CORRIGÉ

2de

Algorithmique et programmation Python

Corrigé détaillé
6 exercices

> Ce qui est noté au contrôle, ce n'est pas le résultat seul : c'est le raisonnement posé, puis le calcul mené jusqu'au bout avec l'unité.

1 Suivre un programme à la main

1. **a.** `a` vaut 10 et `b` vaut **5**. La valeur de `b` a été calculée avec l'ancien `a` : elle ne se met pas à jour toute seule.
2. C'est la conséquence directe de l'affectation : `b` a reçu un nombre, pas une formule.
3. **b.** Les deux étapes.
   $7 \times 2 = 14$
4. Puis.
   $14 - 4 = 10$
   `n` vaut 10.
5. **c.** Six fois : `i` prend 3, 4, 5, 6, 7 et 8. La borne 9 est exclue.
6. Le nombre de tours vaut toujours la différence des bornes : $9 - 3 = 6$.

2 Corriger un programme

1. **a.** Un seul `=` affecte au lieu de comparer. Il faut écrire `if x == 5:`.
2. **b.** Parce que `print` affiche mais ne renvoie rien. Il faut écrire `return m` pour que la valeur soit utilisable dans un calcul.
3. C'est la distinction la plus importante du chapitre : afficher est pour l'humain, renvoyer est pour le programme.
4. **c.** Boucle **infinie** : rien à l'intérieur ne diminue `n`, donc la condition `n > 0` reste vraie indéfiniment.
5. Le programme affiche 10 sans discontinuer, sans jamais s'arrêter ni signaler d'erreur.
6. Il manque une ligne indentée, par exemple `n = n - 1`, qui rapproche de la sortie.

3 Écrire une boucle for

1. **a.** `total = 0`, puis `for i in range(1, 101):` et, indenté, `total = total + i`. On affiche `total` à la fin.
2. La borne 101 est nécessaire pour que 100 soit inclus.
3. **b.** Deux solutions. Soit `for i in range(2, 101, 2):` — le troisième argument est le pas.
4. Soit garder la boucle et ajouter `if i % 2 == 0:` avant l'addition, `%` donnant le reste de la division.
5. **c.** La somme des n premiers entiers vaut $\frac{n(n+1)}{2}$.
   $\frac{100 \times 101}{2} = 5050$
6. Le programme doit afficher 5050 : c'est le contrôle qu'il faut toujours prévoir avant de lancer.

4 Une fonction avec return

1. **a.** `def est_pair(n):` puis, indenté, `return n % 2 == 0`.
2. L'expression `n % 2 == 0` vaut déjà `True` ou `False` : il est inutile d'écrire un `if` pour renvoyer l'un ou l'autre.
3. **b.** `def compte_pairs(liste):`, puis `c = 0`, puis `for x in liste:`, puis `if est_pair(x):` et `c = c + 1`.
4. On termine par `return c`, aligné avec le `for` — donc exécuté après la boucle, et non à chaque tour.
5. Placer le `return` à l'intérieur de la boucle est l'erreur classique : la fonction s'arrêterait au premier élément.
6. C'est là que l'indentation devient un raisonnement : sa position décide de ce que fait le programme.

5 Algorithme d'Euclide

1. **a.** Reste de 91 par 35.
   $91 - 2 \times 35 = 21$
2. Reste de 35 par 21.
   $35 - 21 = 14$
3. Reste de 21 par 14.
   $21 - 14 = 7$
4. Reste de 14 par 7 : il vaut 0. Le PGCD est **7**.
   $14 - 2 \times 7 = 0$
5. **b.** Quatre tours.
6. **c.** On obtient successivement les restes 2, 1, puis 0 : le PGCD vaut **1**.
7. Un PGCD égal à 1 signifie que les deux nombres sont **premiers entre eux** — ils n'ont aucun diviseur commun autre que 1.

---

6 Test de primalité

1. **a.** La racine de 97.
   $\sqrt{97} \approx 9{,}85$
2. Il suffit donc de tester les diviseurs de 2 à 9.
3. **b.** Si $n = a \times b$ avec $a > \sqrt{n}$, alors nécessairement $b < \sqrt{n}$ — sans quoi le produit dépasserait n.
4. Autrement dit, tout diviseur supérieur à la racine a un complice inférieur, déjà testé. Chercher au-delà, c'est retrouver les mêmes couples dans l'autre sens.
5. **c.** On teste 8 diviseurs au lieu de 95.
   $95 - 8 = 87$
6. Quatre-vingt-sept tests économisés sur un petit nombre ; l'écart devient décisif sur de grands entiers.
7. 97 n'étant divisible par aucun de 2 à 9, il est **premier**.

Corrige-toi toi-même. Compte tes points exercice par exercice : c'est ainsi qu'on voit où l'on perd, et ce n'est presque jamais là où on le croit.

Ce qui est évalué	Points	Ce qui fait perdre le point
Distinguer affectation et comparaison	2 pts	`if x = 5:`
Indenter conformément au raisonnement voulu	4 pts	Un `return` placé dans la boucle
Maîtriser les bornes de `range`	3 pts	Oublier la dernière valeur
Garantir la terminaison d'une boucle `while`	3 pts	Une boucle infinie
Employer `return` pour rendre une fonction réutilisable	3 pts	Une fonction qui affiche et renvoie `None`
Prévoir un cas de test dont on connaît la réponse	2 pts	Lancer un programme sans savoir ce qu'il doit donner