviendra…*).
- Le code d'une fonction non appelée n'est jamais exécuté.

Variable locale *vs* globale

- Une variable locale est définie dans une fonction. Elle n'existe pas en dehors de la fonction où elle est définie.
- Une variable globale est définie hors d'une fonction. Son contenu est accessible à tout moment.

```
var taux = 0.2, prixHT = 100;              // variables globales
function calculPrixTTC ( ) {
    let prixTTC = prixHT * (1+taux);       // prixTTC var. locale
    alert("Prix TTC :" + prixTTC);         // prixTTC vaut 120
}
calculPrixTTC( );
alert("Prix TTC :" + prixTTC);             // prix TTC indéfini
```

Variable locale *vs* globale

- Une variable locale est prioritaire sur une variable globale.

```
var prixHT = 100;                          // variable globale
function calculPrixTTC ( ) {
    let prixHT = 200;                      // variable locale
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);         // prixTTC vaut 240
}
calculPrixTTC( );
                        // La valeur 200 est utilisée dans le calcul
```

- En l'absence de variable locale, une variable globale est recherchée.

Variable locale *vs* globale

- L'environnement local est prioritaire sur le global :

```
var prixHT = 100;                                   // prixHT vaut 100
function calculPrixTTC () {          // fonction externe
    let prixHT = 200;                               // prixHT vaut 200
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);          // prix TTC vaut 240
    function calculPrixLuxe () {  // fonction interne
        let prixTTC = prixHT * 1.5;
        alert("Prix TTC :" + prixTTC);
    }
    calculPrixLuxe();
}
calculPrixTTC();
```

- La visibilité des variables est appelée portée lexicale

Argument d'une fonction

- Lors du passage d'un argument, une variable est créée en début de fonction pour contenir la valeur, la variable ou le résultat d'une autre fonction passé en paramètre.
- Toutes les variables créées et initialisées en début de fonction sont détruites à la fin de la fonction.
- Valeur passée en argument

```
function calculPrixTTC (prixHT) {
                // prixHT est créée et initialisée à 100
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);
                // prixHT est supprimée
}
calculPrixTTC(100);       // Appel de la fonction
```

Argument d'une fonction

- Variable passée en argument

```
function calculPrixTTC (prixHT) {        // prixHT = montant
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);
}
let montant = 100;
calculPrixTTC(montant);
```

- Retour d'une fonction passé en argument

```
function calculPrixTTC (prixHT) {
// prixHT vaut la valeur retournée par la fonction prompt()
    let prixTTC = prixHT * 1.2;
    alert("Prix TTC :" + prixTTC);
}
calculPrixTTC(prompt("Prix HT du produit ?"));
```

Argument par défaut

- Valeur par défaut d'un argument

Initialisation de la valeur d'un argument par simple affectation.
En cas d'absence de l'argument lors de l'appel de la fonction, la valeur définie par défaut est utilisée.

```
function calculPrixTTC (prixHT, TVA = 0.2  ) {
    let prixTTC = prixHT * (1 + TVA);
    alert("Prix TTC :" + prixTTC);
}
let montant = 100;
calculPrixTTC(montant, 0.5);          // Prix TTC = 150
calculPrixTTC(montant);               // Prix TTC = 120
calculPrixTTC(montant,      );        // Prix TTC = 120
calculPrixTTC(     , montant);        // Erreur !
```

Paramètre rest

- Fonction avec un nombre d'arguments variable

  JavaScript stocke les arguments dans un tableau dont le nom est mentionné après l'opérateur de décomposition …

  Les valeurs peuvent être utilisées isolément dans la fonction

```
let val1 = 1, val2 = 2, val3 = 3;
function somme(...nombres){
    let maSomme = 0;
    for (let nombre of nombres){
    maSomme += nombre;
  }
  return maSomme;
}
console.log(somme(val1, val2));          // appel avec 2 arg.
console.log(somme(val1, val2, val3));    // appel avec 3 arg.
```

Retour d'une fonction

- L'instruction return permet d'indiquer la valeur que l'on souhaite retourner.
- return met fin à l'exécution de la fonction

```
function calculPrixTTC (prixHT) {
    let prixTTC = prixHT * 1.2;
    return prixTTC;
            // La valeur de prixTTC est retournée
}
let prixProduitHT = prompt("Montant du produit HT ?");
let prixProduit = calculPrixTTC(prixProduitHT);
            // prixProduit reçoit la valeur retournée
alert("Montant du produit TTC : " + prixProduit);
```

Retour d'une fonction

- Plusieurs valeurs de retour grâce à la décomposition (appelée aussi destructuring)

```
function calculPrixTTC (prixHT) {
    let TTCBase = prixHT * 1.2;
    let TTCLuxe = prixHT * 1.33;
    return { TTCBase, TTCLuxe};
            // Les 2 valeurs de prix TTC sont retournées
}
let { TTCBase, TTCLuxe} = calculPrixTTC(100);
            // Les 2 valeurs retournées sont récupérées
console.log(TTCBase);          // TTCBase vaut 120
console.log(TTCLuxe);          // TTCLuxe vaut 133
```

Retour d'une fonction

- Plusieurs valeurs de retour grâce à la décomposition

```
function calculPrixTTC (prixHT) {
    let TTCBase = prixHT * 1.2;
    let TTCLuxe = prixHT * 1.33;
    return { TTCBase, TTCLuxe};
            // Les 2 valeurs de prix TTC sont retournées
}
let {TTCLuxe} = calculPrixTTC(100);
            // 1 seule valeur retournée est récupérée
console.log(TTCLuxe);          // TTCLuxe vaut 133
```

Fonction anonyme

- let maReference = function (argument1, argument2, …) {
  instruction(s);
  };
- Une fonction anonyme n'a pas de nom
- Son résultat est récupéré dans une variable, appelée référence, qui est utilisée pour appeler la fonction
- Utilisée pour passer des fonctions de retour en paramètre ou pour la déclaration de prototype

Fonct. anonyme vs fonct. classique

- La fonction calcul est inconnue avant déclaration

```
console.log(calcul(4,5));          // appel de calcul : erreur !
let calcul = function (val1, val2) {       // déclaration
    return val1+ val2;
};
// Uncaught ReferenceError:
can't access lexical declaration 'calcul' before initialization
```

- calcul est remontée en début de code grâce au mot clé function

```
console.log(calcul(4,5));        // appel de calcul, résultat : 9
function calcul(val1, val2) {
    return val1+ val2;
}                                          // remontée = hoisted
```

Fonction anonyme

- Exemple de fonction anonyme affectée à une variable

```
let prixProduitTTC = function (prix) {
    alert("Prix du produit TTC : " + prix * 1.2);
} ;             // ; à ne pas oublier pour terminer l'instruction !

let montantHT = prompt("Prix du produit HT : ");

prixProduitTTC(montantHT);
    // Appel de la fonction à l'aide de sa référence
```

Fonction anonyme

- Une fonction utilisée une seule fois n'a pas besoin de nom

```
function ajouter(quantite) {
  return function (nombre) {     // seul appel de la fonction
    return nombre + quantite;  };
}
let ajouter4 = ajouter(4);                    // quantite vaut 4
          //  =function (nombre) {
          //      return nombre + 4;  }
alert(ajouter4(5));                           // nombre vaut 5
                                              // affiche 9
// Réutilisation du code
let ajouter100 = ajouter(100);
alert(ajouter100(200));
```

Fonction anonyme

- Exercice : Ecrire une fonction plusPetitQue, qui prend un nombre (x) en argument et retourne une fonction anonyme qui représente le test.
  Lorsque cette fonction anonyme est appelée avec un nombre (y) comme argument, elle retourne :
  - la valeur true si y est plus petit que x
  - false sinon.

  Exemple avec x égal à 10 (plusPetitQueDix)

Fonction anonyme

- Code isolé

  Pour éviter les influences de certaines instructions sur le reste du programme, il peut être nécessaire d'isoler du code

  Cela est possible à l'aide d'une fonction anonyme immédiatement exécutée appelée aussi IIFE (*Immediately Invoked Function Expression*) ou Fonction auto-invoquée

```
( function ( ) {
    // code isolé
} ) ( );
    // ( ) désignent la fonction a exécuter
    // ( ) lancent l'exécution de la fonction anonyme
```

Fonction anonyme

- La fonction est exécutée une seule fois
- Le monde extérieur n'a PAS accès aux variables déclarées dans la fonction
  MAIS la fonction a accès aux variables globales !

```
let unBonjour = "Bonjour";
( function ( ) {
    let message = unBonjour;
    console.log( message);      // affichage de Bonjour !
} ) ( );                        // ( ) lancent l'exécution
console.log(message);           // message est undefined
```

Fonction fléchée

- Définition simplifiée et concise d'une fonction
- Réduction à sa plus simple expression
  - Omission possible des mots clés *function* et/ou *return*

```
(paramètres) => { instructions }
```

- Exemples de fonctions fléchées

```
const msgErreur = (message) => {return `Problème ${message}`}
const msgErreur = message => `Problème ${message}`
// Parenthésage inutile car 1 seul paramètre et 1 seule instruction
console.log(msgErreur("Réseau"));
```

```
const msgErreur = () =>  `Problème 404`
// fonction sans paramètre
console.log(msgErreur());
```

Closure/Fermeture

- Rappel : Les variables locales d'une fonction sont supprimées à la fin de son exécution.

```
function foisDeux () {
    let prix = 100;                    // prix vaut 100
    console.log(prix * 2);             // 200
}
foisDeux ();
console.log(prix);                     // prix undefined
```

Closure/Fermeture

- Une closure est un mécanisme propre à JS qui permet de mémoriser l'environnement local d'une fonction interne intégrée dans une fonction externe, même si elle a déjà exécutée.

```
function compter() {
    let compteur= 0;
    return function() { return ++compteur; };
}
let compteur1 = compter();     // définition d'un compteur
console.log(compteur1());              // compteur1 = 1
console.log(compteur1());              // compteur1 = 2
```

- La variable compteur est protégée et ne peut être modifiée que par l'usage de la référence compteur1()

Closure/Fermeture

- Possibilité de multiplier les références et ainsi disposer de plusieurs compteurs indépendants

```
function compter() {
    let compteur = 0;
    return function() { return ++compteur; };
}
let compteur1 = compter();              // 1er compteur
// compteur1 vaut function() { return ++compteur; }
console.log(compteur1());               // compteur1 = 1
console.log(compteur1());               // compteur1 = 2
let compteur2 = compter();              // 2ème compteur
console.log(compteur2());               // compteur2 = 1
console.log(compteur1());               // compteur1 = 3
```

Array

- Propriété de l'objet Array
  - length : nombre d'éléments dans le tableau

- Méthodes de l'objet Array
  - Array.isArray()
  - *concat()*
  - *includes()*
  - *indexOf()*
  - *lastIndexOf()*
  - *join()*
  - *pop()*
  - *push()*
  - *reverse()*
  - *shift()*
  - *unshift()*
  - *splice()*
  - *slice()*
  - *toString()*
  - *sort()*

Array

- Array.isArray(obj) : vérifie si un objet est un tableau
- *concat(t1, t2…) :* concaténation de plusieurs tableaux t1, t2…
- *includes(elt) : v*érifie si un tableau contient l'élément spécifié
- *indexOf(elt, i) :* recherche un élément dans le tableau à partir de i et renvoie son indice
- *lastIndexOf(elt) :* recherche un él^{nt} dans le tableau, depuis la fin, et renvoie son indice
- *join(sép) :* joint tous les éléments du tableau en une chaîne de caractères (avec sép)
- *pop() :* supprime le dernier élément du tableau et retourne l'élément supprimé
- *push(elts) :* ajoute des él^{nts} à la fin du tableau et renvoie la nouvelle longueur
- *shift() :* supprime le premier élément du tableau, et retourne l'élément supprimé
- *unshift(elts) :* ajoute des él^{nts} au début du tableau et renvoie la nouvelle longueur
- *splice(i, n, elts) :* ajoute/supprime n éléments dans un tableau à l'indice i
- *slice(i_déb, i_fin) :* sélectionne une partie d'un tableau et retourne le nouveau tableau
- *reverse() :* inverse l'ordre des éléments du tableau

- *toString() :* convertit un tableau en une chaîne de caractères et renvoie le résultat

Array

Application d'une fonction aux éléments d'un tableau

- *every(fct)* : vérifie si tous les él^{nts} d'un tableau satisfont la condition décrite dans fct
- *some(fct) :* vérifie si l'un des él^{nts} d'un tableau satisfait la condition décrite dans fct
- *find(fct) :* retourne la valeur du 1er él^{nt} d'un tableau qui satisfait le test décrit dans fct
- *findIndex(fct) :* retourne l'index du 1er él^{nt} d'un tableau qui satisfait le test décrit dans fct
- *filter(fct) :* crée un nouveau tableau contenant les él^{nts} du tableau vérifiant le test de fct
- *forEach(fct) :* applique une fonction fct pour chaque él^{nt} du tableau
- *map(fct) :* crée un nouveau tableau en appliquant une fct à chaque él^{nt} du tableau initial
- *reduce(fct) :* réduit les él^{nts} du tableau à une seule valeur en appliquant une fct à chaque él^{nt} du tableau et en stockant dans un accumulateur le résultat de l'itération précédente
- *sort(fct) :* trie les éléments d'un tableau d'après une fct de comparaison

Tableaux : méthodes & fonctions fléchées

- Exercice : Relevés météorologiques (datameteo.js)
  - Supprimez le 1er relevé, erroné, dans le tableau *dataMeteo*
  - Ajoutez en début de tableau le relevé suivant :
    Angleterre, Londres, -8, 30, 500, 1800

Utilisez désormais des fonctions fléchées et les méthodes intégrées à l'objet Array

- Vérifiez si toutes les températures minimales sont <= 0° C
- Vérifiez si toutes les valeurs pluviométriques sont comprises entre 0 et 3000
- Dupliquez le tableau en réhaussant les températures maximales de 2° C.
  Le nouveau tableau est nommé *dataMeteoRectif*
- Vérifiez si certaines températures maximales dépassent 40° C
  Si c'est le cas, affichez le 1er pays concerné.
  Créez un tableau *paysTMax* (avec les champs pays, capitale, temp. max) ne contenant que des pays qui vérifient cette condition (Temp. Max > 40° C)
- Affichez le tableau *dataMeteoRectif* via une page web :
  - La partie *thead* est à programmer en html
  - Le contenu de *tbody,* lignes et contenu des cellules, est à programmer en JS en créant les éléments nécessaires