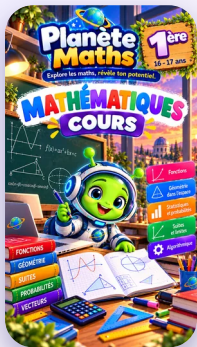


Algorithmique et Python : cours de maths 1ère à télécharger en PDF



Le livre de cours de maths 1ère

Tout le programme en un seul PDF de 103 pages, à imprimer ou à garder sur ta tablette.

[Voir le livre](#)

Un programme, c'est une recette écrite avec une précision de chirurgien : l'ordinateur obéit sans jamais deviner. En 1ère, tu vas utiliser **Python** pour faire travailler les mathématiques à ta place : calculer cent termes d'une suite en une seconde, trouver à partir de quand une quantité dépasse un seuil, encadrer la solution d'une équation, simuler le hasard ou approcher un nombre dérivé. Ce chapitre te donne les outils et surtout la méthode pour **lire**, **écrire** et **vérifier** un programme.

1. Variables, conditions et boucles

D Variable et affectation

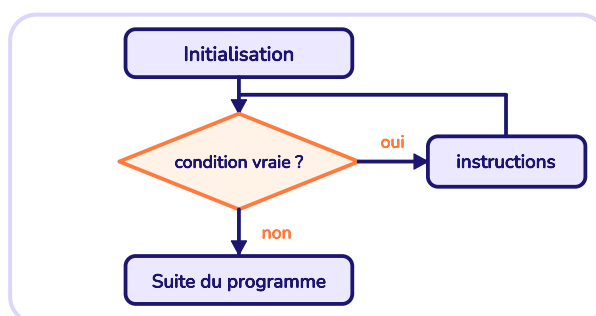
Une variable est un nom qui désigne une valeur stockée en mémoire. L'instruction `a = 5` range la valeur 5 dans la variable a. Le signe `=` est une **affectation**, pas une égalité mathématique : `a = a + 1` a du sens en Python, elle augmente a de 1.

Pour comparer, on utilise `==` (égal), `!=` (différent), `<`, `<=`, `>`, `>=`. Une **condition** choisit les instructions à exécuter, et l'**indentation** (le décalage vers la droite) indique ce qui appartient au bloc :

```
if t < 0:
    message = "gel"
elif t <= 25:
    message = "doux"
else:
    message = "chaud"
```

Pour répéter des instructions, il existe deux boucles.

- La boucle **for** s'emploie quand on connaît le nombre de répétitions : `for k in range(n)` donne à k les valeurs `0, 1, ..., n - 1` ; `range(a, b)` donne `a, ..., b - 1` ; `range(a, b, p)` avance de p en p. Attention : la borne de fin n'est **jamais** atteinte.
- La boucle **while** s'emploie quand on ne sait pas à l'avance combien de fois on répète : elle continue **tant que** la condition est vraie.





Piège

Dans une boucle **while**, il faut que quelque chose change à chaque tour pour que la condition finisse par devenir fausse. Sinon, le programme tourne sans jamais s'arrêter : c'est une **boucle infinie**.

2. Les fonctions Python

Une fonction regroupe des instructions sous un nom et peut recevoir des **paramètres**. Le mot **return** renvoie le résultat ; il termine aussi la fonction. Une fonction Python ressemble beaucoup à une fonction mathématique.

```
def f(x):  
    return 3*x**2 - 4
```

```
print(f(2))
```

Ce programme définit $f(x) = 3x^2 - 4$ puis affiche $f(2) = 12 - 4 = 8$. En Python, la puissance s'écrit ******, le produit ***** (jamais omis), la division **/**. Les nombres décimaux utilisent le **point** : 0.5 et non 0,5.



Méthode : écrire une fonction

1. Choisis un nom parlant et la liste des paramètres.
2. Écris les calculs, avec des variables intermédiaires si besoin.
3. Termine par **return** et teste la fonction sur une valeur dont tu connais le résultat.

3. Les listes

D Liste

Une liste est une suite ordonnée de valeurs écrites entre crochets, comme $L = [4, 9, 1, 7]$. Les positions (indices) commencent à 0 : $L[0]$ vaut 4 et $L[2]$ vaut 1. On trouve le nombre d'éléments avec `len(L)`, on ajoute en fin de liste avec `L.append(10)`, et on peut utiliser `sum(L)`, `max(L)`, `min(L)`.

On parcourt une liste avec `for x in L`, et on en fabrique une avec une **compréhension**. Par exemple `[k**2 for k in range(1, 5)]` donne la liste `[1, 4, 9, 16]`. Les listes servent à conserver les termes d'une suite pour ensuite les afficher ou les tracer.

4. Calculer des termes et des sommes d'une suite

Pour une suite définie par récurrence $u_{n+1} = f(u_n)$, on part de u_0 et on applique n fois la relation : c'est exactement le rôle d'une boucle **for**. Pour une somme, on ajoute chaque terme à un compteur initialisé à zéro (ou au premier terme).

E**Exemple 1 : termes et somme**

Soit (u_n) définie par $u_0 = 5$ et $u_{n+1} = 2u_n - 3$. On veut u_5 et $S = u_0 + u_1 + \dots + u_5$.

```
def termes(n):
```

```
    u = 5
```

```
    S = 5
```

```
    for k in range(n):
```

```
        u = 2*u - 3
```

```
        S = S + u
```

```
    return u, S
```

Les termes successifs sont **5, 7, 11, 19, 35, 67**. L'appel **termes(5)** renvoie donc $u_5 = 67$ et $S = 5 + 7 + 11 + 19 + 35 + 67 = 144$.

5. Chercher un seuil avec while

Quand on cherche le **premier rang** à partir duquel une suite dépasse une valeur, on ne connaît pas le nombre d'étapes : on utilise **while**. La condition de la boucle exprime que le seuil n'est **pas encore** atteint, et on compte les tours.

E**Exemple 2 : un capital qui grandit**

Un capital de 2000 € augmente de 3 % par an. À partir de quelle année dépasse-t-il 3000 € ?

```
def seuil():
```

```
    n = 0
```

```
    capital = 2000
```

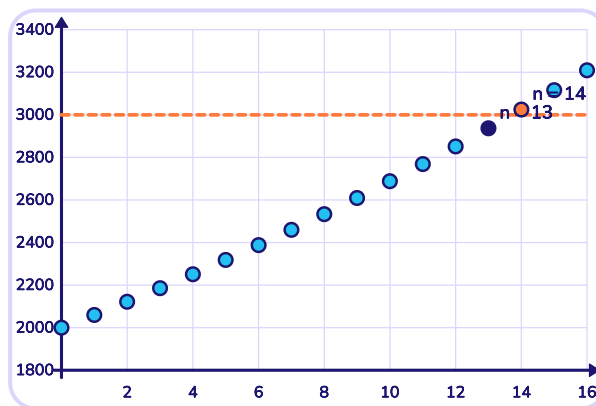
```
    while capital <= 3000:
```

```
        capital = capital * 1.03
```

```
        n = n + 1
```

```
    return n
```

Le programme renvoie 14. En effet $2000 \times 1,03^{13} \approx 2937$ reste inférieur à 3000, alors que $2000 \times 1,03^{14} \approx 3025$ le dépasse.

**!****Astuce de Zyro**

Sur ma planète, on dit qu'un programme se teste toujours deux fois : une fois sur un cas dont on connaît la réponse, et une fois sur le vrai problème. Fais comme moi, tu éviteras bien des pannes !

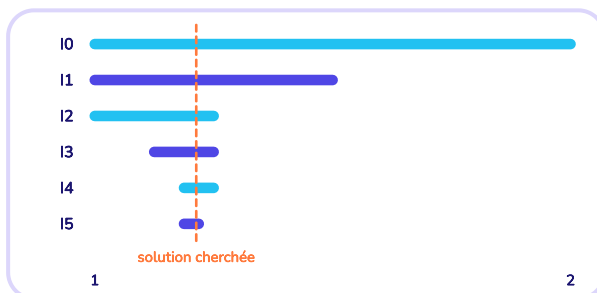
6. La méthode de dichotomie

P

Principe de la dichotomie

Soit f une fonction continue et strictement monotone sur $[a; b]$ telle que $f(a)$ et $f(b)$ soient de signes contraires. Alors l'équation $f(x) = 0$ admet une **unique** solution α dans $[a; b]$ (résultat admis). On l'encadre en coupant l'intervalle en deux : si $f(a)$ et $f(m)$ sont de signes contraires, $\alpha \in [a; m]$, sinon $\alpha \in [m; b]$, où $m = \frac{a+b}{2}$.

À chaque étape, l'amplitude de l'intervalle est **divisée par 2** : après n étapes elle vaut $\frac{b-a}{2^n}$.



E**Exemple 3 : résoudre $x^3 + x = 3$**

Soit $f(x) = x^3 + x - 3$ sur $[1; 2]$. On a $f(1) = -1 < 0$ et $f(2) = 7 > 0$, et f est strictement croissante, donc $f(x) = 0$ a une unique solution α dans $[1; 2]$.

```
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
```

Avec $e = 0,01$, les milieux successifs sont

1,5 ; 1,25 ; 1,125 ; 1,1875 ; 1,21875 ; 1,203125 ; 1,2109375. On obtient l'encadrement $1,2109 < \alpha < 1,2188$, d'amplitude 0,0078.

7. Simuler une loi binomiale

La fonction **random()**, importée avec **from random import random**, renvoie un nombre « au hasard » dans l'intervalle $[0; 1[$. La probabilité que **random()** < **p** est donc égale à **p** : c'est un moyen de simuler une épreuve de Bernoulli de paramètre **p**.

E**Exemple 4 : compter les succès**

Pour simuler une variable aléatoire X de loi binomiale $\mathcal{B}(10; 0,3)$, on répète 10 épreuves indépendantes et on compte les succès :

```
from random import random
```

```
def simulation(n, p):
```

```
    succes = 0
```

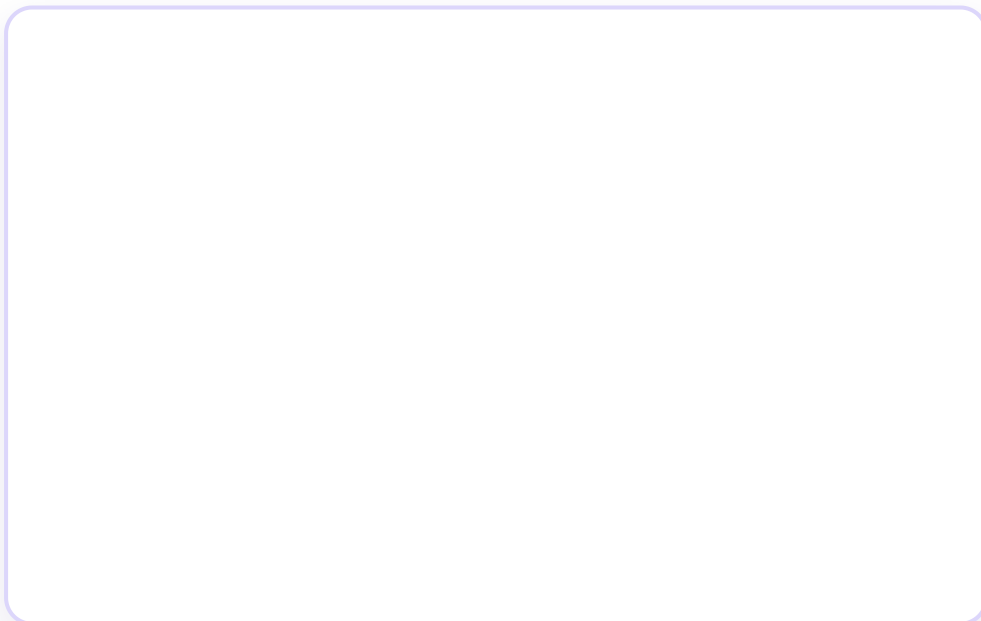
```
    for i in range(n):
```

```
        if random() < p:
```

```
            succes = succes + 1
```

```
    return succes
```

On appelle **simulation(10, 0.3)** par exemple 2000 fois et on calcule la fréquence de chaque valeur. D'après la loi binomiale, $P(X = 3) = \binom{10}{3} \times 0,3^3 \times 0,7^7 \approx 0,267$: la fréquence observée de la valeur 3 est proche de ce nombre, sans lui être exactement égale.



Algorithmique Python 1ère : diagramme en barres comparant les fréquences obtenues par simulation de 2000 essais et les probabilités exactes de la loi binomiale de paramètres 10 et 0,3



Piège

Une simulation donne un résultat **différent à chaque exécution**. Elle fournit une valeur approchée, jamais une valeur exacte, et sa précision augmente avec le nombre de répétitions.

8. Approcher un nombre dérivé

Le nombre dérivé de f en a est la limite du taux d'accroissement $\frac{f(a+h) - f(a)}{h}$ quand h tend vers 0. Pour l'approcher, on calcule ce taux avec un h très petit :

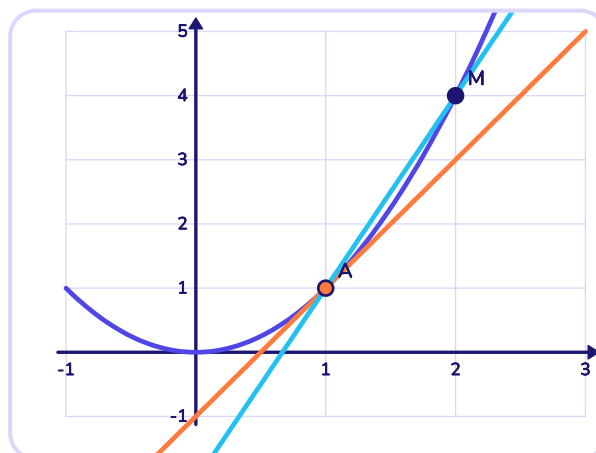
```
def taux(f, a, h):  
    return (f(a + h) - f(a)) / h
```

Ici f est elle-même un paramètre : on peut transmettre une fonction à une autre fonction.



Exemple 5 : nombre dérivé en 2

Soit $f(x) = x^2 + 3x$. Alors $f(2) = 10$ et $\frac{f(2+h) - f(2)}{h} = \frac{7h + h^2}{h} = 7 + h$.
Pour $h = 0,001$, le programme affiche environ 7,001. On conjecture $f'(2) = 7$, ce que le calcul confirme : $7 + h$ tend vers 7.



Sur la figure, la sécante (AM) a pour pente le taux d'accroissement ; quand M se rapproche de A , elle se confond avec la tangente.

À retenir

- Le signe `=` affecte une valeur ; on teste l'égalité avec `==`. L'indentation délimite les blocs.
- **for** `k in range(n)` répète n fois (k de 0 à $n-1$) ; **while** répète tant que la condition est vraie.
- Une fonction se définit avec **def** et renvoie son résultat avec **return** ; les listes s'indexent à partir de 0.
- Termes et sommes d'une suite : boucle **for** avec un accumulateur ; seuil : boucle **while** avec un compteur.
- Dichotomie : on garde la moitié de l'intervalle où f change de signe ; l'amplitude est divisée par 2 à chaque étape.
- Simulation : `random() < p` simule une épreuve de Bernoulli ; les fréquences approchent les probabilités.
- Le taux $\frac{f(a+h) - f(a)}{h}$ avec h petit approche le nombre dérivé $f'(a)$.

Le livre de cours de maths 1ère

Tout le programme en un seul PDF de 103 pages, à imprimer ou à garder sur ta tablette.

Voir le livre

 **Zyro te conseille la suite**

 Exercices de maths

Algorithmique et Python : exercices de maths 1ère

 Contrôles de maths

Algorithmique et Python : contrôle de maths 1ère

 QCM de maths

Algorithmique et Python : QCM de maths 1ère

 Exercices de maths

Second degré : exercices de maths 1ère

 Exercices de maths

Suites numériques : exercices de maths 1ère

 Contrôles de maths

Second degré : contrôle de maths 1ère

Zyro vous emmène sur nos sites partenaires

Embarquez pour d'autres planètes du savoir et leurs applis.

Décollage