Cours d'algorithmique :

Introduction

Définition
Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné
Un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter.

Quelles qualités pour maîtriser l'algorithmique ?
Il faut savoir adopter le point de vue d'une autre personne (ici la machine), avoir de l'intuition (pour anticiper les difficultés et leur trouver des solutions).

De quoi est constitué un algorithme ?
Enfin, les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables d'exécuter que **quatre opérations logiques** (je n'inclus pas ici les opérations de calcul). Ces opérations sont : l'affectation de variables, la lecture / écriture, les tests et les boucles. Un algorithme informatique se ramène donc toujours au bout du compte à la combinaison de ces quatre petites briques de base.

Quel est le rapport entre un algorithme et un programme informatique ?
L'algorithmique exprime les instructions résolvant un problème donné **indépendamment des particularités de tel ou tel langage**. Pour prendre une image, si un programme était une dissertation, l'algorithmique serait le plan, une fois mis de côté la rédaction et l'orthographe. Or, vous savez qu'il vaut mieux faire d'abord le plan et rédiger ensuite que l'inverse…

Comment écrire un algorithme ?
Il a la représentation graphique, avec des carrés, des losanges, etc. qu'on appelait des **organigrammes**. C'est très joli, assez clair pour les petits algorithmes.
Si on veut se pencher sur des problèmes plus ardus, on utilise généralement une série de conventions appelée " pseudo-code ", qui ressemble à un langage authentique, la plupart des problèmes de syntaxe étant mis de côté.

Les variables

Pour employer une image, une variable est une boîte, repérée par une étiquette. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son étiquette.

La première chose à faire avant de pouvoir utiliser une variable est de créer la boîte et de lui coller une étiquette. Ceci se fait tout au début de l'algorithme, avant même les instructions proprement dites. C'est ce qu'on appelle la **déclaration des variables**.

Types numériques

Byte (octet)	0 à 255
Entier simple	-32 768 à 32 767
Entier long	-2 147 483 648 à 2 147 483 647
Réel simple	-3,40E38 à -1,40E-45 pour les valeurs négatives
	1,40E-45 à 3,40E38 pour les valeurs positives
Réel double	1,79E308 à -4,94E-324 pour les valeurs négatives
	4,94E-324 à 1,79E308 pour les valeurs positives

Une déclaration algorithmique de variables aura ainsi cette tête :

Variable g **en Entier Long**

Variables PrixHT, TauxTVA, PrixTTC **en Réel Simple**

Certains langages autorisent d'autres types numériques, notamment :

- le type **monétaire** (avec strictement deux chiffres après la virgule)
- le type **date** (jour / mois / année).

Types non numériques

On dispose donc également du **type alphanumérique** (également appelé **type caractère**) : dans une variable de ce type, on stocke des caractères, qu'il s'agisse de lettres, de signes de ponctuation, d'espaces, ou de chiffres. Une série de caractères (stockée ou non dans une variable alphanumérique, d'ailleurs), s'appelle une **chaîne** de caractères. Et une telle chaîne de caractères est toujours notée entre guillemets.

Un autre type est le type **booléen** : on y stocke uniquement les valeurs logiques VRAI et FAUX.

3. L'instruction d'affectation

3.1 Syntaxe et signification

La seule chose qu'on puisse vraiment faire avec une variable, c'est de **l'affecter, c'est-à-dire lui attribuer une valeur**. En algorithmique, cette instruction se note avec le signe ⇦.

Ainsi : Toto ⇦24 Attribue la valeur 24 à la variable Toto. Ce qui, soit dit en passant, sous-entend impérativement que Toto est une variable de type numérique.

On peut attribuer à une variable la valeur d'une autre variable, telle quelle ou modifiée. Par exemple : Tutu ⇦ Toto .Signifie que la valeur de Tutu est maintenant celle de Toto.

Notez que cette instruction n'a modifié en rien la valeur de Toto : une instruction d'affectation ne modifie que ce qui est situé à gauche de la flèche. Tutu ⇦ Toto + 4. Si Toto contenait 12, Tutu vaut maintenant 16. De même que précédemment, Toto vaut toujours 12.

Tutu ⇦ Tutu + 1 Si Tutu valait 6, il vaut maintenant 7. La valeur de Tutu est modifiée, puisque Tutu est la variable située à gauche de la flèche.

Remarques :
Il y a à droite de la flèche, ce qu'on appelle une **expression**. C'est-à-dire un ensemble de valeurs liées par des **opérateurs**, et dont le résultat final est obligatoirement du même type que la variable située à gauche. Si l'un de ces points n'est pas respecté, la machine sera incapable d'exécuter l'affectation, et déclenchera une erreur (est-il besoin de dire que si aucun de ces points n'est respecté, il y aura aussi erreur !)
Les symboles = et ⇦n'ont pas la même signification.
Il va de soi que l'ordre dans lequel les instructions sont écrites va jouer un rôle essentiel dans le résultat final.

Opérateurs numériques :
Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique :
+ addition - soustraction * multiplication / division ^ puissance
En algorithmique les règles de priorités sont respectées.

Opérateur alphanumérique : &
Cet opérateur permet de **concaténer**, autrement dit d'agglomérer, deux chaînes de caractères.

Exemple

```
Variables A, B, C en Caractère
Début
A ← "Gloubi"
B ← "Boulga"
C ← A & B
Fin
```

La valeur de C à la fin de l'algorithme est "GloubiBoulga"

Opérateurs logiques :

Il s'agit du ET, du OU et du NON. Nous les laisserons de côté… provisoirement, soyez-en sûrs.

Lecture et écriture

1. De quoi parle-t on ?

un certain nombre d'instructions permettent à la machine de dialoguer avec l'utilisateur. Les noms de ces deux opérations , correspondent au point de vue de la machine et non de l'utilisateur, ainsi la **lecture** permet à la machine de lire des information écrite par l'utilisateur. Dans l'autre sens, **l'écriture** permet au programme de communiquer des valeurs à l'utilisateur en les affichant à l'écran.

3. Les instructions de lecture et d'écriture

Tout bêtement, pour que l'utilisateur entre la (nouvelle) valeur de Titi, on mettra :

Lire Titi

Dès que le programme tombe sur cette instruction, **l'exécution s'arrête** ; l'ordinateur attend sagement que l'utilisateur ait frappé quelque chose au clavier, et dès que la touche Enter a été frappée, l'exécution reprend.

Pour écrire quelque chose à l'écran, c'est aussi simple que :

Ecrire Toto

On peut également choisir d'écrire des **libellés** à l'écran, afin de prévenir l'utilisateur de ce qu'il doit frapper (c'est même assez fortement recommandé) :

Ecrire "Entrez votre nom : "

Lire NomFamille

Notez qu'il y a une différence majeure entre afficher un libellé et le contenu d'une variable :

Exemple

Variable Bonjour en Caractère	Variable Bonjour en Caractère
Début	Début
Bonjour ← "Ave Cesar"	Bonjour ← "Ave Cesar"
Ecrire Bonjour	Ecrire "Bonjour"
Fin	Fin

Le premier algorithme affiche le contenu de la variable Bonjour, autrement dit " Ave Cesar ". Le second affiche le libellé " Bonjour ", qui n'a rien à voir avec la variable du même nom.

Les tests

1. De quoi s'agit-il ?

Reprenons le cas de notre " programmation algorithmique du touriste égaré ". Normalement, notre algorithme ressemblera à quelque chose comme : " Allez tout droit jusqu'au prochain carrefour, puis prenez à droite et ensuite la deuxième à gauche, et vous y êtes ".

Mais en cas de doute légitime de votre part, cela pourrait devenir : " Allez tout droit jusqu'au prochain carrefour et là regardez à droite. Si la rue est autorisée à la circulation, alors prenez la et ensuite c'est la deuxième à gauche. Mais si en revanche elle est en sens interdit, alors continuez jusqu'à la prochaine à droite, prenez celle-là, et ensuite la première à droite ".

Ce deuxième algorithme a ceci de supérieur au premier qu'il prévoit, en fonction d'une situation pouvant se présenter de deux façons différentes, deux façons alternatives d'agir. Cela suppose que l'interlocuteur (le touriste) sache analyser la condition que nous avons fixée à son comportement (" la rue est-elle en sens interdit ? ") pour effectuer la série d'actions correspondante.

Eh bien, croyez le ou non, mais les ordinateurs possèdent cette aptitude, sans laquelle d'ailleurs nous aurions bien du mal à les programmer. Nous allons donc pouvoir parler à notre ordinateur comme à notre touriste, et lui donner des séries d'instructions à effectuer selon que la situation se présente d'une manière ou d'une autre.

2. Structure d'un test

Il n'y a que deux formes possibles pour un test ; la forme de gauche est la forme complète, celle de droite la forme simple.

Si booléen **Alors Si** booléen **Alors** Instructions 1 Instructions **Sinon Finsi** Instructions 2 **Finsi**

Ceci appelle quelques explications.
Un booléen est une expression dont la valeur est VRAI ou FAUX. Cela peut donc être (il n'y a que deux possibilités) :

- une variable de type booléen
- une condition

Nous reviendrons dans quelques instants sur ce qu'est une condition en informatique.

Toujours est-il que la structure d'un test est relativement claire. Arrivé à la première ligne (Si…Alors) la machine examine la valeur du booléen. Si ce booléen a pour valeur VRAI, elle exécute la série d'instructions 1. Cette série d'instructions peut être très brève comme très longue, cela n'a aucune importance. A la fin de cette série d'instructions, au moment où elle arrive au mot " Sinon ", la machine sautera directement à la première instruction située après le " Finsi ". De même, au cas où le booléen avait comme valeur " Faux ", la machine saute directement à la première ligne située après le " Sinon " et exécute l'ensemble des " instructions 2 ".

La forme simplifiée correspond au cas où l'une des deux " branches " du Si est vide. Dès lors, plutôt qu'écrire " sinon ne rien faire du tout ", il est plus simple de ne rien écrire.

Exprimé sous forme de pseudo-code, la programmation de notre touriste de tout à l'heure donnerait donc quelque chose du genre :

Exemple

Allez tout droit jusqu'au prochain carrefour **Si** la rue à droite est autorisée à la circulation **Alors** Tournez à droite Avancez Prenez la deuxième à gauche **Sinon** Continuez jusqu'à la prochaine rue à droite Prenez cette rue Prenez la première à droite **Fin si**

3. Qu'est ce qu'une condition ?

Une condition est une comparaison. Cette définition est essentielle !

C'est-à-dire qu'elle est composée de trois éléments :

- une valeur
- un **opérateur de comparaison**
- une autre valeur

Les valeurs peuvent être a priori de n'importe quel type (numériques, caractères…)

Les opérateurs de comparaison sont : =, <> ,<, >, =< ,>=

L'ensemble constitue donc si l'on veut une affirmation, qui a un moment donné est VRAIE ou FAUSSE.

A noter que ces opérateurs de comparaison s'emploient tout à fait avec des caractères. Ceux-ci sont codés par la machine dans l'ordre alphabétique, les majuscules étant systématiquement placées avant les minuscules. Ainsi on a :

"t" < "w" VRAI

"Maman" > "Papa" FAUX

"maman" > "Papa" VRAI

Attention à certains raccourcis du langage courant qui peuvent mener à des non-sens informatiques. Prenons par exemple la condition " Toto est compris entre 5 et 8 ". On peut être tenté de la traduire par : 5 < Toto < 8. Or, une telle expression, qui a du sens en mathématiques, ne veut rien dire en programmation. On va voir tout de suite après comment traduire une telle condition.

4. Conditions composées

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple exposée ci-dessus. Reprenons le cas " Toto est inclus entre 5 et 8 ". En fait cette phrase cache non une, mais **deux** conditions. Car elle revient à dire que " Toto est supérieur à 5 et Toto est inférieur à 8 ". Il y a donc bien là deux conditions, reliées par ce qu'on appelle un **opérateur logique**, le mot ET.

Comme on l'a évoqué plus haut, l'informatique met à notre disposition quatre opérateurs logiques : ET, OU, XOR (le OU exclusif) et NON.

C1 ET C2		C1	
		VRAI	FAUX
C2	VRAI	VRAI	FAUX
	FAUX	FAUX	FAUX

C1 OU C2		C1	
		VRAI	FAUX
C2	VRAI	VRAI	VRAI
	FAUX	VRAI	FAUX

C1 XOR C2		C1	
		VRAI	FAUX
C2	VRAI	FAUX	VRAI
	FAUX	VRAI	FAUX

Quelques mots pour finir là-dessus. L'opérateur NON, lui, inverse une condition :

Condition VRAI ⇔ NON (Condition) FAUX

NON (X > 15) revient à écrire X =< 15

5. Tests imbriqués

Graphiquement, on peut très facilement représenter un SI comme un aiguillage de chemin de fer (ou un aiguillage de train électrique, c'est moins lourd à porter). Un SI ouvre donc deux voies, correspondant à deux traitements différents. Mais il y a des tas de situations où deux voies ne suffisent pas. Par exemple, un programme devant donner l'état de l'eau selon sa température doit pouvoir choisir entre trois réponses possibles (solide, liquide ou gazeuse).

Exemple

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
Ecrire "C'est de la glace"
Finsi
Si Temp > 0 Et Temp < 100 Alors
Ecrire "C'est du liquide"
Finsi
Si Temp > 100 Alors
Ecrire "C'est de la vapeur"
Finsi
Fin
```

Vous constaterez que c'est un peu laborieux. Les conditions se ressemblent plus ou moins, et surtout on oblige la machine à examiner trois tests successifs alors que tous portent sur

une même chose, la température (la valeur de la variable Temp). Il serait ainsi bien plus rationnel d'**imbriquer** les tests de cette manière :

Exemple

```
Variable Temp en Entier
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
Si Temp =< 0 Alors
Ecrire "C'est de la glace"
Sinon
Si Temp < 100 Alors
Ecrire "C'est du liquide"
Sinon
Ecrire "C'est de la vapeur"
Finsi
Finsi
Fin
```

Nous avons fait des économies au niveau de la frappe du programme : au lieu de devoir taper trois conditions, dont une composée, nous n'avons plus que deux conditions simples. Mais aussi, et surtout, nous avons fait des économies sur le temps d'exécution de l'ordinateur. Si la température est inférieure à zéro, celui-ci écrit dorénavant " C'est de la glace " et passe **directement** à la fin, sans être ralenti par l'examen d'autres possibilités (qui sont forcément fausses).

Cette deuxième version n'est donc pas seulement plus simple à écrire et plus lisible, elle est également plus performante à l'exécution.

Les structures de tests imbriqués sont donc un outil indispensable à la simplification et à l'optimisation des algorithmes.

6. Variables Booléennes

Jusqu'ici, nous avons utilisé uniquement des conditions pour faire des choix. Mais vous vous rappelez qu'il existe un type de variables (les booléennes) susceptibles de stocker les valeurs VRAI ou FAUX. En fait, on peut donc entrer des conditions dans ces variables, et tester ensuite la valeur de ces variables.

Reprenons l'exemple de l'eau. On peut le réécrire ainsi :

Exemple

```
Variable Temp en Entier
Variables A, B en Booléen
Début
Ecrire "Entrez la température de l'eau :"
Lire Temp
A ⇦ Temp =< 0
B ⇦ Temp < 100
Si A Alors
Ecrire "C'est de la glace"
Sinon
B Alors
Ecrire "C'est du liquide"
Sinon
Ecrire "C'est de la vapeur"
Finsi
Finsi
Fin
```

A priori, cette technique ne présente guère d'intérêt : on a alourdi plutôt qu'allégé l'algorithme de départ, en lui faisant recourir à deux variables supplémentaires. Mais :

- une variable booléenne n'a besoin que d'un seul bit pour être stockée. L'alourdissement de ce point de vue n'est donc pas considérable.
- dans certains cas, notamment celui de conditions composées très lourdes (avec plein de ET et de OU tout partout) cette technique peut faciliter le travail du programmeur.

7. Quelques jeux logiques

Une remarque pour commencer : dans le cas de conditions composées, les parenthèses jouent un rôle important.

Exemple

```
Variables A, B, C, D, E en Booléen
Variable X en Entier
Début
Lire X
A ⇦ X < 2
B ⇦ X > 12
C ⇦ X < 6
D ⇦ (A ET B) OU C
E ⇦ A ET (B OU C)
Ecrire D, E
Fin
```

Si X = 3, alors on remarque que D sera VRAI alors que E sera FAUX.

S'il n'y a que des ET, ou que des OU, en revanche, les parenthèses ne changent strictement rien.

On en arrive à une autre propriété des ET et des OU, bien plus intéressante. Spontanément, on pense souvent que ET et OU s'excluent mutuellement, et qu'un problème donné s'exprime soit avec un ET, soit avec un OU. Pourtant, ce n'est pas si évident.

Quand faut-il ouvrir la fenêtre de la salle ? Uniquement si les conditions l'imposent, à savoir :

```
Si il fait trop chaud ET il ne pleut pas Alors
Ouvrir la fenêtre
Sinon
Fermer la fenêtre
Finsi
```

Cette petite règle pourrait tout autant être formulée comme suit :

```
Si il ne fait pas trop chaud OU il pleut Alors
Fermer la fenêtre
Sinon
Ouvrir la fenêtre
Finsi
```

Ces deux formulations sont strictement équivalentes. Ce qui nous amène à la conclusion suivante : **toute structure de test requérant une condition composée faisant intervenir l'opérateur ET peut être exprimée de manière équivalente avec un opérateur OU, et réciproquement.**

Ceci est moins surprenant qu'il n'y paraît au premier abord. Jetez pour vous en convaincre un œil sur les tables de vérité, et vous noterez la symétrie entre celle du ET et celle du OU.

Bien sûr, on ne peut pas se contenter de remplacer purement et simplement les ET par des OU ; ce serait un peu facile. La règle d'équivalence est la suivante (on peut la vérifier sur l'exemple de la fenêtre) :

```
Si A ET B Alors Si NON A OU NON B Alors
Instructions 1 Instructions 2
Sinon ⇔Sinon
Instructions 2 Instructions 1
Finsi Finsi
```

les boucles

Et ça y est, on y est, on est arrivés, la voilà, c'est Broadway, la quatrième et dernière structure : ça est les **boucles**. Si vous voulez épater vos amis, vous pouvez également parler de **structures répétitives**, voire carrément de **structures itératives**. Ca calme, hein ? Bon, vous faites ce que vous voulez, ici on est entre nous, on parlera de boucles.

1. A quoi cela sert-il donc ?

Prenons le cas d'une saisie au clavier (une lecture), par exemple, on pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non). Mais l'utilisateur, facétieux ou maladroit, risque fort tôt ou tard de taper autre chose que cela. Dès lors, le programme peut soit planter par une erreur d'exécution (parce que le type de réponse ne correspond pas au type de la variable attendu) soit se dérouler normalement jusqu'au bout, mais en produisant des résultats fantaisistes. Alors, dans tout programme un tant soit peu sérieux, on met en place ce qu'on appelle un **contrôle de saisie** (pour vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

A vue de nez, on pourrait essayer avec un SI. Voyons voir ce que ça donne :

```
Variable Rep en Caractère
Ecrire "Voulez vous un café ? (O/N)"
Lire Rep
Si Rep <> "O" ET Rep <> "N" Alors
Ecrire "Saisie erronnée. Recommencez"
Lire Rep
FinSi
```

C'est impeccable. Du moins tant que l'utilisateur a le bon goût de ne se tromper qu'une seule fois, et d'entrer une valeur correcte à la deuxième demande. Si l'on veut également bétonner en cas de deuxième erreur, il faudrait rajouter un SI. Et ainsi de suite, on peut rajouter des centaines de SI, et écrire un algorithme aussi lourd qu'une blague des Grosses Têtes, on n'en sortira pas, il y aura toujours moyen qu'un acharné flanque le programme par terre. C'est donc une impasse.

2. Une première structure de boucle

La seule issue est donc cette structure de boucle, qui se présente ainsi :

```
TantQue booléen
...
Instructions
...
FinTantQue
```

Le principe est simple : le programme arrive sur la ligne du TantQue. Il examine alors la valeur du booléen (qui, je le rappelle, peut être une variable booléenne ou, plus fréquemment, une condition). Si cette valeur est VRAI, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne FinTantQue. Il retourne ensuite sur la ligne du TantQue, procède au même examen, et ainsi de suite. Le manège enchanté ne s'arrête que lorsque le booléen prend la valeur FAUX. Illustration avec notre problème de contrôle de saisie :

```
Variable Rep en Caractère
Ecrire "Voulez vous un café ? (O/N)"
TantQue Rep <> "O" ET Rep <> "N"
Lire Rep
Si Rep <> "O" ET Rep <> "N" Alors
Ecrire "Saisie erronnée. Recommencez"
FinSi
FinTantQue
```

On remarquera que la présence du bloc " Si " est uniquement là pour l'affichage éventuel du libellé de saisie erronée. En lui-même, le bloc " tant que " est d'une simplicité biblique. Le gag de la journée : c'est d'écrire une structure TantQue dans laquelle le booléen ne devient jamais faux ! L'ordinateur tourne alors dans la boucle comme un dératé et n'en sort plus. Seule solution, quitter le programme avec un démonte-pneu ou un bâton de dynamite. Cette faute de programmation grossière – mais fréquente - ne manquera pas d'égayer l'ambiance collective de cette formation… et accessoirement d'étancher la soif proverbiale de vos enseignants.

3. Boucler en comptant, ou compter en bouclant

Dans le dernier algorithme, vous avez remarqué qu'une boucle était fréquemment utilisée pour augmenter la valeur d'une variable. Il arrive également très souvent qu'on ait besoin d'effectuer un nombre déterminé de passages. Or, a priori, notre structure TantQue ne sait pas à l'avance combien de tours de boucle elle va effectuer (cela dépend d'une condition). C'est pourquoi une autre structure de boucle est à notre disposition :

```
Variable Truc en Entier
Truc ⇦ 0
TantQue Truc < 15
Truc = Truc + 1
Ecrire "Passage numéro : ", Truc
FinTantQue Equivaut à :

Variable Truc en Entier
Pour Truc = 1 à 15
Ecrire "Passage numéro : ", Truc
Truc Suivant Au sens strict, on pourrait donc s'en dispenser, mais c'est tellement agréable de faire moins d'efforts…
```

4. Des boucles dans des boucles

On rigole, on rigole ! De même que les poupées russes contiennent d'autres poupées russes, de même qu'une structure SI … ALORS peut contenir d'autres structures SI … ALORS, une boucle peut contenir d'autres boucles.

```
Variables Truc, Trac en Entier
Pour Truc ⇦ 1 à 15
Ecrire "Il est passé par ici"
Pour Trac ⇦ 1 à 6
Ecrire "Il repassera par là"
Trac Suivant
Truc Suivant
```

Dans cet exemple, le programme écrira une fois "il est passé par ici" puis six fois de suite "il repassera par là", et ceci quinze fois en tout. A la fin, il y aura donc eu 15 x 6 = 90 passages dans la deuxième boucle (celle du milieu). Notez la différence marquante avec cette structure :

```
Variables Truc, Trac en Entier
Pour Truc ⇦ 1 à 15
Ecrire "Il est passé par ici"
Truc Suivant
Pour Trac ⇦ 1 à 6
Ecrire "Il repassera par là"
Trac Suivant
```

Ici, il y aura quinze écritures consécutives de "il est passé par ici", puis six écritures consécutives de "il repassera par là" , et ce sera tout. Si des boucles peuvent être imbriquées (cas n°1) ou successives (cas n°2), elles ne peuvent jamais, au grand jamais, être croisées. Cela n'aurait aucun sens logique, et de plus, bien peu de langages vous autoriseraient ne serait-ce qu'à écrire cette structure aberrante.

```
Variables Truc, Trac en Entier

Pour Truc ⇦...
instructions
Pour Trac ⇦ ...
instructions
Truc Suivant

instructions
Trac Suivant
```

5. Et encore une tournée générale !

Examinons l'algorithme suivant :

```
Variable Truc en Entier
Pour Truc ⇦ 1 à 15
Truc ⇦ Truc * 2
Ecrire "Passage numéro : ", Truc
Truc Suivant Vous remarquerez que nous faisons ici gérer " en double " la variable Truc, ces deux gestions étant contradictoires. D'une part, la ligne " Pour… " augmente la valeur de Truc de 1 à chaque passage. D'autre part la ligne " Truc ⇦ Truc * 2 " double la valeur de Truc à chaque passage. Il va sans dire que de telles manipulations perturbent complètement le déroulement normal de la boucle, et sont sources d'exécutions erratiques. Le gag de la journée : consiste donc à manipuler, au sein d'une boucle Pour, la variable qui sert de compteur à cette boucle. Cette technique est à proscrire absolument… sauf bien sûr, si vous cherchez un prétexte pour régaler tout le monde au bistrot. Mais dans ce cas, n'ayez aucune inhibition, proposez-le directement, pas besoin de prétexte.
```