Les variables

Dans un algorithme des variables vont intervenir.
Il faut les voir comme des boîtes (en fait des cases mémoire) dans lesquelles on y met quelque chose.
Par exemple l'instruction x=3 signifie que dans la boîte réservée à x on y met la valeur 3.

Un autre exemple :

En langage naturel	En python	En javascript
Variables : x , y ,z sont des entiers Initialisation : Affecter à x la valeur 6 Affecter à y la valeur 9 Traitement : Affecter à z la valeur x+y Affecter à z la valeur z+1 Sortie : Afficher z	x=6 y=9 z=x+y z=z+1 print(z)	var x=6 ; var y=9 ; var z=x+y ; z=z+1 ; alert(z) ;

Commentons ce petit programme :
Il y a 3 boîtes x ,y et z ; dans la première on y met 6 et dans la seconde 9.
Ensuite on cherche les contenus des boîtes x et y et on les additionne.
Le résultat est mis dans la boîte z.
L'instruction « z=z+1 » est plus complexe. En effet z désigne à la fois le contenant (à gauche) et le contenu (à droite).
On prend le contenu de la boîte z à qui on ajoute 1 et on met le résultat dans la boîte z.
Au final on affiche le contenu de la boîte z soit 16.
Ainsi on voit dans cet exemple que quand une instruction est exécutée le contenu d'une boîte peut changer.
L'état de la mémoire change à chaque instruction.

Les boucles « pour »

En langage naturel	En python	En javascript

Variables :
i , n sont des entiers
u est un entier
Initialisation :
Affecter à u la valeur 0
Entrée :
Demander à l'utilisateur la valeur de n
Traitement :
Pour i allant de 1 à n
Faire :
Affecter à u la valeur u+i
Fin du pour
Sortie :
Afficher u
	u=0
	n=int(raw_input('Entrer la valeur de n : '))
	for i in range(1 ,n+1) :
	u=u+i
	print(u)
		(La saisie de n se fait par l'intermédiaire d'un input dans le code html de la page web mais à un autre endroit de la page web)
		var u=0;
		for (var i=1 ;i≤n ;i++){
		u=u+i;
		}
		alert(u);

Pour comprendre ce que fait cet algorithme il est souhaitable de prendre une valeur de n particulière petite comme 3 et de remplir un tableau.
Pour chaque valeur de i, toutes les instructions dans le « pour » seront effectuées (ici une seule)
Dans l'exemple il y aura exactement n boucles, donc toutes les instructions du « pour » seront effectuées n fois.
Les variables peuvent alors changer pendant une boucle.
Le tableau suivant montre l'état des variables avant et après les boucles :

i	1	2	3
u avant la boucle	0	1	3
u après la boucle	0+1=1	1+2 soit 3	1+2+3 soit 6

On comprend ainsi que cet algorithme affiche la somme des n premiers entiers.

Les boucles « tant que »

En langage naturel	En python	En javascript

Variables :
i , n sont des entiers
u est un entier
Initialisation :
Affecter à u la valeur 0
Affecter à i la valeur 1
Entrée :
Demander à l'utilisateur la valeur de n
Traitement :
tant que i ≤n
Faire :
Affecter à u la valeur u +i
Affecter à i la valeur i+1
Fin du tant que
Sortie :
Afficher u
	u=0
	n=int(raw_input('Entrer la valeur de n : '))
	while i≤n :
	u=u+i
	i=i+1
	print(u)
		(La saisie de n se fait par l'intermédiaire d'un input dans le code html de la page web mais à un autre endroit de la page web)
		var u=0;
		var i=1;
		while (i≤n){
		u=u+i;
		i=i+1;
		}
		alert(u);

Tant que la condition i≤n est vérifiée les instructions à l'intérieur du « tant que » seront effectuées.
Autrement dit pour sortir du « tant que » la condition i≤n doit être fausse, c'est-à-dire le « tant que » se terminera la première fois que i>n et l'algorithme affichera ce que contiendra alors la boîte u.

Prenons n=3 pour comprendre cet algorithme

i avant la boucle	1	2	3
u avant la boucle	0	1	3
u après la boucle	0+1	1+2	1+2+3 soit 6
i après la boucle	2	3	4

Comme i vaut 4 et 4>3 la boucle tant que s'arrête et le contenu de u est alors 6.
Cet algorithme fait la même chose que l'algorithme précédent.

Si on intervertit les instructions u=u+i et i=i+1 c'est-à-dire si on commence par i=i+1 puis après on fait u=u+i on obtient :

i avant la boucle	1	2	3
u avant la boucle	0	2	5
i après la boucle	2	3	4
u après la boucle	0+2 =2	2+3=5	5+4=9

Le contenu de u est au final 9

Remarque :
Une boucle « tant que » est difficile d'emploi car il faut s'assurer qu'elle se terminera à un moment donné, c'est-à-dire que la condition deviendra fausse après un nombre fini de boucles.
Une boucle « pour » peut être remplacée par une boucle « tant que ».
Dans une boucle « pour » on connaît d'avance le nombre de boucles ce qui n'est pas le cas d'une boucle « tant que » et c'est justement à cause de cela que dans certains cas nous sommes obligés d'utiliser une boucle « tant que ».