

Algorithmique et programmation

Cours de maths - 3ème

Un **algorithme**, c'est une recette : une suite d'instructions précises qu'un ordinateur (ou un robot, ou toi) peut exécuter sans réfléchir. Un **programme**, c'est cette recette écrite dans un langage que la machine comprend. En 3ème, tu vas écrire des programmes avec plusieurs variables, des tests imbriqués, des boucles, des fonctions, et les utiliser pour calculer un PGCD, repérer un nombre premier, simuler le hasard ou dessiner avec un lutin.

Comment lire les programmes de ce chapitre

Les programmes sont écrits comme des blocs de type Scratch, mais en texte clair. Les retraits (décalages vers la droite) montrent ce qui est « à l'intérieur » d'un bloc. Mêmes idées si tu préfères un langage écrit comme Python : seule l'écriture change. Le mot « modulo » donne le reste d'une division euclidienne : $17 \text{ modulo } 5 = 2$, car $17 = 3 \times 5 + 2$.

1. Variables et affectation

Variable

Une variable est une « boîte » portant un nom, qui contient une valeur (un nombre, un texte). **Affecter** une valeur, c'est ranger cette valeur dans la boîte : l'ancienne valeur est effacée.

Attention au sens de l'instruction : « mettre [A] à (A + 1) » ne veut pas dire que A est égal à A + 1 (ce serait impossible !). Cela signifie : on calcule A + 1 avec la valeur actuelle de A, puis on range le résultat dans A.

Exemple : suivre les variables

```
mettre [A] à (6)
mettre [B] à (A + 4)
mettre [A] à (B × 3)
mettre [B] à (A - B)
```

On suit les valeurs ligne par ligne dans un **tableau de suivi** :

Après la ligne	1	2	3	4
A	6	6	30	30
B	—	10	10	20

À la ligne 3, A prend la valeur $10 \times 3 = 30$. À la ligne 4, B prend $30 - 10 = 20$. À la fin : A = 30 et B = 20.

Piège classique

Une variable calculée à partir d'une autre ne « se met pas à jour toute seule » : à la ligne 2 ci-dessus, B a été calculé avec A = 6. Quand A change ensuite, B garde sa valeur tant qu'on ne la recalcule pas.

Méthode : exécuter un programme à la main

1. Fais un tableau avec une ligne par variable.
2. Lis les instructions dans l'ordre, une par une.
3. À chaque affectation, calcule avec les valeurs actuelles et écris la nouvelle valeur.
4. Relis la dernière colonne : c'est le résultat.

2. Tests et conditions imbriquées

Un **test** permet au programme de choisir : une instruction n'est exécutée que si une condition est vraie. La condition est une comparaison (=, ≠, <, ≤, >, ≥) qui donne « vrai » ou « faux ». On peut aussi combiner deux conditions avec **et** (les deux doivent être vraies) ou **ou** (au moins une doit être vraie).

Exemple : tarif d'entrée d'une piscine

```
demander [Quel âge as-tu ?] et attendre
mettre [âge] à (réponse)
si (âge < 6) alors
    dire [gratuit]
sinon
    si (âge < 18) alors
        dire [3 euros]
    sinon
        dire [5 euros]
    fin si
```

fin si

Le second « si » est **imbriqué** : il n'est examiné que si le premier test est faux. Pour 4 ans : « gratuit ». Pour 6 ans : le premier test est faux, le second aussi ($6 < 18$ est vrai ! donc « 3 euros »). Pour 30 ans : les deux tests sont faux, « 5 euros ».

L'ordre des tests compte

Quand plusieurs tests se suivent, le programme s'arrête au premier test vrai. Si tu testais « $\text{âge} < 18$ » avant « $\text{âge} < 6$ », un enfant de 4 ans paierait 3 euros : le test « $\text{âge} < 6$ » ne serait jamais atteint. Range donc tes tests du plus précis au plus général.

Condition avec « et » / « ou »

« $(n > 10) \text{ et } (n < 20)$ » est vrai pour 15 mais faux pour 25. « $(n < 0) \text{ ou } (n > 100)$ » est vrai pour -3 et pour 120, faux pour 50.

3. La boucle « pour » : répéter un nombre connu de fois

Quand on sait à l'avance combien de fois répéter, on utilise la boucle « répéter n fois », ou la boucle « pour i de a à b » qui possède en plus un **compteur** i qui change à chaque tour.

Exemple : somme des entiers de 1 à 5

```
mettre [S] à (0)
pour [i] de (1) à (5)
    ajouter (i) à [S]
dire (S)
```

Tour	i	S après le tour
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

Le programme dit 15. Le dernier tour est celui où $i = 5$ (la valeur finale est comprise).

Avant la boucle, on initialise

Une variable qui sert de total ou de compteur doit recevoir sa valeur de départ (souvent 0) *avant* la boucle, jamais à l'intérieur : sinon elle serait remise à zéro à chaque tour.

4. La boucle « tant que » : répéter jusqu'à ce que ça change

Quand on ne sait pas à l'avance combien de tours seront nécessaires, on utilise « tant que (condition) ». Le programme teste la condition **avant chaque tour** : si elle est vraie, il exécute le bloc puis revient tester ; dès qu'elle est fausse, il passe à la suite.

Exemple : combien de fois peut-on retirer 7 ?

```
mettre [R] à (50)
mettre [k] à (0)
tant que (R ≥ 7)
    mettre [R] à (R - 7)
    ajouter (1) à [k]
dire (k)
dire (R)
```

Après le tour	0	1	2	3	4	5	6	7
R	50	43	36	29	22	15	8	1
k	0	1	2	3	4	5	6	7

Quand $R = 1$, la condition « $R \geq 7$ » est fausse : on sort de la boucle. Le programme dit 7 puis 1 : on retrouve la division euclidienne $50 = 7 \times 7 + 1$ (quotient $k = 7$, reste $R = 1$).

Boucle infinie

Si rien dans la boucle ne peut rendre la condition fausse, le programme tourne sans fin. Vérifie toujours que le bloc fait évoluer une variable de la condition dans le bon sens (ici R diminue, il finira par passer sous 7).

« Pour » ou « tant que » ?

Nombre de tours connu : boucle « pour ». Nombre de tours inconnu, qui dépend d'un résultat (un seuil à atteindre, un reste nul...) : boucle « tant que ». Une boucle « pour » peut toujours se transformer en « tant que » avec un compteur, mais pas l'inverse.

5. Fonctions et procédures avec paramètres

Fonction, procédure, paramètre

Une **procédure** (ou « bloc personnalisé ») est un petit programme auquel on donne un nom, pour le réutiliser sans le recopier. Ses **paramètres** sont des variables qu'on fournit au moment de l'appel. Une **fonction** est une procédure qui **renvoie** un résultat.

Exemple : une fonction qui calcule le prix d'un abonnement

```
définir prix (n)
  si (n ≥ 5) alors
    renvoyer (n × 8 – 6)
  sinon
    renvoyer (n × 8)
fin de la définition
```

Appeler prix(3) donne 24. Appeler prix(7) donne $7 \times 8 - 6 = 50$. Le paramètre n joue le rôle d'une lettre dans une formule : on ne l'écrit qu'une fois, puis on l'utilise avec n'importe quelle valeur.

Pourquoi des fonctions ?

Elles évitent de recopier dix fois les mêmes blocs, rendent le programme lisible (le nom dit ce qu'elle fait) et permettent de corriger une erreur à un seul endroit.

Variable locale

Les paramètres et les variables créés dans une fonction n'existent que pendant son exécution : on ne peut pas les relire ensuite depuis le reste du programme. Pour récupérer un résultat, il faut le renvoyer.

6. Dessiner avec un lutin

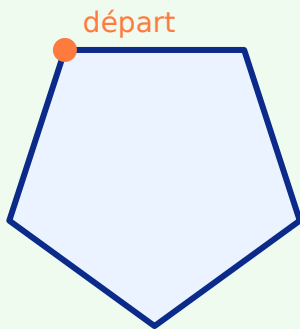
Le lutin se déplace sur l'écran en laissant une trace quand le stylo est baissé. Deux instructions suffisent : « avancer de (c) pas » et « tourner de (a) degrés ». Le lutin tourne sur place ; l'angle donné est l'angle dont il **pivote**, pas l'angle intérieur de la figure.

Exemple : un polygone régulier

```
définir polygone (n) (c)
```

répéter (n) fois
avancer de (c) pas
tourner α de $(360 \div n)$ degrés
fin de la définition

Pour `polygone(5, 40)` : cinq côtés de 40 pas, et le lutin tourne de $360 \div 5 = 72^\circ$ à chaque sommet.

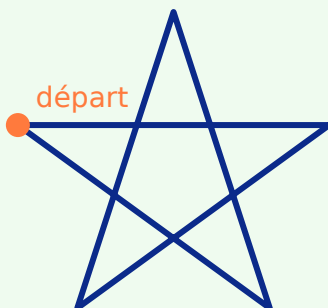


Un tour complet vaut 360°

Pour revenir au point de départ en face de la même direction, la somme des rotations doit être un multiple de 360° . Pour un polygone régulier à n côtés, l'angle de rotation est donc $360 \div n$: 120° pour un triangle équilatéral, 90° pour un carré, 60° pour un hexagone, 45° pour un octogone.

Une étoile

Avec « répéter 5 fois : avancer de 70 pas, tourner de 144° », la somme des rotations vaut $5 \times 144 = 720^\circ$, soit deux tours complets : le lutin dessine une étoile à cinq branches et revient au départ.



Une spirale

Si la longueur du côté *augmente à chaque tour* (avancer de $10 \times i$ pas, tourner de 90°), le lutin ne referme jamais la figure : il dessine une spirale carrée. C'est une boucle «

pour » dont le compteur sert de longueur.

7. Programmes de calcul : PGCD et nombres premiers

Le **PGCD** de deux entiers est leur plus grand diviseur commun. Il sert à simplifier une fraction : **63/105** devient **3/5** en divisant par 21, qui est le PGCD de 63 et de 105. Deux algorithmes classiques le calculent.

Algorithme des soustractions successives

On remplace toujours le plus grand nombre par la différence des deux, jusqu'à obtenir deux nombres égaux : cette valeur est le PGCD (car un diviseur commun de a et b divise aussi $a - b$).

```
tant que (a ≠ b)
  si (a > b) alors
    mettre [a] à (a - b)
  sinon
    mettre [b] à (b - a)
dire (a)
```

Pour $a = 84$ et $b = 36$:

a	84	48	12	12	12
b	36	36	36	24	12

Le PGCD de 84 et 36 est 12.

Algorithme d'Euclide

On remplace (a, b) par $(b, \text{reste de } a \div b)$, jusqu'à obtenir un reste nul : le PGCD est le dernier reste non nul. C'est bien plus rapide quand les nombres sont grands.

```
tant que (b ≠ 0)
  mettre [r] à (a modulo b)
  mettre [a] à (b)
  mettre [b] à (r)
dire (a)
```

Pour $a = 90$ et $b = 36$: $90 \text{ modulo } 36 = 18$, puis $(36, 18)$; $36 \text{ modulo } 18 = 0$, puis $(18, 0)$. Le programme dit 18 : $\text{PGCD}(90, 36) = 18$.

Nombre premier

Un nombre premier est un entier supérieur ou égal à 2 qui n'a que deux diviseurs : 1 et lui-même. Par exemple 2, 3, 5, 7, 11, 13 sont premiers ; $9 = 3 \times 3$ ne l'est pas ; 1 ne l'est pas non plus.

Tester si un nombre est premier

On cherche un diviseur d à partir de 2. Inutile d'aller au-delà de la racine carrée de n : si $n = d \times e$ avec $d \leq e$, alors $d \times d \leq n$. On s'arrête donc dès que $d \times d$ dépasse n .

```
définir estPremier (n)
  mettre [d] à (2)
  tant que ((d × d) ≤ n) et ((n modulo d) ≠ 0)
    ajouter (1) à [d]
  si ((d × d) > n) alors
    dire [premier]
  sinon
    dire [pas premier]
fin de la définition
```

Pour $n = 57$: $d = 2$ (reste 1), $d = 3$ (reste 0) : on s'arrête, $57 = 3 \times 19$, pas premier.
Pour $n = 31$: les restes pour $d = 2, 3, 4, 5$ sont non nuls, puis $d = 6$ donne $36 > 31$: premier.

Le cas $n = 1$

Ce programme répond « premier » pour 1, alors que 1 n'est pas premier. Un bon programmeur teste les cas limites : il faut ajouter « si $n = 1$ » au début.

8. Simuler le hasard

L'instruction « nombre aléatoire entre 1 et 6 » renvoie un entier au hasard, tous les choix étant équiprobables. En répétant l'expérience un grand nombre de fois avec une boucle, on obtient une **simulation** et on mesure la **fréquence** d'un événement : (nombre de fois où il arrive) $\div$ (nombre d'essais).

Exemple : sortir un 5 avec un dé

```
mettre [compte] à (0)
répéter (1000) fois
```

```
mettre [dé] à (nombre aléatoire entre (1) et (6))
si (dé = 5) alors
    ajouter (1) à [compte]
dire (compte ÷ 1000)
```

La probabilité théorique est $\frac{1}{6} \approx 0,167$. Le programme affichera une valeur proche, par exemple 0,171, mais rarement exactement 0,167.

Loi des grands nombres (version intuitive)

Plus on répète l'expérience, plus la fréquence observée a tendance à se rapprocher de la probabilité. Avec 10 lancers, les écarts sont importants ; avec 10 000, ils deviennent petits. Une simulation donne une *estimation*, pas une certitude.

9. Bien écrire et tester un programme

- **Donne des noms clairs** aux variables et aux fonctions (« compte », « prix », pas « x1 »).
- **Teste avec des petites valeurs** dont tu connais déjà le résultat, puis avec des cas limites (0, 1, la valeur exacte du seuil).
- **Découpe** un gros problème en fonctions simples.
- **Repère les erreurs fréquentes** : oubli d'initialiser une variable, parenthèses manquantes ($a + b + c \div 3$ n'est pas la moyenne !), mauvais sens d'une comparaison, boucle qui ne se termine jamais.

Le conseil de Mathbot

« Quand un programme ne fait pas ce que tu veux, ne le relis pas en entier : prends un petit exemple et fais ton tableau de suivi. L'erreur apparaît toujours à la ligne où tes valeurs diffèrent de ce que tu attendais ! »

À retenir

- Une **variable** contient une valeur ; « mettre [A] à (A + 1) » calcule avec l'ancienne valeur puis la remplace.
- On suit un programme avec un **tableau de suivi** des variables, ligne par ligne.
- Les **tests** « si ... sinon » peuvent être imbriqués ; le premier test vrai est le seul exécuté, donc l'ordre compte.
- « Pour » : nombre de tours connu. « Tant que » : on continue tant que la condition est vraie, et il faut qu'elle puisse devenir fausse.
- Une **fonction** a des paramètres et renvoie un résultat ; une procédure effectue des actions.

- Polygone régulier à n côtés : le lutin tourne de $360 \div n$ degrés.
- **PGCD** : soustractions successives jusqu'à l'égalité, ou algorithme d'Euclide (dernier reste non nul).
- Nombre premier : on teste les diviseurs d tant que $d \times d \leq n$.
- Une **simulation** donne une fréquence proche de la probabilité si le nombre d'essais est grand.