

**[mathieu.loiseau@univ-grenoble-alpes.fr](mailto:mathieu.loiseau@univ-grenoble-alpes.fr)**  
**<http://lzbk.univ-grenoble-alpes.fr>**

- Ancrer des concepts génériques de programmation accumulés depuis 2 ans
  - Variable / Type
  - Structure de contrôle
  - Fonction
  - Objet / Classe
  - Événement
- Autres concepts
  - DOM
  - Client
  - Serveur
- **Modélisation**

- Premières séances
  - Rappels, exercices, nouveaux concepts
  - Spécifications du projet
- Fin du semestre → réalisation du projet
- Projet
  - Une librairie de gestion d'une base de questions en JavaScript est mise à votre disposition (d'une manière ou d'une autre...)
  - Vous utilisez/étendez cette librairie pour créer des jeux de quizz à vidéo-projeter en classe de langue (+ exemples de questions)
    - Exemples de jeux
      - Tirs aux buts
      - Désignation aléatoire de l'équipe et calcul des scores
      - Etc.

**Qu'est-ce que vous n'avez PAS  
compris en JavaScript ?**

## **Variables, types de données et structures conditionnelles**

- **Algorithme** (D. Knuth) : Ensemble fini de règles qui donne une suite d'opérations pour résoudre un certain type de problèmes
  - Fini: un algorithme doit toujours se terminer en un nombre fini d'étapes
  - Défini: chaque pas doit être défini de manière non ambiguë
  - Entrée: un algorithme à 0 ou plusieurs entrées issues d'ensembles spécifiés d'objets
  - Sortie: un algorithme à 0 ou plusieurs sorties (quantités en relation de façon spécifiée avec les entrées)
  - Effectivité: chaque opération doit être assez basique pour être exécutée par un homme utilisant un papier et un crayon
- Un **programme** est une manière (dépendant d'un langage) de réaliser un **algorithme**.

- **Algorithme** (D. Knuth) : Ensemble fini de règles qui donne une suite d'opérations pour résoudre un certain type de problèmes
  - Fini: un algorithme doit toujours se terminer en un nombre fini d'étapes
  - Défini: chaque pas doit être défini de manière non ambiguë
  - Entrée: un algorithme à 0 ou plusieurs entrées issues d'ensembles spécifiés d'objets
  - Sortie: un algorithme à 0 ou plusieurs sorties (quantités en relation de façon spécifiée avec les entrées)
  - Effectivité: chaque opération doit être assez basique pour être exécutée par un homme utilisant un papier et un crayon
- Un **programme** est une manière (dépendant d'un langage) de réaliser un **algorithme**.

Ici, nous allons apprendre à écrire des **programmes** implémentant des algorithmes en JavaScript.

- Variable : pour conserver et réutiliser une valeur
  - Le mot clé correspondant est **var**
  - Exemple :

```
var texte = "bonjour le monde !";  
window.alert(texte);  
window.alert("re-"+texte);
```
- Chaque variable a un type
  - Définit « ce que l'on peut faire avec »
  - Permet de détecter des erreurs
  - Allocation de l'espace mémoire
  - Parfois explicite et inaltérable, en JS il est implicite et modifiable pendant l'exécution



- Pour obtenir le type d'une variable : `typeof`  
`typeof nomDeLaVariable`
  - → renvoie une `string`, qui peut valoir :
    - `"undefined"` → pas de valeur
    - `"number"` → nombre
    - `"string"` → chaîne de caractères
    - `"boolean"` → booléen (vrai ou faux)
    - `"symbol"` → mot clé du langage/appel de fonction
    - `"function"` → une fonction (cf. plus loin)
    - `"object"` → n'importe quel autre objet
- Selon le type les opérateurs n'ont pas la même sémantique

- Pour réaliser des opérations sur des variables et/ou des constantes
- En javascript, l'opération effectuée par un opérateur dépend du type des opérands
  - Nombres : +, -, /, \*, % (reste de la division entière)
  - Chaines de caractère : + → concaténation
  - Booléens :
    - && → et logique
    - || → ou logique
    - ! → non
  - Entre une variable et une expression constante :
    - = → affectation

- Pour réaliser des opérations sur des variables et/ou des constantes
- En javascript, l'opération effectuée
  - Nombres : +, -, /, \*, % (reste de la division)
  - Chaines de caractère : + → concaténation
  - Booléens :
    - && → et logique
    - || → ou logique
    - ! → non
  - Entre une variable et une expression
    - = → affectation

```
<script type="text/javascript">  
  var uneChaine, unNombre;  
  uneChaine = "bonjour le";  
  console.log("uneChaine:", uneChaine +  
    " monde !");  
  unNombre = 0;  
  unNombre = unNombre + 1;  
  console.log("unNombre:", "1" + unNombre);  
</script>
```

**Que va afficher la console du navigateur ?  
Pourquoi ?**

Parce  
que...

- Booléens → Utilisés pour les tests
  - Résultat d'une comparaison
  - Instructions conditionnelles
  - Boucles
- Les autres types peuvent être les opérandes de tests d'égalité
  - Conversion
    - Number :
      - 0 → `false`
      - 1, -1, 500 → `true`
    - String :
      - "" → `false`
      - "false", "0" → `true`
- Comparaisons
  - == → Égalité avec transtypage
  - != → Différence avec transtypage
  - === → Égalité sans transtypage
  - !== → Différence sans transtypage
  - > → supérieur
  - >= → supérieur ou égal
  - < → inférieur
  - <= → inférieur ou égal

- Permet d'exécuter une séquence d'instruction différente selon le résultat d'un test logique

```
if(/*test1*/){  
    //Séquence d'instructions à exécuter si test1 est vrai  
}  
else if(/*test2*/){  
    //Séquence d'instructions à exécuter si test1 est faux et test2 est vrai  
}  
//...  
else{  
    //Séquence d'instructions à exécuter si test1 est faux et test2 est faux  
}
```

Voir aussi  
`switch`

## **Et principes d'analyse descendante (en utilisant le DOM)**

- ≈ séquence d'instructions pouvant avoir recours à des paramètres, et renvoyant une valeur calculée en fonction de ces paramètres.

- En JavaScript :

- Définition

```
function addition(par1, par2, etc){  
    //Séquence d'instructions  
    /*Par exemple:*/  
    var resultat = par1 + par2 + etc;  
    return resultat;  
}
```

- Appel

```
console.log(addition(3,2,1));  
//affiche 6  
var sommeNb1a4 = 4 +  
addition(1,2,3);  
//renvoie une valeur ↑
```

- En JS, « procédures » et « fonctions » utilisent le même mot clé (**function**).
  - Par convention on appellera « une procédure » toute fonction qui n'a pas d'instruction **return**.

# Exercice 1 : une fonction

- Créer une fonction `estChiffre` qui renvoie `true` si un caractère est un chiffre.



- Quand le test porte sur une variable qui peut donner lieu à de multiples choix, on peut utiliser le mot-clé `switch`.
- Quand on entre dans un "`case`" on effectue les instructions jusqu'au prochain `break`.
- Le cas par défaut (non-couvert par tous les autres) s'appelle `default`.

```
function estChiffre(caractere){  
    var resultat;  
    switch(caractere.toString()){  
        case "0":  
        case "1":  
        case "2":  
        case "3":  
        case "4":  
        case "5":  
        case "6":  
        case "7":  
        case "8":  
        case "9":  
            resultat = true;  
            break ;  
        default:  
            resultat = false;  
    }  
    return resultat;  
}
```

- Syntaxe

```
for(/*initialisation*/;  
    /*condition de continuation*/;  
    /*incrément : ce qu'on fait à  
la fin de chaque tour*/) {  
    /*instructions répétées tant que la  
condition d'arrêt n'est pas atteinte.  
Attention : si les instructions ne  
permettent pas d'obtenir la condition  
d'arrêt, le programme ne s'arrête jamais */  
}
```

- Exemple

```
console.log("Afficher les  
éléments d'un tableau");  
var tab = ["A", "E", "I", "O", "U"];  
var long = tab.length ;  
for(var i=0;  
    i<long;  
    i++){/*↔ i=i+1*/  
    console.log(tab[i]);  
}
```

- Créer une fonction `estNombre` qui renvoie `true` si une chaîne de caractères représente un nombre.
  - Donnée : `mot.charAt(i)` → (i+1)<sup>e</sup> caractère d'une chaîne
- Créer une fonction `promptNombre` qui demande à l'utilisateur d'entrer un nombre et s'il saisit un nombre le renvoie sous forme de `Number`, renvoie `false` sinon.
  - Donnée : `Number("25")` → `25`

- Syntaxe

```
/*initialisation*/;
```

```
while(/*condition de continuation*/){
```

```
    /*instructions répétées tant que la condition d'arrêt n'est pas  
    atteinte. Attention : si les instructions ne permettent pas d'obtenir la  
    condition d'arrêt, le programme ne s'arrête jamais...*/
```

```
    /*incrément ce qu'on fait à la fin de chaque tour*/
```

```
}
```

## “While not stop”

- Créer une procédure `saisieNombres` qui demande à l'utilisateur d'entrer des nombres, n'accepte que les nombres et les stocke dans un tableau tant qu'il n'a pas entré "stop".