**3W ACADEMY**

# COURS JAVASCRIPT

**From**

**http://javascript.3wa.fr/**

**TABLE OF CONTENTS**

# Préface

Le monde se divise en deux catégories, ceux qui utilisent Internet, et ceux qui le font. Vous avez choisi d'entrer dans la seconde catégorie.

Cet ouvrage a pour objectif de vous donner les premières clés qui vous permettront d'appréhender les concepts de la programmation dans un environnement web. Il est destiné à toute personne un peu curieuse du fonctionnement d'Internet et ayant une pratique courante des différents usages de ce dernier (recherche d'informations, réseaux sociaux, apps mobiles, jeux,...).

A travers ces pages, l'enseignement théorique sera accompagné de différentes formes d'exercices. L'objectif est de tirer le meilleur parti de l'association entre théorie et interactivité. Nous (l'auteur et les divers contributeurs de cet ouvrage) pensons que l'enseignement par la pratique est une méthode efficace. Nous vous encourageons donc fortement à faire (et même refaire) les exercices; nous souhaitons qu'ils vous amènent à découvrir par vous même les concepts. En outre chaque concept sera expliqué par la suite (ce n'est donc pas la peine d'afficher directement la solution). Creusez vous les méninges, soyez curieux, cherchez à comprendre, ces exercices sont faits pour ça ! Et puis même si vous n'y arrivez pas du premier coups, souvenez vous que tous les grands chênes de la forêt ont, eux aussi, un jour, été des glands !

Bonne lecture!

# Notions élémentaires

Dans le vaste monde de la programmation, vous avez choisi d'entrer par la porte du web. Si vous êtes un/une utilisateur(rice) aguerri(e) d'Internet, vous avez peut-être une idée de ce qu'est un "post sur instagram", ou encore un "tweet", mais savez-vous ce qui se cache derrière les mots "serveur" et "url" ?

Dans cette partie nous allons nous interroger sur le fonctionnement global d'Internet, et nous nous attarderons sur le fonctionnement d'une page web. C'est par cet angle que nous allons aborder la programmation (dans un second temps).

# Les Pages web

## Internet est fait de pages

Au fait, qu'est-ce qu'une "page web" ?

Quiz

Question 1 of 1

Cocher la ou les affirmation(s) vraie(s) : Une page web...

- [ ] est complètement écrite avec du code binaire
- [x] est rendue (affichée) via un navigateur
- [x] pourrait être affichée dans un éditeur de texte
- [ ] utilise forcément du javascript
- [ ] est "immortelle"
- [x] est associée à une URL.

Envoyer Solution

Le "web" est un espace sur lequel nous naviguons de page de texte en page de texte donc d'***url*** en ***url***. Une **URL** c'est en fait une adresse permettant à notre **navigateur** (les logiciels Chrome, Firefox, Safari, Internet Explorer, ... sont des navigateurs) de retrouver une page parmi les innombrables pages qui composent Internet.

## Quiz

Comment catégoriser ces éléments?

**Question 1 of 1**

| | Contenu | Présentation | Interactivité |
| --- | --- | --- | --- |
| du texte | ◉ | ○ | ○ |
| une balise HTML | ○ | ◉ | ○ |
| CSS | ○ | ◉ | ○ |
| JS | ○ | ○ | ◉ |
| une image | ◉ | ○ | ○ |

Envoyer Solution

HTML, CSS, et JS, forment le trio de technologies qui est le plus répandu sur le web. Ce sont tous les trois des langages informatiques (HTML => HyperText Markup Language, CSS => Cascading Style Sheets, et JS => JavaScript), ce que l'on appelle communément "du code".

C'est le rôle du **navigateur** d'**interpréter le code** pour rendre (afficher) la page web, et permettre l'**interactivité** conforme aux souhaits du concepteur de ladite page.

Mais concrètement cette adresse qu'indique-t-elle, et où emmène-t-elle ?

## Quiz

**Question 1 of 1**

Cocher la ou les affirmation(s) vraie(s) : URL, ...

- ☑ c'est une adresse vers une machine connectée quelque part à Internet
- ☐ cela veut dire "Unidentified Research Locator"
- ☑ cela veut dire "Uniform Resource Locator"
- ☐ cela permet de communiquer avec un serveur, pour lui commander à boire

Envoyer Solution

Nous venons d'approcher deux choses importantes sur la façon globale dont Internet fonctionne. La première : il y a des machines "connectées à Internet" (la nôtre par exemple) que l'on appelle aussi des **clients** qui parlent avec d'autres machines, elles aussi connectées, que l'on appelle des **serveurs**. Tout ce petit monde se retrouve grâce à un système d'adresses : les **URL** (On dit qu'une **page** est **hébergée** sur un **serveur**, et est disponible à une **URL**). La seconde : Internet se consomme via un **navigateur** (bien que composé en partie de texte lisible)

## Quiz

Comment catégoriser ces éléments?

Question 1 of 1

| | Contenu | Présentation | Interactivité |
| --- | --- | --- | --- |
| du texte | ◉ | ○ | ○ |
| une balise HTML | ○ | ◉ | ○ |
| CSS | ○ | ◉ | ○ |
| JS | ○ | ○ | ◉ |
| une image | ◉ | ○ | ○ |

Envoyer Solution

HTML, CSS, et JS, forment le trio de technologies qui est le plus répandu sur le web. Ce sont tous les trois des langages informatiques (HTML => HyperText Markup Language, CSS => Cascading Style Sheets, et JS => JavaScript), ce que l'on appelle communément "du code".

C'est le rôle du **navigateur** d'**interpréter le code** pour rendre (afficher) la page web, et permettre l'**interactivité** conforme aux souhaits du concepteur de ladite page.

# Ce qu'il faut retenir :

- une page web est associé à une URL
- une page web est hébergée sur un **serveur** (un ordinateur)
- une page web consiste en un ensemble de ressources informatiques, dont du code, qui est interprété par un **client** (le navigateur).

# A quoi sert le code ?

## HTML (HyperText Markup Language)

Le contenu (essentiellement textuel à l'origine) est organisé dans une page via un langage à balise (markup language). Ce langage permet par exemple, de mettre en avant un titre par rapport à un paragraphe en entourant le titre de balises spécifiques (ex: la balise <h1>) et le paragraphe d'une autre balise.

Pour rendre cette information de prépondérance d'une donnée par rapport à une autre, les navigateurs, par défaut, en interprétant le code HTML, changent visuellement l'aspect d'un texte entouré de balises <h1>.

Cela s'écrit de la façon suivante dans le code HTML :

```
<h1>Titre</h1>
<p>Paragraphe</p>
```

et donne ce genre de résultat :

> # Titre
>
> Paragraphe

Il est primordial que l'information dans une page soit bien hiérarchisée. Les moteurs de recherche s'en servent pour indexer votre contenu. Et ces moteurs ne sont que des robots ; le rôle du HTML est d'apporter de la sémantique au contenu pour aider les robots d'indexation à l'analyser.*

# CSS (Cascading Style Sheets)

Si le but premier du langage HTML est de donner de la sémantique au contenu d'une page, la mise en forme, elle, doit se faire via les CSS (feuilles de styles en cascades).

Le code CSS suivant, par exemple, aura pour effet de colorer les titres compris dans des balises h1 en bleu, et les paragraphes en vert.

```
h1 {
    color: blue;
}

p {
    color: green;
}
```

ce qui donnera :

> **Titre**
>
> Paragraphe

les feuilles de styles contiennent des informations que le navigateur saura retranscrire visuellement.

Mais le CSS est capable de faire bien d'autres choses. C'est grâce à lui qu'une page est mise en forme, aussi bien dans les couleurs, que pour la disposition.*

# JS (JavaScript)

Le Javascript est, quant à lui, un langage de programmation qui permet, entre autre, de manipuler les différents éléments d'une page web. Il est, par exemple, très souvent employé pour animer certaines parties des sites (un caroussel d'images, une section qui se déroule, etc...).

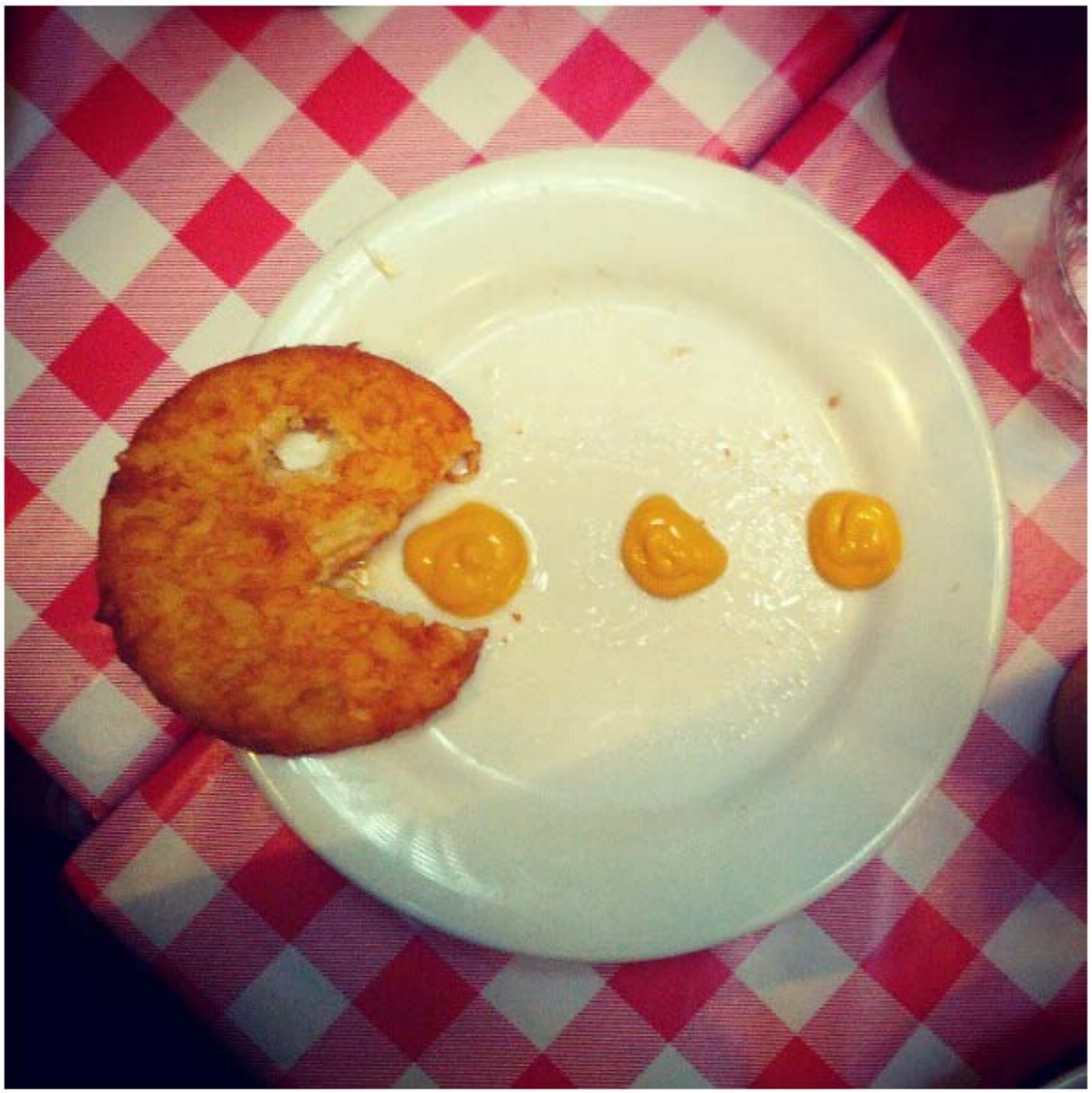
En tant que langage de programmation, c'est un moyen de communiquer avec la machine, de lui programmer des comportements. Pour faire cela il nous faut écrire du code Javascript.

un script javascript sera inclus dans une page html de la façon suivante :

```
<script src="monscript.js"></script>
```

le navigateur comprendra donc qu'il doit charger le fichier ***monscript.js***. Ce dernier devra (dans le cas présent) se trouver dans le même dossier que le fichier html qui lui sert d'hôte.

C'est dans ce fichier que nous allons écrire du code Javascript. Et c'est ce code qui, une fois interprété par le navigateur, permettra par exemple ce genre de caroussel :



ci-dessus. Diaporama d'images (elles defilent grace au JS)

## Ce qu'il faut retenir

- Les langages informatiqes sont des moyens de communiquer notre besoin à la machine pour qu'elle le traite.
- Tous ces langages sont interprétés par le navigateur (le client, par opposition au serveur)
- Le HTML et le CSS sont des langages permettant l'architecturation de l'information et sa mise en forme des pages, le Javascript permet d'en animer le contenu.

*** pour en savoir plus sur l'usage optimal des langages HTML et CSS ainsi que pour vous familiariser avec leur utilisation, reportez-vous au "mois o" dédié.**

# Présentation du javascript

## Un peu d'Histoire

Le Javascript est né dans les années 90, à l'heure des premiers navigateurs web. À l'époque ce monde était (déjà) dominé par Internet Explorer, mais une startup allait boulverser l'Histoire : Netscape.

En 1995, Sun et Netscape annoncent la sortie de JavaScript. Même si ce nouveau langage n'a de commun avec ***Java*** que certaines syntaxes, la référence à ce dernier, est "porteuse". La popularité du navigateur de Netscape (dans lequel est implémenté Javascript) force Microsoft à réagir. Ils implémentent donc à leur tour JScript (un clone de Javascript) dans Internet Explorer 3.0.

C'est en juin 1997 qu'aboutit la première version de la norme ECMAScript, qui tend a standardiser ce langage. Si aujourd'hui cette standardisation n'est pas complètement effective, c'est en bonne voie. À noter malgré tout que la version la plus répandue à ce jour est encore la 1.5 (3e version d'ECMAScript), datant de 1999.

Mais l'époque est de nouveau à l'innovation dans le domaine. À partir de 2004 et l'apparition de GMail, le Javascript a retrouvé du crédit auprès des développeurs. Il se place aujourd'hui comme une sérieuse option dans l'avenir des technologies à moyen/long terme.

## A quoi ressemble du code Javascript ?

Un script javascript est en quelque sorte un petit programme informatique. De nos jours, certains sites comportent tellement de code Javascript que l'on peut les considérer commes des programmes à part entière.

Simple script, ou programme complet, c'est un ouvrage fait de lignes de code. De la même façon qu'un livre est fait de chapitres, de pages, de paragraphes, de phrases, de mots, de lettres et de ponctuation. Un "ouvrage de code" est, de la même façon, composé de ***fichiers***, de de ***structures***, de ***blocs***, d'***instructions***, et ainsi de suite.

## Quiz

**Question 1 of 1**

L'***instruction*** (au sens langage informatique) est une "subdivision" très importante ; à quel élément suivant pourrait-on la comparer :

- ○ Un paragraphe
- ◉ Une phrase
- ○ Un mot

Envoyer Solution

Ce qui suit est une **instruction**.

```
affiche('Hello World');
```

Elle permet d'exécuter une fonction dont le rôle est d'afficher le message : Hello World.

De la même façon qu'une phrase a du sens en elle même, une instruction est, comme son nom l'indique, une petite chose à faire. Mais ainsi que la phrase prend tout son sens dans son contexte au milieu d'autres phrases, c'est un ensemble d'instructions cohérentes entres elles qui compose un programme ou un script.

Mais si une **instruction** est comparable à une phrase, que peut-on comparer à un mot?

Quand nous écrivons, nos phrases sont composées de mots ; ceux-ci étant eux mêmes "disposés" dans un certain ordre. Cet ordre répond aux règles syntaxiques du langage que nous employons (le français, l'anglais, etc...). En informatique c'est la même chose, c'est même plus simple (mais ce n'est pas toujours naturel).

Il y des règles de construction des phrases, celles-ci peuvent être composées de ***mots clés***, ou de ***ponctuation***.

## Exercice

Compléter le code en remplacent les traits de soulignement par les "mots" suivant en les remettants dans le bon ordre :

- = "Hello World"
- var
- msg

NB: Votre instructions devra se terminer par un ";" (en guise de ponctuation)

```
1  var msg = "Hello World";
2
3  affiche(msg);
```

Envoyer Solution

Comme dans nos langues vivantes, les règles syntaxiques dépendent du "type de mots". Nous verrons, au fur et à mesure de l'enrichissement de votre vocabulaire, ces règles de syntaxe.

Je préfère une entrée en matière par l'axe conceptuel. Vous comprenez aisément que les langues vivantes permettent de parler au futur ou au passé. Mais qu'est-il possible d'exprimer à un ordinateur? Ces notions temporelles n'ont pas de sens pour lui. Quels concepts comprend-t-il ? En quoi va-t-il pouvoir nous rendre service ?

# Les bases de la programmation

Nous avons parlé des différents langages qui sont mis en oeuvre dans la fabrication d'une page web. Dans la section suivante nous allons nous attaquer concrètement à ce qu'est la programmation : vous allez découvrir les concepts fondamentaux (la base de la grammaire des langages de programmation), et les mettre en oeuvre par des exercices.

Si les concepts que nous allons étudier sont communs à la majorité des langages de programmation moderne, il sont parfois implémentés quelque peu différemment selon les langages. Dans la mesure où nous travaillons dans le contexte d'un site web, nous utiliserons le langage Javascript comme support. Notez que si la majorité des points étudiés sont valables aussi bien en javascript que dans d'autres langages, ce ne sera pas le cas pour tous.

# Les variables

## Qu'est ce qu'une variable?

Avant tout, il faut comprendre qu'un ordinateur est bête, ce n'est qu'une machine. Son avantage par rapport à nous, humain, c'est qu'elle est très rapide. Cette machine sait faire une montagne de calculs en un rien de temps, mais si on ne lui indique pas ce qu'elle doit faire avec le résultat, elle n'en fera rien.

A votre avis, que va produire le code suivant :

```
((13453 * 234564) / 3 - 6410) * 5;
```

La machine va effectuer le calcul suivant, et puis rien... Le résultat ne sera ni stocké, ni affiché.

Or, il y à fort à parier que si nous avons demandé à la machinque de faire ce calcul, ce n'est pas pour la beauté du geste, mais bien pour en faire quelque chose. Nous avons donc besoin de stocker ce résultat pour l'utiliser plus tard.

Pour ce faire, les langages de programmation mettent à notre disposition les **variables** qui sont en queqlues sortes des boîtes que l'on identifiera par un nom.

On utilisera ce même nom pour faire référence au contenu qui aura été précédemment stocké dans la variable (ce que l'on aura mis dans la boîte).

> Une variable est caractérisée par un nom (identifiant) et la valeur qu'elle contient (qui pourra varier au fil de l'éxécution du programme)

# Créer des variables

On donne généralement aux variables un nom évocateur de ce qu'elles vont contenir. Quand on (re)lit du code et que l'on rencontre une variable "prix", c'est beaucoup plus parlant qu'une variable "abc". Toutefois tous les caractères ne sont pas autorisés dans les noms de variables

## Quiz

**Question 1 of 1**

Cocher les noms de variables qui selon vous seront valides (ceux qui n'ont pas de caractères interdits pour un nom de variable):

- [ ] ma belle variable
- [ ] ma-seconde-variable
- [x] peutÊtreAvecDesAccents
- [x] _3wa
- [ ] nom.avec.des.points
- [x] avec$des$symboles$dollar
- [ ] et*avec*des*etoiles

Envoyer Solution

S'il peut sembler évident qu'un nom de variable ne peut pas contenir d'espace, les autres restrictions ne sont pas aussi évidentes...

> Le nom d'une variable ne peut comporter que des lettres, des chiffres (sans pour autant commencer par un chiffre), et les symboles $ (dollar) et _ (underscore)
>
> Il est fortement recommandé de ne pas utiliser d'accent dans les noms de variables

Imaginons que vous deviez utiliser une varible qui représenterait un "taux nominal d'exploitation", logiquement vous appellerez votre variable "tauxnominalexploitation" (on a retiré le "d'" car c'était déjà assez long comme ça) et ainsi vous vous retrouvez avec un code... indéchiffrable!

Pour rendre le code plus lisible (et notamment les noms des variables), les développeurs suivent souvent une convention de nommage apellée ***camelCase***. Cette "norme" consiste à mettre en majuscule la première lettre qui suivrait le caractère d'espacement (si on avait le droit d'en mettre un), mais pas le premier caractère du nom de la variable.

## Exercice

Corrigez le nom de la variable, pour qu'il suive la convention ***camelCase***

```
1  tauxNominalExploitation
```

Envoyer Solution

Pour utiliser une variable, il faut d'abord la déclarer. Pour ce faire, il suffit d'utiliser le mot clé ***var*** suivi du nom de la nouvelle variable.

## Exercice

Déclarer une variable qui aura comme nom : maSuperVariable

```
1  var maSuperVariable;
```

Envoyer Solution

Vous aurez peut-être remarqué que si on ne respecte pas les majuscules et minuscules dans le nom de la variable, l'exercice n'est pas correct.

En effet "maSuperVariable" et "masupervariable" sont 2 variables différentes!

> Les noms de variables sont sensibles à la casse (majuscule ou minuscule)

# Comment utilise-t-on une variable?

Maintenant que nous savons créer des variables, nous voudrions les utiliser.

Pour affecter une valeur à une variable (stocker quelque chose "dans la boîte") il faut utiliser l'opérateur d'affectation, le symbole "=" (simple égal)

## Exercice

Affecter la valeur 42 dans une variable que l'on nommera "maVar"

```
1 var maVar;
2 maVar = 42;
```

Envoyer Solution

Et maintenant nous voudrions utiliser la valeur stockée dans la variable "maVar".

### Exercice

Affecter (la valeur de) "maVar" dans la variable nommée "uneAutreVar" (NB: considérer que "maVar" est déjà déclarée)

```
1  var uneAutreVar;
2  uneAutreVar = maVar;
```

Envoyer Solution

> Pour utiliser une variable, il suffit d'écrire son nom : la machine comprend, selon le contexte, que l'on veut lui affecter une valeur (quand la variable est directement à gauche d'un "simple égal"), ou bien que l'on veut lire la valeur (tous les autres cas de figure)

Et si, quand j'affecte une valeur à une variable, il y a déjà une valeur dans cette variable, que se passe-t-il?

### Exercice

Les variables "chiffreA" et "chiffreB" contiennent chacune une valeur différente. Faites en sorte d'échanger les valeurs des 2 variables.

NB: les variables chiffreA et chiffreB sont déjà déclarées.

(Indice: vous aurez peut-être besoin d'une 3e variable)

```
1  var chiffreC
2  chiffreC = chiffreA;
3  chiffreA = chiffreB;
4  chiffreB = chiffreC;
```

Envoyer Solution

Vous aurez compris que lorsque l'on affecte une valeur dans une variable, le fait qu'elle contienne déjà une valeur, ou non, n'a pas d'incidence sur l'affectation. Sans précaution particulière, l'ancienne valeur, si elle existait, est perdue.

# Ce qu'il faut retenir à propos des variables

- Un nom de variable est sensible à la casse et ne peut pas contenir n'importe quel caractère.
- Une variable doit être déclarée avant d'être utilisée.
- On affecte une valeur à une variable en la plaçant directement à gauche du signe "=" (ce qui est à droite sera la valeur affectée).

- Avant de continuer -

Dans certains exercices, des informations qui vous aideront, ou vous donneront plus d'informations sur l'objectif à atteindre, apparaitront au sein de l'éditeur de code. Ces informations seront en gris et sur des lignes commençant par // ou seront encadrées par /* et */

Ce sont des ***commentaires***, ils sont très importants ! Quand vous écrivez du code n'hésitez pas à y mettre des commentaires pour expliquer ce que le code doit faire (quand vous relirez ce code, vous me remercierez!). Ces commentaires ne sont pas pris en compte lors de l'éxécution du code, c'est pour les humains, parlez "humain"!

- Bien ! Maintenant nous pouvons continuer ! -

# Opérations sur les variables

Dans le chapitre précédent, nous avons appris à déclarer des variables pour que nous puissions y stocker des valeurs. Nous cherchions notamment à y stocker le résultat de calculs mathématiques.

En effet les ordinateurs sont de très bon calculateurs (très rapides !). Il est donc logique que les langages de programmation proposent des syntaxes pour les calculs mathématiques.

Mais pas de panique, c'est assez naturel !

## Opérations mathématiques : + - * /

Pour faire une addition, on utilise le symbole "+" entre 2 valeurs...

**Exercice**

Compléter le code pour affecter la somme de ***nbA*** et 4 à la variable ***somme***

```
1  var somme = nbA + 4;
```

Envoyer | Solution

C'était compliqué ?

Et bien ce n'est pas plus compliqué pour les autres opérations, on place l'opérateur entre deux termes (variables ou valeurs) et on affecte le résultat dans la variable à gauche du signe "="

si on veut faire des opérations plus complexes, qui rassemblent plusieurs opérations dans la même expression, les règles de priorité des opérateurs s'appliquent de la même façon qu'en algèbre.

Et du coup, si on ajoute des parenthèses ?

## Exercice

Compléter le code pour que la variable ***moy*** contienne le résultat de la moyenne des variables ***nbA***, ***nbB***, ***nbC***, ***nbD***

(la somme de toutes les variables, le tout divisé par 4)

Note : Vous devriez pouvoir faire ça en une seule ligne!

```
1  var moy = (nbA + nbB + nbC + nbD)/ 4;
```

Envoyer Solution

Rien de bien compliqué de ce côté-ci non plus, les parenthèses changent les priorités de la même façon qu'en algèbre.

# Incrémenter... Décrémenter...!

l'***incrémentation*** est une opération "classique" en informatique, il s'agit d'augmenter d'une unité (ou plus), la valeur d'une variable contenant un nombre.

## Exercice

**Décrémenter** la variable nbA d'une unité.

(le contraire d'incrémenter!)

```
1  nbA = nbA - 1;
```

Envoyer Solution

# Ce qu'il faut retenir à propos des opérations sur les variables

- Comme en mathématiques on parle des termes d'une opération pour désigner les valeurs en jeu
- Les symboles utilisés sont les mêmes qu'en algèbre classique, les règles de priorités sont également les mêmes.
- Une opération classique : l'incrémentation (et son pendant, la décrémentation) consiste à ajouter une certaine valeur à une variable et à remplacer le résultat de l'opération dans la même variable.

# Les types de variables

## Le type *Number*

Nous avons vu dans les chapitres précédents qu'il était possible de calculer puis d'affecter une valeur à une variable. Nous avons d'ailleurs expérimenté la manipulation de nombres. Nous avons donc travaillé avec des variables dites de type ***Number*** (nombre).

### Quiz

**Question 1 of 1**

Quelles valeurs, parmi celles ci-dessous correspondent selon vous au type **Number**:

- [x] 18294
- [ ] "3452"
- [x] -23
- [x] 0
- [x] 3.4453
- [ ] "mathias"

Envoyer | Solution

> Le type **Number** regroupe les nombres entiers positif ou négatif (ex : 1, -4, 23, 42, -3 543, etc...) ainsi que les nombres à virgules (ex : 1.34, 43.7, 3453,03, -32.6, etc...)

Les ordinateurs, savent donc manipuler les chiffres et les nombres, rien d'étonnant à cela : il y a même certains ordinateurs que l'on apelle des "supers calculateurs". Mais leur usage ne serait-il pas limité si l'on ne pouvait faire que des calculs algébriques?

Par ailleurs, si les entiers positifs font partie du type ***Number***, pourquoi est-ce que ***"3452"*** (dans les propositions de réponse), n'est pas de type ***Number***?

# Le type *String*

Il apparait donc assez rapidement que nous aurons besoin, dans nos programmes, de manipuler autre chose que des nombres. On peut en effet avoir besoin d'afficher, de stocker, de manipuler des mots ou des phrases, ou même des suites de chiffres qui n'ont pas de signification mathématique (un code barre par exemple)... De façon plus générale, on se rend compte que nous aurons besoin de manipuler des caractères (que ces caractères soient des lettres ou des chiffres).

Dans le quizz précédent, si vous regardez attentivement, vous remarquerez que les chiffres 3, 4, 5 et 2 sont entourés par des "guillemets doubles" (***double quotes***). Dans ce cas, les caractères 3, 4, 5, et 2 n'ont pas une signification numérique, mais sont compris par l'ordinateur comme une suite de caractères. Ces données sont regroupées dans un second type appelé ***String*** (chaine de caractères)

## Quiz

**Question 1 of 1**

Quelles valeurs, parmi celles ci-dessous correspondent selon vous au type **String**:

- [ ] #Toto#
- [x] "3452"
- [x] '-23'
- [ ] "o'
- [ ] str
- [x] "mathias"

Envoyer Solution

> Une chaine de caractères doit obligatoirement être délimitée soit par des "guillemets doubles" (symbole : ") soit par des "guillemets simples" ou apostrophe (symbole : '); aucun autre délimiteur n'est valable.

Vous aurez également noté que les délimiteurs vont par paire : si une chaine de caractères commence par un guillemet simple, elle se terminera obligatoirement par un guillemet simple, et il en va de même avec les guillemets doubles.

## Exercice

Corriger le code suivant pour qu'il ne comporte plus d'erreur

```
1  var greetings = "Bonjour, comment t'appelles tu?";
```

Envoyer Solution

Vous l'aurez compris, comme les délimiteurs vont par paires, si l'on a besoin d'utiliser une apostrophe dans une chaine de caractères, il faut que cette dernière soit délimitée par des guillemets doubles (et inversement).

Vous vous posez une question... et si j'additionne des ***string***, ça fait quoi?

## Exercice

"Additionner" les chaines de caractères ***greet*** et ***firstname*** et stocker le résultat dans la variable ***result***

```
1 var greet = "Hello ";
2 var firstname = "Mathias";
3 var result = "";
4
5 result = greet + firstname;
6
7 // le code ci dessous affichera le contenu de la variable result
8 affiche(result);
```

Envoyer Solution

Humm en effet, l'ordinateur ne fait pas vraiment une "addition" de lettres entre elles (ce qui donnerait quoi d'ailleurs... euh...). Il fait ce qui s'appelle une ***concaténation***, on "colle" les chaines bout à bout.

# Le type *Boolean*

Vous le savez sûrement, dans un ordinateur, tout est binaire, noir ou blanc, 1 ou 0, vrai ou faux.

C'est donc naturellement qu'il existe un type ***booléen*** (Boolean), du point de vue d'une machine, ça l'est, je vous assure!

Ce type regroupe seulement 2 valeurs, un peu particulières mais faciles à manipuler : ***vrai*** (true) et ***faux*** (false).

```
var maVar;

maVar = true; // la variable maVar vaut "vrai"

maVar = false; // maintenant elle vaut "faux"
```

Il n'y a pas non plus d'addition à proprement parler chez les booléens. Il n'y a d'ailleurs pas vraiment de multiplication, non plus.

Pour ces valeurs, il y a un ensemble de règles qui définissent quels sont les opérateurs et quel est leur fonctionnement, les priorités, etc... On parle d'algèbre de Boole.

Il existe principalement 2 opérateurs le ***ET*** (on utilisera "&&" dans le code) et le ***OU*** (on utilisera "||" dans le code), ça peut paraitre un peu étrange comme ça, mais en fait on peut presque tout déduire avec juste du bon sens.

## Quiz

Prenons par exemple les conditions à remplir pour la gratuité d'accès aux musées nationnaux (en France) : — avoir moins de 25 et être de nationalité d'un pays membre de l'UE — être demandeur d'emploi

**Question 1 of 2**

Pour avoir la gratuité d'accès il faut répondre à l'un des critères. On peut en deduire que le resultat de "true OR false" sera :

- (•) true
- ( ) false

**Question 2 of 2**

Pour remplir le premier critère il faut à la fois avoir moins de 25 et être de nationnalité d'un pays de l'UE. On peut donc en deduire que le résultat de "true AND false" sera :

- ( ) true
- (•) false

Envoyer Solution

Voici les tables complètes pour ces 2 opérations

| ***AND*** | **true** | **false** |
| --- | --- | --- |
| true | true | false |
| false | false | false |

| ***OR*** | **true** | **false** |
| --- | --- | --- |
| true | true | true |
| false | true | false |

vraiment pratique, vous verrez!

# Ce qu'il faut retenir à propos des types de variables

- Il existe différents types de variable (vous connaissez pour l'instant les types Number, String, et Boolean)
- En javascript, le type d'une variable n'est pas figé (il change au gré des valeurs que l'on affecte à la variable)
- Le comportement des opérateurs peut varier en fonction du type des variables (une ***addition*** n'est pas identique à une ***concaténation***, même si c'est le même opérateur)
- les opérations entre booléens se font selon les règles de l'algèbre de Boole.

# Les tableaux

On s'aperçoit rapidement que la manipulation des variables que nous avons vue jusqu'à maintenant ne sera pas aisée dès que nous devrons manipuler des données plus complexes.

Par exemple, si l'on veut écrire un programme qui aura pour rôle de faire des opérations sur une liste d'élèves et leurs notes respectives. Les variables devant être déclarées au moment où l'on écrit le code (cela ne peut pas se faire dynamiquement), comment alors créer autant de variables que d'élèves (alors qu'au moment d'écrire le code nous ne savons pas combien il y aura d'élèves)?

Pour répondre à cette problématique, il existe un type de variable dont nous n'avons pas encore parlé : le type "Array" (tableau).

Les tableaux sont des sortes d'agrégats de variables : plusieurs valeurs sont regroupées sous un même nom de variable.

Donc si les "tableaux" sont un type de variable, quand on utilise des tableaux il faut donc leur donner un nom et les déclarer!

## Quiz

**Question 1 of 1**

Cocher la/les affirmation(s) vraie(s) à propos du code suivant

```
var monPremierTableau = [];
```

- [x] Nous avons déclaré une variable
- [ ] Le fait qu'elle comporte le mot Tableau en a fait un tableau
- [x] Nous lui affectons une valeur tout de suite après sa déclaration
- [ ] Cette valeur est "vide" une valeur spéciale en attendant d'y mettre un tableau
- [x] Cette valeur représente un tableau vide (qui ne contient aucune valeur pour le moment)

Envoyer Solution

En effet, pour "décrire" un tableau en javascript, on utilise les caractères "[" (crochet ouvrant) "]" (crochet fermant) en guise de délimitation. On parle de la "notation tableau".

Pour remplir le tableau, il faudra donc mettre les valeurs entre les crochets, et on utilisera le signe "," (virgule) pour séparer les différentes valeurs

## Exercice

Compléter le code pour que le tableau contienne les valeurs 10, 5, et 3 (dans cet ordre)

```
1  var monPremierTableau = [10, 5, 3];
```

Envoyer Solution

Maintenant que notre tableau contient plusieurs valeurs, nous voudrions les utiliser. Or l'ordinateur ne sera pas capable de savoir quelle valeur choisir si on utilise juste le nom "monPremierTableau" (ce nom fait maintenant référence au tableau dans sa globalité). On utilisera alors un indice (ou index) pour identifier chacune des valeurs dans le tableau.

## Quiz

### Question 1 of 1

Cocher la/les affirmation(s) vraie(s) à propos du code suivant

```
var monPremierTableau = [10, 5, 3];
var val1 = monPremierTableau[1];
```

- [ ] Après l'exécution du code, le tableau ne contiendra plus qu'une seule valeur
- [ ] On affecte le tableau dans son ensemble à la variable val1
- [x] La variable val1 sera de type Number
- [ ] La variable val1 contiendra la valeur 10
- [x] La variable val1 contiendra la valeur du tableau qui est à l'indice 1
- [x] La variable val1 contiendra la valeur 5

Envoyer Solution

En rajoutant derrière le nom de la variable un indice entre crochet, on indique que l'on fait référence non plus au tableau dans sa globalité, mais juste à la valeur du tableau qui est à l'indice précisé.

Attention à ne pas confondre la "notation tableau" (ex : [val1, val2, ...] — décrit un tableau et son contenu) avec cette notation qui permet de préciser sur quel indice du tableau nous allons travailler.

Il faut noter également une petite subtilité en informatique, les indices de tableau commencent toujours à 0, le premier élément du tableau est donc stocké à l'indice 0 du tableau, à l'indice 1 on trouvera donc le second élément.

soit pour le tableau suivant :

```
var tab = [10, 5, 3];
```

on aura

| indice | tab[indice] | valeur |
|---|---|---|
| 0 | tab[0] | 10 |
| 1 | tab[1] | 5 |
| 2 | tab[2] | 3 |

À noter que lors de l'initialisation du tableau, on ne fait nullement mention des indices (mais uniquement des valeurs) ces derniers sont gérés automatiquement en fonction de l'ordre des valeurs.

Le nombre d'éléments dans un tableau n'est pas figé, à tout moment on peut écrire dans un tableau, et à l'instar de la lecture des valeurs d'un tableau, il faut préciser à quel indice on veut écrire.

Si l'indice n'existe pas (pas de valeur précédemment stockée à cet indice) alors, l'emplacement est créé pour nous. Si en revanche il y a déjà une valeur qui est stockée (à l'index demandé), le tableau se comportera alors comme n'importe quelle variable et l'ancienne valeur (et uniquement la valeur à l'indice indiqué) sera écrasée par la nouvelle.

# La longueur du tableau

Si on peut ajouter des valeurs dans un tableau sans avoir jamais à lui déclarer de taille, il peut être intéressant d'avoir un moyen de connaître le nombre total d'éléments dans le tableau.

Pour ce faire les tableaux ont une propriété spéciale, la propriété "length". Elle s'utilise de la façon suivante :

```
var monTab = [10, 5, 3];
var longueurTab = monTab.length;
```

La variable "longueurTab" sera de type Number et contiendra la valeur 3.

Pour connaitre la longueur (le nombre d'éléments) d'un tableau, ajouter ".length" après le nom du tableau.

## Quiz

**Question 1 of 1**

Soit le code suivant, quelle sera la longueur du tableau monTab

```
var monTab = [];
monTab[2] = 5;
```

- [ ] 1
- [ ] 2
- [x] 3

Envoyer Solution

En effet les indices de tableaux commencent toujours à 0 et ne sont jamais discontinus.

À noter que si le dernier indice d'un tableau est 2, la longueur du tableau sera de 3. Et à l'inverse, avec la longueur du tableau, on peut déduire le dernier indice qui sera donc "longueur (3) - 1 = dernier indice (2)"

# Quel type de donnée peut-on stocker dans un tableau?

Jusqu'à maintenant nous n'avons stocké dans les tableaux que des valeurs entières (de type Number). Mais en réalité un tableau peut contenir tout type de donnée. En javascript, on peut même mettre différents types de données dans un même tableau :

## Exercice

Déclarer un tableau monTab, lui affecter la chaine de caractères "Mathias" à l'indice 0, puis la valeur 30 à l'indice 1.

```
1  var monTab = [];
2  monTab[0] = "Mathias";
3  monTab[1] = 30;
4  // on aurait également pu le faire en une seule ligne :
5  // var monTab = ["Mathias", 30];
```

Envoyer Solution

Si les valeurs contenues dans un tableau peuvent être de tout type, pourrait-on mettre dans un tableau un autre tableau?

## Exercice

Compléter le code pour mettre les valeurs 10, 5 et 3 (dans cet ordre) dans le tableau numsTab; et mettre ce dernier à l'indice 1 de l'autre tableau (tabParent).

```
1  var numsTab = [10, 5, 3];
2  var tabParent = [];
3
4  tabParent[1] = numsTab;
```

Envoyer Solution

oui, les tableaux peuvent s'imbriquer les uns dans les autres. Un tableau peut accueillir tous les types de données sans limitation.

# Ce qu'il faut retenir à propos du type Array (tableau)

- Les tableaux sont des variables qui peuvent agréger plusieurs valeurs
- les différentes "cases" du tableau sont repérées par un indice (ou index)
- La propriété ".length" donne la nombre d'éléments dans un tableau
- On peut mettre n'importe quel type de donnée dans un tableau (les tableaux peuvent donc s'imbriquer les uns dans les autres)

# Les structures conditionnelles "if"

Bon, est-ce vraiment bête un ordinateur ? Quand même, ça à l'air de faire beaucoup de choses, ça doit bien réfléchir un peu !

Humm, et bien, on peut faire en sorte que la machine analyse la situation et réagisse différemment selon des scénarii que vous aurez définis.

## Quiz

**Question 1 of 1**

Si on exécute le code suivant, que va-t-il se passer ?:

```
if (ageCapitaine > 50)
{
    affiche("le capitaine n'est plus tout jeune");
}
```

- affiche systèmatiquement "le capitaine n'est plus tout jeune"
- n'affiche jamais rien
- affichera "le capitaine n'est plus tout jeune" si ageCapitaine est strictement supérieur à 50

Envoyer Solution

Jusqu'à maintenant, vous aviez peut être remarqué que l'exécution du code était linéaire, toutes les lignes de code étaient traitées de haut en bas sans exception.

Grâce à la structure ***if*** nous avons la possibilité de conditionner l'exécution d'une partie du code (le code qui est compris entre ***{*** et ***}***). En effet ce code ne sera exécuté que si la **condition** (ce qui se trouve ***entre les parenthèses*** juste après le mot clé ***if***) est vérifiée.

concrètement, voici comment se déroule l'exécution du code (le même que dans le quiz précédent)

```
1. if (ageCapitaine > 50)
2. {
3.    affiche("le capitaine n'est plus tout jeune");
4. }
5. // suite du code
```

Sur la première ligne la variable ***ageCapitaine*** est comparée à la valeur 50, dans le cas où la **condition est vérifiée** (si l'âge du capitaine est strictement supérieur à 50) le code compris entre { (ligne 2) et } (ligne 4) sera exécuté sinon, il ne le sera pas (on saute directement à la ligne 5). Enfin, que la condition soit vérifiée ou non, on continue normalement (à partir de la ligne 5).

La syntaxe du ***if*** utilise un ***bloc de code*** (le code qui est compris entre accolades). On dit que l'exécution du code est conditionnée par le ***if*** qui le précède.

On met à la suite

- le mot clé ***if***.
- la ***condition*** qui doit être entre parenthèses.
- le ***bloc de code*** qui doit être exécuté si la condition est vérifiée.

À noter que l'on rencontrera plus tard des blocs de code associés à d'autres mots clés que le ***if***

# Les opérateurs de comparaison

Nous pouvons donc comparer des valeurs entres elles avec l'opérateur ">" (supérieur à). Mais ce n'est pas le seul...

Voici la liste des opérateurs de comparaison qui existe :

| Opérateurs | Significations |
|---|---|
| > | Strictement supérieur à ... |
| < | Strictement inférieur à ... |
| >= | Supérieur ou égal à ... |
| <= | Inférieur ou égal à ... |
| == | Équivalent à ... |
| === | Strictement égal à ... |
| != | Non équivalent à ... |
| !== | Strictement différent de ... |

Les opérateurs de comparaison fonctionnent de façon assez similaire à leurs cousins arithmétiques. Ils mettent en jeu 2 termes qui viennent se placer de part et d'autre de l'opérateur, et produisent un résultat.

Lors d'une opération de comparaison le résultat est toujours un booléen qui indique si oui, ou non ***l'expression est vérifiée***.

exemples :

```
ageCapitaine > 50; // vaut vrai si l'age du capitaine est strictement supérieur à 50
30 !== ageCapitaine; // vaut vrai si l'age du capitaine est différent de 30
```

## Exercice

Compléter l'extrait de code suivant pour afficher le message "Vous avez gagné!" uniquement si la variable votreScore est strictement supérieure à la variable monScore

```
1  if (monScore < votreScore) {
2      msg = "Vous avez gagné!";
3  }
4
5  // ATTENTION : ne pas modifier en dessous de cette ligne
6  affiche(msg);}
```

Envoyer Solution

Ce n'est pas très compliqué convenons-en, mais je vois bien qu'il y a quelques questions qui restent en suspens... Pourquoi est ce que pour l'égalité c'est "===" (trois signes égal accolés)? Qu'est-ce que l'opérateur d'équivalence (celui avec seulement 2 signes égal accolés)?

Et bien mettons nous en situation et voyons comment cela fonctionne.

## Exercice

corriger le code suivant pour que le message soit affiché

```
1  var num = 15;
2  var numVerif = 15;
3
4  // ne pas changer la condition
5  if (num === numVerif) {
6      numHasBeenVerified = true; // faite passer cette variable à true si la condition est bien
7      affiche("Hello, and welcome on board!");
8  }
```

Envoyer Solution

Dans l'exercice précédent le code original ne marche pas, car les variables ***num*** et ***numVerif*** sont de type différent. En effet quand l'une est un Number (un nombre), l'autre est une String (une chaine de caractère).

L'opérateur d'égalité strict va, quand on lui soumet 2 valeurs, comparer le type de donnée, puis leur valeur effective. Si les 2 termes de la comparaison sont identiques en tous points (type de donnée et valeur) l'évaluation de l'opération donne la valeur vraie.

Pour corriger le code et faire s'afficher le message, il y avait donc plusieurs possibilités.

La première d'entre elles : changer d'opérateur. En utilisant l'opérateur d'équivalence (le "==" double égal) l'ordinateur comprend qu'il va potentiellement avoir à comparer des termes ayant des types de données différents. Et comme il est logique (l'ordinateur ne compare pas ensemble des choux avec des pommes... bêtement logique!), il convertit automatiquement une des variables et utilisera le résultat de la conversion pour la comparaison.

Bien que cet opérateur soit "pratique", il est vivement recommandé de ne l'utiliser qu'avec la plus grande prudence : la conversion automatique de type (transtypage) répond à des règles parfois "étranges" (et pour le coup pas toujours logiques!).

## Quiz

Qu'affichera le code ci dessous... :

```
if (val1 == val2) {
    affiche("valeurs équivalentes");
}
```

### Question 1 of 4

... pour les valeurs :

```
var val1 = 0, val2 = false
```

- (•) "valeurs équivalentes“
- ( ) rien

### Question 2 of 4

... pour les valeurs :

```
var val1 = "zero", val2 = 0
```

- ( ) "valeurs équivalentes“
- (•) rien

Question 3 of 4

... pour les valeurs :

```
var val1 = "2zero", val2 = 20
```

- ○ "valeurs équivalentes“
- ◉ rien

Question 4 of 4

... pour les valeurs :

```
var val1 = "20", val2 = 20
```

- ◉ "valeurs équivalentes“
- ○ rien

Envoyer Solution

En javascript, les variables pouvant changer de types dynamiquement (à l'éxécution, au gré des valeurs), il faut au maximum éviter de se reposer sur la conversion automatique mise en oeuvre par l'opérateur d'équivalence.

# Expressions conditionelles avancées

Malheureusement toutes les conditions ne se résument pas à une simple comparaison entre 2 valeurs, c'est souvent plus compliqué. On sera amené à combiner plusieurs comparaisons.

Pour définir un intervalle par exemple, imaginons que nous devions vérifier que l'âge (dans la variable ageCapitaine) est compris entre 25 et 59 ans. Un opérateur de comparaison ne fonctionne qu'avec 2 termes, on sait donc faire : – d'une part

```
ageCapitaine >= 25 // qui sera évalué à true ou false
```

– d'autre part

```
ageCapitaine <= 59 // qui sera évalué à true ou false
```

or nous avons vu qu'il était possible de faire des opérations avec les valeurs booléennes.

```
if (ageCapitaine >= 25 && ageCapitaine <= 59) { /*...*/ }
```

On utilise les opérateurs booléens entre des expressions de comparaison. Ainsi une fois les expressions évaluées, l'ordinateur va effectuer les opérations booléeenes (selon les règles de l'algèbre de Boole).

## Exercice

Complétez le code (remplacer les commentaires dans la condition du if) pour que la variable msg contienne la bonne valeur.

```
1  // la variable droids a été déclarée et est un tableau de chaines de caractères
2
3  var msg = "these are not the droid we are looking for";
4  if(droids[0] == "r2d2" || droids[1] == "c3po") {
5      msg = "these are the droid we are looking for!"
6  }
```

Envoyer Solution

# Et sinon ?

Le ***if*** est une structure fondementale, elle permet d'exécuter une portion de code si et seulement si une certaine condition est vérifiée. La question qui se pose donc désormais est : que peut-on faire dans le cas inverse (la condition n'est pas vérifiée) ?

## Exercice

compléter le code pour que la variable maVar prenne la valeur "est équivalent à true" ou "n'est pas équivalent à true" en fonction de la valeur de maVar

```
1  //Note: maVar, et msg sont déclarées, la valeur de maVar est générée aléatoirement
2
3  if (maVar == true) {
4      msg = "est équivalent à true";
5  }
6  if (maVar != true) {
7      msg = "n'est pas équivalent à true";
8  }
9
10 // Ne pas modifier en dessous de cette ligne
11 // affiche la chaine contenu dans la variable msg
12 affiche(msg);
```

Envoyer Solution

Il y a plusieurs solutions pour répondre au problème posé dans l'exercice précédent. La plus simple (dans l'état actuelle de ce que nous savons) consisterait à ajouter un autre ***if*** juste après la fermeture du premier. Ce second ***if***, lui, testerait la condition inverse à celle du premier : le premier compare les valeurs avec l'opérateur == le second devra comparer les mêmes valeurs avec l'opérateur ***!=***

Toutefois cette solution, bien que fonctionnellement correcte (le programme a bien le comportement attendu), n'est pas la meilleure :

En effet, une fois que les valeurs auront été comparées pour déterminer si elles sont équivalentes (le premier ***if*** avec ==), l'ordinateur devra refaire une comparaison de ces 2 mêmes valeurs pour déterminer si elles sont "non équivalentes" (le second ***if*** avec ***!=***).

On comprend que l'on demande à l'ordinateur de faire plusieurs fois un même traitement (la comparaison) alors qu'il n'y a pas besoin : les 2 cas de figure étant contraire l'un de l'autre (si la première condition est vérifiée, la seconde est obligatoirement fausse, et inversement).

Il existe donc un mot clé ***else*** qui permet de répondre à cette problématique, on s'en sert ainsi :

```
if (maVar == true) {
    msg = "maVar est équivalent à true";
}
else {
    msg = "maVar est équivalent à false";
}
```

le mot clé ***else*** ne peut pas exister sans qu'il soit associé à un ***if***, il est la jointure qui relie les blocs de code à exécuter en cas de condition vraie (celui juste après la condition du if), et le bloc à exécuter dans le cas contraire (le bloc de code après le ***else***)

# Et si j'ai plus de 2 cas de figures ?

Imaginons que nous ayons une variable de type Number qui contienne un "code pays". Nous souhaiterions afficher le nom du pays correspondant (et pas son code), il faudrait que nous fassions une suite de if / else qui ressemblerait à quelque chose comme ça :

```
// la valeur de code pays est initialisée en amont dans le code
if (codePays == 1) {
    // affiche "France"
}
else {
    if (codePays == 2) {
       // affiche "Belgique"
    }
    else {
        if (codePays == 3) {
            // affiche "Espagne"
        }
        // else ...
    }
}
```

Dans le cas présent, on n'a que 3 pays, mais c'est déjà assez pour se rendre compte que ça peut vite devenir illisible.

Pour remédier à ce problème, il existe une structure appellée "switch" qui permet de gérer ce genre de situation avec un code plus lisible.

le code suivant (avec le "switch") est équivalent au code précédent (avec les "if")

```
switch (codePays) {
    case (1):
        // affiche "France"
        break;
    case (2):
        // affiche "Belgique"
        break;
    case (3):
        // affiche "Espagne"
        break;
    // case (...):
        // ...
        // break;
    default:
        // affiche "Code pays inconnu"
}
```

Avec cette façon de faire, la condition se fait toujours sur la variable codePays, et pour chaque "case" (cas de figure), on teste l'équivalence de la variable (en l'occurrence codePays) avec la valeur du cas (les cas étant étudiés les un après les autres).

Une fois que l'on a trouvé un cas qui convient (quand codePays est équivalent à la valeur saisie entre les parenthèses du cas) on éxécute le code qui correspond. L'instruction ***break*** indique que ce n'est pas la peine de tester les cas suivants (si il n'y en avait pas, les autres cas seraient étudiés, et potentiellement l'un d'eux pourrait marcher)

On voit par ailleurs qu'il y a un cas un peu particulier à la fin de la structure : le mot clé "default", c'est une façon d'avoir un bloc de code par défaut; il fonctionne comme les autres cas sauf qu'on ne lui associe pas de valeur particulière. À noter toutefois que ce mot clé n'est pas obligatoire.

# Ce qu'il faut retenir à propos des "if"

- Les structures conditionnelles cassent la linéarité de l'exécution du code (certains blocs de code ne seront pas forcément exécutés)
- L'évaluation des conditions utilise des opérateurs de comparaison
- L'opérateur == (équivalence) est différent de l'opérateur === (égalité stricte)
- On peut traiter plusieurs cas sans répéter à chaque fois toute l'expression soit avec le mot clé "else" soit avec la structure "switch"

# Les boucles

Comme nous l'avons vu les ordinateurs sont particulièrement utiles quand il s'agit de faire des suites d'opérations et de manipuler des variables. Imaginons que nous ayons le même traitement à faire pour un certain nombre de valeurs (afficher tous les élements d'un tableau).

Il me faudrait donc une structure de code qui permettrait d'isoler les actions à faire dans un bloc et de préciser à l'ordinateur qu'il faudrait exécuter ce bloc autant de fois que nécessaire (pour chaque valeur du tableau par exemple).

Pour ce faire les langages informatiques disposent de différentes instructions regroupées sous le terme générique de boucle.

Une boucle c'est donc une structure constituée d'un bloc de code qui va potentiellement s'exécuter plusieurs fois à la suite. et d'une expression qui va conditionner l'exécution ou la réexécution de ce bloc.

Chaque éxécution du bloc de code associé à une boucle est appellée une itération. On parlera alors de la condition d'arrêt des itérations, ou de leur nombre, pour qualifier une boucle.

# la boucle "for"

Imaginons que nous voulions afficher tous les nombres entre 1 et 50 (inclus); l'idée serait donc de faire une boucle : un bloc de code qui aurait pour rôle d'afficher le contenu d'une variable et qui se répéterait autant de fois que nécessaire.

pour ce faire nous allons écrire quelque chose comme ceci :

```
for (var i = 1; i <= 50; i = i + 1) {
    // code pour afficher la valeur de i
}
// quand la boucle aura fini ses itérations le code reprend ici
```

Que se passe-t-il dans le code précédent? eh bien nous avons utilisé la boucle ***for***, cette boucle se compose ainsi :

- le mot clé ***for***
- la déclaration d'une variable qui va servir de compteur
- une condition qui tant que l'expression est évaluée à vrai, provoque une nouvelle éxécution de la boucle.
- un "pas", une instruction d'incrémentation du compteur (cette instruction sera automatiquement exécutée juste après chaque exécution du bloc de code de la boucle)
- le bloc de code à répéter

On pourrait imaginer "traduire" ce code en français par : pour "i" variant de 1 à 50 inclus (tant que i <= 50) avec un pas de 1; à chaque pas on fait quelque chose (ici, on affiche la valeur de "i")

La boucle for nous permet donc de parcourir un ensemble de valeur avec un pas (une valeur d'incrémentation à chaque tour de boucle).

Ne pourrait-on pas s'en servir pour parcourir toutes les valeurs d'un tableau

## Exercice

Soit le tableau monTab (ci dessous); compléter le code pour concaténer bout à bout dans la variable "chaineResult" toutes les chaines contenues dans le tableau monTab.

```
1  // var monTab = [ ... ]; - monTab est un tableau constitué de 3 chaines de caractères;
2  var chaineResult = "";
3  for (var i=0; i <= monTab.length - 1; i = i + 1) {
4      chaineResult = chaineResult + monTab[i];
5  }
6  affiche(chaineResult);
```

Envoyer Solution

Comme nous l'avons appris précédemment, pour tous les tableaux, les indices commencent à 0 et vont jusqu'à "tab.length – 1" (où tab représente notre tableau).

Or nous voudrions concaténer la valeur du tableau à chacun des indices du tableau variant de 0 à tab.length – 1 inclus et avec un pas de 1.

La boucle "for" nous permet donc de répéter un bloc d'instructions un certain nombre de fois. Ce nombre de fois devra être calculé avant de commencer les itérations, et ne devrait pas varier au cours de l'éxécution des itérations.

Si Par contre on ne peut pas calculer à l'avance combien d'itérations seront nécéssaires pour arriver à nos fins, il faut employer un autre type de boucle.

Si par exemple on nous donne un tableau de personnes (une liste de nom sous forme de chaine de caractères). Nous voudrions savoir si le nom "Mathias" y figure. Et surtout nous voudrions que la boucle s'arrête dès qu'elle rencontre le nom dans le tableau (inutile de perdre plus de temps).

# La boucle "while"

C'est un autre type de boucle, ce sont les boucles qui commencent par le mot clé ***"while"*** et que l'on pourra "traduire" par "tant que". Elle s'utilise ainsi (en reprenant l'exemple de la recherche dans une liste).

```
// on déclare un compteur
var i = 0;

// on répète les itérations de la boucle tant que
// le compteur i n'a pas atteint la valeur max
// et que la valeur du tableau à l'indice i est différente de "Mathias"
while (i < tabPersonnes.length && tabPersonnes[i] != "Mathias") {
    // j'incrémente le compteur
    i = i + 1;
}
// quand la boucle aura fini ses itérations le code reprend ici
```

Dans la condition j'indique que je veux faire des itérations ***tant que*** "i" est toujours compris entre les indices mini et maxi du tableau (commençant au mini et incrémentant jusqu'au maxi) et que la valeur à l'indice en cours est différente de la valeur que je cherche.

Quand le programme sera sorti de cette boucle, si "i" est toujours strictement inférieur à la longueur du tableau (l'indice maximum), c'est que tabPersonnes[i] est égal à "Mathias", on a trouvé la position de "Mathias" dans le tableau de référence : il est à l'indice "i".

Dans notre exemple ci dessus, la boucle commence donc par le mot clé "while", puis ensuite sur la même ligne, entre parenthèses, se trouve la condition, et enfin le bloc de code qui doit être éventuellement répété.

Une fois avant chaque itération la condition est évaluée et si le résultat est "false", les itérations s'arrêtent immédiatement et le cours de l'éxécution reprend après la boucle.

## Exercice

Compléter le code pour trouver l'indice où il faudrait insérer la valeur de la variable "prenom", pour que le tableau "tabPrenoms" reste trié dans l'ordre alphabétique.

Vous devrez utiliser la variable "cpt" pour compter le nombre d'itération(s) effectuée(s) Et mettre le resultat (l'indice auquel il faudrait insérer) dans la variable insertAt

```
1  var personne = "Jules";
2  var tabPrenoms = [  "Ana",
3                      "André",
4                      "Beata",
5                      "Bertrand",
6                      "Julie",
7                      "Mathias",
8                      "Patricia",
9                      "Victor"  ];
10
11 var insertAt; // doit contenir l'indice où il faudrait insérer
12 var cpt = 0; // doit contenir le nombre d'itération(s) effectuée(s)
13
14 while (cpt < tabPrenoms.length && tabPrenoms[cpt] < personne) {
15     // j'incrémente le compteur
16     cpt = cpt + 1;
17 }
18 insertAt = cpt;
```

Envoyer Solution

Comme dans le précédent exemple, nous avons parcouru le tableau tant que les valeurs que nous trouvions ne remplissaient pas le critère (valeur du tableau inférieur à valeur à insérer). Mais que se serait-il passé si aucune des valeurs dans le tableau ne remplissait le critère ? À force d'être incrémentée dans la boucle, à un moment, la variable cpt aurait égalé la longueur du tableau, et c'est la première partie de l'expression (cpt < tabPrenoms.length) qui aurait été évalué à "false", ce qui aurait provoqué l'arrêt des itérations.

# Les boucles infinies

De façon générale, il est très important, avec les boucles, de faire attention à ce que l'on appelle les "boucles infinies". Dans l'exemple précédent (recherche de position dans l'ordre alphabétique), imaginons que l'on simplifie la condition en enlevant "cpt < tabPrenoms.length"; mais aussi que l'on ait "Vincent" dans la variable "personne".

La boucle pourrait tourner (virtuellement) à l'infini sans jamais trouver de valeur qui réponde à la condition suivante : tabPrenoms[cpt] < personne

Les boucles de ce type, celles dont les conditions ne permmettent à aucun moment ou pour aucune valeur de sortir de la boucle, s'appellent des boucles infinies.

Dans la réalité des faits, cela provoque une erreur de la machine, ou du programme.

# Ce qu'il faut retenir à propos des boucles

- Elles servent à répéter un bloc de code un nombre indéterminé de fois.
- Le nombre de répétitions est lié à une condition et au résultat de son évaluation au fil des itérations.
- Attention aux boucles infinies

# Les fonctions

Nous commençons à connaitre un bon nombre d'outils pour résoudre de petits problèmes via un programme informatique (variables, boucles, tableaux, etc...). Mais si on voit un peu plus grand, à l'échelle d'un programme de gestion de classe et d'élèves par exemple, on imagine aisément qu'il y aura des traitements (ex: calcul de moyenne) qui vont être faits à différents moments, et avec différents jeux de données. Pourtant, pour chacun de ces traitements les instructions à effectuer seront les mêmes : somme des éléments parcourus avec une boucle puis division par le nombre d'éléments.

Nous allons donc répéter plusieurs fois, et à différents endroits du code un ensemble d'instructions similaires.

Pour éviter de se répéter, les langages de programmation offrent la possibilité de définir des ***fonctions***.

Exemple de "fonction" en javascript

```
1. function doNothing()
2. {
3.     // il n'y a rien à faire...
4. }
```

Une fonction consiste en 2 éléments:

— D'une part la ***signature*** de la fonction :

```
1. function doNothing()
```

C'est le mot clé ***fonction*** suivi du ***nom*** suivi, entre ***parenthèses*** des ***paramètres*** (nous reviendrons sur les paramètres).

— D'autre part le corps de la fonction:

```
2. {
3.     // il n'y a rien à faire...
4. }
```

Entre accolades comme tout "***bloc de code***" qui se respecte, c'est la liste des ***instructions à exécuter*** à chaque fois que l'on ***invoque*** la fonction

Jusqu'à maintenant nous n'étions pas capables d'écrire un petit morceau de code pour "mettre de coté" et déclencher son éxécution à la demande. Grâce aux fonctions c'est maintenant le cas. Il paraît donc évident maintenant que pour déclencher l'éxécution du code (pour invoquer la fonction) il va falloir utiliser le nom de la fonction.

```
// simplement comme ceci
doNothing();
```

Simplement le nom de la fonction suivi d'un jeu de parenthèses. Mais que peut-il donc y avoir entre ces parenthèses?

On pourra définir dans la signature de chaque fonction un certain nombre de ***paramètres***. Ces ***variables*** (car les paramètres sont des variables) sont valables uniquement pendant l'éxécution de la fonction. Après l'éxécution de la fonction les variables sont détruites.

```
var c;
function somme (a, b) {
    c = a + b;
}
somme(10, 5);
affiche(c);
```

Les variables "a" et "b" sont des paramètres de la fonction "somme". Dans la fonction la variable "a" prend la valeur qui a été passée en paramètre au moment de l'invocation de la fonction (en l'occurrence la valeur 10 pour "a" et 5 pour "b").

## Quiz

### Question 1 of 1

Cochez la(les) affirmation(s) vraie(s) à propos de l'exemple de code suivant.

```
var c;
var a = 2;
function somme (a, b) {
    c = a + b;
}
affiche(a);
somme(10, 5);
affiche(c);
```

- [x] `affiche(a)` est un appel à la fonction "affiche"
- [ ] `affiche(a)` aura pour effet d'afficher "10"
- [ ] `affiche(c)` aura pour effet d'afficher "15"
- [x] à l'extérieur de la fonction la variable "a" vaut 2
- [ ] à l'intérieur de la fonction la variable "a" vaut 2, et "c" vaudra 7

Envoyer Solution

Ce que nous pouvons observer est assez particulier au langage Javascript. Dans le système quand on demande à accéder à une variable, l'ordinateur cherche cette variable dans un "contexte" (un "scope"). Un "scope" est limité, dans le code, par les accolades (ouvrante et fermante) d'une fonction.

```
1. function maFunc () {
2.     var maVar = 10;
3.     var result;
4.
5.     result = maVar + 2;
6. }
```

À la ligne 2 et 3 je déclare 2 variables, elles existent donc désormais dans le scope de la fonction. À la ligne 5 je peux utiliser la variable result ainsi que la variable maVar car elles existent dans le contexte (le scope) où s'éxécute cette ligne de code (on est dans la même fonction). En somme, un scope est créé pour tout le code entre les accolades de la ligne 1 et 6.

Si on ne crée pas de fonction et que l'on déclare des variables, elles sont créées dans un scope global.

Les scopes créés par les fonctions s'imbriquent (on peut écrire une fonction dans une autre fonction, on ne l'a pas fait, mais vous aurez rapidement l'occasion de le faire par vous-même).

Or les variables suivent certaines règles par rapport au scope :

- une variable est définie dans un scope et existe pour tous les scopes enfants.
- une variable définie dans un scope masque les variables éventuellement récupérées du scope parent.

Ce concept, certes compliqué, est important. Il semblera (rapidement) plus naturel avec un peu de pratique.

Quoi qu'il en soit, c'est une bonne pratique de limiter au maximum la "porosité" de la fonction. Une fonction doit être au maximum indépendante et ne pas être "infiltrée" par des variables provenant du scope parent. Les paramètres nous permettent en fait d'échanger des valeurs entre "l'intérieur et l'extéieur de la fonction".

```
function afficheMeilleur (listePersonnes) {
    var meilleur;
    for (var i = 0; i<listePersonnes.length; i = i + 1) {
        // cherche parmi la liste passée en paramètre à la fonction
        // la meilleure personne et stocke le resultat
        // dans la variable "meilleur" (déclarée dans la même fonction)
    }
    affiche(meilleur);
}

var listeEleves = [/*{ ... }, { ... }, ...*/];

afficheMeilleur(listeEleves);
```

Dans la fonction, on utilise le paramètre, et non directement la variable listeEleves (qui a été déclarée à l'extérieur de la fonction)

Quand on utilise une variable dans une fonction, il faut faire en sorte, autant que possible, que cette variable soit déclarée dans la même fonction (on dira que c'est une variable locale) ou que cette variable soit issue d'un des paramètres de la fonction.

Pour autant, cette porosité offerte par le système des paramètres est unilatérale :

```
// soit la fonction "foo", elle accèpte un paramètre.
// la valeur passée en paramètre sera copiée dans la variable bar
function foo (bar) {
    bar = 'baz';
}

var anyVar = 'val'; // on déclare une variable et on lui donne la valeur "val"
foo(anyVar);        // on appelle la fonction "foo" avec en paramètre la variable anyVar
affiche(anyVar);    // affiche "val", la variable anyVar n'est pas modifiée
```

La question se pose donc de l'existence d'un mécanisme de porosité de la fonction de l'intérieur vers l'extérieur.

## Quiz

Considérez le code suivant

```
function sayHello (param) {
    var newString = param + " I'm Mathias";
    return newString;
}

var msg = 'Hello World';
var newGreetings;

newGreetings = sayHello(msg);
affiche(msg);
affiche(newGreetings);
```

**Question 1 of 1**

Cocher la(les) affirmation(s) vraie(s)

- [x] ce code affiche "Hello World" puis "Hello World I'm Mathias"
- [ ] ce code affiche 2 fois "Hello World" puis "Hello World I'm Mathias"
- [ ] ce code affiche 2 fois "Hello World I'm Mathias"
- [ ] la variable "param" existe dans le "scope global"
- [x] la variable "newGreetings" existe dans le "scope global"
- [x] dans le scope globale msg ne change pas de valeur
- [ ] la fonction sayHello ne retourne rien
- [x] la valeur retournée par la fonction sayHello est récupérée dans la variable newGreetings

Envoyer Solution

Le mécanisme des fonctions étant prévu pour permettre l'éxécution d'une portion de code à la demande, le besoin de retourner le résultat de l'éxécution de cette portion de code, apparait assez rapidement.

Les langages nous permettent donc de retourner le résultat par le biais de l'instruction "return".

Ainsi donc une fonction est un bloc de code auquel on fait référence en l'invoquant (ce qui a pour effet d'exécuter le code du bloc). Chaque éxécution pourra être paramétrée avec un jeu de variables, et éventuellement retourner un résultat (et ces 2 biais doivent rester les seuls échanges entre l'extérieur et l'intérieur de la fonction).

# Ce qu'il faut retenir à propos des fonctions

- Les fonctions se définissent par une ***signature*** et un ***corps*** (bloc de code).
- Une fonction peut être exécutée plusieurs fois (il faut l'invoquer dans le code).
- Une fonction accepete des paramètres, ces derniers permettront de modifier son comportement.
- Une variable déclarée dans une fonction n'existe que pour la fonction
- Une fonction peut retourner une valeur

# Les objets

Faisons un petit point de parcours.

Nous avons commencé par parler des ***variables*** : un ***type de donnée*** et un ***nom*** qui servent de ***conteneur*** à une ***valeur***. De variable contenant des types de données simples (***booléens***, ***nombres***, ***chaine de caractère***, etc...) nous sommes passés à des structures ***tableau*** (avec des ***indices*** pour repérer chaque valeur).

Et puis nous venons de parler des ***fonctions*** qui sont des ***blocs*** de code qui embarquent un ***"comportement"***. Un ensemble d'***instructions*** dont on pourra ***déclencher l'éxécution*** à loisir.

L'idée qui est venue au concepteur de langage informatique a été de regrouper ces 2 notions (des ***données structurées***, et des ***"comportements"***) dans un objet, une chose, un truc, un machin, un bidule.

– seule la première dénomination a perduré dans la littératurre informatique, allez savoir pourquoi –

Un ***objet***, dans le domaine de la programmation informatique désigne un conteneur, une variable qui à l'instar d'un tableau contient plusieurs valeurs, mais dont les valeurs sont indexées (identifiées) par une ***clé*** (en ***chaine de caractères***) et non par ***indice*** numérique contrairement à un tableau. Et qui peut en plus "contenir" des ***fonctions***, des ***"comportements"*** (qui seront eux aussi ***identifiés*** par des ***clés***).

En Javascript, la façon la plus simple de déclarer un object consiste à décrire une suite de clés et de valeurs le tout entouré d'accolades :

```
1. {
2.     "cle1" : "valeur 1",
3.     "cle2" : function () { var msg = "Hello World"; },
4.     "cle3" : 42
5. }
```

- la première accolade ouvrante indique que l'on ouvre une liste de clés/valeurs.

```
1. {
```

- entre la clé et la valeur il doit y avoir le caractère ":"

```
2.    "cle1" : "valeur 1",
```

- entre chaque couple clé/valeur, il devra y avoir une virgule (après "valeur 1" ci-dessus, et après "}" ci-dessous. Mais pas après 42, c'est la dernière valeur.

```
3.    "cle2" : function () { var msg = "Hello World"; },
4.    "cle3" : 42
```

- la dernière accolade fermante, indique que l'on ferme la liste de clés/valeurs. et donc c'est la fin de la déclaration de notre objet.

```
5. }
```

Comme pour un tableau on pourra employer la syntaxe avec des crochets en suffixe du nom de variable pour faire référence à une valeur en particulier, ou appeler une fonction d'un objet.

## Exercice

Compléter le code pour appeller la fonction sayHello de l'objet "robot"

```
1  var robot = {
2      name: "r2-d2",
3      sayHello: function () {
4          msg = 'Hello World!'
5          affiche(msg);
6      }
7  }
8
9  robot['sayHello']();
```

Envoyer Solution

Mais il existe une syntaxe moins verbeuse; il suffit d'ajouter un "." derrière le nom de l'objet puis de nommer, par sa clé, la propriété ou la fonction désirée :

```
robot.sayHello();
// remplace
// robot['sayHello']();

// et on peut faire référence à la valeur dans la clé "name" ainsi
robot.name;
```

Les ***objets*** sont une structure de données très importante. Le paradigme qui en découle est à la source de nombreux concepts dans le domaine de l'architecture logicielle.

Il est très important que vous soyez à l'aise avec leur manipulation. Et même si ce que nous en voyons à travers cette introduction aux objets est très vulgarisé, gardez en tête que la ***programmation orientée objet*** (POO) est un concept clé.

## Exercice

**Faites la moyenne de tous les élèves et enregistrez le résultat dans la variable "moy".**

```
1  // var eleves = [ { prenom : "Ana", note: 15 }, { prenom : ... , ... } ];
2  var moy = 0;
3  var somme = 0;
4  var eleve;
5
6  for (var i=0; i<eleves.length; i=i+1) {
7      eleve = eleves[i];
8      somme = somme + eleve.note;
9  }
10 moy = somme / eleves.length;
```

Envoyer Solution

# Ce qu'il faut retenir à propos des Objets

- Les objets sont un type de données (au même titre que le type tableau, le type booléen, etc...).
- A l'instar d'un tableau, un objet regroupe plusieurs valeurs chacune étant associée à une clé (on dit que c'est une propriété de l'objet).
- Les objets peuvent aussi "contenir" des fonctions qui sont elles aussi associées chacune à une clé, on dit que ce sont des méthodes de l'objet).

# Les bonnes pratiques

## La conception d'algorithme

Un algorithme est une suite finie d'instructions permettant de résoudre un problème.

La tache du développeur est d'inventer les algorithmes pour répondre au problème qui se pose à lui. Les problèmes posés étant de plus en plus compliqués, il va de soi que les algorithmes sont devenus eux aussi de plus en plus compliqués.

Plus l'application finale est grande, et plus l'architecturation du code est importante.

Mais à quelque niveau que ce soit, l'idée principale est de se dire que l'on doit assembler des briques. Chaque brique devant avoir son rôle et sa place : à l'échelle d'une fonction, une brique serait une instruction. Chaque instruction est indépendante des autres, mais combinées ensemble elles font quelque chose.

### Quiz

Choisissez les opérations dans l'ordre dans lequel il faudra qu'elles s'exécutent pour déterminer si la variable "isMax" est supérieure à tous les élément du tableau

Question 1 of 4

Première étape :

- ○ determiner si on a parcouru toutes les valeurs du tableau
- ○ comparer "isMax" avec un élément du tableau
- ◉ parcourir les élements du tableau
- ○ si "isMax" est plus petit qu'un des éléments du tableau sortir de la boucle

**Question 2 of 4**

Deuxième étape :

- ○ determiner si on a parcouru toutes les valeurs du tableau
- ◉ comparer "isMax" avec un élément du tableau
- ○ parcourir les élements du tableau
- ○ si "isMax" est plus petit qu'un des éléments du tableau sortir de la boucle

**Question 3 of 4**

Troisième étape :

- ○ determiner si on a parcouru toutes les valeurs
- ○ comparer "isMax" avec un élément du tableau
- ○ parcourir les élements du tableau
- ◉ si "isMax" est plus petit qu'un des éléments du tableau sortir de la boucle

**Question 4 of 4**

Quatrième étape :

- ◉ determiner si on a parcouru toutes les valeurs
- ○ comparer "isMax" avec un élément du tableau
- ○ parcourir les élements du tableau
- ○ si "isMax" est plus petit qu'un des éléments du tableau sortir de la boucle

Envoyer Solution

Une fois ces étapes traduites en code informatiques, notre machine peut comprendre notre algorithme, reproduire les opérations, et ainsi répondre à notre problème.

Dans l'exemple ci-dessus nous avions 4 opérations, vous pensez bien qu'il n'y a pas que 4 instructions ou opérations à faire dans les applications complexes de notre quotidien. Mais le cerveau humain aura toujours plus de facilité à gérer si le nombre d'opérations est faible.

Dans la conception d'algorithmes plus conséquents, voire dans la conception de l'architecture d'un programme informatique dans son ensemble, on va pouvoir regrouper ces petits algorithmes dans des fonctions, puis dans des classes, et dans des modules. Et par le jeu des boites, qui masquent la mécanique interne, nous ne manipulons plus que des noms, et des "comportements".

Un ensemble de fonctions dans un programme peut être vu de la même façon : chaque fonction à un rôle bien distinct, et est indépendante des autres fonctions. Mais l'ensemble de ces fonctions, invoquées dans le bon ordre, produit quelque chose.

En somme la création d'algorithme consiste à découper en taches plus simples un problème et à organiser les opérations à faire pour y répondre. Selon la taille des opérations on pourra éventuellement les redécouper en opérations encore plus petites, et ainsi de suite jusqu'a arriver au niveaux des instructions.

En tant que développeur, vous devrez donc être capables d'effectuer ce découpage et cette conception de l'ordre des opérations à effectuer pour apporter une solution.

# Be smart

L'algorithmique est une discipline hautement intellectuelle qui demande de conceptualiser des problèmes physiques en solution informatique. À l'instar d'un traducteur, le developpeur doit "traduire" la problématique pour qu'un ordinateur soit en mesure de la comprendre. Mais il doit ensuite produire le code qui permettra à l'ordinateur d'apporter une solution au problème donné à la base. Avec la part grandissante que prend l'informatique dans notre quotidien les problématiques complexes sont de plus en plus nombreuses. De surccroit, le temps et surtout le coût de developpement sont devenus des enjeux cruciaux.

Pour raccourcir les temps de développement et diminuer les coûts liés à la maintenance des applications informatiques, il existe plusieurs doctrines plus ou moins précises quant à la façon d'écrire/d'organiser son code.

Une bonne façon d'aborder la question est de suivre la "philosophie" KISS (Keep It Simple & Smart) chaque subdivision (à tous les niveau : un module, une fonction, etc...) doit pouvoir se définir simplement : quel est son rôle? que doit-on lui fournir? quel résultat cela nous permet-il d'obtenir? Si on ne peut pas répondre aisément (pour schématiser : par une ou deux phrase(s) simple(s)) à ces questions c'est sûrement qu'il y a un problème de conception (où en tous cas, qu'il risque d'y en avoir à l'usage de cette partie du code).

# Ce qu'il faut retenir

- Un algorithme est un ensemble d'étapes permettant de répondre à un problème
- Organiser et segmenter son code (pour l'instant avec des fonctions)
- KISS (Keep It Simple and Smart)