# Bases javascript

## Variables

voir MDN Variables et litéraux Déclaration explicite (avec le mot clé var) : variable dont la portée dépend de l'emplacement de la déclaration (peut être locale)

```
var i=0;//variable globale
function uneFonction(){
    //Variable locale
    var varLocale;
}
```

Déclaration implicite (sans le mot clé var ⇒ déconseillé) : variable globale

```
j=0;
```

Une variable non initialisée renvoie **undefined**

var a un e portée de fonction, let une portée de bloc :

```
var i=0;//variable globale
function uneFonction(){
    //Variable locale à la fonction
    var varLocale;
    if(condition){
        let x; //x n'est connu que dans ce bloc if
    }
}
```

Déclarer une constante (toujours initialisée) :

```
const VRAI=true; //constante globale
function uneFonction(){
    //constante locale à la fonction
    const LOCALE='LOCALE';
}
```

## Fonctions

voir MDN Fonctions

Déclarations Une fonction de portée globale

```
function square(nombre) {
  return nombre * nombre;
}
```

Déclaration avec une expression de fonction :

Une fonction déclarée avec une expression

```
var square=function (nombre) {
  return nombre * nombre;
};
```

Fonction en argument et fonction anonyme :

Fonction prenant une autre fonction en argument

```
function arrayMap(f,a) {
  var result = [], // Créer un nouveau tableau Array
  var i;
  for (i = 0; i < a.length; i++)
    result[i] = f(a[i]);
  return result;
}

arrayMap(function(x) {return x * x * x}, [0, 1, 2, 5, 10]);
//retourne le tableau [0, 1, 8, 125, 1000], par passage d'une fonction anonyme.

arrayMap((x)=> x * x * x, [0, 1, 2, 5, 10]);
// avec lambda expression (Arrow function)
```

## Portée et fonctions

Il est impossible d'accéder aux variables définies dans une fonction en dehors de cette fonction : ces variables n'existent que dans la portée de la fonction. En revanche, une fonction peut accéder aux différentes variables et fonctions qui appartiennent à la portée dans laquelle elle est définie. Une fonction définie dans une autre fonction peut également accéder à toutes les variables de la fonction “parente” et à toute autre variable accessible depuis la fonction “parente”.

```
// Les variables suivantes sont globales
var num1 = 20,
    num2 = 3,
    nom = "Messi";

// Cette fonction est définie dans la portée globale
function multiplier() {
  return num1 * num2;
}

multiplier(); // Renvoie 60

// Un exemple de fonction imbriquée
function getScore () {
  var num1 = 2,
      num2 = 3;
  function ajoute() {
    return nom + " a marqué " + (num1 + num2);
  }
```

```
  return ajoute();
}

getScore(); // Renvoie "Messi a marqué 5"
```

## Closure

Une closure permet de stocker une variable locale utilisée par une fonction

```
(function() {

})();
```

Exemple : La variable locale nom (Mozilla) est stockée dans la variable maFonction, correspondant à créerFonction()

```
function créerFonction() {
  var nom = "Mozilla";
  function afficheNom() {
    alert(nom);
  }
  return afficheNom;
}

var maFonction = créerFonction();
maFonction();
```

## Classe et encapsulation

Voir classes avec ECMASCRIPT

Il est possible de définir une fonction publique accédant à des fonctions et des variables privées en utilisant des closures. Cette façon de procéder est également connue comme le patron de conception module, et permet de créer des classes :

Une classe pour créer des compteurs

```
var creerCompteur = function() {
  var compteurPrivate = 0;
  function changeValeur(val) {
    compteurPrivate += val;
  }
  return {
    increment: function() {
      changeValeur(1);
    },
    decrement: function() {
      changeValeur(-1);
    },
```

```
    valeur: function() {
      return compteurPrivate;
    }
  };
};
```

Utilisation de la classe creerCompteur : Instanciations de compteurs

```
var compteur1 = creerCompteur();
var compteur2 = creerCompteur();
alert(compteur1.valeur()); /* Affiche 0 */
compteur1.increment();
compteur1.increment();
alert(compteur1.valeur()); /* Affiche 2 */
compteur1.decrement();
alert(compteur1.valeur()); /* Affiche 1 */
alert(compteur2.valeur()); /* Affiche 0 */
```

# Prototype

Mais les closures sont coûteuse (en mémoire et temps d'exécution), il est donc préférable d'implémenter l'équivalent en utilisant le prototype : Par contre, dans ce cas, la variable compteur est redevenue publique.

```
function creerCompteur() {
  this.compteur= 0;
}
creerCompteur.prototype = {
  decrement: function() {
    this.compteur--;
  },
  increment: function() {
    this.compteur++;
  },
  valeur: function() {
    return this.compteur;
  }
};
```

Utilisation de la classe et instanciation d'un objet avec l'opérateur new :

```
var compteur=new creerCompteur();
compteur.increment();
alert(compteur.compteur);//Affiche 1 -> accès direct à compteur possible car il est
public
```

# JSON

JSON : Javascript Object Notation, utilisé par un grand nombre de langages de programmation, est un format de

données textuelles permettant la sérialisation de tableaux, d'objets, de données.

Les données sont stockées sous forme de couples clé/valeur :

```
"dataName" : "dataValue"
```

Les objets sous forme d'ensemble de couples clé/valeurs matérialisés par des accolades :

```
{
    "Obj1member1Name" : "member1Value",
    "Obj1member2Name" : "member2Value"
}
```

Les tableaux sous forme de liste d'éléments (valeurs, arrays ou objets), séparés par des virgules, et entourés de crochets [] : Tableau d'objets

```
[
    {"Obj1member1Name" : "member1Value",
    "Obj1member2Name" : "member2Value"},
    {"Obj2member1Name" : "member1Value",
    "Obj2member2Name" : "member2Value"},
]
```

Tableau associatif d'objets

```
[
    "USD": {
        "symbol": "$",
        "name": "US Dollar",
        "symbol_native": "$",
        "decimal_digits": 2,
        "rounding": 0,
        "code": "USD",
        "name_plural": "US dollars"
    },
    "CAD": {
        "symbol": "CA$",
        "name": "Canadian Dollar",
        "symbol_native": "$",
        "decimal_digits": 2,
        "rounding": 0,
        "code": "CAD",
        "name_plural": "Canadian dollars"
    }
]
```

# Tableaux

```
var values=[{"code":"AA"},{"code":"BB"}];
```

## Parcours classique

```
for(var i=0;i<values.length;i++){
   console.log( values[i].code);
}
```

## foreach

```
values.forEach(function(element) {
  console.log(element.code);
});
```