# 2nde -Algorithmique & programmation (Python)

Édition élève - exercices seuls

**Thèmes :** variables, affectations, conditions, boucles, fonctions, calcul numérique, géométrie algorithmique, simulations et probabilités.
**Notation :** les fractions mathématiques sont composées avec `\dfrac` pour une meilleure lisibilité.

## Exercice 1 - Variables, types, conversions - pièges de input()

On exécute ce script :

```
age = input("Age ? ")
x = input("Nombre réel ? ")
print(age + 1)
print(x * 2)
```

(a) Expliquer précisément l'erreur provoquée par `print(age + 1)`.

(b) Si l'utilisateur saisit `3.5` pour `x`, indiquer exactement ce qu'affiche `print(x * 2)`.

(c) Corriger le script pour afficher l'âge augmenté de 1 et le double de $x$ comme des nombres.

(d) Donner une entrée pour laquelle `int(...)` échoue et expliquer pourquoi.

## Exercice 2 - Affectation, priorités, // et % - lecture de trace

On pose `a = 17` et `b = 5`.

(a) Calculer `a + b * 2` puis `(a + b) * 2`.

(b) Calculer `a / b`, `a // b` et `a % b`, puis relier les deux derniers résultats à la division euclidienne.

(c) Faire la trace de `a = a + b; b = a - b; a = a - b` et donner les valeurs finales.

(d) Expliquer ce que réalise `a, b = b, a` et pourquoi cette écriture est préférable en Python.

## Exercice 3 - Tests - intervalles et logique booléenne

Écrire des conditions Python correspondant exactement aux ensembles indiqués.

(a) Tester si $x \in [-2\,;\,5[$.

(b) Tester si $x \notin [0\,;\,1]$.

(c) Tester si $x \in ]-\infty\,;\,-3] \cup [2\,;\,+\infty[$.

(d) Tester si $x \in [1\,;\,4]$ et $x \neq 3$.

## Exercice 4 - Logique - lois de De Morgan

On considère la condition :

```
not((x >= 0 and x <= 1) or (x == 3))
```

(a) Réécrire la condition sans `not`, uniquement avec `and`, `or`, comparaisons et `\char33 =`.

(b) Décrire l'ensemble des réels qui valident cette condition.

(c) Tester mentalement la condition pour $x = 0{,}5$, $x = 2$, $x = 3$ et $x = -1$.

(d) Écrire une version lisible utilisant deux variables booléennes intermédiaires.

## Exercice 5 - Boucle while - seuil et arrêt garanti

On définit $u_0 = 1$ et $u_{n+1} = 1{,}15u_n + 2$.

(a) Écrire un programme qui calcule le plus petit entier $n$ tel que $u_n \geq 100$.

(b) Calculer $u_0, u_1, u_2, u_3$.

(c) Justifier rigoureusement, à un niveau de 2nde, que la boucle finit.

(d) Donner le couple $(n, u_n)$ obtenu à l'arrêt, avec $u_n$ arrondi au millième.

## Exercice 6 - Boucle for - somme de carrés et vérification

On veut calculer $S = 1^2 + 2^2 + \cdots + n^2$.

(a) Écrire un programme utilisant `for` qui calcule $S$ pour un entier naturel $n$.

(b) Calculer $S$ à la main pour $n = 5$.

(c) Vérifier pour $n = 5$ la formule $S = \dfrac{n(n+1)(2n+1)}{6}$.

(d) Écrire un programme qui vérifie automatiquement cette formule pour tous les entiers de 1 à 50.

## Exercice 7 - Division euclidienne - convertir une durée

On dispose d'une durée entière `t = 9876` secondes. On veut l'écrire sous la forme heures, minutes, secondes.

(a) Calculer le nombre entier d'heures complètes.

(b) Calculer le nombre de minutes complètes restant après retrait des heures.

(c) Calculer le nombre de secondes restantes et donner la décomposition de 9876 s.

(d) Écrire une fonction `decompose(t)` qui renvoie les trois valeurs, en refusant une durée négative.

## Exercice 8 - Fonction par morceaux - tarif progressif

Un service facture une consommation $x \geq 0$ selon le tarif suivant : $2$ € par unité jusqu'à 10 unités ; puis $1{,}5$ € par unité supplémentaire jusqu'à 25 ; puis $1{,}2$ € par unité supplémentaire au-delà.

(a) Écrire une fonction Python `prix(x)` qui renvoie `None` si $x < 0$.

(b) Calculer `prix(8)` et `prix(18)`.

(c) Calculer `prix(30)` en justifiant la continuité des paliers.

(d) Expliquer pourquoi, dans un bloc `if/elif/else`, il suffit de tester `x <= 10`, puis `x <= 25`.

## Exercice 9 - Fonction à plusieurs arguments - distance dans le plan

On rappelle que la distance entre $A(x_A, y_A)$ et $B(x_B, y_B)$ vaut $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.

(a) Écrire une fonction `distance(xA, yA, xB, yB)` utilisant `math.sqrt`.

(b) Calculer exactement la distance entre $A(-2; 3)$ et $B(4; -5)$.

(c) Vérifier que la fonction renvoie 10 pour ces coordonnées.

(d) Expliquer pourquoi la distance ne change pas si l'on échange A et B.

## Exercice 10 - Algorithme d'Euclide - PGCD et invariant

On veut calculer le PGCD de deux entiers positifs avec l'algorithme d'Euclide.

```
def pgcd(a, b):
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
```

(a) Faire la trace pour `a=9999` et `b=3663`.

(b) Donner le PGCD obtenu.

(c) Expliquer pourquoi remplacer $(a, b)$ par $(b, a\%b)$ conserve les diviseurs communs.

(d) Adapter la fonction pour accepter des entrées négatives.

## Exercice 11 - Simulation - pièce équilibrée puis biaisée

On simule des lancers indépendants d'une pièce et on estime la fréquence de Pile.

(a) Écrire une fonction `freq_pile(N)` pour une pièce équilibrée.

(b) Expliquer pourquoi le résultat n'est généralement pas exactement $0{,}5$.

(c) Modifier la fonction pour une pièce donnant Pile avec probabilité $p = 0{,}62$.

(d) Ajouter une protection pour `N <= 0` et expliquer l'effet d'un grand N.

## Exercice 12 - Simulation - au moins un 6 en quatre lancers

On lance un dé équilibré quatre fois et on s'intéresse à l'événement « obtenir au moins un 6 ».

(a) Écrire une simulation de N expériences qui estime cette probabilité.

(b) Calculer la probabilité théorique exacte avec l'événement contraire.

(c) Donner une valeur décimale approchée au dix-millième.

(d) Expliquer ce que l'on doit observer pour un grand nombre d'expériences, par exemple `N=50000`.

## Exercice 13 - Boucles imbriquées - compter exactement les passages

On considère le programme :

```
c = 0
for i in range(n):
    for j in range(i, n):
        c += 1
```

(a) Pour $n = 4$, lister les couples $(i, j)$ parcourus.

(b) Exprimer le nombre final `c` en fonction de $n$.

(c) Calculer `c` pour $n = 10$.

(d) Comparer approximativement le nombre de passages quand on remplace $n$ par $2n$.

## Exercice 14 - Calcul itératif - terme d'une suite et somme partielle

On définit $u_0 = 2$ et $u_{n+1} = 2u_n + 1$.

(a) Écrire une fonction `terme(n)` qui calcule $u_n$.

(b) Calculer $u_1, u_2, u_3$.

(c) Écrire une fonction qui calcule $S_n = u_0 + u_1 + \cdots + u_n$, puis donner $S_3$.

(d) Conjecturer une expression de $u_n$ et la vérifier en la remplaçant dans la relation de récurrence.

## Exercice 15 - Boucle while - premier entier dépassant un seuil

On cherche le plus petit entier $n$ tel que $1 + 2 + \cdots + n \geq 1000$.

(a) Écrire un programme `while` qui calcule ce plus petit n.

(b) Expliquer pourquoi la boucle s'arrête.

(c) Utiliser $1 + 2 + \cdots + n = \dfrac{n(n+1)}{2}$ pour prévoir la valeur de n.

(d) Vérifier la minimalité en comparant les sommes aux rangs $n - 1$ et $n$.

## Exercice 16 - Débogage - erreur de borne dans range

Le but de la fonction suivante est de calculer $1 + 2 + \cdots + n$, mais elle est fausse :

```
def somme(n):
    S = 0
    for k in range(n):
        S = S + k
    return S
```

(a) Tester mentalement la fonction pour $n = 1, 2, 3$.

(b) Identifier précisément l'erreur.

(c) Donner deux corrections différentes de la boucle.

(d) Ajouter une validation qui renvoie 0 si $n < 1$.

## Exercice 17 - Multiples et diviseurs - fonctions booléennes

On travaille avec des entiers. L'opérateur `%` permet de tester une divisibilité.

(a) Écrire `est_multiple(a, b)` qui indique si a est un multiple de b, en refusant `b=0`.

(b) Utiliser la fonction pour décider si 357 est un multiple de 7.

(c) Écrire une fonction qui renvoie le plus grand multiple positif de a inférieur ou égal à b, avec $a > 0$ et $b \geq 0$.

(d) Déterminer le plus grand multiple de 7 inférieur ou égal à 100 et expliquer le calcul.

## Exercice 18 - Nombres premiers - test par divisions successives

Un entier $n \geq 2$ est premier s'il n'a pas de diviseur autre que 1 et lui-même.

(a) Écrire une fonction `est_premier(n)` utilisant une boucle `while`.

(b) Expliquer pourquoi il suffit de tester jusqu'à ce que $d^2 > n$.

(c) Tester mentalement 1, 2, 9, 17 et 221.

(d) Optimiser le programme en traitant 2 séparément puis en ne testant que les diviseurs impairs.

## Exercice 19 - Statistiques - lire une fonction à cinq valeurs

On considère cinq valeurs réelles $x_1, \ldots, x_5$. On veut calculer leur moyenne, leur variance de population et leur écart-type sans utiliser de liste.

```
from math import sqrt

def stats5(a, b, c, d, e):
    m = (a+b+c+d+e)/5
    V = ((a-m)**2 + (b-m)**2 + (c-m)**2
         + (d-m)**2 + (e-m)**2)/5
    return m, V, sqrt(V)
```

(a) Expliquer ce que représentent les trois valeurs renvoyées.

(b) Calculer la moyenne pour $(2, 4, 4, 4, 6)$.

(c) Calculer exactement la variance pour ces cinq valeurs.

(d) Donner l'écart-type approché au millième et interpréter son rôle.

## Exercice 20 - Flottants - comparaison à une précision donnée

On exécute :

```
S = 0.0
for k in range(10):
    S += 0.1
print(S)
```

(a) Donner le résultat attendu en mathématiques exactes.

(b) Expliquer pourquoi Python peut afficher une valeur comme `0.9999999999999999`.

(c) Écrire un test robuste pour vérifier que S est égal à 1 à $10^{-9}$ près.

(d) Proposer une méthode évitant l'accumulation d'erreurs pour ce calcul particulier.

## Exercice 21 - Balayage - encadrer $\sqrt{2}$ au millième

On veut encadrer $\sqrt{2}$ au millième sans utiliser `sqrt`. On cherche un entier n tel que $(n/1000)^2 \leq 2 < ((n+1)/1000)^2$.

(a) Écrire une boucle `while` qui détermine n avec uniquement des calculs entiers.

(b) Donner la valeur de n obtenue.

(c) En déduire un encadrement de $\sqrt{2}$ au millième.

(d) Expliquer pourquoi tester `n*n <= 2000000` évite les flottants pendant la recherche.

## Exercice 22 - Boucle while - méthode de Babylone pour $\sqrt{a}$

Pour $a > 0$, on définit $x_0 = a$ et $x_{n+1} = \frac{1}{2}\left(x_n + \frac{a}{x_n}\right)$.

(a) Écrire une fonction qui s'arrête quand deux approximations successives diffèrent de moins de $10^{-6}$.

(b) Calculer $x_1$ et $x_2$ pour $a = 2$ à quatre décimales.

(c) Expliquer le rôle du seuil `eps`.

(d) Ajouter les validations nécessaires pour `a <= 0` ou `eps <= 0`.

## Exercice 23 - Dichotomie - approximer $\sqrt{2}$

On sait que $1 < \sqrt{2} < 2$. On coupe répétitivement l'intervalle en deux et on conserve la moitié contenant $\sqrt{2}$.

(a) Écrire une fonction `racine2_dicho(eps)` qui renvoie le milieu d'un intervalle de largeur inférieure à eps.

(b) Donner les trois premiers intervalles obtenus à partir de `[1,2]`.

(c) Expliquer pourquoi le test `m*m < 2` permet de choisir la bonne moitié.

(d) Indiquer la valeur approchée obtenue pour `eps=1e-6`.

## Exercice 24 - Premier rang - puissance dépassant 2

Un capital est multiplié chaque année par $1{,}07$. On cherche le plus petit entier n tel que $1{,}07^n > 2$.

(a) Écrire une boucle `while` qui calcule ce plus petit n sans utiliser de logarithme.

(b) Donner la valeur de n.

(c) Vérifier numériquement la minimalité avec les puissances aux rangs n-1 et n.

(d) Adapter le programme en une fonction `premier_rang(q, seuil)` avec validations.

## Exercice 25 - Simulation - somme de deux dés au moins égale à 10

On lance deux dés équilibrés et indépendants. On note S la somme des deux résultats.

(a) Écrire une simulation de N expériences estimant $P(S \geq 10)$.

(b) Déterminer exactement les couples donnant une somme au moins égale à 10.

(c) Calculer la probabilité théorique exacte.

(d) Expliquer comment utiliser la théorie pour contrôler la cohérence d'une simulation.

## Exercice 26 - Géométrie algorithmique - tester l'alignement de trois points

Pour $A(x_A, y_A)$, $B(x_B, y_B)$ et $C(x_C, y_C)$, on peut tester l'alignement avec le déterminant $D = (x_B - x_A)(y_C - y_A) - (y_B - y_A)(x_C - x_A)$.

(a) Écrire une fonction `alignes(xA,yA,xB,yB,xC,yC)` qui renvoie un booléen.

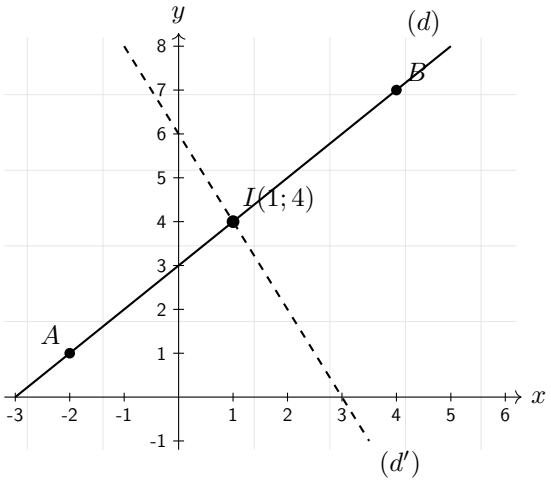
(b) Tester $A(1;2)$, $B(4;5)$ et $C(7;8)$.

(c) Tester les mêmes A et B avec $D(7;9)$.

(d) Expliquer pourquoi cette méthode est préférable au calcul et à la comparaison de deux pentes.

## Exercice 27 - Équations de droites - coefficient directeur et intersection

On considère la droite $(d)$ passant par $A(-2;1)$ et $B(4;7)$, ainsi que la droite $(d')$ d'équation $y = -2x + 6$.




(a) Calculer le coefficient directeur de $(d)$ puis déterminer une équation de $(d)$.

(b) Écrire une fonction qui, pour deux points d'abscisses différentes, renvoie les coefficients m et p de $y = mx + p$.

(c) Calculer exactement les coordonnées du point d'intersection I de $(d)$ et $(d')$.

(d) Vérifier par substitution que I appartient aux deux droites et identifier ce point sur la figure.

## Exercice 28 - Balayage numérique - rechercher un maximum

On étudie $f(x) = -x^2 + 6x + 1$ sur $[0;6]$. On veut approcher le maximum par balayage avec un pas h.

(a) Écrire une fonction `maximum_balayage(h)` qui teste les points $0, h, 2h, \ldots, 6$.

(b) Avec $h = 0{,}5$, déterminer le point où le maximum est trouvé.

(c) Vérifier algébriquement la valeur exacte du maximum en écrivant $f(x) = 10 - (x-3)^2$.

(d) Expliquer l'effet d'un pas plus petit sur la précision et le nombre de calculs.

## Exercice 29 - Approximation géométrique - longueur d'une courbe

On approxime la longueur de la courbe $y = x^2$ entre $x = 0$ et $x = 2$ par une ligne brisée reliant n points régulièrement espacés.

(a) Écrire une fonction `longueur(n)` qui additionne les distances entre points consécutifs.

(b) Pour $n = 4$ sous-intervalles, donner les cinq abscisses utilisées.

(c) Calculer la longueur approchée pour $n = 4$ à $10^{-4}$ près.

(d) Expliquer pourquoi augmenter n améliore en général l'approximation.

## Exercice 30 - Simulation conditionnelle - arbre à deux étapes

Une population contient 40 % d'individus du groupe A et 60 % du groupe B. Un test est positif avec probabilité 0,7 dans A et 0,3 dans B.

(a) Écrire une simulation qui génère d'abord le groupe puis le résultat du test et estime $P(A \mid T)$, où T signifie « test positif ».

(b) Calculer exactement $P(T)$.

(c) Calculer exactement $P(A \mid T)$.

(d) Expliquer pourquoi, dans la simulation, il faut diviser le nombre de cas « A et T » par le nombre total de cas T, et non par N.