

Algorithmique et programmation Python

Mathématiques · Seconde
Contrôle n°1 · 55 min · sur 20

Exercice 1 — Variables et indentation

4 pts 10 min

Comprendre affectation et indentation

Réponds à chaque question.

LA RÈGLE DU COURS

Une ligne mathématiquement absurde devient parfaitement claire en programmation, parce que le signe n'y veut pas dire la même chose. Et le décalage des lignes n'est pas un confort de lecture.

1. Que signifie exactement l'affectation `n = 5` ? (1 pt)**Range 5 dans la case appelée `n`.**

→ Ce n'est pas l'affirmation d'une égalité.

2. Pourquoi `n = n + 1` a-t-il un sens en programmation ? (1 pt)**Parce qu'on remplace le contenu de `n` par ce contenu augmenté de 1.**

→ Mathématiquement, l'égalité serait absurde.

3. À quoi sert l'indentation en Python ? (1 pt)

Elle indique ce qui s'exécute à l'intérieur d'un bloc : c'est la syntaxe elle-même.

→ Ce qui revient à gauche s'exécute dans tous les cas.

4. Quelle différence y a-t-il entre `=` et `==` ? (1 pt)**Un seul signe égal affecte une valeur ; le double signe compare.**

→ Dans un test, on demande : c'est donc le double signe.

Exercice 2 — Les deux boucles

4 pts 11 min

Choisir et écrire une boucle

Réponds à chaque question.

LA RÈGLE DU COURS

L'une répète un nombre de fois connu d'avance, l'autre tant qu'une condition tient. La première a une borne piégeuse, la seconde un risque qui ne produit aucun message d'erreur.

1. Quand emploie-t-on une boucle `for` ? (1 pt)**Quand le nombre de répétitions est connu à l'avance.**

→ C'est ce qui la distingue de l'autre boucle.

2. Quelles valeurs prend `i` dans `for i in range(1, 11)` ? (1 pt)**Les valeurs 1 à 10.**

→ La borne de droite est exclue : c'est l'erreur d'un débutant sur deux.

3. Quand emploie-t-on une boucle `while` ? (1 pt)**Quand le nombre de tours n'est pas connu : on répète tant qu'une condition reste vraie.**

→ Il faut garantir que la condition finira par devenir fausse.

4. Que se passe-t-il si cette condition ne peut jamais devenir fausse ? (1 pt)

Le programme tourne indéfiniment, sans message d'erreur.

→ C'est la seule erreur qui ne produit rien : ni affichage, ni plantage.

Exercice 3 — Fonctions

4 pts 10 min

Écrire et comprendre une fonction

Réponds à chaque question.

LA RÈGLE DU COURS

Deux instructions ont l'air voisines et ne font pas du tout la même chose. L'une permet de réutiliser un résultat dans un calcul ; l'autre le montre et l'oublie.

1. Qu'est-ce qu'une fonction, en programmation ? (1 pt)

Un traitement regroupé sous un nom, avec des paramètres en entrée.

→ Les paramètres sont les valeurs qu'on lui fournit.

2. Que fait l'instruction ``return`` ? (1 pt)

Elle renvoie une valeur à celui qui a appelé la fonction, et interrompt son exécution.

→ L'interruption fait partie de son effet.

3. Que fait ``print``, et en quoi diffère-t-elle ? (1 pt)

Elle affiche, sans rien renvoyer.

→ Une fonction qui affiche sans renvoyer ne peut pas être réutilisée dans un calcul.

4. Quelle question se pose avant de rendre une fonction ? (1 pt)

Renvoie-t-elle avec ``return``, ou se contente-t-elle d'afficher ?

→ C'est la faute qui rend une fonction juste mais inutilisable.

Exercice 4 — Les algorithmes

5 pts 14 min

Connaître les algorithmes du programme

Réponds à chaque question.

LA RÈGLE DU COURS

Trois algorithmes reviennent, et chacun a son piège propre : une initialisation, un critère d'arrêt, et l'endroit exact où placer une conclusion.

1. Avec quoi initialise-t-on la recherche d'un maximum, et pourquoi pas avec 0 ? (1 pt)

Avec le premier élément de la liste ; une liste de nombres négatifs donnerait sinon un maximum faux.

→ Zéro n'appartient pas forcément à la liste.

2. Quel est le principe de l'algorithme d'Euclide ? (1 pt)

On remplace le plus grand par le reste de sa division par le plus petit, et on recommence jusqu'à ce que le reste soit nul.

→ Le PGCD est alors le dernier diviseur employé.

3. Déroule Euclide sur 48 et 18 : donne les restes successifs. (1 pt)

$48 - 2 \times 18 = 12$, puis $18 - 12 = 6$, puis $12 - 2 \times 6 = 0$.

→ Le reste devient nul au troisième tour.

4. Quel est le PGCD de 48 et 18 ? (1 pt)

6

→ C'est le dernier diviseur employé avant que le reste ne s'annule.

5. Jusqu'où suffit-il de tester les diviseurs pour savoir si n est premier ? (1 pt)

Jusqu'à $\sqrt{n}$.

→ Si n a un diviseur supérieur à sa racine, il en a nécessairement un inférieur.

Repérer une erreur et la corriger

Chaque ligne est fautive. Écris ce qu'il fallait écrire.

LA RÈGLE DU COURS

Trois fautes reviennent. Un test écrit avec le signe de l'affectation. Une conclusion placée à l'intérieur de la boucle. Et l'absence de message d'erreur prise pour une preuve.

1. « `if n = 3:` » (1 pt)

``if n == 3:``

→ Un seul signe égal range une valeur ; dans un test, on compare.

2. « Dans le test de primalité, on écrit `return True` à l'intérieur de la boucle. » (1 pt)

Il va à l'extérieur, aligné avec la boucle.

→ À l'intérieur, la fonction déclarerait 9 premier dès le premier diviseur qui ne convient pas.

3. « Mon programme s'exécute sans message d'erreur : il est donc juste. » (1 pt)

Il faut le tester sur un cas dont on connaît déjà la réponse.

→ Une boucle infinie ne produit aucun message, et un résultat faux non plus.

Barème — ce que le correcteur regarde

Les lignes chiffrées couvrent les 5 exercices et totalisent 20 points.

EXERCICE	ATTENDU	POINTS	POINT PERDU SI
Ex.1	Variable, affectation et indentation	4 points — 1 par question	Une affectation lue comme une égalité mathématique.
Ex.2	Les deux boucles et leurs bornes	4 points — 1 par question	Une borne de droite comptée comme incluse.
Ex.3	Fonctions : renvoyer et afficher	4 points — 1 par question	Une fonction qui affiche sans rien renvoyer, jugée réutilisable.
Ex.4	Les algorithmes à connaître	5 points — 1 par question	Un maximum initialisé à zéro.
Ex.5	L'erreur repérée, et la réponse juste donnée	3 points — 1 par ligne	Une conclusion tirée avant d'avoir tout testé.

ET MAINTENANT ?

16 à 20

Le chapitre est tenu. Entraîne-toi maintenant sur le seul souci d'efficacité du programme : tester les diviseurs jusqu'à la racine plutôt que jusqu'à $n - 1$. Pour un million, on passe d'un million de tests à mille.

10 à 15

Refais l'exercice 4 en déroulant chaque algorithme à la main sur un cas dont tu connais la réponse. C'est ce que fait un programmeur avant même de lancer le programme.

moins de 10

Reprends la fiche de cours et retiens la distinction qui commande tout le reste : un `=` range, un `==` demande. Dans un test, on demande toujours.