Plaquette 5 – Problèmes de synthèse : programmes complets à écrire et à corriger

Algorithmique et Python — Seconde

Corrections détaillées

Boîte à outils

Méthode pour lire un programme. On construit un tableau de suivi, une colonne par variable, une ligne par tour de boucle. C'est long mais infaillible.

Les six erreurs les plus fréquentes.

Erreur	Symptôme
return dans la boucle	la fonction s'arrête au premier tour
mauvaise indentation	instruction exécutée trop souvent, ou pas assez
= au lieu de ==	erreur de syntaxe
borne de *range* oubliée	il manque le dernier terme
accumulateur mal initialisé	résultat décalé, ou toujours nul
mise à jour manquante dans un *while*	boucle infinie

Le piège de l'indentation. En Python, l'indentation **définit** les blocs. Un *return* décalé d'un niveau change complètement le comportement du programme.

Écrire un programme à partir d'un énoncé. Quatre étapes :

1. identifier les **entrées** (les paramètres de la fonction) ; 2. identifier la **sortie** (ce que renvoie *return*) ; 3. choisir la structure : *for* si le nombre de tours est connu, *while* sinon ; 4. traiter les cas limites : liste vide, $n = 0$, division par zéro.

Tester un programme. On l'essaie sur un cas dont on connaît la réponse à la main, puis sur un cas limite. Un programme non testé n'est pas terminé.

Contrôler un résultat. Trois moyens : une formule mathématique, une seconde méthode de calcul, ou un ordre de grandeur.

Correction de l'exercice 1 – Prévoir une sortie

— **Rôle de* a*** : le maximum ✓
— **Rôle de* b*** : le minimum ✓
— **Initialisation** : $a = b = 5$ ✓
— **Après** $x = 5$: $a = 5$ et $b = 5$ ✓
— **Après** $x = 2$: $a = 5$ et $b = 2$ ✓
— **Après** $x = 8$: $a = 8$ et $b = 2$ ✓
— **Après** $x = 1$: $a = 8$ et $b = 1$ ✓
— **Après** $x = 9$: $a = 9$ et $b = 1$ ✓
— **Valeur renvoyée** :

$$9 - 1 = 8$$

— **Nom mathématique de cette quantité** : l'**étendue** de la série ✓
— **Contrôle** : $\max(L) = 9$ et $\min(L) = 1$ ✓
— **Version courte équivalente** : *return max(L) - min(L)* ✓

— **Sur une liste constante** : le résultat serait 0 ✓
— **Signe du résultat** : toujours positif ou nul ✓
— **Justification** : le maximum est supérieur ou égal au minimum ✓
— **Sur une liste vide** : *L[0]* provoque une erreur ✓
— **Sur une liste à un élément** : le résultat est 0 ✓

Réponse. Elle renvoie 8 : c'est l'**étendue** de la liste ($9 - 1$).

Correction de l'exercice 2 – Corriger une indentation

— **Diagnostic** : le *return* est **dans** la boucle ✓
— **Conséquence** : la fonction se termine au **premier** tour ✓
— **Valeur renvoyée** : $0 + L[0]$, soit le premier élément ✓
— **Contrôle sur* [3, 5, 7]*** : la fonction renvoie 3 ✓
— **Correction** : sortir le *return* d'un niveau d'indentation ✓

```
def somme(L):
    s = 0
    for x in L:
        s = s + x
    return s
```

— **Vérification après correction** : sur *[3, 5, 7]*, on obtient 15 ✓
— **Contrôle** : 3 + 5 + 7 = 15 ✓
— **Règle générale** : un *return* dans une boucle **interrompt** l'exécution ✓
— **Quand est-ce voulu ?** Dans une recherche, pour s'arrêter dès qu'on a trouvé ✓
— **Exemple légitime** : une fonction qui teste la présence d'une valeur ✓
— **Comment repérer l'erreur ?** Tester sur une liste de plusieurs éléments ✓
— **Symptôme caractéristique** : le résultat ne dépend **pas** de la fin de la liste ✓
— **Test discriminant** : comparer *somme([1, 2])* et *somme([1, 100])* ✓
— **Avant correction** : les deux renvoient 1 ✓
— **Après correction** : 3 et 101 ✓
— **Leçon** : en Python, l'indentation n'est pas cosmétique, elle **définit** la structure ✓

Réponse. Le *return* doit être **après** la boucle, à l'indentation de la fonction.

Correction de l'exercice 3 – Corriger un range

— **Diagnostic** : *range(n)* parcourt 0 à $n - 1$ ✓
— **Somme calculée** : $0 + 1 + \cdots + (n - 1)$ ✓
— **Somme attendue** : $1 + 2 + \cdots + n$ ✓
— **Écart** : exactement n ✓
— **Contrôle pour** $n = 5$: la fonction renvoie 10 au lieu de 15 ✓
— **Détail** : 0 + 1 + 2 + 3 + 4 = 10 ✓
— **Première correction possible** : *range(1, n + 1)* ✓
— **Seconde correction possible** : *range(n + 1)*, le 0 n'ajoutant rien ✓
— **Vérification pour** $n = 5$:

$$1 + 2 + 3 + 4 + 5 = 15$$

— **Contrôle par la formule** : $\dfrac{5 \times 6}{2} = 15$ ✓
— **Contrôle pour** $n = 100$: $\dfrac{100 \times 101}{2} = 5050$ ✓

— **Résultat de la version erronée** : $\frac{99 \times 100}{2} = 4950$ ✓
— **Écart** : 100, conforme au diagnostic ✓
— **Version en une ligne** : *return n* (n + 1) // 2* ✓
— **Avantage** : un temps de calcul **constant**, au lieu de *n* tours ✓
— **Cas** $n = 0$: les deux versions renvoient 0 ✓
— **Cohérence** : la somme d'aucun terme vaut bien 0 ✓

Réponse. Il faut *range(1, n + 1)* : la version initiale somme 0 à $n - 1$, il manque donc n.

Correction de l'exercice 4 – Corriger un accumulateur

— **Diagnostic** : l'accumulateur est initialisé à 0 ✓
— **Conséquence** : $0 \times x = 0$ à chaque tour ✓
— **Résultat** : toujours 0, quelle que soit la liste ✓
— **Correction** : initialiser à 1 ✓
— **Justification** : 1 est l'élément **neutre** de la multiplication ✓

```
def produit(L):
    p = 1
    for x in L:
        p = p * x
    return p
```

— **Vérification sur* [2, 3, 5]*** :

$$1 \times 2 \times 3 \times 5 = 30$$

— **Contrôle** : $2 \times 3 \times 5 = 30$ ✓
— **Sur une liste vide** : le résultat est 1 ✓
— **Cohérence mathématique** : le produit vide vaut 1, par convention ✓
— **Analogie avec la somme** : elle s'initialise à 0, l'élément neutre de l'addition ✓
— **Règle générale** : l'accumulateur part toujours de l'élément **neutre** de l'opération ✓
— **Cas particulier légitime du** 0 : si la liste contient un 0, le produit est nul ✓
— **Contrôle** : *produit([2, 0, 5])* renvoie 0 ✓
— **Pour le maximum** : l'élément neutre n'existe pas, d'où l'initialisation à *L[0]* ✓
— **Application** : la factorielle est le produit des entiers de 1 à n ✓
— **Contrôle** : *produit([1, 2, 3, 4, 5])* renvoie $120 = 5!$ ✓

Réponse. Il faut *p = 1* : l'élément neutre de la multiplication, et non 0.

Correction de l'exercice 5 – Écrire : notes au-dessus de la moyenne

— **Structure** : d'abord la moyenne, puis le filtrage ✓

```
def auDessus(L):
    s = 0
    for x in L:
        s = s + x
    m = s / len(L)
    R = []
    for x in L:
        if x > m:
            R.append(x)
    return R
```

— **Nécessité de deux parcours** : la moyenne doit être **complète** avant la comparaison ✓
— **Erreur classique** : comparer à une moyenne partielle dans un seul parcours ✓
— **Somme** : $8 + 15 + 11 + 18 + 6 + 12 = 70$ ✓
— **Effectif** : 6 ✓
— **Moyenne** :

$$\frac{70}{6} \approx 11{,}667$$

— **Notes supérieures** : 15, 18 et 12 ✓
— **Résultat** :

$$[15, 18, 12]$$

— **Contrôle** : 11 est inférieur à 11,667, il est donc exclu ✓
— **Attention** : 11 et 12 encadrent la moyenne ✓
— **Nombre de notes au-dessus** : 3 ✓
— **Nombre en dessous** : 3 également ✓
— **Coïncidence** : la moyenne partage ici la série en deux moitiés égales ✓
— **Ce n'est pas général** : la moyenne n'est pas la médiane ✓
— **Médiane de la série** : la série triée est $[6, 8, 11, 12, 15, 18]$ ✓
— **Calcul** : $\frac{11 + 12}{2} = 11{,}5$ ✓
— **Comparaison** : $11{,}5 < 11{,}667$ ✓

Réponse. *[15, 18, 12]*, la moyenne valant $\frac{70}{6} \approx 11{,}67$.

Correction de l'exercice 6 – Écrire : suite récurrente

— **Structure** : une boucle *for* de n tours ✓

```
def u(n):
    x = 2
    for i in range(n):
        x = 0.8 * x + 3
    return x
```

— **Déroulé** : $u_0 = 2$ ✓
— $u_1 = 1{,}6 + 3 = 4{,}6$ ✓
— $u_2 = 3{,}68 + 3 = 6{,}68$ ✓
— $u_3 = 5{,}344 + 3 = 8{,}344$ ✓
— $u_4 = 6{,}6752 + 3 = 9{,}6752$ ✓
— $u_5 = 7{,}74016 + 3$ ✓
— **Résultat** :

$$u_5 = 10{,}74016$$

— **Croissance** : la suite est croissante ✓
— **Ralentissement** : les écarts diminuent ✓
— **Écarts successifs** : 2,6 ; 2,08 ; 1,664 ; 1,3312 ; 1,06496 ✓
— **Observation** : chaque écart vaut 0,8 fois le précédent ✓
— **Limite** : elle vérifie $\ell = 0{,}8\ell + 3$ ✓
— **Résolution** : $0{,}2\ell = 3$, donc

$$\ell = 15$$

— **Contrôle** : $0{,}8 \times 15 + 3 = 15$ ✓
— **Valeurs suivantes** : $u_{10} \approx 13{,}604$ et $u_{20} \approx 14{,}850$ ✓
— **Forme explicite** : $u_n = 15 - 13 \times 0{,}8^n$ ✓
— **Contrôle pour** $n = 0$: $15 - 13 = 2$ ✓
— **Contrôle pour** $n = 5$: $15 - 13 \times 0{,}32768 = 10{,}74016$ ✓

Réponse. $u_5 = 10{,}74016$, et la suite converge vers $\ell = 15$.

Correction de l’exercice 7 – Écrire : résolution d’équation par balayage

— **Principe** : avancer par petits pas jusqu’à dépasser ✓

```
x = 2
```

```
pas = 0.01
while x * x < 7:
    x = x + pas
print(x)
```

— **Valeur exacte** : $\sqrt{7} \approx 2{,}6458$ ✓
— **Contrôle des bornes** : $2^2 = 4 < 7$ et $3^2 = 9 > 7$ ✓
— **Test en** 2,64 : $6{,}9696 < 7$, la boucle continue ✓
— **Test en** 2,65 : $7{,}0225 > 7$, la boucle s’arrête ✓
— **Résultat affiché** :

$$\approx 2{,}65$$

— **Encadrement obtenu** : $2{,}64 < \sqrt{7} < 2{,}65$ ✓
— **Contrôle** : $\sqrt{7} \approx 2{,}6458$, bien dans l’intervalle ✓
— **Nombre de tours** : environ 65 ✓
— **Contrôle** : $\dfrac{2{,}65 - 2}{0{,}01} = 65$ ✓
— **Problème des décimaux** : les additions successives de 0,01 accumulent une petite erreur ✓
— **Solution plus sûre** : boucler sur un **entier** et diviser ✓
— **Code correspondant** : *for k in range(0, 101): x = 2 + k / 100* ✓
— **Comparaison avec la dichotomie** : 100 tours ici contre 7 pour la même précision ✓
— **Contrôle** : $\dfrac{1}{2^7} \approx 0{,}0078 < 0{,}01$ ✓
— **Avantage du balayage** : il trouve **toutes** les solutions d’un intervalle ✓
— **Avantage de la dichotomie** : bien plus rapide, mais une seule solution ✓

Réponse. Le programme affiche ≈ 2,65, encadrant $\sqrt{7} \approx 2{,}6458$.

Correction de l’exercice 8 – Écrire : conversion de durée

— **Principe** : divisions euclidiennes successives ✓

```
def convertit(t):
```

```
    h = t // 3600
    reste = t % 3600
    m = reste // 60
    s = reste % 60
    return h, m, s
```

— **Nombre de secondes dans une heure** : 3600 ✓
— **Heures** : $10\,000 \,//\, 3600 = 2$ ✓

— **Contrôle** : $2 \times 3600 = 7200$ ✓
— **Reste** : $10\,000 - 7200 = 2800$ secondes ✓
— **Minutes** : $2800 \,//\, 60 = 46$ ✓
— **Contrôle** : $46 \times 60 = 2760$ ✓
— **Secondes** : $2800 - 2760 = 40$ ✓
— **Résultat** :

$$2 \text{ h } 46 \text{ min } 40 \text{ s}$$

— **Contrôle inverse** : $2 \times 3600 + 46 \times 60 + 40 = 7200 + 2760 + 40 = 10\,000$ ✓
— **Piège de l'opérateur** : $t \,/\, 3600$ donnerait $2{,}7\overline{7}$, un décimal ✓
— **Nécessité de //**** : pour obtenir un nombre entier d'heures ✓
— **Piège du reste** : réutiliser t au lieu de *reste* fausserait les minutes ✓
— **Résultat de ce piège** : $10\,000 \,//\, 60 = 166$ minutes ✓
— **Contrôle du piège** : 166 minutes dépasse largement une heure ✓
— **Pour** 86 400 **secondes** : on obtiendrait 24 h 0 min 0 s ✓
— **Contrôle** : c'est exactement une journée ✓
— **Pour ajouter les jours** : diviser d'abord par 86 400 ✓

Réponse. 10 000 s = 2 h 46 min 40 s.

Correction de l'exercice 9 – Écrire : tableau de signes

— **Structure** : une fonction, une boucle, et un test à trois branches ✓

```
def f(x):
    return x * x - 5 * x + 6

for x in range(7):
    if f(x) > 0:
        print(x, "positif")
    elif f(x) == 0:
        print(x, "nul")
    else:
        print(x, "négatif")
```

— **Rôle de* elif***** : tester une seconde condition seulement si la première est fausse ✓
— **Calcul en** 0 : 6, positif ✓
— **Calcul en** 1 : $1 - 5 + 6 = 2$, positif ✓
— **Calcul en** 2 : $4 - 10 + 6 = 0$, **nul** ✓
— **Calcul en** 3 : $9 - 15 + 6 = 0$, **nul** ✓
— **Calcul en** 4 : $16 - 20 + 6 = 2$, positif ✓
— **Calcul en** 5 : $25 - 25 + 6 = 6$, positif ✓
— **Calcul en** 6 : $36 - 30 + 6 = 12$, positif ✓
— **Racines lues** :

$$x = 2 \quad \text{et} \quad x = 3$$

— **Contrôle par factorisation** : $x^2 - 5x + 6 = (x - 2)(x - 3)$ ✓
— **Développement de contrôle** : $x^2 - 3x - 2x + 6 = x^2 - 5x + 6$ ✓
— **Signe entre les racines** : négatif, mais aucun entier ne s'y trouve ✓
— **Contrôle** : $f(2{,}5) = 6{,}25 - 12{,}5 + 6 = -0{,}25$ ✓
— **Conclusion** : le balayage entier **manque** la partie négative ✓
— **Leçon** : un balayage à pas trop grand peut masquer un changement de signe ✓

— **Correction** : balayer au pas de 0,1 révélerait les valeurs négatives ✓
— **Somme et produit des racines** : 2 + 3 = 5 et 2 × 3 = 6, conformes aux coefficients ✓

Réponse. Racines $x = 2$ et $x = 3$; le balayage entier manque cependant l'intervalle négatif $]2\,;3[$.

Correction de l'exercice 10 – Corriger une condition

— **Diagnostic** : l'opérateur *or* au lieu de *and* ✓
— **Analyse de la condition** : *x >= 10 or x <= 20* ✓
— **Conséquence** : tout nombre vérifie l'une des deux conditions ✓
— **Justification** : si $x < 10$, alors $x \leq 20$ est vrai ✓
— **Et si** $x > 20$: alors $x \geq 10$ est vrai ✓
— **Résultat** : la fonction compte **tous** les éléments ✓
— **Contrôle sur* [5, 15, 25]*** : elle renvoie 3 au lieu de 1 ✓
— **Correction** : remplacer *or* par *and* ✓
— **Condition corrigée** : *if x >= 10 and x <= 20* ✓
— **Écriture plus lisible en Python** : *if 10 <= x <= 20* ✓
— **Vérification sur* [5, 15, 25]*** :

1

— **Vérification sur* [8, 10, 12, 20, 22]*** : on compte 10, 12 et 20 ✓
— **Résultat** : 3 ✓
— **Bornes incluses** : les inégalités sont **larges** ✓
— **Pour exclure les bornes** : utiliser > et < ✓
— **Résultat alors** : 1 seul élément, le 12 ✓
— **Règle de vérification** : tester la condition sur un point **hors** de l'intervalle ✓
— **Test discriminant** : $x = 0$ doit donner faux, ce que l'*or* ne faisait pas ✓

Réponse. Il faut *and* et non *or* (ou l'écriture *10 <= x <= 20*).

Correction de l'exercice 11 – Écrire : recherche du plus proche

— **Principe** : parcourir en mémorisant la plus petite distance ✓

```
def plusProche(L, v):
    meilleur = L[0]
    for x in L:
        if abs(x - v) < abs(meilleur - v):
            meilleur = x
    return meilleur
```

— **Rôle de* abs*** : mesurer la distance, sans tenir compte du signe ✓
— **Initialisation** : au premier élément, jamais à une valeur arbitraire ✓
— **Distance de** 3 **à** 9 : 6 ✓
— **Distance de** 11 **à** 9 : 2 ✓
— **Comparaison** : 2 < 6, donc *meilleur* devient 11 ✓
— **Distance de** 7 **à** 9 : 2 ✓
— **Comparaison** : 2 < 2 est **faux**, *meilleur* reste 11 ✓
— **Distance de** 20 **à** 9 : 11 ✓
— **Comparaison** : 11 < 2 est faux ✓
— **Résultat** :

11

— **Remarque importante** : 7 est **aussi** à distance 2 ✓
— **Choix effectué** : l'inégalité stricte garde le **premier** trouvé ✓
— **Avec une inégalité large** : la fonction renverrait 7, le dernier trouvé ✓
— **Contrôle** : les deux réponses sont mathématiquement valables ✓
— **Pour la valeur** 8 : la réponse serait 7, à distance 1 ✓
— **Pour la valeur** 100 : la réponse serait 20 ✓
— **Sur une liste vide** : *L[0]* provoque une erreur ✓

Réponse. 11 (mais 7 est à la même distance : l'inégalité stricte garde le premier trouvé).

Correction de l'exercice 12 – Écrire : triangle de Pascal

— **Principe** : chaque ligne se déduit de la précédente ✓
— **Règle** : chaque terme est la somme des deux au-dessus ✓

```
ligne = [1]
```

```
for k in range(6):
    print(ligne)
    nouvelle = [1]
    for i in range(len(ligne) - 1):
        nouvelle.append(ligne[i] + ligne[i + 1])
    nouvelle.append(1)
    ligne = nouvelle
```

— **Première ligne** : [1] ✓
— **Deuxième** : [1, 1] ✓
— **Troisième** : [1, 2, 1] ✓
— **Quatrième** : [1, 3, 3, 1] ✓
— **Cinquième** : [1, 4, 6, 4, 1] ✓
— **Sixième** :

$$[1, 5, 10, 10, 5, 1]$$

— **Contrôle de la construction** : $10 = 4 + 6$ ✓
— **Autre contrôle** : $5 = 1 + 4$ ✓
— **Somme de chaque ligne** : 1, 2, 4, 8, 16, 32 ✓
— **Régularité** : la somme de la ligne n vaut 2^n ✓
— **Contrôle de la sixième ligne** : $1 + 5 + 10 + 10 + 5 + 1 = 32 = 2^5$ ✓
— **Symétrie** : chaque ligne se lit indifféremment dans les deux sens ✓
— **Lien avec les probabilités** : la ligne n donne les effectifs des lancers de n pièces ✓
— **Contrôle** : 5 pièces donnent 32 issues, dont 10 avec exactement 2 piles ✓
— **Probabilité correspondante** : $\frac{10}{32} = 0{,}3125$ ✓
— **Septième ligne** : [1, 6, 15, 20, 15, 6, 1], de somme 64 ✓

Réponse. [1] ; [1, 1] ; [1, 2, 1] ; [1, 3, 3, 1] ; [1, 4, 6, 4, 1] ; [1, 5, 10, 10, 5, 1].

Correction de l'exercice 13 – Corriger une boucle infinie

— **Diagnostic** : *p = 2* n *au lieu de* p = 2 *p* ✓
— **Conséquence** : p croît **linéairement**, non géométriquement ✓
— **La boucle s'arrête-t-elle vraiment jamais ?** Si, mais très tard ✓
— **Quand ?** Quand $2n > 1000$, soit $n > 500$ ✓
— **Résultat obtenu** : $n = 502$ au lieu de 10 ✓

— **Vrai problème** : le programme ne calcule **pas** ce qu'on veut ✓
— **Correction** : écrire *p = 2* p* ✓

```
n = 0
```

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

— **Déroulé** : $p = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$ ✓
— **Nombre de doublements** : 10 ✓
— **Résultat** :

$$n = 10$$

— **Contrôle** : $2^{10} = 1024 > 1000$ ✓
— **Contrôle de l'étape précédente** : $2^{9} = 512 \leq 1000$ ✓
— **Cohérence** : 10 est bien le plus petit exposant convenable ✓
— **Version alternative avec une puissance** : *while 2 n <= 1000: n = n + 1* ✓
— **Avantage** : plus lisible, mais elle recalcule la puissance à chaque tour ✓
— **Pour dépasser un million** : $n = 20$, car $2^{20} = 1\,048\,576$ ✓
— **Règle utile** : $2^{10} \approx 10^{3}$, donc $2^{20} \approx 10^{6}$ ✓

Réponse. Il faut *p = 2* p* ; le résultat est alors $n = 10$, car $2^{10} = 1024 > 1000$.

Correction de l'exercice 14 – Écrire : moyenne glissante

— **Principe** : parcourir les paires consécutives ✓

```
def glissante(L):
```

```
    R = []
    for i in range(len(L) - 1):
        R.append((L[i] + L[i + 1]) / 2)
    return R
```

— **Borne du* range** : len(L) - 1*, pour que* L[i + 1]* existe ✓
— **Piège** : avec *range(len(L))*, l'accès à *L[i + 1]* provoquerait une erreur au dernier tour ✓
— **Nombre de moyennes** : 3 pour une liste de 4 éléments ✓
— **Première moyenne** : $\frac{4 + 10}{2} = 7$ ✓
— **Deuxième** : $\frac{10 + 6}{2} = 8$ ✓
— **Troisième** : $\frac{6 + 8}{2} = 7$ ✓
— **Résultat** :

[7,0, 8,0, 7,0]

— **Remarque sur le type** : la division renvoie des décimaux ✓
— **Effet de lissage** : les variations sont **atténuées** ✓
— **Étendue de la liste initiale** : $10 - 4 = 6$ ✓
— **Étendue de la liste lissée** : $8 - 7 = 1$ ✓
— **Rapport** : les variations sont divisées par 6 ✓
— **Moyenne de la liste initiale** : $\frac{28}{4} = 7$ ✓

— **Moyenne de la liste lissée** : $\frac{22}{3} \approx 7{,}333$ ✓
— **Pourquoi cet écart ?** Les valeurs extrêmes sont comptées **une fois**, les autres deux fois ✓
— **Application** : le lissage sert à dégager une **tendance** dans des données bruitées ✓
— **Variante à trois valeurs** : moyenner *L[i]*, *L[i+1]* et *L[i+2]* lisserait davantage ✓

Réponse. *[7.0, 8.0, 7.0]* : les variations sont fortement lissées.

Correction de l'exercice 15 – Comparer deux programmes

— **Programme A** : la **somme des carrés** ✓
— **Programme B** : le **carré de la somme** ✓
— **Ce ne sont pas les mêmes** ✓
— **Test pour** $n = 3$ — programme A : $1 + 4 + 9 = 14$ ✓
— **Programme B** : $(1 + 2 + 3)^2 = 36$ ✓
— **Conclusion** :

$$14 \neq 36$$

— **Test pour** $n = 1$: les deux donnent 1 ✓
— **Explication de cette coïncidence** : avec un seul terme, le carré de la somme est la somme des carrés ✓
— **Test pour** $n = 4$ — A : $1 + 4 + 9 + 16 = 30$ ✓
— **B** : $10^2 = 100$ ✓
— **Formule de A** : $\frac{n(n+1)(2n+1)}{6}$ ✓
— **Contrôle pour** $n = 4$: $\frac{4 \times 5 \times 9}{6} = 30$ ✓
— **Formule de B** : $\left(\frac{n(n+1)}{2}\right)^2$ ✓
— **Contrôle pour** $n = 4$: $10^2 = 100$ ✓
— **Rapport pour** n **grand** : $\frac{B}{A}$ croît comme $\frac{3n}{4}$ ✓
— **Contrôle pour** $n = 4$: $\frac{100}{30} \approx 3{,}33$, et $\frac{3 \times 4}{4} = 3$ ✓
— **Identité remarquable** : B est la somme des **cubes** de 1 à n ✓
— **Contrôle pour** $n = 4$: $1 + 8 + 27 + 64 = 100$ ✓
— **Leçon générale** : la somme des carrés n'est **jamais** le carré de la somme, sauf pour un seul terme ✓

Réponse. Non : A donne la somme des carrés (14 pour $n = 3$), B le carré de la somme (36).

Correction de l'exercice 16 – Synthèse : un programme d'analyse complet

a) **Trois indicateurs en un parcours** ✓

```
def indicateurs(L):
    s = 0
    mini = L[0]
    maxi = L[0]
    for x in L:
        s = s + x
        if x < mini:
            mini = x
        if x > maxi:
```

```
            maxi = x
    return s / len(L), mini, maxi
```

— **Initialisation des extremums** : à *L[0]* ✓
— **Division finale** : après la boucle, dans le *return* ✓

b) **Comptage sous condition** ✓

```
def jours(L, seuil):
    c = 0
    for x in L:
        if x > seuil:
            c = c + 1
    return c
```

c) **Écart maximal entre jours consécutifs** ✓

```
def ecartMax(L):
    record = 0
    jour = 0
    for i in range(len(L) - 1):
        e = abs(L[i + 1] - L[i])
        if e > record:
            record = e
            jour = i
    return record, jour
```

— **Rôle de* abs*** : mesurer l'écart sans se soucier du sens ✓
— **Borne du* range** *:* len(L) - 1*, pour accéder à* L[i + 1]* ✓
— **Renvoi du jour** : l'indice du **premier** des deux jours ✓

d) **Résultats pour* T*** ✓

— **Somme** : $14 + 17 + 21 + 19 + 23 + 25 + 18 = 137$ ✓
— **Effectif** : 7 jours ✓
— **Moyenne** :

$$\frac{137}{7} \approx 19{,}571 \text{ °C}$$

— **Minimum** : 14 °C, le premier jour ✓
— **Maximum** : 25 °C, le sixième jour ✓
— **Étendue** : 11 °C ✓
— **Jours au-dessus de** 20 **°C** : 21, 23 et 25 ✓
— **Comptage** :

$$3 \text{ jours}$$

— **Jours au-dessus de la moyenne** : 21, 23 et 25 également ✓
— **Coïncidence** : la moyenne 19,571 est juste sous 20 ✓
— **Écarts consécutifs** : $+3, +4, -2, +4, +2, -7$ ✓
— **Valeurs absolues** : 3, 4, 2, 4, 2, 7 ✓
— **Écart maximal** :

$$7 \text{ °C, entre le } 6^{\text{e}} \text{ et le } 7^{\text{e}} \text{ jour}$$

— **Indice renvoyé** : 5, celui du sixième jour ✓
— **Contrôle** : $|18 - 25| = 7$ ✓
— **Deuxième plus grand écart** : 4 °C, atteint **deux fois** ✓
— **Choix de la fonction** : l'inégalité stricte garde le **premier**, à l'indice 1 ✓

— **Somme des écarts signés** : $3 + 4 - 2 + 4 + 2 - 7 = 4$ ✓
— **Contrôle** : $18 - 14 = 4$, la différence entre le dernier et le premier jour ✓
— **Cohérence** : la somme des variations égale la variation totale ✓
— **Médiane de la série triée** $[14, 17, 18, 19, 21, 23, 25]$: 19 °C ✓
— **Comparaison à la moyenne** : $19 < 19{,}571$ ✓
— **Interprétation** : la moyenne est tirée vers le haut par les deux jours chauds ✓

Réponse. d) moyenne ≈ 19,57 °C, min 14, max 25, 3 jours au-dessus de 20 °C, écart maximal de 7 °C entre le 6^{e} et le 7^{e} jour.