Algorithmes (5)
Instructions itératives ou boucles (nombre d'itérations variable)

Objectif : étudier un nouvel exemple de boucle appelée **boucle « Tantque »**.
Il s'agit des boucles **conditionnelles** ou **boucles avec test**.

I. Exemple introductif

1°) Situation

En septembre 2004, un iceberg de 25 tonnes fait le voyage d'Islande vers Brest. La température locale aidant, il perd 10 % de sa masse chaque jour.
Déterminer à partir de quel jour il reste une quantité de glace inférieure ou égale à un kilogramme.

2°) Analyse du problème

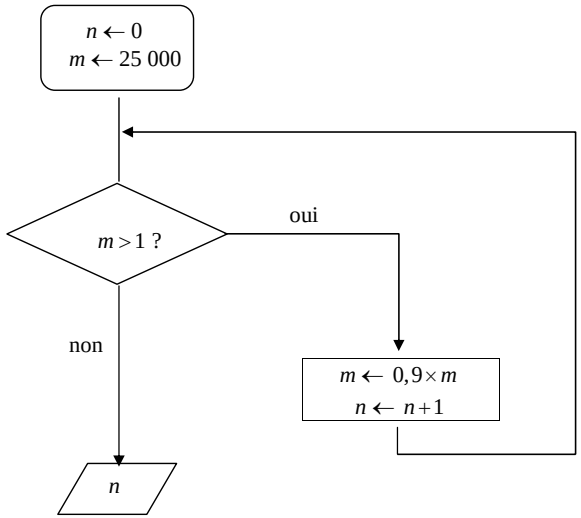
Chaque jour, la masse de l'iceberg diminue de 10 %.

D'un jour à l'autre, la masse de l'iceberg est donc multipliée par $1-\frac{10}{100}=0,9$ (coefficient multiplicateur).

Avec la calculatrice, on pourrait calculer la masse de l'iceberg le 1er jour, puis le 2e jour, puis le 3e jour etc.
Ce serait long et fastidieux. Le recours à un algorithme s'avère particulièrement efficace pour résoudre rapidement le problème.

3°) Présentation en organigramme

On va utiliser une boucle.
On ne connaît pas le nombre d'itérations à l'avance.



Le test est présenté dans un **losange**.

4°) Rédaction d'un algorithme en langage naturel

Les variables qui interviennent sont m et n. Ce sont des nombres.

Initialisations :
m prend la valeur 25 000
n prend la valeur 0

Traitement :
Tantque $m>1$ **Faire**
    m prend la valeur $0,9\times m$
    n prend la valeur $n+1$
FinTantque

Sortie :
Afficher n

Comme on l'a déjà dit, l'algorithme fait intervenir deux variables : m et n.

m désigne la masse de l'iceberg en kg.

n désigne le nombre de jours.

Il n'y a pas d'entrée dans cet algorithme.

Le test d'arrêt est « $m>1$ ». On « sort » de la boucle lorsque $m\leqslant 1$ (d'une certaine manière, on peut dire que le test d'arrêt est $m\leqslant 1$).
Les valeurs de m et n sont modifiées à chaque passage dans la boucle.

Commentaires sur cet algorithme :

1. Cet algorithme sert à déterminer une valeur seuil.

2. Les instructions qui interviennent dans la boucle (« m prend la valeur $0,9\times m$ » et « n prend la valeur $n+1$ »), sont interchangeables car n n'intervient pas dans la première instruction.

La variable n a le rôle d'un compteur (initialisation « n prend la valeur 0 ; instruction « n prend la valeur $n+1$ »).
n commence à 0 et augment de 1 à chaque passage dans la boucle jusqu'à atteindre la valeur finale qui sera affichée en sortie.

II. Notions à connaître

1°) Condition (ou test)

Dans une boucle, le nombre d'itérations **peut dépendre d'une condition** ; dans ce cas, le traitement est répété tant que la condition est vraie.
Lorsque la condition est fausse, on sort de la boucle.

2°) Syntaxe d'une boucle « Tantque »

Pour écrire une boucle avec un test d'arrêt, on utilise la structure :

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

3°) Commentaires

• Le nombre d'itérations n'est pas connu à l'avance. On parle parfois de **boucle non bornée**.

• Pour rendre la lecture plus claire, toutes les actions à effectuer au sein de la boucle sont **écrites en retrait** (on appelle cela des **« indentations »**).

• Avec la structure « Tantque », il est possible que la suite d'instructions ne soit pas exécutée au moins une fois.

III. Programmation en Python

L'expression « tant que » se traduit « while » en anglais. En programmation, on parle de boucle « While ».

En langage naturel
Tantque condition **Faire**   instructions **Tantque**

En langage Python
`while [condition]:`   [instructions]

Langage naturel
On utilise une barre d'indentation.

Langage Python
On doit faire à bien écrire les instructions du bloc while en décalage (indentation). Il faut penser aux deux points à la fin de la ligne.

Lorsque l'on écrit un programme en Python, il est indispensable de bien respecter les indentations. Si on oublie de décaler, on obtient un message d'erreur lorsqu'on le fait tourner.
Les indentations permettent une meilleure lisibilité du programme : on repère ainsi très bien le début et la fin de la boucle.

Il n'existe pas d'instruction pour définir la fin de la boucle. C'est l'indentation, c'est-à-dire le décalage vers la droite d'une ou plusieurs lignes, qui permet de marquer la fin de la boucle.

On peut écrire le programme Python correspondant à l'exemple du **I**.

```
m = 25000
n = 0
while m > 1:
        m = 0.9 * m
        n = n + 1
print(n)
```

Il reste moins d'un kilogramme de glace au bout de **97 jours**.

IV. Bilan sur les boucles

1°) Choix d'une boucle

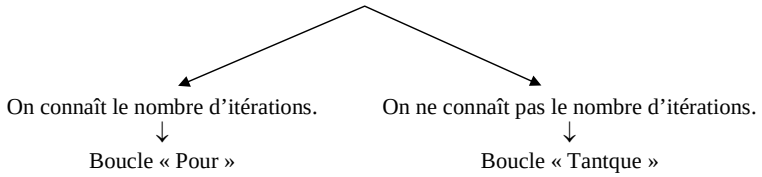
On retiendra :

• Lorsque l'on connaît le nombre d'itérations, on utilise une boucle « Pour ».

• Lorsque l'on ne connaît pas le nombre d'itérations, on utilise une boucle « Tantque ».

Bilan sur les boucles :

choix d'une boucle



2°) Utilisation des boucles « Tantque »

Cette année et l'année prochaine, nous utiliserons beaucoup les boucles « Tantque » pour les problèmes de détermination d'une valeur seuil.

3°) Utilisation de la logique

Les boucles « Tantque » font appel à la logique, en particulier à la négation d'une phrase (cf. cours de logique) pour l'écriture du test d'arrêt.

Exemples importants :

La négation de la phrase « $u > 1$ » est « $u \leqslant 1$ ».

La négation de la phrase « $u = 3$ » est « $u \neq 3$ ».

Savoir-faire exigibles

- Comprendre l'intérêt d'une boucle « Tantque ».

- Savoir lire, comprendre et interpréter un algorithme avec une boucle « Tantque ».

- Comprendre le rôle et l'intérêt des variables intervenant dans une boucle « Tantque » (notamment le rôle du « compteur »).

- Savoir passer d'un algorithme simple rédigé en langage naturel comprenant une boucle « Tantque » à un organigramme et réciproquement.

- Savoir rédiger un algorithme simple comprenant une boucle « Tantque » en respectant la syntaxe pour ce type d'instruction.

- Savoir écrire en langage naturel l'algorithme d'Euclide pour déterminer le PGCD de deux réels.

- Savoir refaire réécrire l'algorithme de l'exemple du cours (exemple-type d'algorithme de détermination d'une valeur seuil).

- Savoir utiliser une initialisation de variable(s).

Résumé

Il est fondamental de retenir qu'une boucle « Tantque » comporte une condition d'arrêt qui permet de répondre à la question « Quand s'arrête l'algorithme ? ».

Par exemple, l'algorithme d'Euclide s'arrête lorsque le reste est nul.

Exercice-type

On considère l'algorithme suivant, rédigé en langage naturel :

Initialisations :
u prend la valeur 1
n prend la valeur 0

Traitement :
Tantque $u \geqslant 0,001$ **Faire**
    u prend la valeur $\frac{u}{2}$
    n prend la valeur $n+1$
FinTantque

Sortie :
Afficher n

1°) Faire tourner l'algorithme « à la main » et indiquer la valeur de n affichée en sortie.

2°) Programmer l'algorithme sur calculatrice et vérifier les résultats du 1°).

Solution :

1°) On remplit un tableau (moyen assez commode de présenter l'évolution des variables).

1ère façon :

Étape	Test $u \geqslant 0,001$	u *	n
Initialisation		1	0
1	Vrai	0,5	1
2	Vrai	0,25	2
3	Vrai	0,125	3
4	Vrai	0,0625	4
5	Vrai	0,03125	5
6	Vrai	0,015625	6
7	Vrai	0,0078125	7
8	Vrai	0,00390625	8
9	Vrai	0,00193125	9
10	Vrai	0,00009765625	10
11	Faux		

* On pourrait donner les résultats sous forme de fraction ($\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ …).
La calculatrice donne les résultats sous forme décimale.

On pourrait représenter l'algorithme par un organigramme.

Pour l'étape 11, on sort de la boucle (on a donc fait 10 itérations) et l'algorithme affiche en sortie $n = 10$ (dernière valeur de la variable n obtenue dans l'algorithme). Le test étant faux, l'algorithme s'arrête ; il serait donc absurde de continuer à donner des valeurs à u (autrement dit, il serait absurde de poursuivre l'algorithme).

2e façon : On peut adopter la présentation suivante (en lignes ou en colonnes) qui est meilleure que la 1ère.

Étape	0	1	2	3	4	5	6	7	8	9	10
Valeur de u	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	$\frac{1}{64}$	$\frac{1}{128}$	$\frac{1}{256}$	$\frac{1}{512}$	$\frac{1}{1024}$
Valeur de n	0	1	2	3	4	5	6	7	8	9	10
Condition $u \geqslant 0,001$	V	V	V	V	V	V	V	V	V	V	F

Cet algorithme affiche le plus petit entier naturel n tel que $\frac{1}{2^n} < 0,001$.
Il s'agit d'un algorithme de détermination de valeur seuil.

2°) Le programme comprend 6 instructions.

Exercices sur les boucles « Tantque »

• Les exercices s'articulent autour des compétences suivantes : comprendre, expliquer, interpréter, modifier, écrire, programmer un algorithme avec une boucle « Tantque ».

• Dans les exercices où un algorithme est donné, il est demandé de le recopier pour s'imprégner de la rédaction.

1 On considère l'algorithme ci-dessous rédigé en langage naturel qui fait intervenir les variables x, réel positif ou nul et n, entier naturel.

Entrée :
Saisir x

Initialisation :
n prend la valeur 0

Traitement :
Tantque $n+1 \leqslant x$ **Faire**
    n prend la valeur $n+1$
FinTantque

Sortie :
Afficher n

Recopier cet algorithme dans un cadre bien centré, sur une même page, en respectant la présentation et la rédaction.

1°) Représenter l'organigramme correspondant à cet algorithme.
2°) Faire tourner l'algorithme « à la main » pour les valeurs suivantes de la variable x saisie en entrée : 2 ; 3,1 ; 9,8.
Afin de mieux voir le déroulement pas à pas de l'algorithme, faire pour chaque valeur, un tableau d'évolution de la variable n sur le modèle suivant.

Étape	Test $x \geqslant n+1$	n
Initialisation		0
1	VRAI	
2		
3		

3°) Comment peut-on définir le nombre de sortie par rapport au nombre d'entrée n ?

4°) Programmer cet algorithme en Python (éventuellement en version fonction) et vérifier qu'il fonctionne correctement.

2 On considère l'algorithme ci-dessous rédigé en langage naturel qui fait intervenir les variables n et u, entiers naturels.

Entrée :
Saisir n

Initialisation :
u prend la valeur n

Traitement :
Tantque $u \geqslant 11$ **Faire**
    u prend la valeur $u-11$
FinTantque

Sortie :
Afficher u

Cet algorithme fait intervenir deux variables (n et u) et comporte une boucle « Tantque ».
La condition de la boucle est « $u \geqslant 11$ ».
Recopier cet algorithme.

1°) Faire tourner cet algorithme « à la main » pour $n = 35$, $n = 50$ puis pour $n = 55$.
Indiquer le nombre de sortie dans chacun des deux cas.

On utilisera dans chaque cas un tableau d'évolution de la variable u selon le modèle suivant :

Étape	$u \geqslant 11$	u
Initialisation		35
1	VRAI	
2		
3		

2°) Quel lien existe-t-il entre le nombre n en entrée et le nombre u obtenu en sortie ? Faire une phrase expliquant ce que représente le nombre u de sortie par rapport au nombre d'entrée n.

Quel est le nombre de sortie lorsque le nombre d'entrée n est égal à 2011 ?

3°) Programmer cet algorithme en Python (éventuellement en version fonction) et vérifier les résultats précédents.

3 **Mise en garde**

Paul pense qu'il a compris comment fonctionne la boucle « Tantque » dans un algorithme.
Voici l'algorithme qu'il a écrit en langage naturel pour obtenir la suite des entiers naturels dont le carré est inférieur ou égal à un réel A strictement positif saisi en entrée.

Entrée :
Saisir A

Initialisation :
n prend la valeur 0

Traitement et sortie :
Tantque $n^2 \leqslant$ A **Faire**
    Afficher *n*
FinTantque

Paul a-t-il vraiment compris le fonctionnement de la boucle « Tantque » ?
Corriger son erreur.
Recopier l'algorithme correct.

4 On dispose d'un dé cubique non truqué dont les faces sont numérotées de 1 à 6.
On considère l'expérience aléatoire qui consiste à lancer le dé plusieurs fois.
On note chaque fois le numéro de la face supérieure.
On s'arrête dès qu'on obtient le numéro 6 pour la première fois.
Écrire une fonction Python qui simule l'expérience aléatoire.

5 On place un euro à la banque sur un compte rémunéré au taux annuel de 3 % (à intérêts composés).
Le montant de l'intérêt est donc égal à 3 % du capital de l'année précédente.
1°) Quel est le coefficient multiplicateur associé à une augmentation de 3 % ?
2°) Rédiger en langage naturel un algorithme permettant de savoir au bout de combien d'années on aura une somme supérieure ou égale à 10 euros.
Réaliser en Python le programme correspondant à cet algorithme et répondre à la question posée.

6 Une expérience consiste à lancer deux dés cubiques équilibrés dont les faces sont numérotées de 1 à 6. On répète cette expérience tant que les deux faces obtenues sont distinctes, on s'arrête lorsqu'elles sont égales et l'algorithme de simulation ci-dessous affiche le nombre de répétitions obtenues.

Initialisations :
x prend la valeur 1
y prend la valeur 2
n prend la valeur 0

Traitement :
Tantque $x \neq y$ **Faire**
    x prend la valeur d'un entier aléatoire de 1 à 6
    y prend la valeur d'un entier aléatoire de 1 à 6
    n prend la valeur $n+1$
FinTantque

Sortie :
Afficher *n*

Recopier cet algorithme.

1°) a) Expliquer la boucle effectuée dans cet algorithme. Peut-on prévoir à l'avance le nombre de passages dans la boucle ?
b) Pourquoi les variables *x* et *y* sont-elles initialisées respectivement à 1 et à 2 au début de l'algorithme ?
2°) Réaliser le programme correspondant en Python sous la forme d'une fonction **jeu**() (il s'agit d'une fonction sans argument).

7 On considère l'algorithme ci-dessous dans lequel *n* est un entier naturel.

Entrée :
Saisir *n*

Traitement et sortie :
Pour *k* variant de 0 à 2*n* avec un pas de 2 **Faire**
    x prend la valeur k^2
    Afficher *x*
FinPour

Recopier cet algorithme.
1°) Que produit cet algorithme ?
2°) Réécrire cet algorithme avec une boucle « Tantque ».

8 Algorithmes de simulation de tirages successifs sans remise dans une urne

1°) Une urne contient cinq boules numérotées de 1 à 5.
On tire successivement deux boules au hasard sans remise.
Voici un algorithme de simulation de cette expérience.

Initialisations :
x prend la valeur 0
y prend la valeur 0

Traitement :
Tantque $x = y$ **Faire**
    x prend la valeur d'un entier aléatoire de 1 à 5
    y prend la valeur d'un entier aléatoire de 1 à 5
FinTantque

Sortie :
Afficher *x* et *y*

Recopier cet algorithme.
a) Expliquer le rôle des variables *x* et *y*.
b) Expliquer pourquoi on a initialisé les variables *x* et *y* à 0.
c) Programmer cet algorithme en Python sous la forme d'une fonction **jeu**().

2°) Une urne contient quatre boules numérotées de 1 à 4.
On tire successivement trois boules au hasard sans remise.
Écrire un algorithme de simulation de cette expérience.
Programmer cet algorithme en Python sous la forme d'une fonction **jeu**().

9 On considère l'algorithme suivant écrit en langage naturel qui fait intervenir les variables *i* et *n*, entiers naturels.

Entrée :
Saisir *n*

Initialisation :
i prend la valeur 0

Traitement :
Tantque $i^2 < n$ **Faire**
    i prend la valeur $i+1$
FinTantque

Sortie :
Afficher *i*

Recopier cet algorithme.
1°) Faire fonctionner cet algorithme « à la main » pour $n = 17$ en détaillant les étapes.
2°) Programmer cet algorithme en Python et l'essayer avec les valeurs de *n* suivantes :
a) $n = 10$; b) $n = 157$; c) $n = 2875$.

3°) Comment peut-on définir (par une phrase en français, pas par une formule mathématique) la valeur de la variable *i* affichée en sortie à partir de l'entier naturel *n* donné en entrée ?
4°) Reprendre les questions précédentes en remplaçant l'instruction « $i^2 < n$ » par « $i^3 < n$ ».

10 Algorithme de dichotomie

On vérifie aisément que $1 < \sqrt{3} < 2$.
On peut donc dire que le nombre $\sqrt{3}$ appartient à l'intervalle [1 ; 2].

En partageant en deux l'intervalle [1 ; 2] (le mot « dichotomie » signifie « partage en deux »), on peut déterminer si $\sqrt{3}$ appartient à l'intervalle [1 ; 1,5] ou [1,5 ; 2].
En recommençant, on peut obtenir ainsi des encadrements de plus en plus précis de $\sqrt{3}$.

On considère l'algorithme suivant écrit en langage naturel.

La variable *p* saisie en entrée désigne la précision.

Entrée :
Saisir *p* (réel strictement positif)

Initialisations :
a prend la valeur 1
b prend la valeur 2

Traitement :
Tantque $b - a > p$ **Faire**
    m prend la valeur $\frac{a+b}{2}$
    Si $m^2 < 3$
        Alors *a* prend la valeur de *m*
        Sinon *b* prend la valeur de *m*
    FinSi
FinTantque

Sortie :
Afficher *a* et *b*

1°) En faisant tourner l'algorithme « à la main », déterminer ce qu'il renvoie lorsque $p = 0,3$, puis $p = 0,1$.
On pourra utiliser la calculatrice pour effecteur les différents calculs.

2°) Programmer cet algorithme en Python sous la forme d'une fonction **dicho**(*p*) et déterminer ce qu'il renvoie si $p = 0,00001$.

11

12 On considère l'algorithme suivant en langage « intermédiaire » :

```
Saisir M
Affecter à S la valeur M
Affecter à I la valeur 2012
Tantque S < 2×M
        Affecter à S son ancienne valeur multipliée par 1,015
        Affecter à I son ancienne valeur à laquelle on ajoute 1
FinTantque
Afficher I
```

En 2012, Clara verse sur un livret d'épargne la somme de 1000 euros.
On souhaite que l'algorithme précédent représente cette situation.

1°) Réécrire l'algorithme ci-dessus en langage naturel selon les consignes habituelles de rédaction.
2°) Que faut-il saisir pour M ?
3°) À quel taux s'élèvent les intérêts sur ce compte-épargne ?
4°) Que représente le résultat affiché à la fin de l'algorithme ?
5°) Programmer l'algorithme en Python. Indiquer ce que renvoie le programme.
6°) Que renvoie le programme pour $M = 50$, $M = 2567$ et pour $M = 150300$?
Donner une interprétation de ces résultats dans le contexte bancaire.

13 Une balle est lâchée d'une hauteur H et rebondit sur le sol. À chaque rebond, elle remonte de la moitié de la hauteur du précédent rebond. Elle s'arrête lorsque la hauteur du rebond devient strictement inférieure à 1 mm.
On cherche à écrire un algorithme qui renvoie le nombre N de rebonds effectués par la balle en fonction de la hauteur H. On donne l'algorithme suivant :

Entrée :
Saisir H (réel strictement positif)

Initialisation :
N prend la valeur 0

Traitement :
Tantque …… **Faire**
    H prend la valeur …….
    N prend la valeur …….
FinTantque

Sortie :
Afficher …….

1°) Recopier et compléter cet algorithme pour qu'il réponde au problème posé.
2°) Le programmer en Python.
3°) Combien de rebonds fera la balle lorsqu'on la laisse tomber d'une hauteur égale à 50 cm ?

Le 19-11-2014

Faire écrire des algorithmes avec boucles « Tantque ».
Il me semble que ce type d'exercice manque à ma liste.

Corrigé

1 Algorithme avec une boucle « Tantque »

Objectif de cet exercice : comprendre le fonctionnement d'un algorithme avec une boucle « Tantque »

Les variables de l'algorithme sont x réel positif ou nul et n entier naturel.

Entrée :
Saisir x

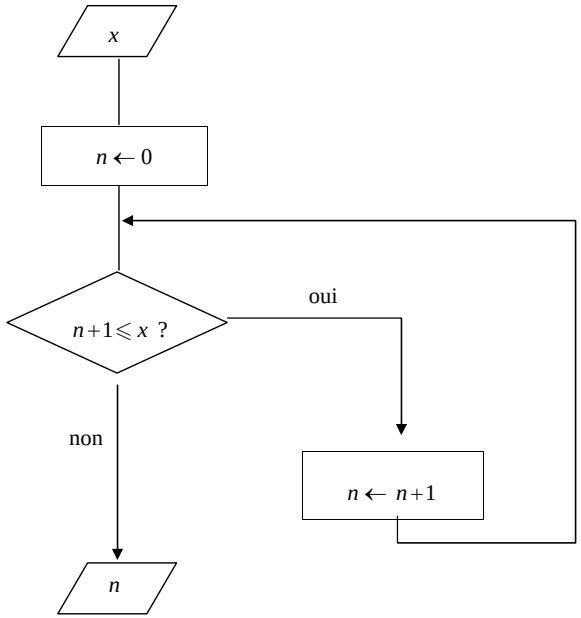
Initialisation :
n prend la valeur 0

Traitement :
Tantque $n+1 \leqslant x$ **Faire**
    n prend la valeur $n+1$
FinTantque

Sortie :
Afficher n

Organigramme :

Les entrées et sorties sont mises dans des parallélogrammes.
Le test est mis dans un losange.



Il ne s'agit pas d'un algorithme de calcul.

$x = 2$	$x = 3{,}1$	$x = 9{,}8$
n prend la valeur 0 $0+1 \leqslant 2$? oui n prend la valeur 1 $1+1 \leqslant 2$? oui n prend la valeur 2 $2+1 \leqslant 2$? non Le nombre de sortie est **2**.	n prend la valeur 0 $0+1 \leqslant 3{,}1$? oui n prend la valeur 1 $1+1 \leqslant 3{,}1$? oui n prend la valeur 2 $2+1 \leqslant 3{,}1$? oui n prend la valeur 3 $3+1 \leqslant 3{,}1$? non Le nombre de sortie est **3**.	n prend la valeur 0 $0+1 \leqslant 9{,}8$? oui n prend la valeur 1 $1+1 \leqslant 9{,}8$? oui n prend la valeur 2 $2+1 \leqslant 9{,}8$? oui n prend la valeur 3 $3+1 \leqslant 9{,}8$? oui n prend la valeur 4 $4+1 \leqslant 9{,}8$? oui n prend la valeur 5 $5+1 \leqslant 9{,}8$? oui n prend la valeur 6 $6+1 \leqslant 9{,}8$? oui n prend la valeur 7 $7+1 \leqslant 9{,}8$? oui n prend la valeur 8 $8+1 \leqslant 9{,}8$? oui n prend la valeur 9 $9+1 \leqslant 9{,}8$? non Le nombre de sortie est **9**.

À chaque fois, on continue tant que la condition est vraie.

On peut tester l'algorithme pour d'autres nombres positifs que des décimaux, par exemple π ou des fractions.

Version très claire avec présentation en tableau montrant l'évolution de la variable n :

$x = 2$

Étape	Test $x \geqslant n+1$	n
Initialisation		0
1	VRAI	1
2	VRAI	2
3	FAUX	

$x = 3{,}1$

Étape	Test $x \geqslant n+1$	n
Initialisation		0
1	VRAI	1
2	VRAI	2
3	VRAI	3
4	FAUX	

$x = 9{,}8$

Étape	Test $x \geqslant n+1$	n
Initialisation		0
1	VRAI	1
2	VRAI	2
3	VRAI	3
4	VRAI	4
5	VRAI	5
6	VRAI	6
7	VRAI	7
8	VRAI	8
9	VRAI	9
10	FAUX	

Version proposée par Valentine de Gaulle (à qui j'avais demandé en guise de punition de faire les tableaux) :

Les variables x et n apparaissent dans le tableau d'évolution.
On peut simplement dire qu'il n'est pas intéressant de montrer l'évolution de la variable x au cours du déroulement de l'algorithme puisque son contenu est constant du début à la fin.

$x = 2$

Étape	Test $x \geqslant n+1$	x	n
Initialisation		2	0
1	VRAI	2	1
2	VRAI	2	2
3	FAUX		

$x = 3{,}1$

Étape	Test $x \geqslant n+1$	x	n
Initialisation		3,1	0
1	VRAI	3,1	1
2	VRAI	3,1	2
3	VRAI	3,1	3
4	FAUX		

$x = 9{,}8$

Étape	Test $x \geqslant n+1$	x	n
Initialisation		9,8	0
1	VRAI	9,8	1
2	VRAI	9,8	2
3	VRAI	9,8	3
4	VRAI	9,8	4
5	VRAI	9,8	5
6	VRAI	9,8	6
7	VRAI	9,8	7
8	VRAI	9,8	8
9	VRAI	9,8	9
10	FAUX		

Programme en Python :

```
def exo1(x):
    n=0
    while n+1<=x:
        n=n+1
    return n
```

Programmes sur calculatrice :

Calculatrice TI

```
: Prompt X
: 0 → N
: While N+1≤X
: N+1 → N
: End
: Disp N
```

Calculatrice CASIO

```
"X="?→X↵
0→N↵
While N+1≤X ↵
N+1→N ↵
WhileEnd
N◢
```

On peut tester le programme pour les nombres donnés dans l'énoncé mais aussi pour des nombres positifs autres que des décimaux, par exemple π ou des fractions.

Le nombre de sortie est le plus grand entier naturel inférieur ou égal à x, c'est-à-dire sa **partie entière**, notée **$E(x)$**.
Pour un réel positif ou nul, la partie entière est égale à sa troncature à l'unité.

L'algorithme affiche en sortie la partie entière du nombre positif x saisi en entrée.

On peut trouver que cet algorithme ne présente pas d'intérêt et donc que le programme correspondant n'est pas intéressant.
C'est vrai, mais ce premier exercice n'a pas pour but de présenter un algorithme « intéressant ». Il permet surtout de se familiariser avec la notion de boucle « Tantque » et de montrer comment on fait tourner « à la main » un tel algorithme.

2

Entrée :
Saisir n

Initialisation :
u prend la valeur n

Traitement :
Tantque $u \geqslant 11$ **Faire**
    u prend la valeur $u-11$
FinTantque

Sortie :
Afficher u

1°) On fait fonctionner l'algorithme « à la main » pour $n=35$, $n=50$, $n=55$.

• **Pour $n=35$**

u prend la valeur 35 (valeur de départ).

On rentre dans la boucle.

$35 \geqslant 11$?	oui	u prend la valeur $35-11=24$.
$24 \geqslant 11$?	oui	u prend la valeur $24-11=13$.
$13 \geqslant 11$?	oui	u prend la valeur $13-11=2$.
$2 \geqslant 11$?	non	

On sort de la boucle.
Le nombre de sortie est **2**.

• **Pour $n=50$**

Le nombre de sortie est **6**.

• **Pour $n=55$**

Le nombre de sortie est **0**.

Présentation en tableau

→ Pour $n=35$ (en entrée) :

Étape	$u \geqslant 11$	u
Initialisation		35
1	VRAI	24
2	VRAI	13
3	VRAI	2
4	FAUX	

Le résultat affiché (dernière valeur de u avant sortie de la boucle) est 2.

→ Pour $n=50$ (en entrée) :

Étape	Test $u \geqslant 11$	u
Initialisation		50
1	VRAI	39
2	VRAI	28
3	VRAI	17
4	VRAI	6
5	FAUX	

Le résultat affiché (dernière valeur de u avant sortie de la boucle) est 6.

→ Pour $n = 55$ (en entrée) :

Étape	$u \geqslant 11$	u
Initialisation		55
1	VRAI	44
2	VRAI	33
3	VRAI	22
4	VRAI	11
5	VRAI	0
6	FAUX	

Le résultat affiché (dernière valeur de *u* avant sortie de la boucle) est 0.

2 Version en tableau de Valentine de Gaulle

→ Pour $n = 35$ (en entrée) :

Étape	$u \geqslant 11$	u	n
Initialisation		35	35
1	VRAI	24	35
2	VRAI	13	35
3	VRAI	2	35
4	FAUX		

Le résultat affiché (dernière valeur de *u* avant sortie de la boucle) est 2.

→ Pour $n = 50$ (en entrée) :

Étape	Test $u \geqslant 11$	u	n
Initialisation		50	50
1	VRAI	39	50
2	VRAI	28	50
3	VRAI	17	50
4	VRAI	6	50
5	FAUX		

Le résultat affiché (dernière valeur de *u* avant sortie de la boucle) est 6.

→ Pour $n = 55$ (en entrée) :

Étape	$u \geqslant 11$	u	n
Initialisation		55	55
1	VRAI	44	55
2	VRAI	33	55
3	VRAI	22	55
4	VRAI	11	55
5	VRAI	0	55
6	FAUX		

2°) Pour un entier naturel *n* quelconque, le nombre de sortie est égal au reste de la division euclidienne de *n* par 11 (en effet, prenant un nombre entier, on a retiré 11 autant de fois qu'on le pouvait).

On rappelle que la division euclidienne est la division qui donne un reste et un quotient entier.
Cet algorithme donne juste le reste. Il ne donne pas le quotient.

Exemple :
Pour effectuer la division euclidienne de 69 par 11, on écrit : $69 = 11 \times 6 + 3$. Comme $3 < 11$, on peut dire que le reste de la division euclidienne de 69 est 3 et le quotient est 6.

Programme en Python :

```
def exo2(n):
    u=n
    while u>=11:
          u=u-11
    return u
```

Programmes sur calculatrice

Calculatrice TI

```
: Prompt N
: N → U
: While U⩾11
: U−11 → U
: End
: Disp U
```

Calculatrice CASIO

```
"N="?→ N↵
N → U↵
While 11⩽ U ↵
U−11 → U ↵
WhileEnd↵
U◢
```

Programme sur calculatrice TI :

```
: Prompt N
: N → U
: While U ⩾ 11
: U − 11 → U
: End
: U
```

Remarque : On n'est pas obligé de mettre deux variables dans ce programme.
Cependant il est plus commode d'utiliser 2 variables plutôt qu'une pour la lisibilité du programme.

```
: Prompt N
: While N⩾11
: U−11 → N
: End
: Disp N
```


3

Entrée :
Saisir A

Initialisation :
n prend la valeur 0

Traitement et sortie :
Tantque $n^2 \leqslant$ A **Faire**
    Afficher *n*
FinTantque

Le programme correspondant ne s'arrête jamais (faire l'essai sur la calculatrice) ; on obtient une série de 0 qui ne s'arrête pas.

Commentaire d'un élève :

Tel que l'algorithme est donné, ça fait une boucle infinie.

En effet, il n'y a pas de test d'arrêt.
Ce genre de situation (cauchemar pour les informaticiens) ne se produit pas pour les boucles « Pour » puisque le nombre d'itérations est toujours fini et fixé à l'avance.
On est obligé d'éteindre la calculatrice.

Correction dans l'algorithme : il faut ajouter ***n* prend la valeur $n+1$**.

Entrée :
Saisir A

Initialisation :
n prend la valeur 0

Traitement et sortie :
Tantque $n^2 \leqslant$ A **Faire**
    Afficher *n*
    ***n* prend la valeur $n+1$**
FinTantque

On observera que la sortie se fait au fur et à mesure du traitement.

Programme Python :

```
def exo3(A):
    n=0
    while n**2<=A:
          print (n)
          n=n+1
```

Programmes sur calculatrice :

Calculatrice TI

```
: Prompt A
: 0 → N
: While N² ⩽ A
: Disp N
: N+1 → N
: End
```

Calculatrice CASIO

```
"A="?→A↵
0→N↵
While N² ⩽ A ↵
N◢
N+1→N ↵
WhileEnd↵
```

On peut tester ce programme pour $n = 83$.
On obtient tous les entiers naturels de 0 à 9.

On peut tester pour des valeurs de n plus grandes.
On peut mettre des « Pause » pour permettre que les valeurs s'affichent plus lentement.

On peut aussi utiliser des boucles dans le vide.

Quand on ajoute une Pause dans un programme, nous devons appuyer sur entrer pour « dérouler » le programme.

4 Il s'agit d'une expérience aléatoire de référence.

```
from random import randint

def jeu():
    x=0
    c=0
    while x!=6:
          x=randint(1,6)
          c=c+1
    return c
```



```
from random import randint

def lancer():
    x=randint(1, 6)
    return x

def jeu():
    x=1
    n=0
    while x!=6 :
          x=lancer()
          n=n+1
    return n
```

5 **Algorithme concernant un placement d'argent**

1°) Le coefficient multiplicateur associé à une hausse de 3 % est égal à $1+\frac{3}{100}=1{,}03$.

Chaque année (sauf la première), le capital est égal au capital de l'année précédente multiplié par 1,03.

$$x+\frac{3}{100}\times x = 1{,}03x$$

2°) Il s'agit d'un algorithme permettant de déterminer une valeur seuil.

Algorithme pour savoir au bout de combien d'années on aura au moins 10 euros

Initialisations :
S prend la valeur 1
n prend la valeur 0

Traitement :
Tantque $S<10$ **Faire**
    S prend la valeur $S\times 1{,}03$
    n prend la valeur $n+1$
FinTantque

Sortie :
Afficher n

On peut éventuellement déclarer les variables S et n au début (S réel, n entier naturel).

Programme sur calculatrice

Calculatrice TI

```
: 1 → S
: 0 → N
: While S<10
: S×1,03 → S
: N+1 → N
: End
: Disp N
```

Calculatrice CASIO

```
1 → S ↵
0 → N↵
While S<10 ↵
S×1,03→ S↵
N+1 → N ↵
WhileEnd↵
N◢
```

En faisant tourner le programme, on obtient $n = 78$.
En plaçant 1 € à un taux d'intérêt de 3 %, la somme dépassera 10 € au bout de 78 ans.

La somme dépassera 100 € au bout de 156 ans.

Ce problème se modélise aisément grâce aux suites géométriques.

On verra en Terminale comment résoudre le problème sans avoir recours à un algorithme (en utilisant la fonction « logarithme népérien ») : cela permettrait de savoir par exemple au bout de combien d'années on sera millionnaire, milliardaire… ce qui serait trop long si on le faisait avec un programme.
Pour 1 000 €, 234 ans ; pour 1000 000 €, 468 ans ; pour 1 000 000 000 €, 702 ans

$u_{n+1} = 1,03u_n$

$u_n = (1,03)^n \times 1$

$u_n = (1,03)^n$

On cherche le plus petit entier naturel n tel que $(1,03)^n > 10$.

$n > \dfrac{\ln 10}{\ln 1,03}$

$n > \dfrac{3\ln 10}{\ln 1,03}$

$n > \dfrac{6\ln 10}{\ln 1,03}$

6 Algorithme de simulation de lancers de deux dés

1°)

a) Le rôle de la boucle est précisé dans l'énoncé.
On ne peut pas connaître à l'avance le nombre d'itérations.
Il s'agit d'un algorithme de simulation d'un temps d'attente (temps d'attente d'un double).

Cet algorithme comporte une petite difficulté : la condition d'arrêt est-elle vérifiée ?

b) On a initialisé la variable x à 1 et y à 2 pour pouvoir lancer la première boucle.
Si on donnait à x et y la même valeur, il n'y aurait pas de boucle.

2°) **Programme en Python (version fonction)**

```
from random import randint

def jeu():
    x=1
    y=2
    n=0
    while x!=y :
          x=randint(1, 6)
          y=randint(1, 6)
          n=n+1
    return n
```

On peut séparer en deux fonctions, ce qui permet une meilleure clarté.

```
from random import randint

def lancer():
    x=randint(1, 6)
    y=randint(1, 6)
    return x, y

def jeu():
    x=1
    y=2
    n=0
    while x!=y :
          x,y=lancer()
          n=n+1
    return n
```

Programme sur calculatrice

Calculatrice TI

```
: 1 → X
: 2 → Y
: 0 → N
: While X ≠ Y
: partEnt(NbrAléat * 6) + 1 → X
: partEnt(NbrAléat * 6) + 1 → Y
: N + 1 → N
: End
: Disp N
```

Calculatrice CASIO

```
1 → X ↵
2 → Y ↵
0 → N ↵
While X ≠ Y ↵
partEnt(NbrAléat * 6) + 1 → X ↵
partEnt(NbrAléat * 6) + 1 → Y ↵
WhileEnd ↵
N◢
```

Sur calculatrice TI :

NbrAléat : appuyer sur la touche [math] puis sélectionner PRB.

Pour générer des entiers aléatoires de 1 à 6, on peut aussi utiliser la commande :
randInt(1, 6) → X ou entAléat(1, 6) → X (suivant que la calculatrice est en anglais ou en français).

Cet algorithme permet de simuler l'expérience aléatoire consistant à lancer deux dés et à noter le nombre de lancers à effectuer jusqu'à obtenir un double.

En notant N le nombre de lancers à effectuer avant d'obtenir un double, cet algorithme permet de calculer une valeur approchée de l'espérance de N.

On peut démontrer dans le supérieur que la valeur exacte de cette espérance est égale à $\frac{1}{5}$ (en effet N suit la loi géométrique de paramètre $p = \frac{1}{6}$, donc son espérance est égale à $\frac{p}{q} = \frac{1}{5}$ où $q = 1 - p$).

7 Le but de cet exercice est de réécrire un algorithme avec une boucle « Pour » sous la forme d'un algorithme avec une boucle « Tantque ».

Objectif : comprendre le lien entre les boucles « Tantque » et les boucles « Pour »

1°) Cet algorithme permet d'afficher tous les carrés des entiers naturels pairs inférieurs ou égaux à un entier naturel n donné.

2°) k varie de 0 à $2n$, avec un pas de 2 c'est-à-dire que k va prendre les valeurs 0, 2, 4, … jusqu'à $2n$.

Entrée :
Saisir n

Initialisation :
k prend la valeur 0

Traitement et sortie :
Tantque $k \leqslant 2n$ **Faire**
  x prend la valeur k^2
  Afficher x
  k prend la valeur $k + 2$
FinTantque

On peut faire tourner ce programme pour $n = 5$.
On obtient : 0 ; 4 ; 16 ; 36 ; 64 ; 100 (carrés parfaits pairs de 0 à 100).

Pour aller plus loin…
Réaliser un algorithme qui permet d'afficher tous les carrés parfaits non nuls inférieurs ou égaux à un entier n donné.

8 Algorithmes de simulation de tirages sans remise dans une urne

1°) Avec cet algorithme, on simule le tirage successif sans remise de deux boules dans une urne qui contient 4 boules.

a)
x : le numéro de la première boule tirée
y : le numéro de la deuxième boule tirée

x est un entier aléatoire compris entre 1 et 5 au sens large ($1 \leqslant x \leqslant 5$).
y est un entier aléatoire compris entre 1 et 5 au sens large ($1 \leqslant y \leqslant 5$).

Comme le tirage est effectué sans remise, on doit avoir la condition $x \neq y$.
C'est cela qui va être difficile à traduire dans l'algorithme.

On va utiliser une boucle « Tantque ».

À la fin de l'algorithme, on affiche le couple $(x ; y)$.

Cet algorithme comporte une petite difficulté : la condition d'arrêt est elle vérifiée ?

b) On a initialisé les variables x et y à 0 pour pouvoir lancer la première boucle.
On peut aussi dire que les variables x et y sont initialisées à 0 pour faire « partir » l'algorithme.

c)

```
from random import randint

def jeu():
    x=0
    y=0
    while x==y :
    x=randint(1, 5)
    y=randint(1, 5)
    return x, y
```

Programme sur calculatrice

Calculatrice TI

```
: 0 → X
: 0 → Y
: While  X = Y
: partEnt(NbrAléat * 5) + 1 → X
: partEnt(NbrAléat * 5) + 1 → Y
: End
: Disp X, Y
```

Calculatrice CASIO

```
0 → X↵
0 → Y↵
While X = Y ↵
: partEnt(NbrAléat * 5) + 1 → X↵
: partEnt(NbrAléat * 5) + 1 → Y ↵
WhileEnd↵
X◢
Y◢
```

Pour générer des entiers aléatoires de 1 à 5, on peut aussi utiliser la commande :
randInt(1, 5) → X ou entAléat(1, 5) → X (suivant que la calculatrice est en anglais ou en français).

2°) On utilise trois variables *x*, *y*, *z*.
On met en œuvre une boucle « Tantque ».
La condition va être traduite en utilisant le connecteur « ou ».

Initialisations :
x prend la valeur 0
y prend la valeur 0
z prend la valeur 0

Traitement :
Tantque $(x = y)$ ou $(y = z)$ ou $(z = x)$ **Faire**
  x prend la valeur d'un entier aléatoire de 1 à 4
  y prend la valeur d'un entier aléatoire de 1 à 4
  z prend la valeur d'un entier aléatoire de 1 à 4
FinTantque

Sortie :
Afficher *x*, *y*, *z*

À la fin de l'algorithme, on affiche le triplet (*x* ; *y* ; *z*).
Pour obtenir le « ou » sur calculatrice TI, taper [2nde] [math] (tests) puis sélectionner LOGIQUE.

9 Algorithme de « calcul »

1°) a) Faisons fonctionner l'algorithme pour $n = 17$ en entrée.

Au départ, *i* prend la valeur 0.
$0^2 < 17$? oui donc *i* prend la valeur 1.
$1^2 < 17$? oui donc *i* prend la valeur $1 + 1 = 2$.
$2^2 < 17$? oui donc *i* prend la valeur $2 + 1 = 3$.
$3^2 < 17$? oui donc *i* prend la valeur $3 + 1 = 4$.
$4^2 < 17$? oui donc *i* prend la valeur $4 + 1 = 5$.
$5^2 < 17$? non

Pour $n = 17$, le nombre affiché en sortie est **5**.

Étape	1	2	3	4	5	6
i	0	1	2	3	4	5
Test $i^2 < 17$	V	V	V	V	V	F

dernière valeur de *i*

b)

Pour $n = 10$, le nombre affiché en sortie est **4**.
Pour $n = 157$, le nombre affiché en sortie est **13**.
Pour $n = 2875$, le nombre affiché en sortie est **54**.
On peut vérifier les résultats en programmant l'algorithme sur la calculatrice.

c) Pour *n* entier naturel quelconque, le nombre affiché en sortie est le plus petit entier naturel dont le carré est supérieur ou égal à *n*.

Pour un entier naturel *n* fourni en entrée, cet algorithme renvoie le plus petit entier supérieur ou égal à la racine carrée de *n*.

(Pour $n = 16$, c'était exactement égal).

Remarques :

1. Cet algorithme correspond à une fonction.

2. On pourrait exprimer la valeur en sortie à l'aide de notations ce qui permet d'écrire la valeur à l'aide d'une formule. On utilise les notations $\lfloor x \rfloor$ (partie entière par défaut) et $\lceil x \rceil$ (partie entière par excès) pour un réel x.
$\lfloor x \rfloor$: plus grand entier relatif inférieur ou égal à x
$\lceil x \rceil$: plus petit entier relatif supérieur ou égal à x

2°) $i^3 < n$

Pour $n = 17$, le nombre affiché en sortie est 3.
Pour $n = 10$, le nombre affiché en sortie est 3.
Pour $n = 157$, le nombre affiché en sortie est 6.
Pour $n = 2875$, le nombre affiché en sortie est 15.

On vérifie les résultats en réalisant le programme correspondant sur calculatrice

Pour un entier naturel n fourni en entrée, l'algorithme renvoie le plus petit entier supérieur ou égal à la racine cubique de n.

10 Algorithme de dichotomie

Le but est de ne pas utiliser la touche racine carrée de la calculatrice.
C'est pour cela que la condition « $m^2 < 3$ » n'est pas transformée en « $m < \sqrt{3}$ ».

L'algorithme étudié ici est très ancien.

Le principe de cet algorithme qui est de se limiter à ces méthodes primitives : opérations simples telles qu'additions, soustractions, multiplications, divisions.

1°)

- **$p = 0,3$**

Initialement a prend la valeur 1 et b prend la valeur 2.

$b-a$	$b-a>0,3$?	$m=\frac{a+b}{2}$	$m^2<3$?	a	b
				1	2
1	oui	1,5	oui	1,5	2
0,5	oui	1,75	non	1,5	1,75
0,25	non				

Les valeurs de a et b affichées en sortie sont respectivement 1,5 et 1,75.

- **$p = 0,1$**

$b-a$	$b-a>0,1$?	$m=\frac{a+b}{2}$	$m^2<3$?	a	b
				1	2
1	oui	1,5	oui	1,5	2
0,5	oui	1,75	non	1,5	1,75
0,25	oui	1,625	oui	1,625	1,75
0,125	oui	1,6875	oui	1,6875	1,75
0,0625	non				

Les valeurs de a et b affichées en sortie sont respectivement 1,6875 et 1,75.

2°) On programme cet algorithme sur calculatrice ou sur ordinateur.

Programme Python :

```
def dicho(p):
    a=1
    b=2
    while b-a>p:
          m=(a+b)/2
          if m**2<3:
             a=m
          else: b=m
    return a,b
```

Si $p = 0,00001$, on obtient :
$a = 1,732048035$ (le signe = employé ici n'est pas ce qu'il y a de plus approprié)
$b = 1,732055664$

11 Calcul d'un terme d'indice donné pour une suite définie par récurrence

On déroule l'algorithme « à la main ».

condition		Variable *n*	Variable *u*
initialement		*n* prend la valeur 0	*u* prend la valeur 0
$0 < 3$?	oui	*n* prend la valeur $0+1=1$	*u* prend la valeur $\frac{0+1}{2}=\frac{1}{2}$
$1 < 3$?	oui	*n* prend la valeur $1+1=2$	*u* prend la valeur $\frac{\frac{1}{2}+1}{2}=\frac{3}{4}$
$2 < 3$?	oui	*n* prend la valeur $2+1=3$	*u* prend la valeur $\frac{\frac{3}{4}+1}{2}=\frac{7}{8}$
$3 < 3$?	non		

L'affichage obtenu à la fin de l'algorithme est : 3 pour *n* et $\frac{7}{8}$ pour *u*.

Lien avec les suites :

On considère la suite (u_n) définie sur $\mathbb{N}$ par son premier terme $u_0 = 0$ et la relation de récurrence $u_{n+1} = \frac{u_n + 1}{2}$.

Cet algorithme permet de calculer le terme u_p de cette suite pour un entier naturel *p* donné.

Partie programmation :

Calculatrice TI

```
: Prompt P
: 0 → N
: 1 → U
: While N < P
: N + 1 → N
: (U + 1) / 2 → U
: End
: Disp "N = ", N, "U = ", U
```

Calculatrice CASIO

```
"P = " ?→ P ↵
0 → N ↵
0 → U ↵
While N < P ↵
N + 1 → N ↵
(U + 1) / 2 → U
WhileEnd↵
N◢
U◢
```

Cet exercice est un exercice-type à connaître.

12 Algorithme de seuil

1°) Réécriture de l'algorithme en langage naturel.

Entrée :
Saisir M

Initialisations :
S prend la valeur M
I prend la valeur 2012

Traitement :
Tantque $S < 2 \times M$ **Faire**
    S prend la valeur $S \times 1,015$
    I prend la valeur $I + 1$
FinTantque

Sortie :
Afficher I

2°) Pour M, on doit saisir en entrée la valeur 1000.
3°) Le taux d'intérêt est de 1,5 % par an.
4°) Le résultat affiché à la fin de l'algorithme représente l'année en laquelle le capital aura doublé.
5°) Le programme renvoie 2059.

6°)

Valeur de M en entrée	Valeur de I en sortie
50	2059
2067	2059
150300	2059

Pour chaque valeur de M, le programme renvoie la valeur 2059 (valeur de la variable I en sortie).
Le nombre d'années I au bout duquel le capital a doublé (valeur de I en sortie) ne dépend pas de la somme de départ.
On trouve toujours la valeur 2059 (valeur de I en sortie).

13 Algorithme à compléter

1°)

Entrée :
Saisir H (H en mm)

Initialisation :
N prend la valeur 0

Traitement :
Tantque $H \geqslant 1$ **Faire**
    H prend la valeur $\frac{H}{2}$
    N prend la valeur $N+1$
FinTantque

Sortie :
Afficher N

2°) Programmation

3°) La balle s'arrête au bout de 9 rebonds.

Il ne faut pas oublier que $50 \text{ cm} = 500 \text{ mm}$.

N.B. : On pourrait modéliser le problème à l'aide d'une suite géométrique.