

Algorithmique et programmation

COURS • M22 • TERMINALE

Spécialité de terminale, programme officiel. Cours original rédigé avec une IA. Le cours de ton enseignant fait foi.

Sommaire du cours

Ce que tu dois savoir faire	1
Cours	1
Exemples résolus	7
Méthodes et pièges	9
Vers le PASS	9
Exercices	9
Corrigés	11

Ce que tu dois savoir faire

- Utiliser des variables de type entier, flottant, booléen et chaîne de caractères, et prévoir le résultat d'une suite d'affectations.
- Écrire une instruction conditionnelle avec if, elif, else et des conditions combinées par and, or, not.
- Choisir entre une boucle bornée (for) et une boucle non bornée (while), et écrire une accumulation.
- Définir une fonction en Python avec des paramètres et une valeur renvoyée, et l'appeler.
- Créer, parcourir et modifier une liste, et construire une liste en compréhension.
- Programmer un algorithme de seuil pour une suite ou une grandeur qui évolue.
- Programmer la méthode de dichotomie et prévoir le nombre d'étapes nécessaires pour une précision donnée.
- Programmer la méthode d'Euler pour approcher la solution d'une équation différentielle.
- Simuler une expérience aléatoire et estimer une probabilité par une fréquence.
- Lire un algorithme, dresser un tableau d'état des variables, repérer et corriger une erreur.

Cours

Dans ce chapitre, les programmes sont écrits en Python. Chaque ligne de code est présentée en retrait. En Python, l'**indentation fait partie de la syntaxe** : c'est elle, et non des accolades, qui indique quelles instructions appartiennent à une boucle, à une condition ou à une fonction. On utilise quatre espaces par niveau.

1. Variables, types et affectation

Une **variable** est un nom associé à une valeur stockée en mémoire. L'**affectation** se note avec le signe = : l'instruction `a = 5` range la valeur 5 dans la variable a. Ce n'est pas une égalité mathématique : l'instruction `a = a + 1` signifie « calculer `a + 1` avec l'ancienne valeur de a, puis ranger le résultat dans a ».

Les **types** usuels sont :

Type	Exemple	Usage
int (entier)	7, -3, 2026	comptage, indices
float (flottant)	3.5, 0.1, 2e-3	mesures, calculs approchés
bool (booléen)	True, False	résultat d'un test
str (chaîne)	"PASS", 'bonjour'	texte
list (liste)	[4, 7, 1]	collection ordonnée de valeurs

Le séparateur décimal est le **point** : 3.5 et non 3,5.

Opérations sur les nombres : +, -, * (produit), / (quotient décimal, résultat toujours flottant), // (quotient de la division euclidienne), % (reste de la division euclidienne), (puissance). Ainsi 17 // 5 vaut 3, 17 % 5 vaut 2, 7 / 2 vaut 3.5 et 2⁵ vaut 32. Le test « n est pair » s'écrit `n % 2 == 0`.

Les **flottants** sont des valeurs approchées : 0.1 + 0.2 ne vaut pas exactement 0.3 en machine. C'est pourquoi on évite de tester l'égalité de deux flottants, et pourquoi on compare plutôt à une tolérance.

Entrées et sorties : `print(x)` affiche la valeur de x ; `input()` lit une saisie au clavier **sous forme de chaîne**. Pour obtenir un nombre, on convertit : `n = int(input("n = "))` ou `x = float(input())`.

Une affectation multiple est possible : `a, b = b, a` échange les valeurs de a et b en une seule instruction.

2. Instructions conditionnelles et booléens

Les **opérateurs de comparaison** renvoient un booléen : `==` (égal), `!=` (différent), `<`, `<=`, `>`, `>=`. Attention à ne pas confondre `=` (affectation) et `==` (test d'égalité).

Les conditions se combinent avec `and` (les deux vraies), `or` (au moins une vraie) et `not` (négation).

La structure conditionnelle s'écrit :

```
if x > 0:
    signe = 1
elif x == 0:
    signe = 0
else:
    signe = -1
```

Les deux-points en fin de ligne sont obligatoires, et les instructions de chaque bloc sont indentées. Seul le premier bloc dont la condition est vraie est exécuté ; le bloc `else` s'exécute si aucune condition n'est vraie.

3. Boucles bornées et non bornées

Boucle bornée (for). On l'utilise quand le nombre de répétitions est connu à l'avance. La fonction `range` engendre une suite d'entiers :

- `range(n)` donne 0, 1, ..., n - 1 (n valeurs) ;

- `range(a, b)` donne `a, a + 1, ..., b - 1` (`b` exclu) ;
- `range(a, b, p)` avance de `p` en `p`.

Exemple : calcul de la somme $1 + 2 + \dots + 100$ par **accumulation**.

```
s = 0
for k in range(1, 101):
    s = s + k
print(s)
```

La variable `s`, initialisée à 0 avant la boucle, accumule les termes ; le programme affiche 5050. Pour un produit, on initialise à 1.

Boucle non bornée (while). On l'utilise quand on répète tant qu'une condition est vraie, sans savoir combien de fois. La condition est testée **avant** chaque passage. Il faut s'assurer qu'elle finira par devenir fausse, sinon le programme ne s'arrête jamais.

```
n = 0
while 2 ** n < 1000:
    n = n + 1
print(n)
```

Ce programme affiche 10, le plus petit entier `n` tel que $2^n \geq 1000$ ($2^9 = 512$ et $2^{10} = 1024$).

Choisir la boucle

1. Nombre de tours connu à l'avance : boucle `for` avec `range` • 2. Arrêt quand une condition change : boucle `while`, avec une condition qui finit par devenir fausse

4. Fonctions

Une **fonction** regroupe des instructions réutilisables. Elle reçoit des **paramètres** et **renvoie** une valeur avec `return`.

```
def f(x):
    return x ** 2 - 3 * x + 1

def est_pair(n):
    return n % 2 == 0
```

L'appel `f(2)` renvoie $4 - 6 + 1 = -1$, et `est_pair(7)` renvoie `False`.

Points essentiels :

- **return termine** immédiatement l'exécution de la fonction. Un `return` placé à l'intérieur d'une boucle l'interrompt dès le premier passage.
- **print n'est pas return** : `print` affiche une valeur à l'écran mais la fonction ne la transmet pas ; une fonction sans `return` renvoie la valeur spéciale `None`.
- Les variables créées dans une fonction sont **locales** : elles n'existent pas en dehors.

- Une fonction peut recevoir une autre fonction en paramètre : c'est ce qu'on fait pour la dichotomie ou la méthode d'Euler, qui s'appliquent à une fonction f quelconque.

5. Listes

Une **liste** est une suite ordonnée de valeurs : $L = [4, 7, 1, 9, 3]$.

- Les éléments sont numérotés à partir de **0** : $L[0]$ vaut 4, $L[4]$ vaut 3. $L[-1]$ désigne le dernier élément.
- $\text{len}(L)$ donne le nombre d'éléments (ici 5) ; les indices valides vont de 0 à $\text{len}(L) - 1$.
- $L.\text{append}(x)$ ajoute x à la fin ; $L[i] = x$ remplace l'élément d'indice i .
- $\text{sum}(L)$, $\text{max}(L)$, $\text{min}(L)$ donnent la somme, le maximum, le minimum.

Deux façons de parcourir une liste :

```
for x in L:
    print(x)

for i in range(len(L)):
    print(i, L[i])
```

La première parcourt les valeurs, la seconde les indices (utile quand on a besoin de la position).

Listes en compréhension. On construit une liste en une ligne à partir d'une autre :

```
carres = [k ** 2 for k in range(1, 6)]
grands = [x for x in L if x > 5]
```

carres vaut $[1, 4, 9, 16, 25]$ et grands vaut $[7, 9]$.

On peut aussi construire une liste dans une boucle en partant de la liste vide $[]$ et en utilisant `append` : c'est la méthode pour garder en mémoire les termes successifs d'une suite.

6. Algorithmes de seuil

Une suite (u_n) étant donnée, on cherche le **plus petit rang** n à partir duquel u_n dépasse (ou passe sous) une valeur fixée, appelée seuil. On ne connaît pas ce rang à l'avance : la boucle `while` s'impose.

Structure type, pour une suite définie par u_0 et $u_{n+1} = g(u_n)$, avec un seuil S quand la suite croît :

```
def seuil(S):
    n = 0
    u = 2
    while u < S:
        u = 3 * u - 1
        n = n + 1
    return n
```

Trois points à soigner : l'initialisation (n et u correspondent au même rang), l'ordre des mises à jour (u et n avancent ensemble dans la boucle), et la condition de la boucle, qui est la **négation** de la condition d'arrêt (on continue tant que $u < S$, on s'arrête dès que $u \geq S$). Il faut aussi être sûr que le seuil sera atteint, par exemple parce que la suite tend vers $+\infty$ ou vers une limite au-delà de S , sinon la boucle ne se termine pas.

Quand on connaît une formule explicite, on peut vérifier le résultat par le calcul, par exemple avec le logarithme népérien.

7. Dichotomie

Principe. Soit f une fonction **continue** sur $[a ; b]$ telle que $f(a)$ et $f(b)$ soient de signes contraires. D'après le théorème des valeurs intermédiaires, l'équation $f(x) = 0$ admet au moins une solution dans $[a ; b]$ (une seule si f est strictement monotone). On coupe l'intervalle en deux au milieu $m = (a + b)/2$:

- si $f(a)$ et $f(m)$ sont de signes contraires ($f(a) \times f(m) \leq 0$), une solution est dans $[a ; m]$: on remplace b par m ;
- sinon, une solution est dans $[m ; b]$: on remplace a par m .

On recommence tant que la longueur $b - a$ est supérieure à la précision ϵ voulue.

```
def dichotomie(f, a, b, e):
    while b - a > e:
        m = (a + b) / 2
        if f(a) * f(m) <= 0:
            b = m
        else:
            a = m
    return a, b
```

La fonction renvoie un encadrement $[a ; b]$ de la solution, d'amplitude au plus ϵ .

Nombre d'étapes. À chaque étape la longueur de l'intervalle est divisée par 2. Après n étapes, elle vaut $(b - a)/2^n$. Pour une précision ϵ , il faut le plus petit n tel que $(b - a)/2^n \leq \epsilon$. Sur un intervalle de longueur 1, une précision de 10^{-3} demande 10 étapes ($2^{10} = 1024$), une précision de 10^{-6} en demande 20 ($2^{20} = 1\,048\,576$). La dichotomie est donc **très efficace** : chaque étape apporte un chiffre binaire exact supplémentaire.

Une étape de dichotomie

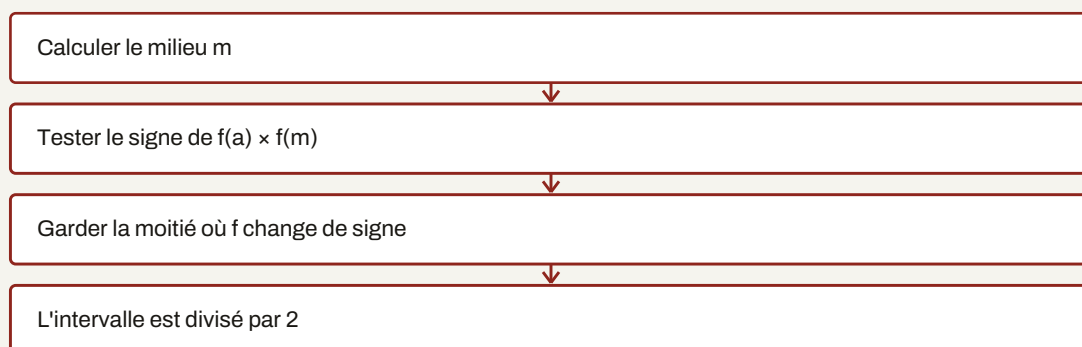


Schéma de principe original • lire les conditions et limites dans le texte

8. Méthode d'Euler

On cherche une fonction y telle que $y' = F(x, y)$ et $y(x_0) = y_0$ (par exemple $y' = y$, ou $y' = 2 - y$). Quand on ne sait pas résoudre exactement, ou pour visualiser une solution, on approche la courbe par une **suite de petits segments**.

Idée. Pour h petit, la dérivée donne l'approximation affine $y(x+h) \approx y(x) + h \times y'(x) = y(x) + h \times F(x, y(x))$. On part de $(x_0; y_0)$ et on avance d'un pas h :

$$x_{k+1} = x_k + h \quad y_{k+1} = y_k + h \times F(x_k, y_k)$$

```
def euler(F, x0, y0, h, n):
    x = x0
    y = y0
    X = [x]
    Y = [y]
    for k in range(n):
        y = y + h * F(x, y)
        x = x + h
        X.append(x)
        Y.append(y)
    return X, Y
```

L'ordre compte : on calcule le nouveau y avec l'**ancien** x , puis on avance x . Les listes X et Y permettent ensuite de tracer la courbe approchée.

Précision. Plus le pas h est petit, meilleure est l'approximation, mais plus il faut d'étapes. L'erreur vient de ce qu'on suit la tangente au lieu de la courbe : pour une fonction convexe (courbe au-dessus de ses tangentes), la méthode d'Euler donne des valeurs trop faibles.

Exemple : pour $y' = y$ avec $y(0) = 1$, la solution exacte est $\exp$. Avec un pas h , chaque étape multiplie y par $(1+h)$; après n étapes de pas $h = 1/n$, on obtient $(1 + 1/n)^n$, qui tend vers e quand n tend vers $+\infty$.

9. Simulation

Le module `random` fournit des nombres pseudo-aléatoires :

- `random()` renvoie un flottant au hasard dans $[0; 1[$;
- `randint(a, b)` renvoie un entier au hasard entre a et b **inclus**.

Pour simuler un événement de probabilité p , on teste `random() < p`, qui est vrai avec probabilité p .

```
from random import random, randint

def experience(p):
    if random() < p:
        return 1
    return 0

def frequence(p, n):
    succes = 0
    for i in range(n):
        succes = succes + experience(p)
    return succes / n
```

Répéter n fois une expérience de Bernoulli et compter les succès simule une **loi binomiale** $B(n, p)$. D'après la loi des grands nombres, la fréquence observée se rapproche de p quand n grandit : c'est ce qui permet d'**estimer une probabilité** par simulation. Pour n assez grand, dans environ 95 % des échantillons, la fréquence est dans l'intervalle $[p - 1/\sqrt{n}; p + 1/\sqrt{n}]$.

10. Lire et corriger un algorithme

Pour comprendre un programme, on dresse un **tableau d'état** : une colonne par variable, une ligne par passage dans la boucle, en notant les valeurs **après** chaque passage. On teste aussi les cas limites (liste vide, $n = 0$, seuil déjà atteint au départ).

Erreurs les plus fréquentes :

- **décalage d'une unité** : `range(1, n)` s'arrête à $n - 1$; un parcours qui commence à l'indice 1 oublie le premier élément ;
- **mauvaise initialisation** : une somme commence à 0, un produit à 1, un maximum au premier élément de la liste (et non à 0, faux si tous les éléments sont négatifs) ;
- **return mal placé** : à l'intérieur de la boucle, il l'arrête au premier tour ;
- **= au lieu de ==** dans un test ;
- **boucle infinie** : la variable de la condition n'est jamais modifiée dans la boucle, ou évolue dans le mauvais sens ;
- **indentation** : une instruction qui devrait être dans la boucle est placée après, ou l'inverse ;
- **mises à jour dans le mauvais ordre** : par exemple avancer x avant de calculer le nouveau y dans Euler.

Exemples résolus

Exemple 1 : un algorithme de seuil

Énoncé. On considère la suite définie par $u_0 = 100$ et $u_{n+1} = 0,8u_n + 10$. a) Écrire une fonction Python qui renvoie le plus petit entier n tel que $u_n < 51$. b) On admet que $u_n = 50 + 50 \times 0,8^n$. Déterminer ce rang par le calcul.

Résolution.

a) La suite décroît vers 50 ($u_1 = 90$, $u_2 = 82$, $u_3 = 75,6$). On continue tant que $u \geq 51$:

```
def seuil():
    n = 0
    u = 100
    while u >= 51:
        u = 0.8 * u + 10
        n = n + 1
    return n
```

b) $u_n < 51$ équivaut à $50 + 50 \times 0,8^n < 51$, soit $0,8^n < 0,02$. La fonction $\ln$ étant strictement croissante, cela équivaut à $n \times \ln(0,8) < \ln(0,02)$, puis, en divisant par $\ln(0,8) < 0$ (ce qui change le sens), $n > \ln(0,02)/\ln(0,8) \approx (-3,912)/(-0,2231) \approx 17,5$. Contrôle : $u_{17} = 50 + 50 \times 0,8^{17} \approx 50 + 50 \times 0,0225 \approx 51,13$ et $u_{18} \approx 50 + 50 \times 0,0180 \approx 50,90$. La fonction renvoie 18.

Exemple 2 : dichotomie

Énoncé. On considère $f(x) = x^3 + x - 1$ sur $[0 ; 1]$. a) Justifier que l'équation $f(x) = 0$ a une unique solution α dans $[0 ; 1]$. b) Dérouler les quatre premières étapes de l'algorithme de dichotomie. c) Combien d'étapes faut-il pour un encadrement d'amplitude 10^{-3} ?

Résolution.

a) f est dérivable et $f'(x) = 3x^2 + 1 > 0$: f est continue et strictement croissante sur $[0 ; 1]$. $f(0) = -1 < 0$ et $f(1) = 1 > 0$. D'après le théorème des valeurs intermédiaires (cas strictement monotone), l'équation a une unique solution α dans $[0 ; 1]$.

b) Tableau d'état :

Étape	a	b	m	f(m)	Nouvel intervalle
1	0	1	0,5	$0,125 + 0,5 - 1 = -0,375$	$[0,5 ; 1]$
2	0,5	1	0,75	$0,421875 + 0,75 - 1 \approx 0,172$	$[0,5 ; 0,75]$
3	0,5	0,75	0,625	$0,244140625 + 0,625 - 1 \approx -0,131$	$[0,625 ; 0,75]$
4	0,625	0,75	0,6875	$0,324951... + 0,6875 - 1 \approx 0,012$	$[0,625 ; 0,6875]$

À chaque étape, $f(a) < 0$: on garde $[a ; m]$ quand $f(m) > 0$, $[m ; b]$ quand $f(m) < 0$. Après quatre étapes, $0,625 \leq \alpha \leq 0,6875$ ($\alpha \approx 0,682$).

c) Après n étapes, l'amplitude vaut $1/2^n$. On veut $1/2^n \leq 10^{-3}$, soit $2^n \geq 1000$. Comme $2^9 = 512$ et $2^{10} = 1024$, il faut 10 étapes.

Exemple 3 : méthode d'Euler et approximation de e

Énoncé. On approche la solution de $y' = y$, $y(0) = 1$, par la méthode d'Euler. a) Avec un pas $h = 0,1$, calculer les trois premières valeurs approchées. b) Montrer que la valeur approchée obtenue en $x = 1$ vaut $1,1^{10}$, et la comparer à e. c) Que donne le programme pour $h = 0,01$?

Résolution.

a) Ici $F(x, y) = y$, donc $y_{k+1} = y_k + 0,1 \times y_k = 1,1 \times y_k$. On obtient $y_0 = 1$, $y_1 = 1,1$, $y_2 = 1,21$, $y_3 = 1,331$, valeurs approchées de $y(0,1)$, $y(0,2)$, $y(0,3)$.

b) Par récurrence immédiate, $y_k = 1,1^k$. On atteint $x = 1$ après 10 pas, donc $y_{10} = 1,1^{10} \approx 2,594$. La valeur exacte est $\exp(1) = e \approx 2,718$: l'erreur est d'environ 0,12. L'approximation est trop faible, car $\exp$ est convexe et la méthode suit les tangentes, situées sous la courbe.

c) Avec $h = 0,01$, il faut 100 pas et l'on obtient $1,01^{100} \approx 2,705$, soit une erreur d'environ 0,013 : diviser le pas par 10 a divisé l'erreur par environ 10. L'appel s'écrit :

```
def F(x, y):
    return y

X, Y = euler(F, 0, 1, 0.01, 100)
print(Y[-1])
```

Méthodes et pièges

- Avant d'écrire une boucle, demande-toi si tu connais le nombre de tours : oui, for ; non, while.
- Dans une boucle while, la condition écrite est celle pour **continuer** : c'est la négation de la condition d'arrêt demandée par l'énoncé.
- Vérifie qu'une boucle while se termine : la variable testée doit être modifiée dans la boucle et se rapprocher de la condition d'arrêt.
- Initialise une somme à 0, un produit à 1, un maximum au premier élément de la liste.
- Retiens que range(a, b) s'arrête à b - 1 et que les indices d'une liste commencent à 0.
- Ne confonds pas print et return : une fonction qui doit fournir un résultat à un autre calcul doit le renvoyer.
- Pour lire un algorithme, dresse un tableau d'état ligne par ligne au lieu de deviner : c'est la méthode qui évite les erreurs de décalage.
- Dans la dichotomie, compare les signes de f(a) et f(m), pas f(m) à 0 seul, et vérifie que f est continue et change de signe sur l'intervalle de départ.
- Dans la méthode d'Euler, calcule le nouveau y avec l'ancien x, puis avance x.
- N'écris jamais de test d'égalité entre flottants : compare la différence à une tolérance.

Vers le PASS

En PASS, la programmation n'est pas évaluée pour elle-même, mais les algorithmes de ce chapitre structurent le raisonnement numérique. La méthode d'Euler est exactement la façon dont on simule un bilan : une concentration de médicament qui diminue de façon proportionnelle à sa valeur ($C' = -kC$), une population, une charge qui varie. Les algorithmes de seuil répondent à des questions du type « au bout de combien de temps la concentration passe-t-elle sous le seuil d'efficacité ? ». La simulation et la loi des grands nombres préparent les biostatistiques, et la dichotomie illustre comment on résout numériquement une équation qu'on ne sait pas résoudre à la main.

Suite en PASS : Équations différentielles et bilans

Exercices

Niveau 1 : s'entraîner

1. Donne les valeurs de a et b après l'exécution de ces instructions, en détaillant chaque étape. Quel est le rôle de ce programme ?

```

a = 3
b = 5
a = a + b
b = a - b
a = a - b

```
2. Donne la valeur de chacune des expressions Python suivantes : $17 // 5$, $17 \% 5$, $2^{**} 5$, $7 / 2$, $10 \% 2 == 0$.
3. Quelle valeur affiche le programme suivant ? Justifie par un tableau d'état.

```
s = 0
for k in range(1, 6):
    s = s + k * k
print(s)
```

4. Écris une fonction `est_pair(n)` qui renvoie `True` si l'entier `n` est pair et `False` sinon, puis une fonction `compte_paires(L)` qui renvoie le nombre d'entiers pairs d'une liste `L`.
5. On définit $L = [4, 7, 1, 9, 3]$.
 - a) Donne `L[0]`, `L[2]`, `L[-1]` et `len(L)`. b) Donne la liste $M = [2 * x \text{ for } x \text{ in } L]$ et la liste $N = [x \text{ for } x \text{ in } L \text{ if } x > 3]$. c) Après l'instruction `L.append(6)`, que valent `len(L)` et `sum(L)` ?

Niveau 2 : type bac

6. Une ville compte 2000 habitants en 2026, et sa population augmente de 3 % par an.
 - a) Écris une fonction Python qui renvoie le nombre d'années nécessaires pour que la population atteigne ou dépasse 3000 habitants. b) Détermine ce nombre par le calcul.
7. Un élève a écrit la fonction suivante, censée renvoyer la moyenne des valeurs d'une liste non vide `L`.


```
def moyenne(L):
    s = 0
    for i in range(1, len(L)):
        s = s + L[i]
    return s / len(L)
```

- a) Que renvoie `moyenne([10, 12, 14])` ? Est-ce le résultat attendu ? b) Corrige la fonction. c) Un autre élève place `return s / len(L)` à l'intérieur de la boucle (indenté sous le `for`). Que renvoie alors sa fonction pour `[10, 12, 14]` avec un parcours correct commençant à l'indice 0 ?
8. On cherche une valeur approchée de $\sqrt{3}$ comme solution de $x^2 - 3 = 0$ sur $[1 ; 2]$.
 - a) Justifie que la méthode de dichotomie s'applique. b) Déroule à la main la fonction dichotomie du cours avec $f(x) = x^2 - 3$, $a = 1$, $b = 2$ et $\epsilon = 0,1$. Donne l'encadrement final et le nombre d'étapes. c) Combien d'étapes faudrait-il pour une précision de 10^{-4} ?
9. On considère l'équation différentielle $y' = 2 - y$ avec $y(0) = 0$.
 - a) Écris la relation de récurrence de la méthode d'Euler avec un pas h . b) Avec $h = 0,5$, calcule les valeurs approchées en $x = 0,5 ; 1 ; 1,5 ; 2$. c) La solution exacte est $y(x) = 2 - 2\exp(-x)$. Compare en $x = 2$.

Niveau 3 : vers le PASS

10. On lance deux dés équilibrés à six faces et on s'intéresse à l'événement S : « la somme vaut 7 ».
 - a) Écris une fonction `simule(n)` qui simule n lancers de deux dés et renvoie la fréquence de S . b) Calcule la probabilité exacte de S . c) Pour $n = 1000$, dans quel intervalle la fréquence obtenue se trouve-t-elle dans environ 95 % des cas ?
11. Après une injection, la concentration $C(t)$ d'un médicament dans le plasma (en $\text{mg} \cdot \text{L}^{-1}$, t en heures) vérifie $C' = -0,2C$, avec $C(0) = 10$.
 - a) Montre qu'avec la méthode d'Euler de pas $h = 1$, on obtient $C_{k+1} = 0,8C_k$. b) Calcule la valeur approchée de $C(3)$ et compare-la à la valeur exacte $10\exp(-0,6)$. c) Écris un algorithme de seuil donnant

le premier instant entier où la valeur approchée passe sous $1 \text{ mg} \cdot \text{L}^{-1}$, et détermine ce rang. d) Compare à l'instant exact, sachant que $C(t) = 10\exp(-0,2t)$. Explique le sens de l'écart.

12. On veut résoudre $f(x) = 0$ sur $[0; 1]$ avec une précision de 10^{-6} , pour une fonction f continue et strictement monotone qui change de signe.

a) Méthode de balayage : on calcule f en $0, 10^{-6}, 2 \times 10^{-6}$, etc. Combien d'évaluations de f faut-il au pire ? b) Combien d'étapes de dichotomie faut-il ? c) Conclut sur l'efficacité comparée des deux méthodes.

13. On considère la fonction suivante, où n est un entier positif ou nul.

```
def mystere(n):
    c = 0
    while n > 0:
        c = c + n % 10
        n = n // 10
    return c
```

a) Dresse le tableau d'état pour l'appel `mystere(2026)`. b) Que calcule cette fonction ? Que renvoie `mystere(0)` ? c) Pourquoi la boucle se termine-t-elle toujours ?

Corrigés

1. Étape par étape : $a = 3, b = 5$; puis $a = 3 + 5 = 8$; puis $b = 8 - 5 = 3$; puis $a = 8 - 3 = 5$. **À la fin, $a = 5$ et $b = 3$: le programme échange les valeurs de a et b sans variable auxiliaire (en Python, $a, b = b, a$ fait la même chose).**

2. $17 // 5 = 3$ (quotient euclidien, $17 = 5 \times 3 + 2$) ; $17 \% 5 = 2$ (reste) ; $2 \cdot 5 = 10$; $17 / 5 = 3.4$ (flottant) ; $10 \% 2 == 0$ vaut `True`** (10 est pair).

3. `range(1, 6)` donne $k = 1, 2, 3, 4, 5$.

k	s après le passage
1	$0 + 1 = 1$
2	$1 + 4 = 5$
3	$5 + 9 = 14$
4	$14 + 16 = 30$
5	$30 + 25 = 55$

Le programme affiche 55, la somme des carrés des entiers de 1 à 5.

4. Un entier est pair si son reste dans la division par 2 est nul.

```
def est_pair(n):
    return n % 2 == 0

def compte_pairs(L):
    c = 0
    for x in L:
        if est_pair(x):
```



```
        c = c + 1
    return c
```

Par exemple, `compte_pairs([4, 7, 1, 8])` renvoie 2. Une version en une ligne : `return len([x for x in L if x % 2 == 0])`.

5. a) $L[0] = 4$, $L[2] = 1$ (troisième élément, car on compte à partir de 0), $L[-1] = 3$ (dernier élément), $\text{len}(L) = 5$.

b) $M = [8, 14, 2, 18, 6]$ et $N = [4, 7, 9]$ (on garde, dans l'ordre, les éléments strictement supérieurs à 3). c) L devient $[4, 7, 1, 9, 3, 6]$: $\text{len}(L) = 6$ et $\text{sum}(L) = 4 + 7 + 1 + 9 + 3 + 6 = 30$.

6. a) Une hausse de 3 % revient à multiplier par 1,03. On compte les années tant que la population est sous 3000 :

```
def annees():
    n = 0
    p = 2000
    while p < 3000:
        p = p * 1.03
        n = n + 1
    return n
```

b) La population au bout de n années vaut $2000 \times 1,03^n$. On veut $2000 \times 1,03^n \geq 3000$, soit $1,03^n \geq 1,5$, soit $n \times \ln(1,03) \geq \ln(1,5)$ ($\ln$ croissante), donc $n \geq \ln(1,5)/\ln(1,03) \approx 0,4055/0,02956 \approx 13,7$. Contrôle : $2000 \times 1,03^{13} \approx 2937$ et $2000 \times 1,03^{14} \approx 3025$. La fonction renvoie 14 : le seuil est atteint en 2040.

7. a) La boucle parcourt $i = 1$ et $i = 2$ seulement : $s = 12 + 14 = 26$, et la fonction renvoie $26/3 \approx 8,67$, alors que la moyenne vaut $(10 + 12 + 14)/3 = 12$. Le premier élément (indice 0) est oublié : c'est une erreur de décalage d'une unité.

b) Correction : on parcourt tous les indices, ou directement les valeurs.

```
def moyenne(L):
    s = 0
    for x in L:
        s = s + x
    return s / len(L)
```

c) Le return est exécuté dès le premier passage, avec $s = 10$: la fonction renvoie $10 / 3 \approx 3,33$. Le return doit être placé après la boucle, au même niveau d'indentation que le for.

8. a) $f(x) = x^2 - 3$ est continue sur $[1 ; 2]$, strictement croissante sur cet intervalle ($f'(x) = 2x > 0$), avec $f(1) = -2 < 0$ et $f(2) = 1 > 0$. L'équation a une unique solution dans $[1 ; 2]$, qui est $\sqrt{3}$, et la dichotomie s'applique.

b) Tableau d'état ($f(a)$ reste négatif) :

Étape	a	b	m	f(m)	b - a après l'étape
1	1	2	1,5	$2,25 - 3 = -0,75$	$a = 1,5 ; 0,5$

Étape	a	b	m	f(m)	b - a après l'étape
2	1,5	2	1,75	$3,0625 - 3 = 0,0625$	$b = 1,75 ; 0,25$
3	1,5	1,75	1,625	$2,640625 - 3 \approx -0,359$	$a = 1,625 ; 0,125$
4	1,625	1,75	1,6875	$2,84765625 - 3 \approx -0,152$	$a = 1,6875 ; 0,0625$

Après l'étape 4, $b - a = 0,0625 \leq 0,1$: la boucle s'arrête. **Encadrement : $1,6875 \leq \sqrt{3} \leq 1,75$, obtenu en 4 étapes** ($\sqrt{3} \approx 1,732$). c) Il faut $1/2^n \leq 10^{-4}$, soit $2^n \geq 10\,000$. Or $2^{13} = 8192$ et $2^{14} = 16\,384$: **14 étapes**.

9. a) Ici $F(x, y) = 2 - y$, donc $y_{k+1} = y_k + h(2 - y_k)$.

b) Avec $h = 0,5$: $y_{k+1} = y_k + 0,5(2 - y_k) = 0,5y_k + 1$. $y(0,5) \approx 0 + 0,5 \times 2 = 1$; $y(1) \approx 1 + 0,5 \times 1 = 1,5$; $y(1,5) \approx 1,5 + 0,5 \times 0,5 = 1,75$; $y(2) \approx 1,75 + 0,5 \times 0,25 = 1,875$. c) Valeur exacte : $y(2) = 2 - 2\exp(-2) \approx 2 - 2 \times 0,1353 \approx 1,729$. L'approximation 1,875 est trop forte d'environ 0,15 : la solution exacte est concave ($y'' = -y' = -2\exp(-x) < 0$), donc sous ses tangentes, et Euler, qui suit les tangentes, la surestime. Un pas plus petit réduirait l'écart.

10. a) On compte les lancers dont la somme vaut 7 :

```
from random import randint

def simule(n):
    c = 0
    for i in range(n):
        if randint(1, 6) + randint(1, 6) == 7:
            c = c + 1
    return c / n
```

b) Il y a 36 couples équiprobables, dont 6 donnent 7 : (1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1). $P(S) = 6/36 = 1/6 \approx 0,167$. c) $1/\sqrt{1000} \approx 0,0316$. L'intervalle est $[1/6 - 0,0316 ; 1/6 + 0,0316]$, soit **environ $[0,135 ; 0,198]$** .

11. a) Euler avec $h = 1$: $C_{k+1} = C_k + 1 \times (-0,2C_k) = 0,8C_k$.

b) $C_3 = 10 \times 0,8^3 = 5,12 \text{ mg} \cdot \text{L}^{-1}$. Valeur exacte : $10\exp(-0,6) \approx 5,49 \text{ mg} \cdot \text{L}^{-1}$. Euler sous-estime de 0,37 $\text{mg} \cdot \text{L}^{-1}$ environ. c) Algorithme :

```
def seuil_concentration():
    t = 0
    C = 10
    while C >= 1:
        C = 0.8 * C
        t = t + 1
    return t
```

$C_k = 10 \times 0,8^k < 1$ équivaut à $0,8^k < 0,1$, soit $k > \ln(0,1)/\ln(0,8) \approx 10,3$. Contrôle : $C_{10} \approx 10 \times 0,107 \approx 1,07$ et $C_{11} \approx 10 \times 0,0859 \approx 0,86$. **La fonction renvoie 11 (heures)**. d) Exactement : $10\exp(-0,2t) < 1$ équivaut à $t > \ln(10)/0,2 = 5\ln(10) \approx 11,5 \text{ h}$. La méthode d'Euler fait décroître la concentration trop vite : la courbe exacte est convexe, au-dessus de ses tangentes, et Euler suit les tangentes, donc sous-estime C et

annonce le passage sous le seuil trop tôt.

12. a) Le balayage teste les points $k \times 10^{-6}$ pour k de 0 à 10^6 : au pire **environ 10^6 évaluations (1 000 001)**.

b) Il faut $1/2^n \leq 10^{-6}$, soit $2^n \geq 10^6$. Comme $2^{19} = 524\,288$ et $2^{20} = 1\,048\,576$, **20 étapes** suffisent (deux évaluations de f par étape dans la version du cours, soit 40 évaluations au plus). c) **La dichotomie est des dizaines de milliers de fois plus rapide** : son nombre d'étapes croît comme le nombre de chiffres de précision voulus (environ 3,3 étapes par chiffre décimal), alors que le balayage est multiplié par 10 à chaque chiffre supplémentaire.

13. a) Tableau d'état (valeurs après chaque passage) :

Passage	$n \% 10$	c	n
départ		0	2026
1	6	6	202
2	2	8	20
3	0	8	2
4	2	10	0

n vaut 0, la boucle s'arrête : **mystere(2026) renvoie 10**. b) $n \% 10$ est le dernier chiffre de n et $n // 10$ supprime ce chiffre : **la fonction calcule la somme des chiffres de n** . Pour $n = 0$, la condition $n > 0$ est fausse dès le départ : **mystere(0) renvoie 0**. c) Tant que $n > 0$, $n // 10 < n$: n est un entier positif qui **décroît strictement** à chaque passage. Une suite d'entiers positifs strictement décroissante ne peut pas être infinie : n atteint 0 et la boucle se termine.