

Baccalauréat général – spécialité mathématiques

Algorithmique et programmation

Décrire une méthode, l'automatiser et vérifier son résultat

Langage naturel, Python, fonctions et listes

Objectifs

- Lire un algorithme, suivre les valeurs des variables et repérer le résultat renvoyé.
- Écrire un programme avec affectations, conditions, boucles et fonctions.
- Générer, parcourir et filtrer une liste en distinguant valeurs et indices.
- Tester un programme sur des cas simples et relier son résultat à la question mathématique.

Cadre du cours. On vise la session 2027 du baccalauréat général, spécialité mathématiques. Le programme de référence est celui du Bulletin officiel spécial n° 8 du 25 juillet 2019. Il demande la consolidation des notions de programmation déjà rencontrées et la programmation modulaire ; les exemples ci-dessous utilisent Python, mais le raisonnement reste indépendant d'un logiciel particulier. Les exercices sont créés pour ce cours et ne sont pas des sujets officiels.

Organisation conseillée. Prévoir cinq séances de 35 à 45 minutes :

1. diagnostic, variables, affectation et lecture d'un algorithme ;
2. conditions et boucles ;
3. fonctions et listes ;
4. recherche de seuil et simulation ;
5. exercices de synthèse, tests et rappel différé.

Table des matières

1	Avant de commencer : diagnostic et rappels	3
2	1. Construire un algorithme	4
3	2. Fonctions et listes	6
4	3. Conditions, recherche et simulation	7
5	4. Exercices gradués	8
6	Bilan et rappel différé	10

1 Avant de commencer : diagnostic et rappels

Durée : 12 minutes, sans exécuter les programmes. Pour chaque question, écrire les étapes sur papier avant de chercher une commande.

- D1.** On exécute successivement $x \leftarrow 3$, puis $x \leftarrow x + 2$. Quelle est la valeur finale de x ?
- D2.** Que teste la condition `x >= 5` ?
- D3.** Quelles valeurs produit `range(4)` en Python ?
- D4.** Que renvoie une fonction qui contient `return 7` ?
- D5.** Dans la liste `[12, 8, 5]`, quel est l'indice du premier élément en Python ?
- D6.** Pourquoi une boucle qui ne modifie jamais sa condition d'arrêt peut-elle ne pas se terminer ?

1.1 Rappels ciblés

Une **variable** possède un nom et une valeur. Une affectation remplace la valeur précédente : $x \leftarrow x + 1$ signifie « calculer l'ancienne valeur de x plus 1, puis la ranger dans x ». Ce n'est pas une égalité algébrique à résoudre.

Une condition est une proposition vraie ou fausse. Une boucle répète un bloc selon une règle ; une fonction reçoit éventuellement des paramètres et renvoie un résultat avec **return**.

En Python, les indices d'une liste commencent à 0. Ainsi, dans `[12, 8, 5]`, l'indice 0 désigne 12, l'indice 1 désigne 8, et l'indice 2 désigne 5. La fonction `range(n)` produit $0, 1, \dots, n - 1$.

À retenir

Avant d'exécuter un programme, écrire : les valeurs initiales, la condition testée, le nombre de répétitions et la valeur renvoyée. La programmation est une traduction précise d'une méthode ; elle ne dispense pas d'expliquer ce que le résultat signifie.

2 1. Construire un algorithme

2.1 Entrées, traitement, sortie

Un algorithme décrit une suite finie d'actions. Pour le lire, repérer :

1. les **entrées** fournies au départ ;
2. les **variables** intermédiaires ;
3. les **conditions** qui choisissent un chemin ;
4. les **boucles** qui répètent une action ;
5. la **sortie** affichée ou renvoyée.

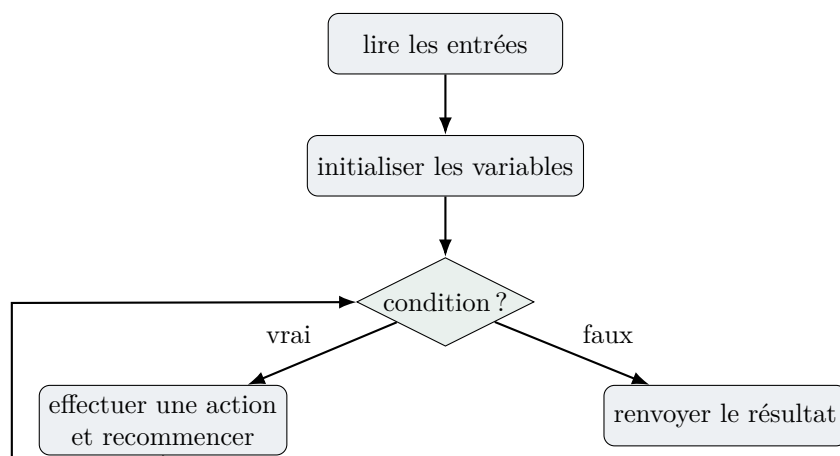


FIGURE 1 – Un schéma courant : initialiser, tester, agir, puis renvoyer.

2.2 Affectation, condition et boucle

En langage naturel, on utilise souvent la flèche

`somme ← somme + valeur.`

En Python, cette affectation s'écrit avec le signe égal. Une condition `if` choisit un bloc, et une boucle `for` répète sur une collection ou une suite d'indices.

Exemple entièrement détaillé

On veut calculer $1 + 2 + \dots + n$, pour un entier $n \geq 1$.

Méthode en langage naturel :

Entrée : un entier n
`somme ← 0`
 Pour k allant de 1 à n
 `somme ← somme + k`
 Renvoyer `somme`

Traduction Python :

```
def somme_entiers(n):
```

```
somme = 0
for k in range(1, n+1):
    somme = somme + k
return somme
```

Pour $n = 4$, on suit les quatre passages :

k	ancienne somme	nouvelle somme
1	0	1
2	1	3
3	3	6
4	6	10

Le programme renvoie donc 10, ce qui correspond à $1 + 2 + 3 + 4 = 10$. Le test sur $n = 1$, qui renvoie 1, vérifie également la première valeur.

2.3 Un invariant simple

Après le passage d'indice k dans l'exemple, la variable `somme` contient exactement $1 + 2 + \dots + k$. Cette phrase, vraie avant et après chaque passage, explique pourquoi le résultat final est correct. Elle joue le rôle d'un contrôle du raisonnement.

3 2. Fonctions et listes

3.1 Découper une tâche

Une fonction informatique rassemble une tâche sous un nom réutilisable. Elle peut avoir zéro, un ou plusieurs paramètres. Les variables créées à l'intérieur sont locales à la fonction, et `return` transmet le résultat au programme appelant.

```
def carre(x):  
    return x * x
```

```
resultat = carre(3)
```

Après l'appel, `resultat` vaut 9. Une fonction ne doit pas seulement calculer : son nom, ses paramètres et son résultat doivent correspondre à la question.

3.2 Générer et parcourir une liste

Une liste est une collection ordonnée, dans laquelle une valeur peut apparaître plusieurs fois. On peut la créer en extension, par ajouts successifs ou en compréhension.

```
valeurs = [2, 5, 8]
```

```
carres = []  
for valeur in valeurs:  
    carres.append(valeur * valeur)
```

```
carres_bis = [valeur * valeur for valeur in valeurs]
```

Les deux dernières listes valent `[4, 25, 64]`. La méthode par compréhension est concise ; la boucle explicite est souvent plus facile à lire au début.

Pour parcourir une liste :

- `for valeur in valeurs` utilise directement les valeurs ;
- `for i in range(len(valeurs))` utilise les indices et permet d'écrire `valeurs[i]`.

La fonction `append` ajoute une valeur à la fin. La longueur est obtenue avec `len`.

Exemple entièrement détaillé

On veut calculer la moyenne d'une liste non vide `valeurs`.

```
def moyenne(valeurs):  
    total = 0  
    for valeur in valeurs:  
        total = total + valeur  
    return total / len(valeurs)
```

Pour `[10, 12, 14]`, la boucle construit successivement les totaux 10, 22, puis 36. La fonction renvoie

$$\frac{36}{3} = \boxed{12}.$$

La condition « liste non vide » est importante : diviser par `len(valeurs)` n'aurait pas de sens pour une liste de longueur nulle.

4 3. Conditions, recherche et simulation

4.1 Filtrer avec une condition

Une condition peut être combinée avec `and`, `or` et `not`. Pour construire une liste des valeurs positives :

```
def positives(valeurs):
    resultat = []
    for valeur in valeurs:
        if valeur > 0:
            resultat.append(valeur)
    return resultat
```

La liste produite conserve l'ordre d'origine et ne contient que les valeurs qui vérifient la condition.

4.2 Rechercher un premier rang

Pour chercher le premier entier satisfaisant une condition, on initialise un rang, on teste, puis on l'augmente tant que la condition n'est pas vérifiée.

```
def premier_seuil(condition, limite):
    n = 0
    while n <= limite and not condition(n):
        n = n + 1
    if n <= limite:
        return n
    return None
```

Dans une copie, il faut préciser ce que signifie `condition(n)` et prévoir le cas où aucun rang ne convient dans la limite choisie. La borne `limite` protège contre une boucle infinie.

4.3 Simuler une expérience

Pour une expérience aléatoire, une simulation fournit une réalisation du modèle. Par exemple :

```
from random import random

def compter_succes(n, p):
    compteur = 0
    for i in range(n):
        if random() < p:
            compteur = compteur + 1
    return compteur
```

La comparaison `random() < p` crée un succès de probabilité p dans le modèle usuel. Une simulation donne un résultat aléatoire ; elle ne remplace ni la formule ni la justification mathématique.

À retenir

Une fonction doit avoir un objectif vérifiable. Une boucle doit faire évoluer une variable ou parcourir une liste. Pour une recherche, indiquer la condition d'arrêt et le cas où la valeur recherchée n'existe pas. Pour une simulation, distinguer une réalisation, une fréquence observée et une probabilité théorique.

5 4. Exercices gradués

Les corrections détaillées sont dans le fascicule séparé.

Exercice 1 – Trace d’affectations

Compétence : suivre une variable. On part de $x = 4$, puis on exécute successivement $x \leftarrow 2x$, $x \leftarrow x - 3$, $x \leftarrow x^2$. Donner la valeur finale et les valeurs intermédiaires.

Indice.

Après chaque ligne, remplace l’ancienne valeur par la nouvelle.

Exercice 2 – Condition

Compétence : traduire une condition. Écrire en Python une fonction `signe_nombre(x)` qui renvoie 1 si $x > 0$, -1 si $x < 0$, et 0 sinon.

Indice.

Il faut trois cas : strictement positif, strictement négatif, puis le cas restant.

Exercice 3 – Boucle et liste

Compétence : parcourir une liste. Écrire une fonction qui reçoit une liste de nombres et renvoie la somme de ses éléments, puis donner le résultat pour $[3, -1, 4]$.

Indice.

Initialise une variable à zéro et ajoute chaque valeur avec une boucle `for`.

Exercice 4 – Fonction et compréhension

Compétence : générer une liste. Écrire une fonction qui reçoit une liste `valeurs` et renvoie la liste des doubles des valeurs strictement positives.

Indice.

Combine le filtre `valeur > 0` et le calcul `2 * valeur`.

Exercice 5 – Moyenne

Compétence : écrire une fonction modulaire. Compléter la fonction moyenne du cours pour qu’elle renvoie aussi `None` si la liste est vide. Tester mentalement sur `[]` et `[4, 6]`.

Indice.

Teste `len(valeurs) == 0` avant la boucle et la division.

Exercice 6 – Lecture d’une boucle

Compétence : déterminer la sortie d’un programme. Que renvoie le programme suivant ?

```
liste = []
for k in range(1, 5):
    if k % 2 == 0:
        liste.append(k)
```

Explique le rôle de `%`.

Indice.

`k % 2` est le reste de la division de k par 2.

Exercice 7 – Premier seuil

Compétence : écrire une recherche arrêtée. Écrire une fonction qui renvoie le premier entier n de 0 à 20 tel que $n^2 \geq 50$, et `None` si aucun ne convient.

Indice.

Utilise une boucle `while` avec la condition `n <= 20 and n*n < 50`.

Exercice 8 – Simulation et interprétation

Compétence : distinguer résultat simulé et probabilité. Écrire une fonction qui simule 1000 lancers d'une pièce équilibrée et renvoie la fréquence de piles. Que devrait-on observer sans exiger une valeur exacte ?

Indice.

Compte les succès de probabilité 0,5, puis divise par 1000.

6 Bilan et rappel différé

6.1 Erreurs fréquentes

- Confondre une affectation avec une équation à résoudre.
- Oublier que `range(n)` s'arrête à $n - 1$.
- Mélanger une valeur de liste et son indice.
- Diviser par la longueur d'une liste vide.
- Oublier `return`, ou afficher une valeur alors que la fonction doit la renvoyer.
- Écrire une boucle qui n'avance pas vers sa condition d'arrêt.
- Conclure qu'une simulation donne exactement la probabilité théorique.

6.2 Auto-évaluation

- ☐ Je sais suivre une affectation ligne après ligne.
- ☐ Je sais écrire une condition avec plusieurs cas.
- ☐ Je sais choisir entre parcourir les valeurs et parcourir les indices.
- ☐ Je sais écrire une fonction qui renvoie un résultat.
- ☐ Je sais contrôler l'arrêt d'une boucle et interpréter une simulation.

6.3 Rappel différé

R1. Que vaut `list(range(3))` en Python ?

R2. Quelle est la différence entre `return` et `print` dans une fonction ?

R3. Pourquoi une variable de somme est-elle souvent initialisée à zéro ?

R4. Citer un test simple permettant de vérifier une fonction de moyenne.

En cas de blocage. Écrire d'abord la méthode en français : « je reçois..., j'initialise..., je parcours..., je teste..., je renvoie... ». Puis traduire chaque verbe en instruction Python.