Algorithmie

Découverte des bases

Algorithme est un mot dérivé du nom du mathématicien al_Khwarizmi qui a vécu au 9ème siècle et qui était membre de l'académie des sciences à Bagdad.

Pourquoi un cours d'algo ?

Quel que soit le langage de programmation utilisé, la logique est la même : cette logique de traitement, c'est l'algorithmie. Cela veut dire que nous pourrions obtenir le même résultat avec des langages différents, si la logique est la même.

Un algorithme prend des **données en entrée**, exprime un **traitement** particulier et fournit des **données en sortie**.

Le but est d'obtenir d'une machine qu'elle fasse un travail automatisé, en suivant une procédure qu'on lui décrit.

Avant même d'écrire un algorithme dans un langage particulier, il faut :

- expliciter son raisonnement,
- décrire les étapes à réaliser,
- dans un langage simple.

Quels éléments ?

Les outils de comparaison

Outil	Utilité	Outil	Utilité
=	Égalité	!=	Différence
>	Supériorité	?	Existant
<	Infériorité	0	Vide
≥ ou ≤	Sup ou égal, inf ou égal	v/f	vrai/faux

Exemple :

si tomate **est <** 4 cm **alors** c'est une tomate cerise !

Les outils d'analyse

Outil	Utilité	Outil	Utilité
&&	ET	||	OU

Exemple :

si tomate **est <** 4 cm **&&** tomate **est** rouge, **alors** c'est une tomate cerise mûre !

Les outils de conditions

SI ... ALORS ... (SINON ...)

La condition est l'opérateur conditionnel le plus fréquent. En fonction d'une comparaison, on demande la réalisation de telle ou telle action.

Exemple :

si age **est <** 18 **alors** Vous êtes mineur **sinon** vous êtes majeur.

TANT QUE ... FAIRE ...

La boucle **tant que** nous permet de répéter une opération tant que le résultat ne permet pas de sortir de la boucle. Attention, il faut bien contrôler dans la condition quelque chose qu'on modifie dans la boucle, sinon, on crée une boucle infinie !

Bonne version	Boucle infinie !
Pomme = 1 ; **Tant que** Pomme < 10 ajouter 1 à Pomme // Dès que Pomme atteint 10, on sort de la boucle.	Pomme = 1 ; **Tant que** Banane < 10 ajouter 1 à Banane ou Pomme = 1 ; **Tant que** Pomme < 10 écrire le nombre de Pommes.

POUR CHAQUE ... FAIRE ...

La boucle **pour chaque**, contrairement à la boucle tant que, comprend automatiquement qu'elle va réaliser une action pour chaque occurrence. Elle s'arrête automatiquement lorqu'on atteint la dernière occurrence à lire.

Exemple :

jour = [lundi, mardi, mercredi, jeudi, vendredi]

Pour chaque jours de la semaine, mettre mon réveil à 7h00.

Contrairement à la boucle **tant que**, on ne modifie pas la valeur observée dans la condition : on n'ajoute pas 1 pour changer de jour dans l'exemple.

Rompre une boucle

Il est possible, lorsqu'on a réussi une opération, de stopper l'exécution d'une boucle, plutôt que de la laisser s'exécuter jusqu'au bout. Par exemple, si une erreur arrive lors de l'exécution d'une opération, sur une des boucles, on ne va pas poursuivre : on stoppe la boucle (**break**).

Les variables

Pour faire les comparaisons, on utilise des variables. Le but est qu'à l'intérieur de cette variable, on va pouvoir y faire passer les valeur à tester.

Exemple :

si le fruit est une pomme, alors on fera une tarte aux pommes.

Dans la variable fruit, on va pouvoir faire passer différentes valeurs à tester, afin de les vérifier. Et dans le cas où la variable est vérifiée, la condition pourra se réaliser.

Une variable peut avoir comme valeur :

- une chaîne de caractère, (Ex : fruit = pomme)
- un nombre, (Ex : pommes = 10)
- une valeur booléenne : vrai ou faux, 0 ou 1 (Ex : fruit = VRAI)
- un tableau de valeurs. (Ex : fruits = [pomme, abricot, pêche])

La boucle **Pour chaque** se base forcément sur une variable contenant un tableau : elle sert à lire chaque ligne du tableau les unes après les autres.

Démonstration :

tableau = [2,4,6,8]

Pour chaque ligne de tableau, ajouter 2 à la ligne. Retourner le tableau à la fin :

tableau de sortie : [4,6,8,10]

Un tableau est conçu avec des clés et des valeurs. La clé permet de retrouver la place de la valeur dans le tableau.

De base, un tableau qui a que des valeurs (ex [pomme, abricot, pêche]), a implicitement des clés qui sont des chiffres, commençant par zéro.

[0 => pomme,
1 => abricot,
2 => pêche]

mais on peut aussi nommer ces clés, et pouvoir ainsi rechercher avec des noms :

[$clé_1$ => $valeur_1$,

$clé_2$ => $valeur_2$, …]

Ainsi, dans une boucle **Pour** on peut récupérer les clés et les valeurs de chaque ligne :

pour chaque ligne du tableau,

    si la clé est égale à cela,

    alors afficher la valeur.

Les fonctions

Les fonctions sont une manière d'embarquer un bout d'algorithme, qu'on va appeler.

L'origine d'une fonction est très souvent la factorisation : si je fais la même action à plusieurs endroit de mon algorithme, j'écris plusieurs fois le même code. Pour éviter ça, on crée une fonction qui englobe ce code, une fois, et qui est appelée là où il faut dans mon algorithme.

La fonction se construit en plusieurs temps :

1. la définition : on lui donne un nom.
2. Ses besoins : on définit les paramètres qui lui sont nécessaires pour fonctionner.
3. Algorithme à réaliser.
4. Retour de la fonction : la réponse suite à l'opération (résultat… par défaut, si la fonction s'est bien passée, renvoie vrai sinon faux).

exemple :

fonction **dire_bonjour**(prenom) {

    Afficher « Bonjour prenom ! »

    retourne « tout s'est bien passé »

}

Appeler la fonction

Une fois que la fonction est définie, il faut l'appeler pour qu'elle s'exécute. Il n'est pas utile que les fonctions aient été déclarées dans l'ordre dans lequel on les appellera.

Pour l'appeler, on va le faire par son nom, puis lui donner les paramètres nécessaires.

Avec notre exemple :

dire_bonjour(Georges) va afficher « Bonjour Georges ! »

Il n'est pas forcément utile, dans ce cas, d'écouter le retour de la fonction, mais on pourrait le faire.

Par exemple :

réponse = dire_bonjour(Georges) ;

si réponse == « tout s'est bien passé », alors la fonction s'est exécutée, on peut passer à la suite par exemple. Sinon, on peut exécuter un code autre…

On peut tout à fait appeler une fonction dans une autre fonction. On ne peut pas appeler une fonction à l'intérieur d'elle-même (sinon on crée une boucle).

Attention, les paramètres donnés à une fonction peuvent être normés. Par exemple si une fonction a besoin d'un tableau pour fonctionner, on ne peut pas lui passer une chaîne de caractère ou un booléen à la place, la fonction va casser.