Plaquette 1 – Suites, seuils, sommes et simulations

Algorithmique et Python — Première spé maths

Corrections détaillées

Boîte à outils

Schémas d'algorithmes à connaître.

— **Terme de rang** n d'une suite définie par récurrence : une boucle *for* qui applique n fois la relation $u_{n+1} = f(u_n)$.

— **Algorithme de seuil** : une boucle *while* qui avance tant que la condition n'est pas atteinte, et un compteur qui mémorise le rang.

— **Somme** : un accumulateur $s = 0$ puis $s = s + terme$ dans la boucle.

— **Simulation** : *randint(a, b)* (entier au hasard) ou *random()* (réel de $[0\,;1[$), répétée n fois, puis fréquence $= \dfrac{\text{succès}}{n}$.

— Formules mathématiques associées : $u_n = u_0 + nr$ (arithmétique), $u_n = u_0 q^n$ (géométrique),

$$1 + q + \cdots + q^n = \frac{1 - q^{n+1}}{1 - q} \quad (q \neq 1)$$

— *range(n)* parcourt $0, \ldots, n-1$; *range(1, n+1)* parcourt $1, \ldots, n$.

— *print* affiche, *return* renvoie : une fonction réutilisable fait un *return*.

— Une boucle *while* doit **progresser** vers sa condition d'arrêt, sinon le programme ne s'arrête jamais.

Correction de l'exercice 1 – Prévoir la sortie d'une boucle

La relation appliquée est $u_{n+1} = 2u_n - 3$, avec $u_0 = 5$, et la boucle tourne 4 fois.

Tour	1	2	3	4
u	7	11	19	35

— $2 \times 5 - 3 = 7$; $2 \times 7 - 3 = 11$; $2 \times 11 - 3 = 19$; $2 \times 19 - 3 = 35$.

— Le programme affiche 35, qui est donc u_4.

Piège. *range(4)* fait quatre tours : la variable i ne sert pas ici, elle ne compte que les répétitions.

Réponse. 35 (c'est u_4).

Correction de l'exercice 2 – Fonction qui calcule un terme

On traduit la relation de récurrence dans une boucle *for*.

```
def terme(n):
    u = 200
    for i in range(n):
        u = 0.9 * u + 15
    return u
```

— Vérification à la main :

— $u_1 = 0{,}9 \times 200 + 15 = 195$;

— $u_2 = 0{,}9 \times 195 + 15 = 190{,}5$;

— $u_3 = 0{,}9 \times 190{,}5 + 15 = 186{,}45$.
— Donc *terme(3)* renvoie 186,45.
— Remarque : la suite décroît vers 150, valeur pour laquelle $0{,}9x + 15 = x$.

Réponse. *terme(3)* renvoie 186,45.

Correction de l'exercice 3 – Algorithme de seuil

Multiplier par 1,12 chaque heure : suite géométrique $p_n = 8\,000 \times 1{,}12^{\,n}$.

```
def seuil():
    p = 8000
    n = 0
    while p <= 20000:
        p = p * 1.12
        n = n + 1
    return n
```

— On peut contrôler le résultat par le calcul : $1{,}12^8 \approx 2{,}476$, donc $p_8 \approx 19\,808 < 20\,000$; $1{,}12^9 \approx 2{,}773$, donc $p_9 \approx 22\,185 > 20\,000$.
— La fonction renvoie donc 9 : la population dépasse 20 000 au bout de 9 heures.

Piège. La condition est *while p <= 20000* : avec *while p < 20000*, on s'arrêterait aussi à une égalité, ce qui n'est pas « dépasser ».

Réponse. 9 heures ($p_9 \approx 22\,185$).

Correction de l'exercice 4 – Somme des termes d'une suite

On combine l'accumulateur et le calcul du terme général.

```
def somme(n):
    s = 0
    for k in range(n + 1):
        s = s + 3 + 5 * k
    return s
```

— *range(n + 1)* est indispensable pour inclure le terme u_n.
— Vérification par la formule de la somme d'une suite arithmétique : il y a 21 termes, le premier vaut 3, le dernier $u_{20} = 3 + 100 = 103$.

$$S = 21 \times \frac{3 + 103}{2} = 21 \times 53 = 1\,113$$

— Donc *somme(20)* renvoie 1 113.

Réponse. *somme(20)* renvoie 1 113.

Correction de l'exercice 5 – Somme géométrique

```
def sommeGeo(n):
    q = 0.8
    s = 0
    for k in range(n + 1):
        s = s + q**k
    return s
```

— Formule du cours : $\dfrac{1 - q^{\,n+1}}{1 - q} = \dfrac{1 - 0{,}8^{11}}{0{,}2}$.
— $0{,}8^{11} \approx 0{,}085\,899$, donc la somme vaut $\dfrac{1 - 0{,}085\,899}{0{,}2} \approx 4{,}570\,5$.

— Le programme renvoie la même valeur (aux arrondis près) : $\approx 4{,}5705$.
— Remarque : quand n grandit, la somme s'approche de $\dfrac{1}{1-0{,}8} = 5$ sans jamais l'atteindre.

Réponse. $\approx 4{,}571$, conforme à $\dfrac{1-0{,}8^{11}}{0{,}2}$.

Correction de l'exercice 6 – Balayage pour approcher une solution

— $f(1) = 1 + 1 - 5 = -3 < 0$ et $f(2) = 8 + 2 - 5 = 5 > 0$: la fonction change de signe, la solution est bien entre 1 et 2.
— On avance tant que $f(x)$ est **négative** : il faut compléter par 0, soit *while f(x) < 0:*.
— La boucle s'arrête au premier x (multiple de 0,01 à partir de 1) tel que $f(x) \geq 0$.
— Calculs : $f(1{,}51) \approx -0{,}047$ et $f(1{,}52) \approx 0{,}032$. Le programme renvoie donc $x \approx 1{,}52$.
— On obtient ainsi l'encadrement $1{,}51 < \alpha < 1{,}52$ de la solution.

Attention. Les additions répétées de 0,01 accumulent de petites erreurs d'arrondi en machine : mieux vaut boucler sur des entiers (*for i in range(100)*) pour un balayage précis.

Réponse. Compléter par 0 ; le programme renvoie $x \approx 1{,}52$ (solution $\approx 1{,}516$).

Correction de l'exercice 7 – Liste de termes

On part d'une liste contenant le premier terme, puis on ajoute les suivants avec *append*.

```
def liste(n):
    L = [1]
    v = 1
    for i in range(n):
        v = 3 * v - 1
        L.append(v)
    return L
```

— Calculs : $v_1 = 2$, $v_2 = 5$, $v_3 = 14$, $v_4 = 41$.
— Donc *liste(4)* renvoie *[1, 2, 5, 14, 41]*.
— Contrôle de la longueur : $n + 1 = 5$ éléments, ce qui est bien le cas.

Réponse. *[1, 2, 5, 14, 41]*.

Correction de l'exercice 8 – Simuler une loi binomiale

— La probabilité qu'une pièce soit défectueuse est 0,04 : on complète par *0.04*, car *random() < 0.04* est vrai avec la probabilité 0,04.
— La boucle intérieure simule un prélèvement de 25 pièces, la boucle extérieure répète N prélèvements.
— La fonction renvoie la **fréquence** des prélèvements contenant au moins une pièce défectueuse.
— Valeur exacte à comparer : $1 - 0{,}96^{25} \approx 1 - 0{,}360 = 0{,}640$.
— Pour $N = 10\,000$, la simulation donne une valeur proche de 0,64, qui varie légèrement d'une exécution à l'autre (fluctuation d'échantillonnage).

Réponse. Compléter par *0.04* ; la simulation approche $1 - 0{,}96^{25} \approx 0{,}640$.

Correction de l'exercice 9 – Compter dans une liste

On accumule la somme puis on divise par le nombre d'éléments, donné par *len(L)*.

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

— Avec *L = [12, 15, 9, 18, 11]* : somme = 65, longueur = 5.

— La fonction renvoie $\frac{65}{5} = 13$.

Piège. *len(L)* et non *len(L) - 1* : la division se fait par le nombre de valeurs. Et la fonction planterait sur une liste vide (division par zéro).

Réponse. *moyenne([12, 15, 9, 18, 11])* renvoie 13.

Correction de l'exercice 10 – Deux suites, un rattrapage

Le premier est arithmétique ($u_n = 300 + 20n$), le second géométrique ($v_n = 180 \times 1{,}06^n$).

```
def rattrapage():
    u = 300
    v = 180
    n = 0
    while v <= u:
        u = u + 20
        v = v * 1.06
        n = n + 1
    return n
```

— Quelques valeurs :

n	**12**	**13**	**14**	**15**
u_n (€)	540	560	580	600
v_n (€)	362,20	383,93	406,96	431,38

— L'écart se réduit lentement ; il faut aller plus loin :

n	**24**	**25**	**26**	**27**
u_n (€)	780	800	820	840
v_n (€)	728,81	772,54	818,89	868,02

— $v_{26} \approx 818{,}89 < 820 = u_{26}$ et $v_{27} \approx 868{,}02 > 840 = u_{27}$: la fonction renvoie 27.

— Morale : une croissance géométrique finit **toujours** par dépasser une croissance arithmétique, même si elle part de plus bas.

Réponse. $n = 27$ ans ($v_{27} \approx 868{,}02$ € contre $u_{27} = 840$ €).

Correction de l'exercice 11 – Synthèse : corriger un programme

— **Erreur 1 : la condition est inversée.** Au départ $p = 1$, donc $p > 1000$ est faux : la boucle ne démarre jamais. Il faut continuer **tant que** p n'a pas dépassé 1 000, c'est-à-dire *while p <= 1000:*.

— **Erreur 2 : le compteur n'est pas incrémenté** et c'est p qui est renvoyé au lieu de n. Il faut ajouter $n = n + 1$ dans la boucle et renvoyer n.

```
def seuil():
    p = 1
    n = 0
```

```
    while p <= 1000:
        p = p * 2
        n = n + 1
    return n
```

— Déroulé : $2^9 = 512 \leq 1\,000$, puis $2^{10} = 1\,024 > 1\,000$.

— La fonction corrigée renvoie 10.

Réponse. Condition *while p <= 1000* et compteur *n = n + 1* avec *return n* ; résultat : 10.