



Cursus SALESFORCE

M2I Formations 2021

Christian Lisangola

Algorithmique

1.

Introduction à l'algorithmique et programmation

C'est quoi un algorithme

C'est un ensemble ordonné d'instructions/opérations finies dans le but de produire un résultat.

- ❑ Il s'agit d'un **ensemble ordonné d'opérations**, ce qui signifie qu'il s'agit d'une chaîne d'instructions précises qui doivent être **suivies dans l'ordre**. Une bonne façon de l'illustrer est avec l'exemple d'une recette de cuisine, qui reste un algorithme simple.
- ❑ Son objectif est **de résoudre un problème**, c'est-à-dire il a un **objectif délimité**. Il ne s'agit pas seulement d'écrire une belle série d'ordre qui ne mènent nulle part, mais plutôt de le faire de manière rationnelle et dans un but précis..

C'est quoi un programme

Lorsqu'un développeur crée un programme, il crée essentiellement un ensemble d'algorithmes.

Un programme informatique **est un ensemble de commandes données à la machine, écrites dans un langage spécifique**, pour effectuer une série d'opérations déterminées afin d'obtenir un résultat.

Introduction L'algorithme

- ❑ La programmation permet de résoudre un problème de manière **automatisée** grâce à l'utilisation des algorithmes.
- ❑ Les instructions utilisées dans le programme représentent **le code source**.
- ❑ Pour que le programme puisse être exécuté par la machine, il faut utiliser un langage que celle-ci peut comprendre : **le langage binaire**.

Introduction à la programmation

Le langage binaire est le langage à 2 états représentés par des 0 et des 1, ce qui correspond à ce que l'ordinateur peut comprendre.

Lorsque le développeur écrit son programme, il le fait grâce à un langage de programmation **qui est ensemble de règles de vocabulaire et de grammaire compréhensible par un humain dans le but de donner des instructions à une machine.**

Pour que la machine arrive à comprendre ces instructions, il faut que ce code soit **traduit** en langage binaire par un **compilateur** ou un **interpréteur**.

Introduction à la programmation

Langage **bas niveau** vs langage de **haut niveau**.

- ❑ Un langage de bas niveau est un langage qui est considéré comme plus proche du langage machine (binaire) plutôt que du langage humain. Il est en général plus difficile à apprendre et à utiliser mais offre plus de possibilités d'interactions avec le hardware de la machine.
- ❑ Un langage de haut niveau est le contraire, il se rapproche plus du langage humain et est par conséquent plus facile à appréhender. Cependant les interactions se voient limiter aux fonctionnalités que le langage met à disposition.

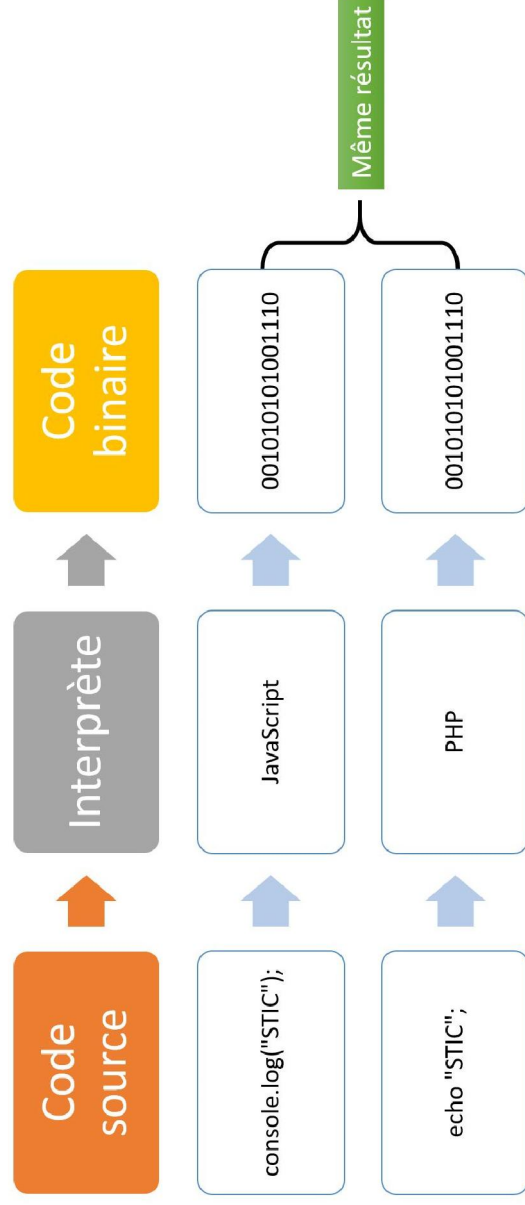
Introduction à la programmation

Compilation vs Interprétation

- ❑ La compilation : consiste à transformer toutes les instructions en langage machine avant que le programme puisse être exécuté. Par conséquent il sera nécessaire de refaire la compilation après chaque modification du code source.
- ❑ L'interprétation : consiste à traduire les instructions en temps réel (on run time). Dans ce cas le code source est lu à chaque exécution et par conséquent les changements apportés au code seront pris en compte directement.

Introduction à la programmation

- Alors c'est pas des 0 et des 1 le code ?



Introduction à la programmation

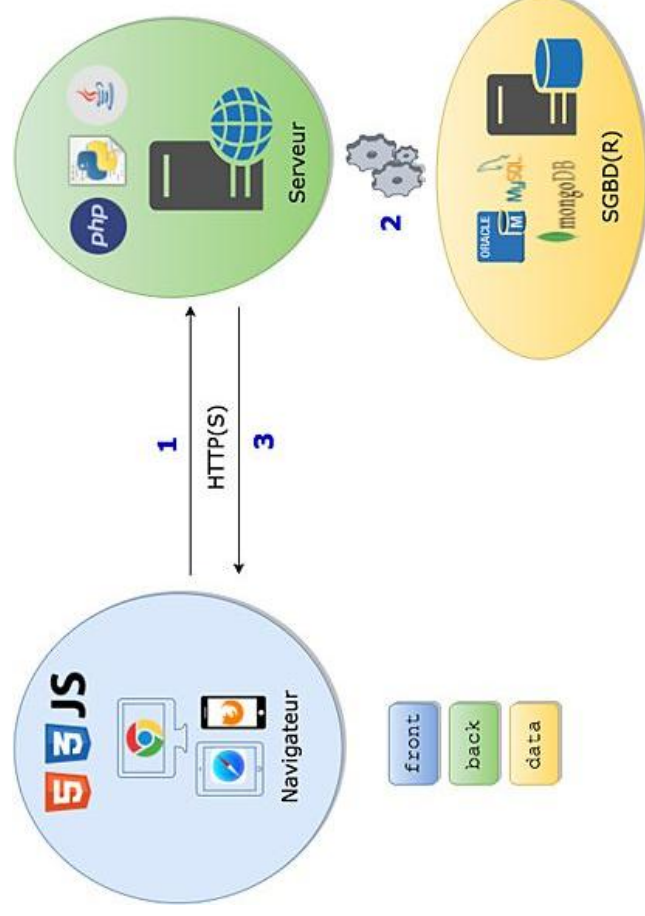
Et les mathématiques dans tout ça ?

- ❑ Développer ne consiste pas à écrire des formules mathématiques (sauf si vous travailler sur des logicielles destinées aux sciences évidemment).
- ❑ Même si les langages sont différents, l'objectif principal d'un langage de programmation reste celui d'instruire la machine pour qu'elle produise des outputs conformes aux objectifs de l'application.
- ❑ On résume souvent par :
 - ❑ $F(X) = Y$, où :
 - ❑ X représente l'input
 - ❑ Y représente l'output
 - ❑ F() représente la fonction qui permet de transformer X en Y



Introduction à la programmation

- Anatomie d'une application interactive



Les éléments fondamentaux :

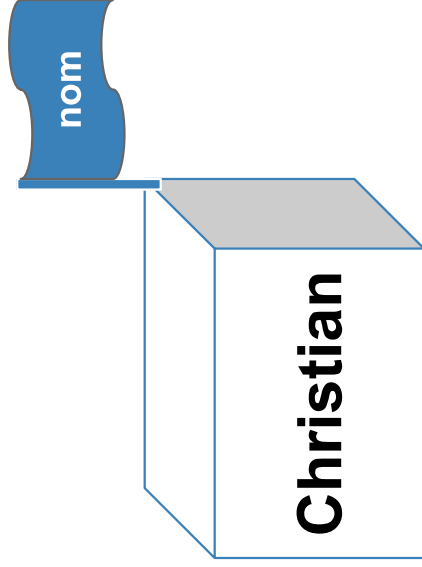
- ☐ Les variables
- ☐ Les opérateurs
- ☐ Les structures conditionnelles
- ☐ Les structures itératives
- ☐ Les fonctions
- ☐ Les tableaux (ou array)
- ☐ Les objets

Installation de Algobox

Afin de mettre en pratique les algorithmes, nous allons utiliser le logiciel algobox :

<https://www.xm1math.net/algobox/download.html>

Les variables



- ❑ Nom de la variable ou **identificateur** : **nom**
- ❑ Contenu : **Christian**

Une variable est une zone que l'on réserve en mémoire pour stocker des données. Par opposition aux constantes, la valeur contenue dans une variable peut changer au cours du temps.

Type de données

Si l'on doit le comparer variable à un récipient, elle peut contenir des données possédant un type bien défini.

Par exemple, on ne doit pas stocker ou garder des baskets dans une marmite.

Les types de données pouvant être contenu dans une variable sont :

- ❑ les suites de caractères (string) : elles sont utilisées pour représenter du texte ;
- ❑ les chiffres (nombre entier, à virgule flottante, etc.) : ils sont utilisés surtout avec des opérateurs mathématiques ;
- ❑ les valeurs booléennes (en anglais : booleans) : soit vrai, soit faux ;
- ❑ les tableaux (array) : ils sont utilisés pour une collection d'éléments
- ❑ les objets : ils sont des conteneurs qui peuvent inclure souvent tout type de données, y compris de sous-objets, des variables (i.e. des propriétés), ou des fonctions (i.e. méthodes)

Déclarations d'une variable

Déclarer une variable, c'est réserver un espace dans la mémoire pour y stocker des données.

```
VARIABLES  
  nom EST_DU_TYPE CHAINE  
  DEBUT_ALGORITHME
```

Affectation et affichage

L'affectation est l'opération qui consiste à attribuer une valeur à une variable.

```
VARIABLES  
  nom EST_DU_TYPE CHAINE  
DEBUT_ALGORITHME  
  nom PREND_LA_VALEUR "Christian Lisangola"  
  AFFICHER nom  
FIN_ALGORITHME
```

Affectation et affichage

L'affectation n'est pas la même chose que l'égalité, car avec l'égalité la relation est maintenue au cours du temps.

Exercise 1

$$A = 5$$

$$B = A$$

$$A = 9$$

$$B : ?$$

Exercise 2

A <- 5

B <- 7

C <- (A + B)/2

A <- 12

B <- 12

C : ?

Opérateurs arithmétiques

Les opérateurs arithmétiques sont : l'addition, la soustraction, la division, la multiplication, etc.

```
VARIABLES
nombre1 EST_DU_TYPE NOMBRE
nombre2 EST_DU_TYPE NOMBRE
somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
nombre1 PREND_LA_VALEUR 5
nombre2 PREND_LA_VALEUR 10
somme PREND_LA_VALEUR (nombre1 + nombre2)/2
AFFICHER somme
FIN_ALGORITHME
```

Lecture des valeurs à partir du clavier

Il est possible que les valeurs des variables proviennent de la saisie de l'utilisateur.

```
VARIABLES
nombre1 EST_DU_TYPE NOMBRE
nombre2 EST_DU_TYPE NOMBRE
somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE nombre1
  LIRE nombre2
  somme PREND_LA_VALEUR (nombre1 + nombre2)/2
  AFFICHER somme
FIN_ALGORITHME
```

Exercice 3

Ecrire un algorithme qui demande à l'utilisateur d'entrer à partir du clavier:

- ☐ La distance parcourue(m)
- ☐ Le temps(sec)

Puis calculer la vitesse selon la formule:

$$\text{vitesse} = \text{distance parcourue} / \text{temps}$$

Puis afficher le résultat dans le format suivant : 345 m/s

Structures conditionnelles

Les structures de contrôle permettent d'exécuter seulement certaines instructions d'un programme selon la vérification d'une ou plusieurs conditions.

La version sémantique la plus répandue des structures de contrôle est « **si... alors...** ».

Par exemple :

- ☐ Si la note est inférieure à 5, alors la note est insuffisante
- ☐ Si l'âge est supérieur ou égal à 18, alors on est majeur
- ☐ Si le mot de passe choisi contient moins de 6 caractères, alors il est trop court

Opérateurs Logiques et de comparaison

Les opérateurs logiques sont : ET/AND, OU/OR, etc.

Les opérateur de comparaisons : >, <, >=, <=, <>, ==

FONCTIONS_UTILISEES

```
VARIABLES
age EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
LIRE age
SI (age>=12 ET age<=17) ALORS
    DEBUT_SI
        AFFICHER "Vous êtes ado"
    FIN_SI
FIN_ALGORITHME
```

FONCTIONS_UTILISEES

```
VARIABLES
pays EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
LIRE pays
SI (pays=="France" OU pays=="Belgique") ALORS
    DEBUT_SI
        AFFICHER "Vous pouvez participer au sondage"
    FIN_SI
SINON
    DEBUT_SINON
        AFFICHER "Vous n'êtes pas éligible pour participer au sondage"
    FIN_SINON
FIN_ALGORITHME
```

Les opérateurs logiques sont : ET(and), OU(or), etc.

Opérateur OU

- ☐ Retourne True quand au moins une des conditions est True
- ☐ Retourne False seulement si toutes les conditions sont false

Opérateur ET

- ☐ Retourne True seulement si toutes les conditions sont évaluées à True
- ☐ Retourne False si au moins une des conditions est évaluée à False

Ecrire un algorithme qui demande l'âge d'un enfant. Ensuite, il l'informe de sa catégorie :

« Poussin » de 7 à 9 ans

« Pupille » de 10 à 11 ans

« Benjamin » de 12 à 13 ans

« Minime » de 14 à 15 ans

« Cadet » 16 à 17 ans

Les structures itératives ou boucles

Les boucles sont à la base d'un concept très utile en programmation : l'itération. L'itération permet d'exécuter de manière répétitives des instructions. Elles peuvent être très utiles par exemple pour appliquer un traitement à une liste d'éléments.

Exemple d'utilisation :

Tant que la liste n'est pas totalement parcouru :

 Modifier un élément de la liste

Ou encore :

Tant que ce chiffre ne dépasse pas 16 :

 Réalise un calcul

Boucle tant que : anatomie

Généralement, cette boucle est utilisée quand le nombre d'itérations n'est pas connu à l'avance.

Tant que (**condition**) **faire**

Debut tant que

Instruction 1

Instruction 2

...

Fin tant que

Boucle tant que : exemple

FONCTIONS_UTILISEES

VARIABLES

code_secret EST_DU_TYPE CHAINE

DEBUT_ALGORITHME

code_secret PREND_LA_VALEUR ""

TANT_QUE (code_secret!="xyz") FAIRE

DEBUT_TANT_QUE

LIRE code_secret

SI (code_secret!="xyz") ALORS

DEBUT_SI

AFFICHER "Code incorrect, veuillez ressayer"

FIN_SI

SINON

DEBUT_SINON

AFFICHER "Félicitations, vous avez trouvé le code secret"

FIN_SINON

FIN_TANT_QUE

FIN_ALGORITHME

Boucle pour : anatomie

Cette boucle est utilisée quand le nombre d'itérations est est connu à l'avance.

Pour **compteur** allant de **départ** à **limite**
 Debut pour
 Instruction 1
 Instruction 2
 ...
 Fin pour

Boucle pour i...

```
VARIABLES
nbr EST_DU_TYPE NOMBRE
compteur EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
LIRE nbr
POUR compteur ALLANT_DE 0 A 12
    DEBUT_POUR
    AFFICHER nbr
    AFFICHER " x "
    AFFICHER compteur
    AFFICHER " = "
    AFFICHERCALCUL * nbr*compteur
    FIN_POUR
FIN_ALGORITHME
```

Ecrire un algorithme qui permet à l'utilisateur de définir une adresse email et un mot de passe.

Ensuite dans un second temps, il sera demandé à l'utilisateur de fournir l'email et le mot de passe:

- Si l'email et le mot de passe ne correspondent pas aux valeurs définies, le message "Identifiants incorrect devra s'afficher", et l'utilisateur devra recommencer la saisie des valeurs.
- Sinon, le message "Bienvenu dans votre espace client" devra s'afficher.

A l'exercice précédent, s'ajoute un nouveau requirement:

Le nombre de fois que l'utilisateur peut saisir des mauvais identifiants est limité à 5, ensuite le programme va s'arrêter avec un message disant

“Vous avez saisi des mauvais identifiants x fois, votre compte est bloqué” .

Ecrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Écrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Ecrivez un programme qui affiche en console les nombres de 1 à n:

- ☐ pour les multiples de trois, il affiche "fizz" à la place du nombre
- ☐ et pour les multiples de cinq de cinq, imprimez "buzz".
- ☐ Pour les nombres qui sont des multiples multiples à la fois de trois et de cinq, imprimez "fizzbuzz".
- ☐ Sinon, il affiche le nombre

Exemple

Si n = 5

1

2

fizz

4

Buzz

Écrire un algorithme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :

$$7 \times 1 = 7$$

$$7 \times 2 = 14$$

$$7 \times 3 = 21$$

...

$$7 \times 10 = 70$$

Ecrire un algorithme qui permettra de calculer plusieurs tables de multiplication.

On va d'abord demander à l'utilisateur de saisir le 1er nombre dont il faut connaître la table de multiple, ensuite on lui demandera le dernier nombre.

Si par exemple le premier nombre est 3 et le dernier 8, on va afficher toutes les tables de multiplications de **3 à 8**, donc : **3,4,5,6,7,8**

Après on demandera à l'utilisateur d'entre le premier nombre et le dernier nombre à multiplier.

Si par exemple il saisit comme premier **nombre 3** et **dernier nombre 7**, on aura;

3 x 3 = 9	4 x 3 = 12	...	8 x 3 = 24
3 x 4 = 12	4 x 4 = 14	...	8 x 4 = 32
3 x 5 = 15	4 x 5 = 20	...	8 x 5 = 40
3 x 6 = 18	4 x 6 = 24	...	8 x 6 = 48
3 x 7 = 21	4 x 7 = 28	...	8 x 7 = 56

Manipulation des chaînes

- ❑ **nomChaine.substr(positionDebut, positionFin)** : Retourne une sous-chaîne
- ❑ **nomChaine.length** : Taille de la chaîne
- ❑ **nomChaine.charCodeAt(pos)** : permet d'obtenir le nombre égal au code ascii de la lettre figurant à la position
- ❑ **nomChaine.toString()** : Transforme un nombre en chaîne

Ecrire un algorithme qui demande à l'utilisateur une chaîne de départ, et ensuite l'algorithme devra produire la chaîne renversée.

Exemple :

Si l'on entre : papa

L'algo devra nous afficher : apap

TP 11 : Le Palindrome

Ecrire un algorithme qui demande à l'utilisateur une chaîne de départ, et ensuite l'algorithme devra lui dire si la chaîne saisie est un palindrome ou non.

Se dit d'un mot, d'un vers, d'une phrase que l'on peut lire indifféremment de gauche à droite ou de droite à gauche.

Exemple : le mot **ressasser** ou la phrase **esope reste ici et se repose.**)

Ecrire un algorithme qui demande à l'utilisateur un nombre de départ, et ensuite l'algorithme devra renverser le nombre.

!Prendre en compte des nombres négatifs

Ecrire un algorithme qui demande à l'utilisateur une chaîne, et ensuite lui retourne le nombre des voyelles et consonnes contenu dans la chaîne.

Exemple:

Entrée : Benjamin

Résultat : 3 voyelles, 5 consonnes

Écrire un algorithme qui demande à l'utilisateur une chaîne, et ensuite un caractère, puis lui retourne le nombre de fois que le caractère saisi se trouve dans la chaîne et dire si c'est une voyelle ou une consonne.

Exemple:

Caractère : i
chaîne : christian

Résultat : i se retrouve 2 fois, et c'est une voyelle

Les tableaux

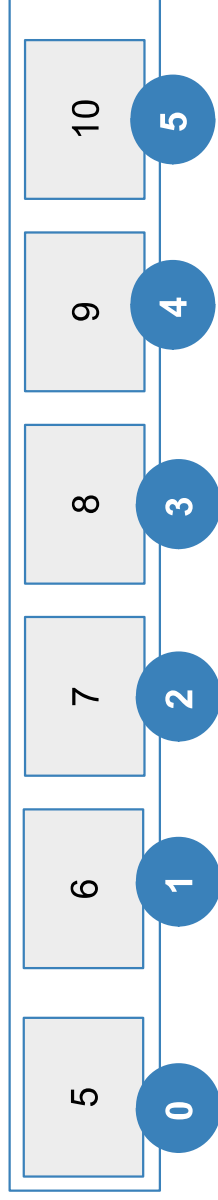
Les tableaux (array) sont des listes indexées d'éléments qui partagent normalement une certaine relation sémantique pour appartenir à la liste. Un exemple tout simple d'array est la liste des courses : on peut indexer cette liste en fonction de l'ordre des articles à acheter :

1. Lait
2. Farine
3. Pommes
4. Fromage

Attention : l'index d'un tableau commence à 0. En prenant l'exemple précédent on dira que l'élément "Lait" est dans la première case du tableau qui est à l'index 0.

De la même façon l'élément "Fromage" est l'élément dans la 4ème case du tableau qui se trouve à l'index 3.

Les tableaux : index



Les tableaux

FONCTIONS_UTILISEES

VARIABLES

nombres EST_DU_TYPE LISTE

n EST_DU_TYPE NOMBRE

i EST_DU_TYPE NOMBRE

DEBUT_ALGORITHME

LIRE n

POUR i ALLANT_DE 0 A n - 1

DEBUT_POUR

LIRE nombres[i]

FIN_POUR

POUR i ALLANT_DE 0 A n - 1

DEBUT_POUR

AFFICHER nombres[i]

FIN_POUR

FIN_ALGORITHME

Exercice 16

Ecrivez un algorithme qui calcule la somme de tous les éléments d'un tableau.

Exercice 17

A partir de l'exercice précédent, ajouter le calcul de la moyenne des nombres contenus dans le tableau.

Exercice 18

Ecrire un algorithme qui demande à l'utilisateur d'initialiser un tableau numérique à partir d'un instruction de lecture, et ensuite le plus grand nombre du tableau et le plus petit.

Exemple :

Tableau : [3, 6, 2, 1, 7, 12, 32]

Plus grand : 32

Plus petit : 1

Exercice 19

Ecrire un algorithme qui demande l'utilisateur de créer 2 listes composées de nombres.

Ensuite, remplir ces 2 listes à partir de la lecture du clavier tout en sachant que ces 2 listes doivent avoir la même taille (aussi définie par une lecture).

Puis, calculer la liste résultante qui va être construite par la somme des éléments aux mêmes index des 2 listes précédentes.

Exemple :

Liste 1 : [2, 4, 5, 7, 8]

Liste 2 : [3, 7, 2, 5, 6]

List 3(résultante) : [5(2+3), 11(4+7), 7(5+2), 12(7+5), 14(8+6)]

Exercice 20

Ecrire un algorithme qui demande l'utilisateur de créer 2 listes composées de nombres.

Ensuite, remplir ces 2 listes à partir de la lecture du clavier tout en sachant que ces 2 listes doivent avoir la même taille (aussi définie par une lecture).

Puis, calculer la liste résultante qui va être construite par la somme des éléments de la première liste et avec l'inverse de la 2ème liste.

Exemple :

Liste 1 : [2, 4, 5, 7, 8]

Liste 2 : [3, 7, 2, 5, 6]

List 3(résultante) : [5(2+3), 11(4+5), 7(5+2), 12(7+5), 14(8+6)]

Les fonctions

Les fonctions représentent une sorte de "programme dans le programme", donc des "sous programmes". On utilise des fonctions pour regrouper des instructions et les appeler sur demande : chaque fois qu'on a besoin de ces instructions, il suffira d'appeler la fonction au lieu de répéter toutes les instructions. Pour accomplir ce rôle, le cycle de vie d'une fonction se divise en deux :

1. Une phase unique dans laquelle la fonction est déclarée
On définit à ce stade toutes les instructions qui doivent être groupées pour obtenir le résultat souhaité.
2. Une phase qui peut être répétée une ou plusieurs fois.

Les fonctions dans la vie courante

Il existe plusieurs objets de tous les jours, qui peuvent nous de bien visualiser le comportement des fonctions

- ❑ Hachoir à viande : Reçoit un input(de la viande), puis produit de la viande hachée
- ❑ Le moule : Reçoit du ciment , puis produit des risques.

Les fonctions

```
FONCTIONS_UTILISEES
FONCTION addition(a,b)
VARIABLES_FONCTION
  DEBUT_FONCTION
    RENVoyer a+b
  FIN_FONCTION
VARIABLES
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
  somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  Lire a
  Lire b
  somme PREND_LA_VALEUR addition(a,b)
  AFFICHER somme
FIN_ALGORITHME
```

Ecrivez une fonction prend en paramètre un nombre et la puissance à laquelle on veut l'élever qui ensuite retourne ce nombre à la puissance mentionnée.

Cette fonction devra ensuite être utilisé dans le programme principal.

Ecrivez une fonction prend en paramètre 2 chaîne dont la première représente un text quelconque et la 2ème un prefix.

Ensuite, faudra que la fonction retourne "VRAI", si le 2ème paramètre est le préfixe du premier et "FAUX" s'il ne l'est pas.

Exemple :

```
maFonction("banner", "bang")  
=> retournera FAUX
```

```
maFonction("hugging","hug")  
=>retournera VRAI
```

Ecrivez une fonction prend en paramètre la distance parcourue et le temps, et qui ensuite retourne la vitesse d'un mobile en MRU.

Formule : $v = \text{distance} / \text{temps}$

Cette fonction devra ensuite être utilisé dans le programme principal.

Ecrire un fonction qui permet de valider la longueur d'une chaîne.

Il doit avoir 3 paramètres : chaine,min,max.

Elle retourne 1(vrai), si la taille de est entre min et max

Elle retourne 0(faux), si la taille n'est pas entre min et max

A partir de la fonction du TP 24, créer un programme qui permet à l'utilisateur de créer un compte avec :

- Pseudo : (min : 5, max : 10)
- Nom : (min : 4, max : 30)
- Prénom : (min : 2, max : 30)
- Mot de passe : (min : 8, max : 50)

Tant que les requirements ne sont pas remplis, on demande à l'utilisateur de recommencer.

Complexité algorithmique

La complexité algorithmique nous permet de mesurer l'efficacité d'un algorithme.

S'il faut mesurer l'efficacité ou performance d'un algorithme en se basant sur son temps d'exécution, alors les temps d'exécutions seront différents selon les ordinateurs en fonction de la configuration matérielle.

Donc, utiliser le temps d'exécution comme metric ne peut pas donner une idée claire sur les performances d'un algorithme.

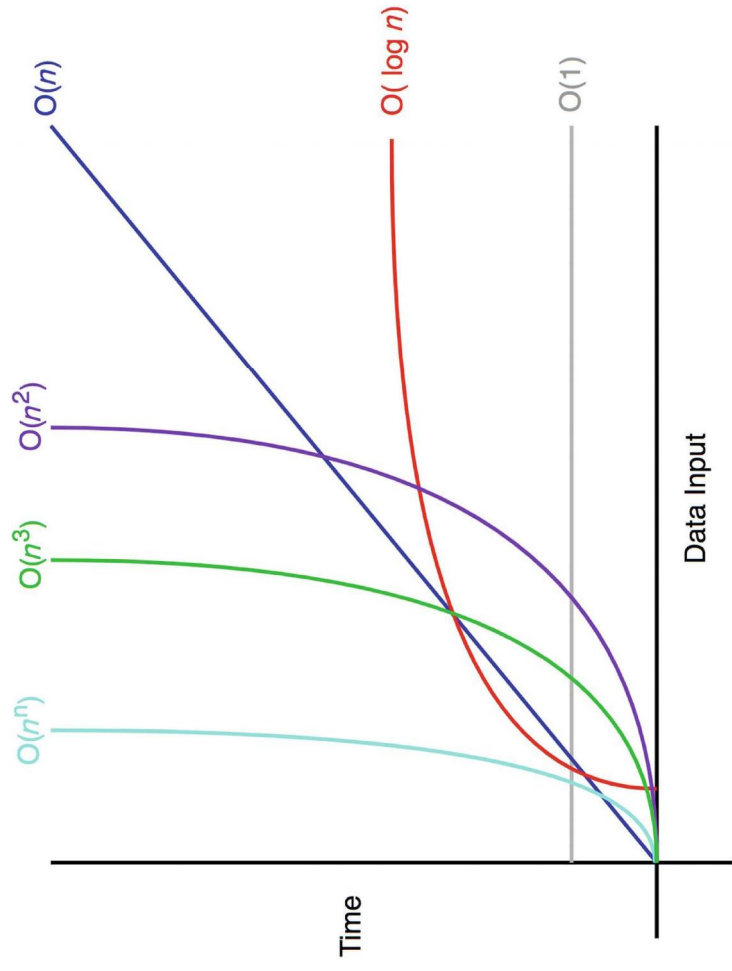
Pour avoir un standard qui soit indépendant de la configuration, on se base sur la Big O notation (Notation grand O).

La Big O, nous permet d'évaluer les performances d'un algorithme sur base de :

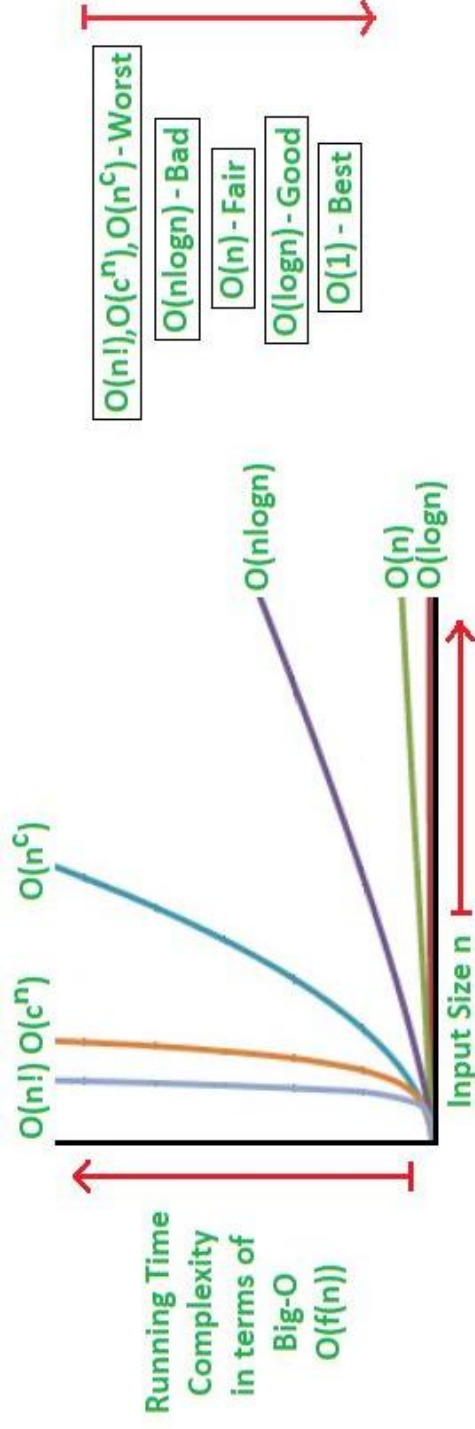
- ❑ Runtime complexity : Nombre d'opération exécutées par le CPU
- ❑ Space complexity : Quantité de mémoire utilisé

L'expression utilisée sera $O(n)$ où n constitue le nombre d'opérations pour le runtime complexity et le space complexity.

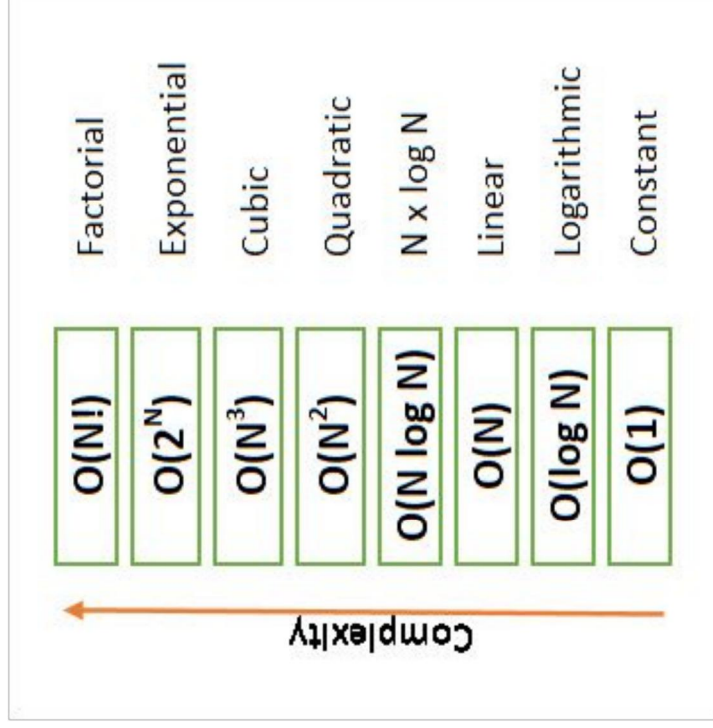
Big O



Big O



Big O



- ☐ Calcul de la somme des éléments du tableau : $O(n)$
- ☐ Calcul de la somme de 2 nombres : $O(1)$
- ☐ Calcul de la somme des diagonales d'une matrice??
- ☐ Retirer un valeur au début d'un tableau : $O(n)$
- ☐ Retirer un valeur à la fin du tableau : $O(1)$
- ☐ Fibonacci(recursif) sans mémorisation : 2^n
- ☐ Fibonacci(recursif) avec mémorisation : $O(n)$
- ☐ Tri à bulles : $O(n^2)$