# Algorithmique

## 1 Premiers exemples d'algorithmes

### A. En classe

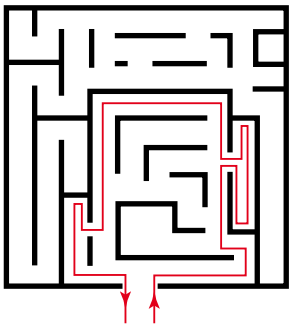
Suivre la liste d'instructions ci-dessous.

Prendre une copie double.
Écrire son nom en haut à gauche sur la 1re ligne et sa classe sur la ligne en dessous.
À la ligne suivante et à 8 cm du bord gauche de la feuille écrire : « Maths : Contrôle n°1 ».
Si la copie est à grands carreaux, tirer un trait horizontal sur toute la largeur de la feuille sur la 5e ligne en dessous du dernier texte écrit.
Si la copie est à petits carreaux, faire de même mais sur la 8e ligne en dessous du texte.
Ranger la copie pour le prochain contrôle.

### B. Sortir d'un labyrinthe

Vérifiez que la liste d'instructions ci-dessous donne bien le parcours rouge sur le labyrinthe ci-contre :

Entrez dans le labyrinthe ci-contre.
Posez la main droite sur le mur à votre droite.
Tant que vous n'avez pas atteint la sortie, longez systématiquement un mur en le gardant, sans jamais le lâcher, à main droite.
Lâchez le mur et sortez du labyrinthe.



Cette liste d'instructions n'est pas forcément performante, mais elle est ... absolument infaillible !

**Exercice :** L'appliquer pour sortir du labyrinthe de Lavergne (Lot) à imprimer sur le site.

### C. Un programme de construction géométrique

Suivre le programme de construction suivant. Que permet-il de construire ?

Placer deux points A et B dans le plan.
Tracer le cercle C de centre A passant par B.
Tracer le cercle C' de centre B passant par A.
Tracer la droite passant par les points d'intersection des deux cercles.
Appeler I le point d'intersection de cette droite et de la droite (AB).

### D. Un programme de calcul

Appliquer les instructions ci-contre aux nombres : 5 ; 7 ; 3 ; 8,3
Qu'observez-vous ?

Soustraire 4
Multiplier le résultat par 2
Ajouter 8

**Point histoire**

Le nom « algorithme » vient de Al-Khwarizmi, mathématicien perse du IXe siècle qui a introduit ce concept dans son traité « Al-Jabr wa-al-Muquabalah » où il décrit des procédures pas à pas de résolution d'équations. « Al-Jabr » est lui-même à l'origine du mot « algèbre ».

**Bilan**

Un algorithme est une liste d'instructions à suivre qui, à partir de données, permettent d'obtenir des résultats clairement définis en un nombre fini d'étapes.

Données → Traitement : liste d'instructions → Résultats

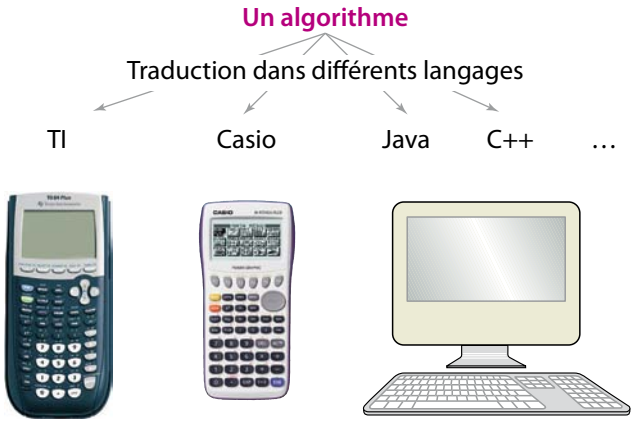
## 2 Algorithme et programme informatique

De nombreux algorithmes sont connus depuis des millénaires, comme l'algorithme d'Euclide.

Depuis l'avènement de l'informatique, on peut faire exécuter des algorithmes automatiquement par des ordinateurs.

Pour cela il faut écrire l'algorithme sous une forme bien particulière, celle d'un programme informatique écrit dans un langage que peut « comprendre » la machine : un langage de programmation (Java, C++, Php, Python, etc.).

Cependant ces langages utilisent des instructions et des structures analogues. Ce sont elles qui seront utilisées dans l'algorithme, indépendamment d'un langage.




**Bilan**

Trois étapes pour écrire un programme informatique :
– analyser le problème posé ;
– écrire un algorithme indépendamment d'un langage de programmation ;
– traduire dans un langage que « comprend » la machine que l'on va utiliser.

→ L'algorithmique

### Gérer les interactions entre la machine et l'utilisateur : entrées et sorties

Si un utilisateur veut faire exécuter par une calculatrice ou un ordinateur le « programme de calcul » ci-contre, il faut une communication entre l'utilisateur et la machine :
– la machine doit lui demander à quel nombre appliquer ce programme. Il s'agit d'une **entrée** demandée à l'utilisateur ;
– la machine doit appliquer le programme de calcul. Il s'agit du **traitement** effectué par la machine ;
– la machine doit communiquer le résultat à l'utilisateur : il s'agit de la **sortie** vers l'utilisateur.

**« Programme de calcul » comme en 3e**

Soustraire 4
Multiplier le résultat par 2
Ajouter 8

↓

**Algorithme**

ENTRÉE : Demander un nombre
TRAITEMENT : Soustraire 4
Multiplier le résultat par 2
Ajouter 8
SORTIE : Annoncer le résultat

**Bilan**

Pour écrire un algorithme, on doit envisager :
– **une phase préparatoire** : on y repère en particulier les informations à demander à l'utilisateur (nombres, mots, points, listes de nombres, etc.) appelées **entrées** ;
– **le traitement** : c'est la liste d'instructions qui seront appliquées aux données (on fait des calculs, on crée des points, des droites, des cercles, etc.) ;
– **la sortie des résultats** : ils seront en général affichés à l'écran à la fin ou petit à petit en cours de traitement (ce seront des nombres, des mots, des graphiques, des listes de nombres, etc.).

## Exercices

Écrire, en repérant les entrées, le traitement et la sortie, un algorithme :

1 qui demande à l'utilisateur un nombre et lui renvoie le carré de ce nombre ;

2 qui demande à l'utilisateur deux nombres et lui renvoie la moyenne de ces deux nombres ;

3 qui demande à l'utilisateur de placer (sur un logiciel de géométrie par exemple) trois points A, B et C et qui construit et affiche à l'écran le point D tel que ABCD est un parallélogramme.

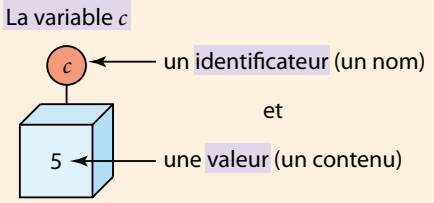
# 3 Variable et affectation

## A. Un espace de la mémoire

L'ordinateur doit stocker les données entrées par l'utilisateur et les résultats éventuels de ses calculs dans des zones de sa mémoire, et il doit savoir où il les a rangés !
Il repère donc chaque zone par une « adresse » en lui donnant un nom.

**Variable :** on peut s'imaginer une variable comme un espace de la mémoire :
– qui est désigné par un nom, qu'on choisit. Ce nom est appelé étiquette ou **identificateur** ;
– qui peut contenir une « **valeur** » (un nombre, un mot, une liste de nombres, etc.).

**Affectation :** une affectation est l'attribution d'une valeur (d'un contenu) à une variable.




La valeur 5 a été **affectée** à la variable *c*.

Dans certains langages, il est obligatoire de déclarer les variables au début du programme en donnant leurs noms, voire en indiquant si elles vont contenir des nombres, des mots, etc.

## B. Les instructions liées aux variables : « saisir », « prend la valeur », « afficher »

**Algorithme 1**

VARIABLES :
*a*, *b*, *m* nombres

ENTRÉES : Saisir *a*
Saisir *b*

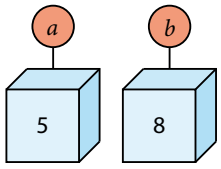
TRAITEMENT :
*m* prend la valeur $(a + b)/2$

SORTIE : Afficher *m*

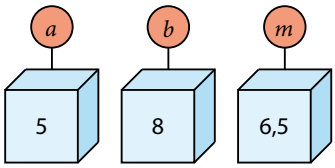
**Explications**

→ On annonce les noms des différentes variables utilisées et leur type : elles contiendront des nombres.

→ On demande à l'utilisateur de donner deux valeurs qui seront stockées dans les variables *a* et *b*.
Si l'utilisateur entre 5 puis 8 on aura en mémoire :




→ On déclenche le calcul de $(5 + 8)/2$ et on range le résultat 6,5 dans la variable *m*. On aura en mémoire :




→ On fait afficher à l'écran non pas la lettre *m* mais le contenu de la variable *m*, c'est-à-dire le nombre 6,5.

# Exercices @ @

1. Quel est le résultat affiché par l'algorithme si l'utilisateur entre les valeurs 10 pour *a* et 14 pour *b* ?
2. Modifier cet algorithme pour qu'il donne la moyenne de trois notes demandées à l'utilisateur.
3. Écrire, en utilisant variables et affectation, l'algorithme figurant page précédente.

*Voir exercice résolu 1*

## Bilan

**Saisir *a* (ou demander *a*, lire *a*)** permet de demander à l'utilisateur d'introduire une donnée.
Cette instruction joue un double rôle :
– elle crée la variable nommée *a* ;
– elle lui affecte la valeur entrée par l'utilisateur.
Elle provoque l'arrêt de l'algorithme dans l'attente d'une valeur entrée au clavier par l'utilisateur.

***m* prend la valeur (*a* + *b*)/2** permet de déclencher le calcul de (*a* + *b*)/2 avec les valeurs de *a* et *b*, puis d'affecter à la variable *m* le résultat de ce calcul. De même « *m* prend la valeur 4 » affecte 4 à *m*.
**Important !** Noter la dissymétrie de l'affectation

*m* prend la valeur (*a* + *b*)/2
la variable *m* — valeurs (contenus) des variables *a* et *b*

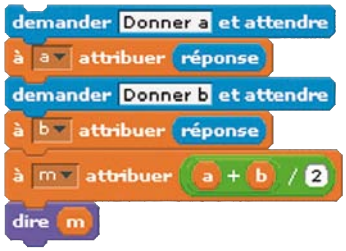
**Afficher *m*** permet d'afficher à l'écran la valeur (le contenu) de la variable *m*.

**Programmes créés à partir de l'algorithme 1 page précédente dans différents langages**

Algobox

```
VARIABLES
    a EST_DU_TYPE NOMBRE
    b EST_DU_TYPE NOMBRE
    m EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE a
    LIRE b
    m PREND_LA_VALEUR (a+b)/2
    AFFICHER m
FIN_ALGORITHME
```

Scratch (langue : Français Canada)




Calculatrices Casio

```
"A="?→A
"B="?→B
(A+B)÷2→M
M
```

Xcas (Xcasfr)

```
saisir(a);
saisir(b);
m:=(a+b)/2;
afficher(m):;
```

Scilab

```
1 a=input("a=");
2 b=input("b=");
3 m=(a+b)/2;
4 disp(m);
```

Calculatrices TI

```
:Input "A=",A
:Input "B=",B
:(A+B)/2→M
:Disp M
```

On met entre guillemets dans un algorithme les messages affichés tels quels à l'utilisateur.
L'affectation se fait par le symbole : = sur Xcas, par la flèche → sur les calculatrices et par un signe d'égalité sur Scilab.

## Comment faire pour lire et comprendre un algorithme qui est donné ?

On exécute les instructions pas et pas, comme le ferait une machine. Il peut être très utile de faire un état des différentes variables pas à pas, en choisissant des valeurs pour les entrées demandées à l'utilisateur comme dans les explications données en 3.B.

Plutôt que les dessins faits en 3.B, on peut présenter cet état des variables dans un tableau comme ci-contre en indiquant à chaque ligne les contenus des variables.

| Instructions | Contenus des variables | | |
|---|---|---|---|
| Saisir *a* | *a* : 5 | | |
| Saisir *b* | *a* : 5 | *b* : 8 | |
| m prend la valeur (*a* + *b*)/2 | *a* : 5 | *b* : 8 | *m* : 6,5 |

## C. Réduire le nombre de variables

Une variable $z$ a pour valeur $v$. Si on affecte à cette variable $z$ une nouvelle valeur $e$, l'ancienne valeur $v$ est effacée et remplacée par $e$.

Ainsi l'instruction « **$b$ prend la valeur $b + 2$** » déclenche le calcul de $b + 2$ pour le contenu de $b$ puis range le résultat dans $b$ en écrasant le contenu précédent.

**Exemple :**

| Instructions | Contenus des variables | |
|---|---|---|
| $a$ prend la valeur 5 | $a$ : 5 | |
| $b$ prend la valeur 6 | $a$ : 5 | $b$ : 6 |
| $a$ prend la valeur $b + 3$ | $a$ : 9 | $b$ : 6 |
| **$b$ prend la valeur $b + 2$** | $a$ : 9 | $b$ : 8 |

**Exercice :**

Recopier les tableaux ci-dessous (ou utiliser ceux disponibles sur le site) et compléter l'état des variables en supposant que l'utilisateur entre 5 comme valeur de $x$. À quel programme de calcul correspondent ces deux algorithmes ? Comparer leurs avantages et leurs inconvénients.

**Algorithme 2**

| VARIABLES : $a$, $b$, $c$, $d$, $x$ nombres | Contenus des variables | | | | |
|---|---|---|---|---|---|
| ENTRÉES : Saisir $x$ | $x$ : | | | | |
| TRAITEMENT : $a$ prend la valeur $x + 4$ | $x$ : | $a$ : | | | |
| $b$ prend la valeur $2 \times a$ | $x$ : | $a$ : | $b$ : | | |
| $c$ prend la valeur $b - 3$ | $x$ : | $a$ : | $b$ : | $c$ : | |
| $d$ prend la valeur $c - x$ | $x$ : | $a$ : | $b$ : | $c$ : | $d$ : |
| SORTIE : Afficher $d$ | | | | | |

**Algorithme 3**

| VARIABLES : $a$, $x$ nombres | Contenus des variables | |
|---|---|---|
| ENTRÉES : Saisir $x$ | $x$ : | |
| TRAITEMENT : $a$ prend la valeur $x + 4$ | $x$ : | $a$ : |
| $a$ prend la valeur $2 \times a$ | $x$ : | $a$ : |
| $a$ prend la valeur $a - 3$ | $x$ : | $a$ : |
| $a$ prend la valeur $a - x$ | $x$ : | $a$ : |
| SORTIE : Afficher $a$ | | |

*Voir exercice résolu 1*

**Bilan**

Réduire le nombre de variables dans un algorithme permet de diminuer la place prise en mémoire par un programme dans une machine. En revanche l'algorithme peut perdre en lisibilité.
Compte tenu de la puissance des machines à l'heure actuelle, on cherche souvent un compromis entre la réduction du nombre de variables et la conservation d'une bonne lisibilité.

## 4 Structure alternative : « Si ...Alors...Sinon »

**Exemple:**

Appliquer l'algorithme ci-contre aux nombres 18, 21 et 237.

**Exercice :** Modifier l'algorithme 4 pour qu'il affiche si un nombre donné est un multiple de 7.

**Algorithme 4**

```
Si n a pour reste 0 dans la division par 2 Alors
    afficher "nombre pair"
    Sinon
    afficher "nombre impair"
FinSi
```

La structure alternative s'exprime ainsi :

```
Si condition Alors
    suite d'instructions 1 (si la condition est vraie)
    Sinon
    suite d'instructions 2 (si la condition est fausse)
FinSi
```

La condition ne doit offrir que deux réponses : vraie ou fausse !
Quand la condition est vraie, on exécute la suite d'instructions 1 ; quand elle est fausse, on exécute la suite d'instructions 2.

Dans certains cas, il n'y a pas d'instruction à effectuer quand la condition est fausse. On écrit alors :

**Si** *condition* **Alors**
| *suite d'instructions 1 (si la condition est vraie)*
**FinSi**

Si la condition est vraie, la suite d'instructions 1 est exécutée. Si la condition est fausse, on passe à la suite de l'algorithme.

**La structure alternative de l'algorithme 4 dans différents langages (programmes complets sur le site)**

Algobox

```
SI (n%2==0) ALORS
    DEBUT_SI
    AFFICHER "le nombre est pair"
    FIN_SI
    SINON
        DEBUT_SINON
        AFFICHER "le nombre est impair"
        FIN_SINON
```

Scratch


TI

```
:If ent(N/2)=N/2
:Then
:Disp "PAIR"
:Else
:Disp "IMPAIR"
:End
```

Xcas

```
si irem(n,2)==0 alors
afficher("nombre pair");
sinon
afficher("nombre impair");
fsi:;
```

Scilab

```
2 if reste(n,2)==0 then
3   disp("nombre pair")
4 else
5   disp("nombre impair")
6 end
```

Casio Graph 35+

```
If MOD(N,2)=0↵
Then "PAIR"⌟
Else "IMPAIR"⌟
IfEnd
```

**Attention !** Le test d'une égalité s'écrit souvent avec un double signe d'égalité ==
Aide : « *n*%2 », « irem (*n*, 2) », « reste(*n*, 2) », « mod(*n*, 2) » donnent le reste de la division euclidienne de *n* par 2.
Les calculatrices TI82, 83 ne disposent pas de cette fonction « reste » : on teste alors si *n*/2 est un entier (en testant s'il est égal à sa partie « avant la virgule » ent(*n*/2). Il faut s'adapter au langage de programmation utilisé …

*Voir exercices résolus 2 et 3*

# 5 Structures itératives : boucles

## A. Quand on connaît le nombre de répétitions

### Exemple

L'algorithme ci-contre donne le résultat obtenu en augmentant de 4 % un montant *S* en euros (rappel : le résultat est $S \times (1 + 4/100) = S \times 1{,}04$).
On dépose sur un livret d'épargne un montant *S* et il augmente chaque année de 4 %. On veut connaître le montant obtenu au bout de 10 ans.
On répète pour cela 10 fois de suite l'augmentation de 4 %. On obtient l'algorithme 5 :

*i* prend la valeur 1, l'*instruction* s'effectue
*i* prend la valeur 2, l'*instruction* s'effectue
etc … *i* prend la valeur 10, l'*instruction* s'effectue.

C'est fini.

On parle d'une boucle et l'on dit que l'on passe « 10 fois dans la boucle ».

VARIABLE : *S* nombre
ENTRÉE : Saisir *S*
TRAITEMENT : *S* prend la valeur $S \times 1{,}04$
SORTIE : Afficher *S*

**Algorithme 5**

VARIABLE : *S* nombre, *i* nombre
ENTRÉE : Saisir *S*
TRAITEMENT :
**Pour i allant** de 1 **à** 10 **Faire**
| *S prend la valeur S × 1.04*
**FinPour**
SORTIE : Afficher *S*

# Exercices

1 Modifier l'algorithme 5 ci-dessus pour qu'il donne le montant obtenu au bout de 15 ans.

2 Modifier l'algorithme 5 ci-dessus pour qu'il demande à l'utilisateur le nombre *n* d'années où l'argent est laissé sur le compte et qu'il affiche le montant alors obtenu.

*Voir exercice résolu 4, question 1*

# Algorithmique

**Bilan**

La répétition d'une suite d'instructions un certain nombre de fois s'appelle une **boucle** ou une **structure itérative**.

Une répétition 100 fois d'une suite d'instructions s'écrit :

**Pour *i* allant de 1 à 100 Faire**
| *Suite d'instructions*
**FinPour**

Dans l'algorithme ci-dessus, *i* est le **compteur** de la boucle. Il est **incrémenté** (augmenté) de 1 à chaque passage dans la boucle. On peut lui donner un autre nom que *i*. Il peut aller de 4 à 200, ou même de 1 à *n* où *n* est une variable contenant un nombre entier (au moins égal à 1).

**Les programmes écrits à partir de l'algorithme 5 dans différents langages**

Algobox

```
VARIABLES
    S EST_DU_TYPE NOMBRE
    i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE S
    POUR i ALLANT_DE 1 A 10
        DEBUT_POUR
        S PREND_LA_VALEUR S*1.04
        FIN_POUR
    AFFICHER S
FIN_ALGORITHME
```

Scratch


TI

```
:Input "S=",S
:For(I,1,10)
:S*1.04→S
:End
:Disp S
```

Xcas

```
saisir(S);
pour k de 1 jusque 10 faire
S:=S*1.04;
fpour;
afficher(S):;
```

Attention. Le nom *i* est réservé et ne peut être utilisé.

Scilab

```
1 S=input("S=");
2 for i=1:10
3   S=S*1.04;
4 end
5 disp(S);
```

Casio Graph 35+

```
"S="?→S↵
For 1→I To 10↵
S×1.04→S↵
Next↵
S↵
```

**Exercice**

Quels affichages obtient-on avec les algorithmes ci-contre ? On dressera pour chacun un état des variables dans la boucle :

| | | |
|---|---|---|
| Boucle | *i* : 1 | *n* : |
| | *i* : 2 | *n* : |
| | … | … |

**Algorithme 6**

VARIABLES : *n*, *i* nombres
TRAITEMENT :
  Pour *i* allant de 1 à 9 Faire
  | *n* prend la valeur $7 \times i$
  | SORTIE : Afficher *n*
  FinPour

**Algorithme 7**

VARIABLES : *n*, *i* nombres
INITIALISATION :
  *n* prend la valeur 1
TRAITEMENT :
  Pour *i* allant de 1 à 9 Faire
  | *n* prend la valeur $2 \times n$
  FinPour
SORTIE : Afficher *n*

## B. Quand on connaît un test d'arrêt mais pas le nombre de répétitions

**Exemple :** On dépose sur un livret d'épargne une somme inférieure ou égale à 5 000 €. Elle augmente chaque année de 4 %. On veut savoir au bout de combien d'années elle aura dépassé 6 000 €.

Il s'agit encore d'une boucle mais on ne connaît pas le nombre de passages dans la boucle.
On va répéter l'augmentation jusqu'à ce que le montant devienne supérieur à 6 000 €.
Il faudra créer une variable qui compte le nombre de passages dans la boucle c'est-à-dire le nombre d'années. Cette variable s'appelle un compteur. On a deux structures possibles.

### Algorithme 8 : Tant que …

VARIABLES : $S$, $n$ nombres
ENTRÉE : Saisir $S$
INITIALISATION : $n$ prend la valeur 0
TRAITEMENT :
  **Tantque** $S \leqslant 6000$ **Faire**
    $S$ prend la valeur $S \times 1{,}04$
    $n$ prend la valeur $n + 1$
  **FinTantque**
SORTIE : Afficher $n$

### Algorithme 9 : Répéter … Jusqu'à

VARIABLE : $S$, $n$ nombres
ENTRÉE : Saisir $S$
INITIALISATION : $n$ prend la valeur 0
TRAITEMENT :
  **Répéter**
    $S$ prend la valeur $S \times 1{,}04$
    $n$ prend la valeur $n + 1$
  **Jusqu'à** $S > 6000$
SORTIE : Afficher $n$

## Bilan

Pour écrire une boucle avec un test d'arrêt on a deux structures possibles :

**Tantque** *condition* **Faire**
| *Suite d'instructions*
**FinTantque**

On commence par tester si la condition est vraie.
Si elle est vraie, on exécute la suite d'instructions et on recommence.
Si elle est fausse, on s'arrête et on sort de la boucle.

**Répéter**
| *Suite d'instructions*
**Jusqu'à** *condition*

Après avoir exécuté la suite d'instructions, on teste si la condition est vraie. Si elle est vraie, on s'arrête et on sort de la boucle ; si elle est fausse, on recommence la suite d'instructions.

Avec une structure « répéter jusqu'à », la suite d'instructions est exécutée au moins une fois.
Avec la structure « Tant que », elle peut ne pas l'être puisque l'on peut ne pas entrer dans la boucle.

La boucle « Tant que » de l'algorithme 8 dans différents langages (programmes complets sur le site) @ @

**Algobox**

```
TANT_QUE (S<=6000) FAIRE
    DEBUT_TANT_QUE
    S PREND_LA_VALEUR S*1.04
    n PREND_LA_VALEUR n+1
    FIN_TANT_QUE
```

**Calculatrice TI**

```
:While S≤6000
:S*1.04→S
:N+1→N
:End
```

**Scilab**

```
while S<=6000
  S=S*1.04;
  n=n+1;
end
```

**Calculatrice Casio**

```
While S≤6000↵
S×1.04→S↵
N+1→N↵
WhileEnd↵
```

**Xcas**

```
tantque S<=6000 faire
S:=S*1.04;
n:=n+1;
ftantque;
```

La boucle « Répéter jusqu'à » de l'algorithme 9 dans différents langages (programmes complets sur le site) @ @

**Scratch**


**Xcas**

```
repeter
S:=S*1.04;
n:=n+1;
jusqua S>6000;
```

**Calculatrice TI**

```
:Repeat S>6000
:S*1.04→S
:N+1→N
:End
```

## Exercices @ @

1 Vérifier en dressant l'état des variables pour une somme $S$ de 4 800 € que les algorithmes 8 et 9 ci-dessus font bien afficher le même nombre d'années $n$.
2 Modifier les algorithmes 8 et 9 pour faire afficher le nombre d'années nécessaire pour doubler la somme $S$ de départ.
3 S'inspirer des algorithmes ci-dessus pour demander un nombre entier naturel à l'utilisateur et renvoyer le plus petit entier naturel $n$ tel que $2^n$ soit plus grand que ce nombre.

*Voir exercice résolu 4*

# Algorithmique

## 1 Passer du langage naturel à l'écriture d'un algorithme avec variables et affectation

### Énoncé

**1.** Appliquer le « programme de calcul » ci-contre plusieurs fois.
Qu'observez-vous ?
**2.** Écrire un algorithme correspondant à ce « programme de calcul »
de façon formalisée.

Choisir deux entiers $a$ et $b$ entre 1 et 9.
Multiplier $a$ par 5.
Ajouter 7.
Doubler le résultat obtenu.
Ajouter $b$.
Soustraire 14.

### Solution

**1.** Pour $a = 3$, $b = 8$ on obtient 38. Pour $a = 2$ et $b = 1$ on obtient 21.
Il semble que l'on obtient le nombre formé de $a$ dizaines et $b$ unités.

**2.** On peut introduire une variable par donnée ou par résultat à stocker dans la mémoire de l'ordinateur (algorithme 1).
On peut ensuite réduire le nombre de variables (algorithme 2) :

**Algorithme 1**

VARIABLES : $a$, $b$, $c$, $d$, $e$, $f$, $g$ nombres
ENTRÉES : Saisir $a$
Saisir $b$
TRAITEMENT :
$c$ prend la valeur $5 \times a$
$d$ prend la valeur $c + 7$
$e$ prend la valeur $2 \times d$
$f$ prend la valeur $e + b$
$g$ prend la valeur $f - 14$
SORTIE : Afficher $g$

ou

**Algorithme 2**

VARIABLES : $a$, $b$, $c$ nombres
ENTRÉES : Saisir $a$
Saisir $b$
TRAITEMENT :
$c$ prend la valeur $5 \times a$
$c$ prend la valeur $c + 7$
$c$ prend la valeur $2 \times c$
$c$ prend la valeur $c + b$
$c$ prend la valeur $c - 14$
SORTIE : Afficher $c$

## 2 Écrire un algorithme utilisant une structure alternative « Si…Alors… »

### Énoncé

Écrire un algorithme qui demande l'heure (en heures et minutes) à Paris en été et renvoie l'heure à Tokyo.
Le décalage horaire à Tokyo est + 7 h en été.

### Solution

**Analyse :** on commence en prenant des exemples :
– s'il est 5 h 20 à Paris, il est 12 h 20 à Tokyo ;
– s'il est 20 h 30 à Paris, il est 20 h 30 + 7 h soit 3 h 30 le lendemain matin à Tokyo.
Il faut donc d'abord demander à l'utilisateur le nombre d'heures et de minutes à Paris.
Ensuite il faut ajouter 7 au nombre d'heures.
Mais si le nombre d'heures est supérieur à 24, il faut enlever 24 et prévenir qu'on est le lendemain…
Il faut enfin afficher l'heure à Tokyo.

**Algorithme**

VARIABLES : $h$, $m$ nombres
**// $h$ est le nombre d'heures, $m$ celui de minutes**
ENTRÉES :
Saisir $h$
Saisir $m$
TRAITEMENT :
$h$ prend la valeur $h + 7$
Si $h > 24$ Alors
afficher « A Tokyo il est déjà demain »
$h$ prend la valeur $h - 24$
FinSi
SORTIE : Afficher $h$ « heures » $m$ « minutes »

**Aide :** le symbole // annonce souvent un commentaire dans un programme. Il en facilite la lecture.

**Aide :** cette instruction provoque l'affichage du contenu de $h$ suivi du mot « heures », du contenu de $m$ et du mot « minutes ».

# Exercices résolus

## 3 Utiliser une structure « Si …Alors…Sinon… »

### Énoncé

Voici un programme de calcul :
Choisir un nombre entier. S'il est pair, le diviser par 2,
sinon le multiplier par 3 et ajouter 1.

**1.** Appliquer ce programme de calcul à 8 et à 11.

**2.** Écrire l'algorithme correspondant à ce programme de calcul.

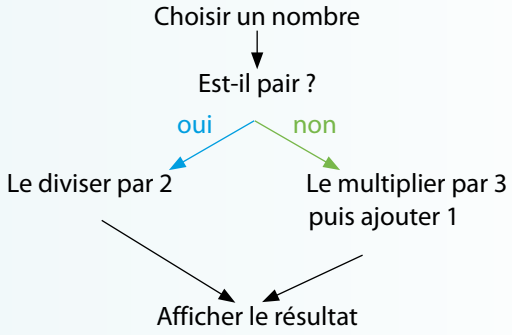
*@ @ En répétant cet algorithme on obtient une célèbre conjecture ! TP disponible sur le site.*

### Solution

**1.** Avec 8, on obtient 4.
Avec 11, on obtient 34.

**2.**
```
VARIABLES : n, p nombres
ENTRÉE : Saisir n
TRAITEMENT :
Si n est pair Alors
    p prend la valeur n/2
    Sinon
    p prend la valeur 3 × n +1
FinSi
SORTIE : Afficher p
```

**Aide :** On représente parfois une structure alternative par un schéma.




## 4 Utiliser les structures itératives pour répéter des lancers de dé

### Énoncé

On notera NbAlea (1 ; 6) le tirage d'un nombre aléatoire entier entre 1 et 6.

**1.** Écrire un algorithme qui simule 100 lancers d'un dé et qui compte le nombre de 6 obtenus.

**2.** Écrire un algorithme qui simule le lancer d'un dé jusqu'à l'apparition d'un 6 et qui affiche le nombre de lancers effectués pour obtenir ce premier 6.

### Solution

**1. Analyse** : il faut répéter 100 fois le lancer d'un dé.
C'est une boucle « Pour … FinPour ». Il faut aussi créer une variable (compteur) qui compte le nombre de 6 :
– au début elle aura la valeur 0 (on l'initialise à 0) car on n'a pas de 6 ;
– à chaque lancer, on augmentera sa valeur de 1 si on obtient un 6.

**Algorithme 1**
```
VARIABLES : C, i, D entiers
INITIALISATION :
    C prend la valeur 0  // C est le compteur
TRAITEMENT :
    Pour i allant de 1 à 100 Faire
        D prend la valeur NbAlea(1; 6)
        Si D = 6 Alors
            C prend la valeur C + 1
        FinSi
    FinPour
SORTIE : Afficher C
```

**2. Analyse** : il faut aussi répéter le lancer du dé. On ne connaît pas le nombre de répétitions mais on connaît la condition d'arrêt : obtenir 6.
C'est une boucle « Tant que … » ou « Répéter …jusqu'à ». Comme on va effectuer au moins un lancer, on choisit plutôt « Répéter… jusqu'à ».
On a aussi besoin d'un compteur qui compte le nombre de lancers : il aura pour valeur 0 au début et on augmentera sa valeur de 1 à chaque lancer.

**Algorithme 2**
```
VARIABLES : C, D entiers
INITIALISATION :
    C prend la valeur 0     // C est le compteur
TRAITEMENT :
    Répéter
        D prend la valeur NbAlea(1; 6)
        C prend la valeur C + 1
    Jusqu'à D = 6
SORTIE : Afficher C
```