

Algorithmique & Python

Dichotomie ■ Euler ■ Simulation ■ Riemann ■ Suites

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

Les algorithmes du programme de Terminale

Terminale — Spécialité Mathématiques — Programme officiel



Table des matières

1	🔍 Pourquoi étudier l'algorithmique en maths ?	3
1.1	De quoi parle-t-on ?	3
1.2	Les 7 algorithmes du programme	3
1.3	L'idée directrice	3
2	📖 Cours Python — Les bases pour débutants	4
2.1	Les variables : des boîtes avec une étiquette	4
2.2	Les types de données	4
2.3	Les opérations de calcul	5
2.4	Afficher et lire : <code>print</code> et <code>input</code>	6
2.5	Les comparaisons et les booléens	6
2.6	La conditionnelle : <code>if</code> / <code>elif</code> / <code>else</code>	6
2.7	La boucle <code>for</code> : répéter un nombre connu de fois	7
2.8	La boucle <code>while</code> : répéter tant qu'une condition est vraie	9
2.9	Les fonctions : créer ses propres outils	10
2.10	Les listes : stocker plusieurs valeurs	11
2.11	Le module <code>random</code> : simuler le hasard	12
2.12	Récapitulatif : les structures à connaître au bac	13
3	🎓 Les algorithmes du programme	14
3.1	Algorithme de dichotomie (TVI)	14
3.2	Méthode d'Euler (équations différentielles)	15
3.3	Sommes de Riemann (intégrales)	16
3.4	Simulation aléatoire et loi des grands nombres	16
3.5	Suites récurrentes : calcul de termes et recherche de seuil	17
3.6	Recherche d'extremum et tri	18
3.7	Calcul de coefficients binomiaux et triangle de Pascal	18
3.8	Visualisation graphique	19
3.9	Résumé : quel algorithme pour quel problème ?	20
4	🧰 Boîte à outils — Réflexes pour le bac	21
5	✍ Exercices	23
6	🔑 Problème — La méthode de Newton ★★	26
7	✔ Corrigés détaillés	27

1 ? Pourquoi étudier l'algorithmique en maths ?

1.1 De quoi parle-t-on ?

L'algorithmique est la science de la **résolution méthodique** de problèmes. En Terminale, Python est l'outil qui permet de **mettre en œuvre** les algorithmes pour résoudre des problèmes que le calcul à la main ne peut pas traiter : trouver un zéro de fonction, approcher une intégrale, simuler des expériences aléatoires, résoudre des équations différentielles numériquement.

1.2 Les 7 algorithmes du programme



1.3 L'idée directrice

L'idée directrice :

Chaque algorithme répond à une question précise du cours de maths. L'algorithmique n'est pas un chapitre isolé : c'est un **outil transversal** qui donne vie à tous les autres chapitres. Un algorithme bien compris, c'est un théorème **concret**.

2 Cours Python — Les bases pour débutants

Intuition | Python, c'est quoi ?

Python est un **langage de programmation** : une façon de donner des instructions à un ordinateur. On écrit du texte (le **code**), l'ordinateur l'exécute ligne par ligne, de haut en bas. C'est comme une recette de cuisine : chaque ligne est une étape.

Où écrire du Python ? Au lycée, on utilise généralement **Thonny**, **Spyder**, **EduPython** ou le site repl.it. Tu écris dans l'éditeur (zone blanche) et tu cliques sur « Exécuter » (ou F5).

2.1 Les variables : des boîtes avec une étiquette

Définition | Variable

Une **variable** est une **boîte** dans la mémoire de l'ordinateur. Elle a un **nom** (l'étiquette) et contient une **valeur**. Le symbole `=` signifie « mettre la valeur dans la boîte » (on dit **affecter**).

Créer et modifier des variables

```
1 x = 5           # la boîte x contient 5
2 nom = "Alice"   # la boîte nom contient le texte "Alice"
3 pi = 3.14159     # la boîte pi contient 3.14159
4
5 x = x + 3        # on prend la valeur de x (5), on ajoute 3,
6                  # et on remet le résultat (8) dans x
7 print(x)         # affiche 8
```

Attention | `=` n'est pas l'égalité mathématique !

En maths, $x = x + 3$ est impossible. En Python, `x = x + 3` signifie :

1. Lire la valeur actuelle de `x` (côté droit du `=`).
2. Calculer `x + 3`.
3. Stocker le résultat dans `x` (côté gauche du `=`).

Le `=` est une **affectation** (flèche \leftarrow), pas un test d'égalité !

Pour tester l'égalité, on utilise `==` (deux signes égaux) :

`x == 8` renvoie `True` (vrai) ou `False` (faux).

2.2 Les types de données

✓ Propriété | Les 4 types à connaître

Type	Signification	Exemples	Usage en maths
<code>int</code>	Entier	0, -3, 42, 1000	$n \in \mathbb{N}$, compteurs
<code>float</code>	Décimal (flottant)	3.14, -0.5, 2.0	$x \in \mathbb{R}$, calculs
<code>str</code>	Texte (chaîne)	"Bonjour", "42"	Affichage
<code>bool</code>	Booléen (vrai/faux)	True, False	Tests, conditions

⚠ Attention | Piège : "42" \neq 42

"42" est un **texte** (deux caractères), 42 est un **nombre** (entier).
On ne peut pas calculer "42" + 1 (erreur!), mais on peut calculer 42 + 1 = 43.

2.3 Les opérations de calcul

✓ Propriété | Opérations arithmétiques

Python	Signification	Exemple	Résultat
<code>a + b</code>	Addition	7 + 3	10
<code>a - b</code>	Soustraction	7 - 3	4
<code>a * b</code>	Multiplication	7 * 3	21
<code>a / b</code>	Division réelle	7 / 2	3.5
<code>a // b</code>	Division entière	7 // 2	3
<code>a % b</code>	Reste (modulo)	7 % 2	1
<code>a ** b</code>	Puissance a^b	2 ** 10	1024

💡 Exemple | Attention à la division !

```
1 print(7 / 2)      # 3.5 (division réelle, résultat float)
2 print(7 // 2)     # 3   (division entière, partie entière)
3 print(7 % 2)      # 1   (reste : 7 = 3 * 2 + 1)
```

Au bac, on utilise presque toujours / (division réelle).

2.4 Afficher et lire : print et input

Afficher un résultat avec print

```

1 x = 42
2 print(x)                # affiche : 42
3 print("La valeur est", x) # affiche : La valeur est 42
4 print(f"x vaut {x}")     # affiche : x vaut 42 (f-string)
5 print(f"x    = {x**2}")  # affiche : x    = 1764

```

Intuition | Quand utiliser print ?

`print` **affiche** un résultat à l'écran. C'est différent de `return` (qui **renvoie** une valeur depuis une fonction). Au bac, on te demande souvent « qu'affiche ce programme ? » — il faut repérer les `print`.

2.5 Les comparaisons et les booléens

Propriété | Opérateurs de comparaison

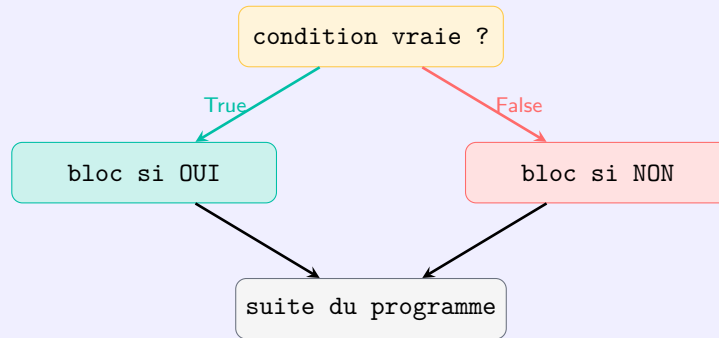
Python	Signification	Exemple	Résultat
<code>==</code>	Égal à	<code>5 == 5</code>	<code>True</code>
<code>!=</code>	Différent de	<code>5 != 3</code>	<code>True</code>
<code><</code>	Strictement inférieur	<code>3 < 5</code>	<code>True</code>
<code><=</code>	Inférieur ou égal	<code>5 <= 5</code>	<code>True</code>
<code>></code>	Strictement supérieur	<code>3 > 5</code>	<code>False</code>
<code>>=</code>	Supérieur ou égal	<code>7 >= 3</code>	<code>True</code>

On peut combiner avec `and` (et), `or` (ou), `not` (non) :
`(x > 0) and (x < 10)` est vrai si $0 < x < 10$.

2.6 La conditionnelle : if / elif / else

Définition | if : exécuter du code sous condition

La structure `if` permet d'exécuter certaines lignes **seulement si** une condition est vraie.



if / elif / else — Syntaxe complète

```

1 note = 14
2
3 if note >= 16:
4     print("Très bien")           # exécuté si note >= 16
5 elif note >= 12:
6     print("Bien")                # exécuté si 12 <= note < 16
7 elif note >= 10:
8     print("Passable")           # exécuté si 10 <= note < 12
9 else:
10    print("Insuffisant")         # exécuté si note < 10
11
12 # Ici, affiche "Bien" car 14 >= 12 et 14 < 16
  
```

⚠ Attention | L'indentation est obligatoire !

Le code à l'intérieur d'un if doit être **décalé de 4 espaces** (ou 1 tabulation). C'est l'**indentation**. Sans elle, Python ne sait pas quel code appartient au if.

```

1 # CORRECT :
2 if x > 0:
3     print("positif")           # indenté = appartient au if
4
5 # ERREUR :
6 if x > 0:
7 print("positif")              # pas indenté = erreur !
  
```

Règle simple : après chaque ligne qui finit par : (deux-points), on indente le bloc suivant.

2.7 La boucle for : répéter un nombre connu de fois

📖 Définition | Boucle for

La boucle for répète un bloc de code un **nombre fixé** de fois. La variable de boucle prend successivement chaque valeur d'une séquence.

Comprendre for et range

```

1 # Affiche 0, 1, 2, 3, 4 (5 valeurs, de 0 à 4)
2 for i in range(5):
3     print(i)
4
5 # Affiche 1, 2, 3, 4, 5 (de 1 à 5)
6 for i in range(1, 6):          # ATTENTION : 6 est exclu !
7     print(i)
8
9 # Affiche 0, 2, 4, 6, 8 (de 0 à 8, de 2 en 2)
10 for i in range(0, 10, 2):
11     print(i)

```

✓ Propriété | Les 3 formes de range

Écriture	Valeurs produites	Nombre de valeurs
<code>range(n)</code>	$0, 1, 2, \dots, n-1$	n valeurs
<code>range(a, b)</code>	$a, a+1, \dots, b-1$	$b - a$ valeurs
<code>range(a, b, p)</code>	$a, a+p, a+2p, \dots$	jusqu'à $< b$

Règle d'or : range s'arrête **toujours avant** la borne supérieure.

Pour aller de 1 à n : `range(1, n+1)`.

💡 Exemple | Motif classique : calculer une somme $\sum_{k=1}^n k^2$

```

1 # Calculer S = 1 + 2 + 3 + ... + 100
2 S = 0                                # Initialiser l'accumulateur à 0
3 for k in range(1, 101):              # k va de 1 à 100
4     S = S + k**2                      # Ajouter k à S à chaque tour
5 print(S)                             # Affiche 338350

```

Comprendre le mécanisme :

Tour	k	S (après)
1	1	$0 + 1 = 1$
2	2	$1 + 4 = 5$
3	3	$5 + 9 = 14$
4	4	$14 + 16 = 30$
\vdots	\vdots	\vdots
100	100	$328350 + 10000 = 338350$

✂ Méthode | Le motif « accumulateur » (très fréquent au bac)

Pour calculer une somme, un produit ou un compteur :

```

1 # SOMME : S = a_1 + a_2 + ... + a_n
2 S = 0                                # neutre de l'addition

```



```

3 for k in range(1, n + 1):
4     S = S + terme(k)           # ou S += terme(k)
5
6 # PRODUIT : P = a_1    a_2    ...    a_n
7 P = 1                         # neutre de la multiplication
8 for k in range(1, n + 1):
9     P = P * terme(k)          # ou P *= terme(k)
10
11 # COMPTEUR : combien de k vérifient une condition ?
12 C = 0
13 for k in range(1, n + 1):
14     if condition(k):
15         C = C + 1             # ou C += 1

```

2.8 La boucle while : répéter tant qu'une condition est vraie

Définition | Boucle while

La boucle while répète un bloc de code **tant que** la condition est vraie. On l'utilise quand on **ne sait pas à l'avance** combien de répétitions seront nécessaires.

while — recherche de seuil

```

1 # Trouver le plus petit n tel que 2^n > 1 000 000
2 n = 0
3 while 2**n <= 1000000:      # tant que 2^n <= 1 000 000
4     n = n + 1               # passer au n suivant
5 print(n)                   # affiche 20 (car 2^20 = 1 048 576)

```

Comprendre le mécanisme :

Tour	n	2^n
—	0	$1 \leq 1\,000\,000 \rightarrow$ on continue
1	1	$2 \leq 1\,000\,000 \rightarrow$ on continue
2	2	$4 \leq 1\,000\,000 \rightarrow$ on continue
\vdots	\vdots	\vdots
19	19	$524\,288 \leq 1\,000\,000 \rightarrow$ on continue
20	20	$1\,048\,576 > 1\,000\,000 \rightarrow$ on sort !

Attention | Boucle infinie — le danger du while

Si la condition ne devient **jamais fausse**, le programme tourne à l'infini et ne s'arrête jamais !
Exemple :

```

1 n = 1
2 while n > 0:                # n est toujours > 0 car on l'augmente !
3     n = n + 1               # BOUCLE INFINIE : appuyer Ctrl+C pour arrêter

```

Toujours vérifier que la condition finira par devenir fausse.

✂ Méthode | for ou while ? Comment choisir

for

Tu sais **combien** de fois
répéter (nombre fixé)

Exemples : $\sum_{k=1}^n$, afficher
 n termes, n lancers de dé

?

while

Tu ne sais pas quand
t'arrêter (condition)

Exemples : $u_n > M$ (seuil), $b -$
 $a < \varepsilon$ (dichotomie), $|u_n - \ell| < \varepsilon$

2.9 Les fonctions : créer ses propres outils

📖 Définition | Fonction Python

Une **fonction** est un bloc de code réutilisable. On la définit une fois avec `def`, puis on l'**appelle** autant de fois qu'on veut.

`def` = je **définis** la recette. Appel = je **lance** la recette.

🐍 Définir et appeler une fonction

```
1 # DÉFINIR la fonction (rien ne s'exécute encore)
2 def carre(x):
3     return x ** 2          # renvoie x
4
5 # APPELER la fonction (maintenant ça s'exécute)
6 resultat = carre(5)        # resultat = 25
7 print(carre(3))           # affiche 9
8 print(carre(7) + 1)       # affiche 50
```

⚠ Attention | return ≠ print !

```
1 def f(x):
2     return x + 1          # RENVOIE la valeur (on peut la réutiliser)
3
4 def g(x):
5     print(x + 1)          # AFFICHE la valeur (on ne peut pas la ré
6                             utiliser)
7
8 a = f(5)                  # a vaut 6 (la valeur est stockée dans a)
9 b = g(5)                  # affiche 6, mais b vaut None (rien n'est renvoyé !)
```

Au bac, on utilise `return` pour les calculs et `print` pour l'affichage.

🐍 Fonctions mathématiques classiques

```
1 import math                # OBLIGATOIRE pour utiliser les fonctions
2     maths
3
4 def f(x):
5     return x**2 - 3*x + 1    # f(x) = x^2 - 3x + 1
```

```

6 def g(x):
7     return math.exp(x) - 2*x          # g(x) = e^x - 2x
8
9 def h(x):
10    return math.log(x) + math.sqrt(x)  # h(x) = ln(x) + x
11
12 def derive_approx(f, x, h=1e-6):
13    return (f(x + h) - f(x)) / h      # f'(x)      [f(x+h) - f(x)] / h
14
15 # Appels :
16 print(f(2))          # 4 - 6 + 1 = -1
17 print(g(0))          # 1 - 0 = 1
18 print(h(1))          # 0 + 1 = 1

```

✓ Propriété | Fonctions de la bibliothèque math

Python	Mathématiques	Exemple
<code>math.sqrt(x)</code>	\sqrt{x}	<code>math.sqrt(9) = 3.0</code>
<code>math.exp(x)</code>	e^x	<code>math.exp(1) ≈ 2.718</code>
<code>math.log(x)</code>	$\ln(x)$	<code>math.log(1) = 0.0</code>
<code>math.cos(x)</code>	$\cos(x)$ (radians)	<code>math.cos(0) = 1.0</code>
<code>math.sin(x)</code>	$\sin(x)$ (radians)	<code>math.sin(math.pi) ≈ 0</code>
<code>math.pi</code>	π	3.14159265...
<code>math.factorial(n)</code>	$n!$	<code>math.factorial(5) = 120</code>
<code>abs(x)</code>	$ x $	<code>abs(-7) = 7</code>

Il faut écrire `import math` **une seule fois** en début de programme.

2.10 Les listes : stocker plusieurs valeurs

📖 Définition | Liste

Une **liste** est une variable qui contient **plusieurs valeurs**, numérotées à partir de **0**. On la crée avec des crochets `[]`.

🔧 Créer, lire et modifier une liste

```

1 # Créer une liste
2 L = [10, 20, 30, 40, 50]
3
4 # Lire un élément (ATTENTION : on compte à partir de 0 !)
5 print(L[0])          # 10 (premier élément)

```

```

6 print(L[2])      # 30  (troisième élément)
7 print(L[-1])     # 50  (dernier élément)
8 print(len(L))    # 5   (nombre d'éléments)
9
10 # Ajouter un élément à la fin
11 L.append(60)     # L = [10, 20, 30, 40, 50, 60]
12
13 # Modifier un élément
14 L[1] = 99        # L = [10, 99, 30, 40, 50, 60]

```

⚠ Attention | Les indices commencent à 0 !

Indice	0	1	2	3	4
Valeur	10	20	30	40	50

L[5] provoque une **erreur** (il n'y a que 5 éléments, d'indices 0 à 4).

🔗 Construire une liste avec une boucle

```

1 # Liste des carrés de 0 à 9
2 carres = []                # liste vide
3 for k in range(10):
4     carres.append(k**2)
5 print(carres)              # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
6
7 # Même chose en UNE ligne (compréhension de liste)
8 carres = [k**2 for k in range(10)]
9
10 # Fonctions utiles sur les listes
11 print(sum(carres))         # 285 (somme)
12 print(max(carres))        # 81 (maximum)
13 print(min(carres))        # 0  (minimum)

```

2.11 Le module random : simuler le hasard

🔗 Les 3 fonctions essentielles de random

```

1 import random
2
3 # Nombre décimal aléatoire entre 0 (inclus) et 1 (exclu)
4 x = random.random()        # ex: 0.7234...
5
6 # Entier aléatoire entre a et b (les deux inclus)
7 d = random.randint(1, 6)    # simule un dé : 1, 2, 3, 4, 5 ou 6
8
9 # Choisir un élément au hasard dans une liste
10 couleur = random.choice(["rouge", "bleu", "vert"])

```

💡 Exemple | Simuler un lancer de pièce

```

1 import random
2
3 def pile_ou_face():
4     if random.random() < 0.5:      # 50% de chance
5         return "Pile"
6     else:
7         return "Face"
8
9 # Ou plus simplement : simuler un Bernoulli de paramètre p
10 def bernoulli(p):
11     if random.random() < p:
12         return 1      # succès
13     else:
14         return 0      # échec

```

2.12 Récapitulatif : les structures à connaître au bac

✓ Propriété | Antisèche Python pour le bac

Structure	Modèle
Affectation	<code>x = valeur</code>
Affichage	<code>print(x)</code> ou <code>print(f"texte {x}")</code>
Condition	<code>if condition:</code> (puis <code>elif</code> / <code>else</code>)
Boucle fixe	<code>for k in range(n):</code>
Boucle condition	<code>while condition:</code>
Fonction	<code>def nom(param): return résultat</code>
Liste vide	<code>L = []</code> puis <code>L.append(valeur)</code>
Somme	<code>S = 0</code> puis <code>S += terme</code> dans une boucle
Import maths	<code>import math</code> (<code>math.exp</code> , <code>math.log</code> , <code>math.sqrt</code>)
Import hasard	<code>import random</code> (<code>random.random()</code> , <code>random.randint</code>)

3 Les algorithmes du programme

3.1 Algorithme de dichotomie (TVI)

★ Théorème | Rappel : Théorème des valeurs intermédiaires

Si f est **continue** sur $[a, b]$ et $f(a) \cdot f(b) < 0$, alors f s'annule au moins une fois sur $]a, b[$.

Définition | Principe de la dichotomie

On cherche un zéro de f sur $[a, b]$. L'idée : couper l'intervalle en deux à chaque étape.

1. Calculer $m = \frac{a+b}{2}$ (milieu).
2. Si $f(a) \cdot f(m) \leq 0$: le zéro est dans $[a, m]$, poser $b = m$.
3. Sinon : le zéro est dans $[m, b]$, poser $a = m$.
4. Répéter jusqu'à $b - a < \varepsilon$ (précision voulue).

Après n étapes, l'intervalle a une largeur $\frac{b-a}{2^n}$.

Dichotomie — trouver un zéro de f

```

1 def dichotomie(f, a, b, eps):
2     """Retourne une valeur approchée d'un zéro de f sur [a,b]."""
3     while b - a > eps:
4         m = (a + b) / 2
5         if f(a) * f(m) <= 0:
6             b = m
7         else:
8             a = m
9     return (a + b) / 2

```

Exemple | Résoudre $x^2 = 2$ (\Leftrightarrow trouver $\sqrt{2}$)

```

1 def f(x):
2     return x**2 - 2
3
4 resultat = dichotomie(f, 1, 2, 1e-10)
5 print(resultat)    # 1.4142135623...

```

$f(1) = -1 < 0$ et $f(2) = 2 > 0$. Après $\lceil \log_2(\frac{1}{10^{-10}}) \rceil = 34$ étapes, on a $\sqrt{2} \approx 1,4142135624$.

✓ Propriété | Complexité

Après n étapes, la précision est $\frac{b_0 - a_0}{2^n}$. Pour atteindre une précision ε : $n \geq \frac{\ln(\frac{b_0 - a_0}{\varepsilon})}{\ln 2}$. C'est une complexité **logarithmique** : très rapide.

3.2 Méthode d'Euler (équations différentielles)

Définition | Principe d'Euler

On approche la solution de $y' = f(t, y)$ avec $y(t_0) = y_0$ en remplaçant la dérivée par un taux d'accroissement :

$$y'(t) \approx \frac{y(t+h) - y(t)}{h} \implies y(t+h) \approx y(t) + h \cdot f(t, y(t))$$

Avec un pas $h > 0$ et des points $t_k = t_0 + kh$:

$$y_{k+1} = y_k + h \cdot f(t_k, y_k)$$

Méthode d'Euler — $y' = f(t, y)$, $y(t_0) = y_0$

```

1 def euler(f, t0, y0, h, n):
2     """Méthode d'Euler : n pas de taille h."""
3     T = [t0]
4     Y = [y0]
5     t, y = t0, y0
6     for k in range(n):
7         y = y + h * f(t, y)
8         t = t + h
9         T.append(t)
10        Y.append(y)
11    return T, Y

```

Exemple | $y' = -2y + 6$, $y(0) = 1$ (solution exacte : $y(t) = -2e^{-2t} + 3$)

```

1 def f(t, y):
2     return -2 * y + 6
3
4 T, Y = euler(f, 0, 1, 0.01, 500) # t de 0 à 5
5 # Y[-1]      3.0000 (converge vers y_infini = 3)

```

Attention | Le pas h doit être petit

Plus h est petit, plus l'approximation est précise, mais plus le calcul est long. Un bon compromis au bac : $h = 0,01$ ou $h = 0,001$. L'erreur d'Euler est proportionnelle à h (méthode d'ordre 1).

Propriété | Cas $y' = ay + b$: formule d'Euler explicite

Pour $y' = ay + b$, la récurrence d'Euler devient :

$$y_{k+1} = y_k + h(ay_k + b) = (1 + ah)y_k + hb$$

C'est une suite arithmético-géométrique de raison $q = 1 + ah$.

Si $|1 + ah| < 1$ (i.e. $-2 < ah < 0$ pour $a < 0$), la suite converge vers $y_\infty = -\frac{b}{a}$.

3.3 Sommes de Riemann (intégrales)

Définition | Principe des sommes de Riemann

On approche $\int_a^b f(x) dx$ en découpant $[a, b]$ en n sous-intervalles de largeur $h = \frac{b-a}{n}$:

Somme à gauche : $S_n^- = h \sum_{k=0}^{n-1} f(a + kh)$

Somme à droite : $S_n^+ = h \sum_{k=1}^n f(a + kh)$

Somme aux milieux : $S_n^m = h \sum_{k=0}^{n-1} f(a + (k + \frac{1}{2})h)$

Quand $n \rightarrow +\infty$: $S_n^- \rightarrow \int_a^b f(x) dx$ (et de même pour les autres).

Sommes de Riemann — 3 méthodes

```

1 def riemann_gauche(f, a, b, n):
2     h = (b - a) / n
3     return h * sum(f(a + k * h) for k in range(n))
4
5 def riemann_droite(f, a, b, n):
6     h = (b - a) / n
7     return h * sum(f(a + k * h) for k in range(1, n + 1))
8
9 def riemann_milieux(f, a, b, n):
10    h = (b - a) / n
11    return h * sum(f(a + (k + 0.5) * h) for k in range(n))

```

Exemple | Calculer $\int_0^1 e^{-x^2} dx$ (pas de primitive connue !)

```

1 import math
2 def f(x):
3     return math.exp(-x**2)
4
5 print(riemann_milieux(f, 0, 1, 10000))    #      0.74682
6 # Valeur exacte :      /2      erf(1)      0.74682...

```

C'est l'**intérêt majeur** de Riemann : calculer des intégrales dont on ne connaît pas de primitive.

Propriété | Précision

L'erreur des sommes à gauche/droite est en $O(\frac{1}{n})$. L'erreur de la méthode des milieux est en $O(\frac{1}{n^2})$: bien meilleure pour le même coût.

3.4 Simulation aléatoire et loi des grands nombres

Simuler une expérience aléatoire

```

1 import random
2
3 # Pile ou Face (Bernoulli de paramètre 0.5)
4 def bernoulli(p):
5     return 1 if random.random() < p else 0

```



```

6
7 # Simuler n lancers et calculer la fréquence
8 def frequence(p, n):
9     return sum(bernoulli(p) for _ in range(n)) / n
10
11 print(frequence(0.5, 10000)) # 0.50 (LGN)

```

Illustrer la loi des grands nombres

```

1 import random
2 import matplotlib.pyplot as plt
3
4 def loi_grands_nombres(p, n):
5     """Trace la convergence de la fréquence vers p."""
6     X = []
7     S = 0
8     for k in range(1, n + 1):
9         S += bernoulli(p)
10        X.append(S / k)
11    plt.plot(X, linewidth=0.8)
12    plt.axhline(y=p, color='red', linestyle='--')
13    plt.xlabel('n')
14    plt.ylabel('Fréquence')
15    plt.title('Loi des grands nombres')
16    plt.show()
17
18 loi_grands_nombres(0.3, 10000)

```

Simuler une loi binomiale

```

1 def binomiale(n, p):
2     """Simule  $X \sim B(n, p)$ ."""
3     return sum(bernoulli(p) for _ in range(n))
4
5 # Histogramme de 10000 simulations de  $B(20, 0.3)$ 
6 echantillon = [binomiale(20, 0.3) for _ in range(10000)]
7 plt.hist(echantillon, bins=range(22), density=True,
8         edgecolor='black', alpha=0.7)
9 plt.title('Histogramme  $B(20, 0.3)$ ')
10 plt.show()

```

3.5 Suites récurrentes : calcul de termes et recherche de seuil

Calculer les termes d'une suite $u_{n+1} = f(u_n)$

```

1 def termes_suite(f, u0, n):
2     """Retourne la liste  $[u_0, u_1, \dots, u_n]$ ."""
3     U = [u0]
4     u = u0
5     for k in range(n):
6         u = f(u)

```

```

7         U.append(u)
8     return U
9
10 # Exemple :  $u_{n+1} = (2 + u_n)$ ,  $u_0 = 0$ 
11 import math
12 U = termes_suite(lambda u: math.sqrt(2 + u), 0, 20)
13 print(U[-1]) # 2.0 (la suite converge vers 2)

```

Recherche de seuil : trouver n tel que $u_n > M$

```

1 def seuil(f, u0, M):
2     """Retourne le plus petit n tel que  $u_n > M$ ."""
3     u = u0
4     n = 0
5     while u <= M:
6         u = f(u)
7         n += 1
8     return n
9
10 # Exemple :  $u_{n+1} = 1.05 * u_n$  (croissance 5%),  $u_0 = 100$ 
11 # Combien d'années pour dépasser 200 ?
12 n = seuil(lambda u: 1.05 * u, 100, 200)
13 print(n) # 15 (il faut 15 ans pour doubler)

```

Attention | Boucle infinie !

Si la suite ne dépasse jamais M (par exemple suite décroissante), la boucle `while` ne s'arrête **jamais**. En pratique, on ajoute un compteur de sécurité :

```

1 def seuil_securise(f, u0, M, max_iter=100000):
2     u, n = u0, 0
3     while u <= M and n < max_iter:
4         u = f(u)
5         n += 1
6     return n if u > M else None

```

3.6 Recherche d'extremum et tri

3.7 Calcul de coefficients binomiaux et triangle de Pascal

Triangle de Pascal

```

1 def pascal(n):
2     """Retourne le triangle de Pascal jusqu'à la ligne n."""
3     T = [[1]]
4     for k in range(1, n + 1):
5         ligne = [1]
6         for j in range(1, k):
7             ligne.append(T[k-1][j-1] + T[k-1][j])
8         ligne.append(1)
9         T.append(ligne)

```

```

10     return T
11
12 # Coefficients binomiaux C(10, k)
13 T = pascal(10)
14 print(T[10])    # [1, 10, 45, 120, 210, 252, 210, 120, 45, 10, 1]

```

🔗 Coefficient binomial par formule directe

```

1 import math
2
3 def binom(n, k):
4     """C(n, k) = n! / (k! * (n-k)!"""
5     return math.factorial(n) // (math.factorial(k) * math.factorial(n
6         - k))
7
8 # Ou directement : math.comb(n, k) (Python 3.8+)
9 print(math.comb(10, 3))    # 120

```

3.8 Visualisation graphique

🔗 Tracer une fonction avec matplotlib

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x = np.linspace(-2, 4, 1000)
5 y = x**2 - 3*x + 1
6
7 plt.figure(figsize=(8, 5))
8 plt.plot(x, y, color='steelblue', linewidth=2, label=r'$f(x)=x^2-3x+1$')
9 plt.axhline(0, color='black', linewidth=0.5)
10 plt.axvline(0, color='black', linewidth=0.5)
11 plt.grid(alpha=0.3)
12 plt.legend(fontsize=12)
13 plt.xlabel('x')
14 plt.ylabel('f(x)')
15 plt.title('Graphe de f')
16 plt.show()

```

🔗 Visualiser la méthode d'Euler

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def euler_plot(f, t0, y0, h, n):
5     T, Y = euler(f, t0, y0, h, n)
6     plt.plot(T, Y, 'o-', markersize=2, label=f'Euler (h={h})')
7
8 # y' = -2y + 6, y(0) = 1

```

```

9  f = lambda t, y: -2*y + 6
10 euler_plot(f, 0, 1, 0.5, 10)      # pas grossier
11 euler_plot(f, 0, 1, 0.1, 50)     # pas moyen
12 euler_plot(f, 0, 1, 0.01, 500)   # pas fin
13
14 # Solution exacte
15 t = np.linspace(0, 5, 200)
16 plt.plot(t, -2*np.exp(-2*t) + 3, 'r--', linewidth=2, label='Exacte')
17 plt.legend()
18 plt.title("Méthode d'Euler : effet du pas h")
19 plt.show()

```

3.9 Résumé : quel algorithme pour quel problème ?

✓ Propriété | Tableau récapitulatif

Problème	Algorithme	Chapitre lié
Zéro de f	Dichotomie	Continuité, TVI
$\int_a^b f$	Sommes de Riemann	Calcul intégral
$y' = f(t, y)$	Méthode d'Euler	Éq. différentielles
Fréquence, estimation	Simulation	Probabilités, LGN
Termes de u_n	Boucle for	Suites
Seuil $u_n > M$	Boucle while	Suites, limites
Maximum de f	Balayage	Dérivées, variations
$\binom{n}{k}$	Triangle de Pascal	Dénombrement

4 Boîte à outils — Réflexes pour le bac

Méthode | Les 10 réflexes algorithmiques

1. **Lire un algorithme = exécuter à la main.** Faire un tableau de valeurs des variables étape par étape.
2. **for = nombre d'itérations connu, while = condition d'arrêt.**
3. **range(n) = 0, 1, ..., n - 1** (n termes). **range(1, n+1) = 1, ..., n.**
4. **Initialiser les variables** avant la boucle. Oublier l'initialisation est l'erreur #1.
5. **Accumulateur** : $S = 0$ puis $S += \text{terme}$ dans la boucle (pour sommer).
6. **Dichotomie** : vérifier $f(a) \cdot f(b) < 0$ avant de lancer. Sinon, pas de zéro garanti.
7. **Euler** : un petit pas h donne une meilleure approximation mais plus de calculs.
8. **Simulation** : toujours faire **beaucoup** de répétitions ($n \geq 1000$) pour la LGN.
9. **Indentation = structure.** Un décalage en trop ou en moins change tout le programme.
10. **Tester son code** avec des cas simples dont on connaît la réponse.

Attention | Top 6 des erreurs Python au bac

1. **Confusion = (affectation) et == (test d'égalité).**
2. **Oublier que range(a, b) s'arrête à b - 1.** D'où le +1 fréquent.
3. **Division entière // vs. division réelle /.** En Python 3, $7/2 = 3.5$ mais $7//2 = 3$.
4. **Oublier d'initialiser** le compteur ou l'accumulateur ($n = 0$, $S = 0$).
5. **Boucle while infinie** : la condition ne devient jamais fausse. Toujours vérifier la terminaison.
6. **Mauvaise indentation.** Le code dans une boucle ou un if doit être indenté de 4 espaces.

🔧 Méthode | Mots-clés à repérer dans les énoncés 🔑

Tu lis dans l'énoncé. . .	Tu penses à . . .
« compléter l'algorithme »	lire, comprendre la structure, trouver les lignes manquantes
« que retourne / affiche »	exécuter à la main (tableau de valeurs)
« écrire un programme »	identifier : <code>for</code> ou <code>while</code> , quelle formule
« valeur approchée d'un zéro »	dichotomie
« approximation de $\int_a^b f$ »	sommes de Riemann
« approcher la solution de $y' = \dots$ »	méthode d'Euler
« simuler / fréquence »	<code>random</code> , boucle, compteur
« déterminer n tel que »	boucle <code>while</code> (recherche de seuil)
« afficher les termes »	boucle <code>for</code> , suite récurrente

🔧 Méthode | Exécuter un algorithme à la main (méthode du tableau)

Pour comprendre ce que fait un algorithme, on trace un **tableau d'évolution des variables** :

Étape	k	S	u	condition
Init	–	0	1	–
$k = 0$	0	1	2	–
$k = 1$	1	3	4	–
$k = 2$	2	7	8	–
\vdots				

C'est la méthode la plus sûre pour répondre aux questions « que vaut S après la boucle ? ».

5 Exercices

Exercice 1 — Lire un algorithme

Que retourne la fonction suivante pour $n = 5$?

```
1 def mystere(n):
2     S = 0
3     for k in range(1, n + 1):
4         S = S + k**2
5     return S
```

Exercice 2 — Compléter un algorithme

Compléter le programme pour qu'il calcule $n!$ (factorielle de n) :

```
1 def factorielle(n):
2     F = ... # à compléter
3     for k in range(..., ...): # à compléter
4         F = ... # à compléter
5     return F
```

Exercice 3 — Suite récurrente

On définit $u_0 = 1$ et $u_{n+1} = \frac{u_n + 3}{2}$.

- Écrire un programme qui affiche les 20 premiers termes.
- Vers quelle valeur semble converger la suite ?
- Écrire un programme qui détermine le plus petit n tel que $|u_n - 3| < 10^{-6}$.

Exercice 4 — Dichotomie

- Utiliser la dichotomie pour résoudre $e^x = 3x$ sur $[1, 2]$ à 10^{-8} près.
- Combien d'étapes sont nécessaires ?
- Adapter pour trouver $\sqrt[3]{7}$ (zéro de $x^3 - 7$ sur $[1, 2]$).

Exercice 5 — Méthode d'Euler

- Appliquer Euler à $y' = y$, $y(0) = 1$ avec $h = 0,1$ et $n = 10$ (donc $t \in [0, 1]$). Comparer y_{10} à e .
- Même question avec $h = 0,01$ et $n = 100$. L'approximation est-elle meilleure ?
- Appliquer Euler à $y' = -y + \sin(t)$, $y(0) = 0$ avec $h = 0,01$ sur $[0, 10]$. Tracer la solution.

Exercice 6 — Sommes de Riemann

- Calculer $\int_0^1 x^3 dx$ par les sommes à gauche avec $n = 100, 1000, 10000$. Comparer à la valeur exacte $\frac{1}{4}$.
- Calculer $\int_0^\pi \sin(x) dx$ par la méthode des milieux. Comparer à 2.
- Estimer $\int_0^1 \frac{1}{1+x^2} dx$ et vérifier que c'est $\approx \frac{\pi}{4}$.

Exercice 7 — Simulation de probabilités

- Simuler 10 000 lancers de deux dés et estimer la probabilité d'obtenir une somme égale à 7.
- Simuler le paradoxe des anniversaires : pour n personnes, estimer par simulation la probabilité d'une coïncidence.
- Simuler la marche aléatoire : un pion part de 0 et avance de $+1$ ou -1 (équiprobable) à chaque pas. Tracer 5 trajectoires sur 1000 pas.

Exercice 8 ★★☆☆ — Lecture et compréhension

Que calcule la fonction suivante ?

```

1 import math
2 def f(x):
3     return math.log(x) - 2
4
5 def algo(eps):
6     a, b = 1, 100
7     while b - a > eps:
8         m = (a + b) / 2
9         if f(a) * f(m) <= 0:
10            b = m
11        else:
12            a = m
13    return (a + b) / 2

```

- Quelle équation résout cet algorithme ?
- Que retourne `algo(0.001)` ? Donner la valeur exacte.

Exercice 9 ★★☆☆ — Seuil et suite géométrique

On place 1000€ à un taux annuel de 3% (intérêts composés).

- Écrire la suite (u_n) du capital après n années.
- Écrire un programme qui détermine le nombre d'années pour doubler le capital.
- Vérifier avec la formule $n = \frac{\ln 2}{\ln 1,03} \approx 23,4$.

Exercice 10 ★★☆☆ — Exécuter à la main

Exécuter à la main l'algorithme suivant et donner la valeur de S en sortie :

```

1 S = 0
2 u = 1
3 for k in range(5):
4     S = S + u
5     u = u * 2
6 print(S)

```

Exercice 11 ★★☆☆ — Euler et équation différentielle

On considère l'équation $y' = -0,5y + 2$, $y(0) = 10$.

- Déterminer la solution exacte.
- Appliquer Euler avec $h = 0,5$ et faire les 4 premières étapes à la main.
- Comparer avec les valeurs exactes en $t = 0,5, 1, 1,5, 2$.

Exercice 12 ★★☆☆ — Intervalle de confiance par simulation

On ne connaît pas p (probabilité de succès d'un événement). On effectue $n = 1000$ essais et on observe $f_n = 0,62$.

- Donner un intervalle de confiance théorique pour p au seuil 95% (Tchebychev).
- Écrire un programme qui simule 10 000 échantillons de taille 1000 avec $p = 0,6$ et vérifie que la fréquence tombe dans l'intervalle $[p - \frac{1}{\sqrt{20n}}; p + \frac{1}{\sqrt{20n}}]$ environ 95% du temps.

Exercice 13 ★★ ★ — **Méthode de Monte-Carlo**

Pour estimer π : on tire aléatoirement n points (x, y) dans le carré $[0, 1]^2$ et on compte ceux qui tombent dans le quart de disque $x^2 + y^2 \leq 1$.

- a) Expliquer pourquoi la fréquence des points dans le disque converge vers $\frac{\pi}{4}$.
- b) Écrire le programme. Estimer π pour $n = 10^4$ et $n = 10^6$.
- c) Quelle précision attend-on pour $n = 10^6$? (Utiliser $\sigma(\bar{X}_n) \approx \frac{1}{2\sqrt{n}}$.)

Exercice 14 ★★ ★ — **Comparer Euler et solution exacte**

On considère $y' = \cos(t) - y$, $y(0) = 0$.

- a) La solution exacte est $y(t) = \frac{1}{2}(\sin t + \cos t - e^{-t})$. Vérifier en dérivant.
- b) Programmer Euler avec $h = 0,1$, $h = 0,01$, $h = 0,001$ sur $[0, 10]$.
- c) Tracer sur le même graphique les 3 approximations et la solution exacte.
- d) Mesurer l'erreur maximale pour chaque h . Vérifier qu'elle est proportionnelle à h .

6 🐼 Problème — La méthode de Newton ★★

🔥 Problème style prépa

La méthode de Newton est une alternative à la dichotomie pour trouver les zéros d'une fonction. Elle est beaucoup plus rapide mais nécessite de connaître la dérivée.

Soit $f : \mathbb{R} \rightarrow \mathbb{R}$ dérivable, de dérivée f' . On cherche α tel que $f(\alpha) = 0$.

Partie A — Le principe géométrique

1. Écrire l'équation de la tangente à la courbe de f au point d'abscisse x_n .
2. Cette tangente coupe l'axe des abscisses en un point x_{n+1} . Montrer que :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3. Illustrer géométriquement : à chaque étape, on remplace la courbe par sa tangente et on prend le zéro de cette tangente comme nouvelle approximation.

Partie B — Programmation

4. Écrire une fonction Python `newton(f, df, x0, eps, max_iter)` qui implémente la méthode de Newton.
5. Appliquer à $f(x) = x^2 - 2$ (donc $f'(x) = 2x$) avec $x_0 = 1$. Combien d'itérations pour atteindre 10^{-15} ?
6. Comparer avec la dichotomie : combien d'itérations faut-il en dichotomie pour la même précision ?

Partie C — Convergence

7. On note $e_n = x_n - \alpha$ l'erreur à l'étape n . En utilisant un développement de Taylor de f autour de α , montrer que :

$$e_{n+1} \approx \frac{f''(\alpha)}{2f'(\alpha)} \cdot e_n^2$$

8. Que signifie $e_{n+1} \approx C \cdot e_n^2$? Expliquer pourquoi Newton est **quadratiquement** convergente : le nombre de décimales correctes **double** à chaque étape.
9. Pour $f(x) = x^2 - 2$ et $x_0 = 1$, calculer x_1, x_2, x_3, x_4 et compter les décimales correctes de $\sqrt{2}$.

Partie D — Limites de la méthode

10. Que se passe-t-il si $f'(x_n) = 0$? Donner un exemple concret.
11. Tester Newton sur $f(x) = x^3 - 2x + 2$ avec $x_0 = 0$. Que constate-t-on ? (La méthode peut ne pas converger si x_0 est mal choisi.)
12. Proposer une stratégie hybride : utiliser la dichotomie pour s'approcher du zéro, puis Newton pour affiner.

7 Corrigés détaillés

Exercice 1

On trace le tableau d'exécution pour $n = 5$:

k	k^2	S
—	—	0
1	1	1
2	4	5
3	9	14
4	16	30
5	25	55

La fonction retourne $S = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = \boxed{55}$.

En général, `mystere(n)` retourne $\sum_{k=1}^n k^2 = \frac{n(n+1)(2n+1)}{6}$.

Exercice 2

```
1 def factorielle(n):
2     F = 1
3     for k in range(1, n + 1):
4         F = F * k
5     return F
```

F commence à 1 (élément neutre de la multiplication) et k va de 1 à n .

Exercice 3

a) et b) :

```
1 u = 1
2 for n in range(20):
3     print(f"u_{n} = {u:.10f}")
4     u = (u + 3) / 2
```

On observe que $u_n \rightarrow 3$. Vérification : le point fixe de $f(x) = \frac{x+3}{2}$ est $\ell = \frac{\ell+3}{2} \Rightarrow \ell = 3$.

c) :

```
1 u = 1
2 n = 0
3 while abs(u - 3) >= 1e-6:
4     u = (u + 3) / 2
5     n += 1
6 print(n)    # 21
```

Il faut $n = 21$ étapes. En effet, $u_n - 3 = -2 \cdot \left(\frac{1}{2}\right)^n$, donc $|u_n - 3| < 10^{-6} \iff 2 \cdot 2^{-n} < 10^{-6} \iff n > \frac{\ln(2 \times 10^6)}{\ln 2} \approx 20,9$.

Exercice 4

a) On cherche le zéro de $f(x) = e^x - 3x$ sur $[1, 2]$.

$f(1) = e - 3 \approx -0,282 < 0$ et $f(2) = e^2 - 6 \approx 1,389 > 0$.

```
1 import math
```

```

2 def f(x):
3     return math.exp(x) - 3*x
4
5 resultat = dichotomie(f, 1, 2, 1e-8)
6 print(resultat)      #      1.51213455

```

b) Nombre d'étapes : $n \geq \frac{\ln(1/10^{-8})}{\ln 2} = \frac{8 \ln 10}{\ln 2} \approx 26,6$, donc 27 étapes.

c) Zéro de $g(x) = x^3 - 7$ sur $[1, 2]$: $g(1) = -6 < 0$, $g(2) = 1 > 0$.

```

1 resultat = dichotomie(lambda x: x**3 - 7, 1, 2, 1e-8)
2 print(resultat)      #      1.91293118

```

Exercice 5

a) $y' = y$, $y(0) = 1$, solution exacte $y(t) = e^t$.

Euler avec $h = 0,1$: $y_{k+1} = y_k + 0,1 \cdot y_k = 1,1 \cdot y_k$. Donc $y_k = 1,1^k$.

$y_{10} = 1,1^{10} \approx 2,5937$. Valeur exacte : $e \approx 2,7183$. Erreur $\approx 4,6\%$.

b) Avec $h = 0,01$: $y_{100} = 1,01^{100} \approx 2,7048$. Erreur $\approx 0,5\%$. Oui, bien meilleure.

c)

```

1 import math
2 T, Y = euler(lambda t, y: -y + math.sin(t), 0, 0, 0.01, 1000)
3 plt.plot(T, Y)
4 plt.title("y' = -y + sin(t)")
5 plt.show()

```

Exercice 6

a) $\int_0^1 x^3 dx = \frac{1}{4} = 0,25$.

n	Somme à gauche	Erreur
100	0,245025	5×10^{-3}
1000	0,249500	5×10^{-4}
10000	0,249950	5×10^{-5}

L'erreur diminue comme $\frac{1}{n}$ (comme prévu).

b) $\int_0^\pi \sin(x) dx = 2$. Milieux avec $n = 10000$: $\approx 2,0000000$ (erreur $< 10^{-8}$). La méthode des milieux est bien meilleure (erreur en $\frac{1}{n^2}$).

c) $\int_0^1 \frac{1}{1+x^2} dx = \arctan(1) = \frac{\pi}{4} \approx 0,7854$. Vérifié.

Exercice 7

a)

```

1 import random
2 compteur = 0
3 N = 10000
4 for _ in range(N):
5     d1 = random.randint(1, 6)
6     d2 = random.randint(1, 6)
7     if d1 + d2 == 7:
8         compteur += 1

```

```
9 print(compteur / N)      #      0.167 (théorique : 6/36 = 1/6)
```

b)

```
1 def anniversaire_simulation(n, N=10000):
2     compteur = 0
3     for _ in range(N):
4         dates = set()
5         collision = False
6         for _ in range(n):
7             d = random.randint(1, 365)
8             if d in dates:
9                 collision = True
10                break
11            dates.add(d)
12        if collision:
13            compteur += 1
14    return compteur / N
15
16 print(anniversaire_simulation(23))      #      0.507
```

c)

```
1 for _ in range(5):
2     positions = [0]
3     x = 0
4     for _ in range(1000):
5         x += random.choice([-1, 1])
6         positions.append(x)
7     plt.plot(positions, linewidth=0.5)
8 plt.title("5 marches aléatoires")
9 plt.show()
```

Exercice 8

a) $f(x) = \ln(x) - 2$. L'algorithme cherche le zéro de f , c'est-à-dire la solution de $\ln(x) = 2$, soit $x = e^2$.

b) `algo(0.001)` retourne une valeur approchée de $e^2 \approx 7,389$, à 0,001 près.

Exercice 9

a) $u_0 = 1000$ et $u_{n+1} = 1,03 \cdot u_n$. Donc $u_n = 1000 \times 1,03^n$.

b)

```
1 u = 1000
2 n = 0
3 while u <= 2000:
4     u = 1.03 * u
5     n += 1
6 print(n)      # 24
```

Il faut 24 années pour que le capital dépasse 2000€.

c) $1,03^n > 2 \iff n > \frac{\ln 2}{\ln 1,03} \approx 23,45$. Donc $n = 24$, cohérent.

Exercice 10

Tableau d'exécution :

k	S (avant)	u (avant)	S (après)
0	0	1	1
1	1	2	3
2	3	4	7
3	7	8	15
4	15	16	31

$$S = 1 + 2 + 4 + 8 + 16 = \boxed{31} = 2^5 - 1.$$

En général, cet algorithme calcule $\sum_{k=0}^{n-1} 2^k = 2^n - 1$.

Exercice 11

a) $y' = -0,5y + 2$ est de la forme $y' = ay + b$ avec $a = -0,5$, $b = 2$. Solution générale : $y(t) = Ce^{-0,5t} + 4$. Avec $y(0) = 10$: $C = 6$, donc $y(t) = 6e^{-0,5t} + 4$.

b) Euler avec $h = 0,5$:

$$y_0 = 10.$$

$$y_1 = 10 + 0,5 \times (-0,5 \times 10 + 2) = 10 + 0,5 \times (-3) = 8,5.$$

$$y_2 = 8,5 + 0,5 \times (-0,5 \times 8,5 + 2) = 8,5 + 0,5 \times (-2,25) = 7,375.$$

$$y_3 = 7,375 + 0,5 \times (-0,5 \times 7,375 + 2) = 7,375 + 0,5 \times (-1,6875) = 6,531.$$

$$y_4 = 6,531 + 0,5 \times (-0,5 \times 6,531 + 2) = 6,531 + 0,5 \times (-1,266) = 5,898.$$

c) Comparaison :

t	Euler ($h = 0,5$)	Exacte	Erreur
0,5	8,500	8,680	2,1%
1,0	7,375	7,639	3,5%
1,5	6,531	6,804	4,0%
2,0	5,898	6,207	5,0%

Exercice 12

a) Tchebychev : $p \in [f_n - \frac{1}{\sqrt{20n}}; f_n + \frac{1}{\sqrt{20n}}]$.

$$\text{Avec } n = 1000 : \frac{1}{\sqrt{20 \times 1000}} = \frac{1}{\sqrt{20000}} \approx 0,00707.$$

$$\text{IC 95\% : } p \in [0,62 - 0,007; 0,62 + 0,007] = [0,613; 0,627].$$

b)

```

1 import random
2 p = 0.6
3 n = 1000
4 marge = 1 / (20 * n)**0.5
5 compteur = 0
6 for _ in range(10000):
7     fn = sum(1 for _ in range(n) if random.random() < p) / n
8     if p - marge <= fn <= p + marge:
9         compteur += 1
10 print(compteur / 10000)      #      0.65 (bien au-dessus de 95%)

```

La borne de Tchebychev est **très pessimiste** : l'intervalle est plus fiable que 95% en pratique (par le TCL, c'est plutôt $\approx 99,8\%$).

Exercice 13

a) L'aire du quart de disque est $\frac{\pi}{4}$, l'aire du carré est 1. Un point aléatoire uniforme dans $[0, 1]^2$ tombe dans le quart de disque avec probabilité $\frac{\pi}{4}$. Par la LGN, la fréquence converge vers $\frac{\pi}{4}$.

b)

```

1 import random
2 def monte_carlo_pi(n):
3     compteur = 0
4     for _ in range(n):
5         x = random.random()
6         y = random.random()
7         if x**2 + y**2 <= 1:
8             compteur += 1
9     return 4 * compteur / n
10
11 print(monte_carlo_pi(10**4))    #      3.14 (2 décimales)
12 print(monte_carlo_pi(10**6))    #      3.1416 (3-4 décimales)

```

c) $\sigma(\bar{X}_n) \approx \frac{1}{2\sqrt{n}}$. Pour $n = 10^6$: $\sigma \approx \frac{1}{2000} = 0,0005$. L'erreur sur $\pi \approx 4\sigma \approx 0,002$, soit ≈ 3 décimales. Monte-Carlo est simple mais lent (précision en $\frac{1}{\sqrt{n}}$).

Exercice 14

a) $y(t) = \frac{1}{2}(\sin t + \cos t - e^{-t})$.

$y'(t) = \frac{1}{2}(\cos t - \sin t + e^{-t})$.

$\cos t - y(t) = \cos t - \frac{1}{2}\sin t - \frac{1}{2}\cos t + \frac{1}{2}e^{-t} = \frac{1}{2}\cos t - \frac{1}{2}\sin t + \frac{1}{2}e^{-t} = y'(t)$. ✓

$y(0) = \frac{1}{2}(0 + 1 - 1) = 0$. ✓

b)-c)

```

1 import math, numpy as np, matplotlib.pyplot as plt
2
3 for h in [0.1, 0.01, 0.001]:
4     n = int(10 / h)
5     T, Y = euler(lambda t, y: math.cos(t) - y, 0, 0, h, n)
6     plt.plot(T, Y, label=f'h={h}')
7
8 t = np.linspace(0, 10, 1000)
9 y_exact = 0.5 * (np.sin(t) + np.cos(t) - np.exp(-t))
10 plt.plot(t, y_exact, 'k--', linewidth=2, label='Exacte')
11 plt.legend()
12 plt.show()

```

d) Erreur maximale :

h	Erreur max
0,1	$\approx 0,048$
0,01	$\approx 0,0048$
0,001	$\approx 0,00048$

L'erreur est bien proportionnelle à h : diviser h par 10 divise l'erreur par 10. C'est la convergence d'ordre 1 d'Euler.

Corrigé du problème — La méthode de Newton

Partie A — Le principe géométrique

1. Tangente en $(x_n, f(x_n))$: $y = f(x_n) + f'(x_n)(x - x_n)$.

2. Cette tangente coupe l'axe $y = 0$:

$$0 = f(x_n) + f'(x_n)(x_{n+1} - x_n) \implies x_{n+1} - x_n = -\frac{f(x_n)}{f'(x_n)}$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

3. Géométriquement : on suit la tangente jusqu'à l'axe des abscisses. Si f est convexe et le point initial bien choisi, les approximations convergent rapidement vers le zéro.

Partie B — Programmation

4.

```
1 def newton(f, df, x0, eps, max_iter=100):
2     x = x0
3     for i in range(max_iter):
4         fx = f(x)
5         if abs(fx) < eps:
6             return x, i
7         dfx = df(x)
8         if dfx == 0:
9             return None, i    # tangente horizontale
10        x = x - fx / dfx
11    return x, max_iter
```

5. $f(x) = x^2 - 2$, $f'(x) = 2x$, $x_0 = 1$:

```
1 resultat, nb = newton(lambda x: x**2 - 2, lambda x: 2*x, 1, 1e-15)
2 print(f"x = {resultat}, itérations = {nb}")
3 # x = 1.4142135623730951, itérations = 5
```

Seulement **5 itérations** pour 15 décimales correctes !

6. En dichotomie sur $[1, 2]$: $n \geq \frac{\ln(1/10^{-15})}{\ln 2} \approx 50$ itérations.

Rapport : Newton 5 vs. dichotomie 50. Newton est environ 10 fois plus rapide.

Partie C — Convergence

7. Taylor de f autour de α ($f(\alpha) = 0$) :

$$f(x_n) = f(\alpha) + f'(\alpha)e_n + \frac{f''(\alpha)}{2}e_n^2 + \dots = f'(\alpha)e_n + \frac{f''(\alpha)}{2}e_n^2 + \dots$$

$$f'(x_n) = f'(\alpha) + f''(\alpha)e_n + \dots \approx f'(\alpha) \text{ (au premier ordre).}$$

$$e_{n+1} = x_{n+1} - \alpha = x_n - \frac{f(x_n)}{f'(x_n)} - \alpha = e_n - \frac{f'(\alpha)e_n + \frac{f''(\alpha)}{2}e_n^2}{f'(\alpha) + \dots}$$

$$\approx e_n - e_n - \frac{f''(\alpha)}{2f'(\alpha)}e_n^2 = \frac{f''(\alpha)}{2f'(\alpha)} \cdot e_n^2.$$

□

8. $e_{n+1} \approx C \cdot e_n^2$ signifie que l'erreur est **mise au carré** à chaque étape. Si $e_n \approx 10^{-k}$, alors $e_{n+1} \approx C \cdot 10^{-2k}$. Le nombre de décimales correctes **double** à chaque itération :

1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 décimales en 5 itérations.

9. Pour $f(x) = x^2 - 2$, $f'(x) = 2x$:

$$x_{n+1} = x_n - \frac{x_n^2 - 2}{2x_n} = \frac{x_n^2 + 2}{2x_n} = \frac{1}{2} \left(x_n + \frac{2}{x_n} \right)$$

(C'est la méthode de Héron / babylonienne !)

$$x_0 = 1. \quad x_1 = \frac{1}{2}(1 + 2) = 1,5. \quad x_2 = \frac{1}{2} \left(1,5 + \frac{2}{1,5} \right) = \frac{1}{2} \times \frac{17}{6} \approx 1,416667.$$

$$x_3 \approx 1,414215686. \quad x_4 \approx 1,414213562373095.$$

$\sqrt{2} \approx 1,414213562373099$. Après 4 itérations : 13 décimales correctes.

Partie D — Limites de la méthode

10. Si $f'(x_n) = 0$, la tangente est horizontale et ne coupe pas l'axe des x : division par zéro. Exemple : $f(x) = x^2$ près de 0, $f'(0) = 0$.

11. $f(x) = x^3 - 2x + 2$, $x_0 = 0$. $f(0) = 2$, $f'(0) = -2$.

$$x_1 = 0 - \frac{2}{-2} = 1. \quad f(1) = 1, \quad f'(1) = 1.$$

$x_2 = 1 - \frac{1}{1} = 0$. On revient à $x_0 = 0$! La méthode **oscille** indéfiniment et ne converge pas. (Le zéro réel est $\approx -1,77$, trop loin de x_0 .)

12. Stratégie hybride :

```
1 def newton_hybride(f, df, a, b, eps):
2     """Dichotomie pour se rapprocher, puis Newton pour affiner."""
3     # Phase 1 : dichotomie grossière
4     x = dichotomie(f, a, b, 0.1)
5     # Phase 2 : Newton pour la précision
6     resultat, _ = newton(f, df, x, eps)
7     return resultat
```

La dichotomie garantit la convergence (robustesse), Newton accélère la précision (rapidité). C'est la stratégie utilisée dans les logiciels professionnels.

Fin de la fiche — Algorithmique & Python

Tu maîtrises maintenant :

- Les bases Python : variables, boucles for/while, fonctions, listes.
 - La dichotomie (zéro de fonction, TVI).
 - La méthode d'Euler (équations différentielles).
 - Les sommes de Riemann (calcul intégral).
 - La simulation aléatoire (probabilités, LGN).
- Les suites récurrentes (termes, seuils, convergence).
- La lecture et l'exécution à la main d'un algorithme.

L'algorithme est au mathématicien ce que le télescope est à l'astronome.

✔ Toutes les fiches sont complètes. Bon courage pour le bac !

