

Plaque 2 - Listes, fonctions et algorithmes élémentaires

Python et calcul scientifique — Licence 1 (L1)

Exercices

Méthode

On programme les algorithmes fondamentaux de l'arithmétique et du traitement de listes (moyenne, crible, Euclide, exponentiation rapide, tris, recherche dichotomique), en se posant à chaque fois trois questions : le programme **termine**-t-il ? est-il **correct** (invariant) ? combien d'**opérations** coûte-t-il (complexité) ? On valide toujours sur un exemple calculable à la main.

Exercice 1 - Moyenne, variance et indice du maximum

On dispose de la liste $notes = [12, 15, 9, 18, 15, 11]$.

- Écrire une fonction $moyenne(L)$ avec une boucle (sans utiliser sum).
- Écrire $variance(L)$ selon la définition $\frac{1}{n} \sum (x_i - \bar{x})^2$ et donner le résultat.
- Vérifier avec la formule de König-Huygens $V = \overline{x^2} - \bar{x}^2$.
- Écrire $argmax(L)$ renvoyant l'indice du premier maximum ; que renvoie-t-elle ici ?

Exercice 2 - Le crible d'Ératosthène

- Écrire une fonction $crible(n)$ renvoyant la liste des nombres premiers $\leq n$: on part d'une liste de booléens, puis pour chaque p premier on raye ses multiples à partir de p^2 .
- Justifier qu'il suffit de traiter les $p \leq \sqrt{n}$, et de rayer à partir de p^2 .
- Le programme donne 15 nombres premiers jusqu'à 50 et 168 jusqu'à 1000. Comparer à l'approximation $\frac{n}{\ln n}$.
- Estimer le nombre d'opérations « rayer » (on admet $\sum_{p \leq n} \frac{1}{p} \approx \ln \ln n$).

Exercice 3 - Euclide et Bézout

- Écrire $pgcd(a, b)$ (algorithme d'Euclide) qui renvoie aussi le nombre de divisions effectuées ; l'appliquer à (1071, 462).
- Justifier la terminaison et la correction de l'algorithme.
- Écrire une fonction **récursive** $bezout(a, b)$ renvoyant (d, u, v) avec $au + bv = d$. On utilisera : si $bu' + (a \bmod b)v' = d$, alors $au + bv = d$ avec $u = v'$, $v = u' - \lfloor a/b \rfloor v'$.
- Vérifier le résultat (21, -3, 7) obtenu pour (1071, 462).

Exercice 4 - Exponentiation rapide

- Écrire une fonction calculant x^n par l'algorithme d'exponentiation rapide (écriture binaire de n) et comptant les multiplications.
- Pour 3^{13} , le programme affiche (1594323, 7). Comparer au calcul naïf.
- Montrer que le nombre de multiplications est au plus $2\lfloor \log_2 n \rfloor + 2$.

d) La fonction intégrée `pow(3, 100, 7)` renvoie 4 : retrouver ce résultat à la main.

Exercice 5 - Écriture en base 2

- a) Écrire une fonction `binaire(n)` renvoyant la chaîne des chiffres binaires de n (divisions successives par 2).
- b) Le programme affiche `11111101010` pour $n = 2026$: vérifier en recomposant 2026.
- c) Combien de chiffres binaires a un entier $n \geq 1$? L'appliquer à 2026.
- d) Écrire la fonction réciproque par l'algorithme de Horner : lire les chiffres de gauche à droite en posant $r \leftarrow 2r + b$ pour chaque chiffre b .

Exercice 6 - Tri par insertion et comparaisons

- a) Écrire un tri par insertion qui compte les comparaisons effectuées.
- b) Le programme donne 12 comparaisons pour `[5, 2, 4, 6, 1, 3]`, 15 pour `[6, 5, 4, 3, 2, 1]` et 5 pour `[1, 2, 3, 4, 5, 6]`. Expliquer les deux cas extrêmes.
- c) En déduire la complexité dans le pire et le meilleur cas pour une liste de taille n .
- d) Quel invariant garantit la correction ?

Exercice 7 - Recherche dichotomique

Soit $L = \text{list}(\text{range}(0, 2000, 2))$ (les 1000 premiers pairs).

- a) Écrire une recherche dichotomique renvoyant l'indice de x (ou -1) et le nombre d'étapes.
- b) Le programme renvoie `(617, 10)` pour $x = 1234$ et `(-1, 10)` pour $x = 1235$; le maximum sur tous les éléments est 10. Justifier.
- c) Combien d'étapes au plus pour une liste d'un million d'éléments ? Et pour une recherche séquentielle ?
- d) Pourquoi la liste doit-elle être triée ?

Exercice 8 - Fibonacci : récursif contre itératif

- a) Écrire `fib_rec(n)` (récursive, selon $F_n = F_{n-1} + F_{n-2}$) en comptant les appels, et `fib_it(n)` (itérative, avec deux variables).
- b) Pour $n = 20$, les deux renvoient 6765, mais la version récursive fait 21 891 appels. Montrer que le nombre d'appels A_n vérifie $A_n = 1 + A_{n-1} + A_{n-2}$, puis $A_n = 2F_{n+1} - 1$.
- c) En déduire la complexité de la version récursive.
- d) `fib_it(100)` renvoie `354224848179261915075` instantanément : pourquoi Python donne-t-il un résultat exact ?

Exercice 9 - Évaluer un polynôme : Horner

Soit $P(x) = 2x^3 - 3x^2 + 5x - 7$, représenté par la liste `[2, -3, 5, -7]` (coefficients du plus haut degré au plus bas).

- a) Écrire `horner(coeffs, x)` fondée sur $P(x) = ((2x - 3)x + 5)x - 7$.

- b) Le programme renvoie 7 pour $x = 2$ et 0.5 pour $x = 1,5$: vérifier à la main.
- c) Comparer le nombre de multiplications avec le calcul direct $\sum a_k x^k$ (puissances recalculées) pour un polynôme de degré n .
- d) Montrer que les valeurs intermédiaires de Horner en $x = a$ sont les coefficients du quotient de P par $X - a$.

Exercice 10 – Triangle de Pascal en listes

- a) Écrire *pascal*(n) renvoyant la ligne n du triangle de Pascal, en construisant chaque ligne à partir de la précédente.
- b) Le programme donne $[1, 6, 15, 20, 15, 6, 1]$ pour $n = 6$. Justifier la construction par la relation de Pascal.
- c) Il affiche aussi $\text{sum}(\text{pascal}(10)) = 1024$ et $\text{pascal}(10)[5] = 252$: interpréter.
- d) Combien d'additions faut-il pour obtenir la ligne n ? Comparer au calcul de $\binom{n}{k}$ par factorielles.

Exercice 11 – Décomposition en facteurs premiers

- a) Écrire *facteurs*(n) renvoyant la liste des facteurs premiers de n (avec multiplicité) par divisions successives, en s'arrêtant quand $d^2 > n$.
- b) Le programme renvoie $[2, 2, 2, 3, 3, 5]$ pour 360, $[2, 1013]$ pour 2026, $[7, 11, 13]$ pour 1001. Justifier que les diviseurs trouvés sont premiers.
- c) Pourquoi le reste $n > 1$ en fin de boucle est-il premier ?
- d) En déduire le nombre de diviseurs de 360.

Exercice 12 – Produit de matrices avec des listes

Une matrice est représentée par la liste de ses lignes.

- a) Écrire *produit*(A, B) calculant AB par trois boucles imbriquées.
- b) Le programme renvoie $[[7, 10], [15, 22]]$ pour A^2 avec $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$: vérifier.
- c) Combien de multiplications pour le produit de deux matrices $n \times n$? Estimer la durée pour $n = 1000$ à 10^8 opérations par seconde.
- d) Avec $F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$, le programme donne $F^2 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$. Comment calculer F^{100} en $O(\log 100)$ produits ?

Exercice 13 – La suite de Syracuse

Pour $n \geq 1$: si n est pair on le divise par 2, sinon on le remplace par $3n + 1$; on s'arrête à 1.

- a) Écrire *syracuse*(n) renvoyant le nombre d'étapes et la valeur maximale atteinte.
- b) Le programme renvoie (16, 52) pour 7 et (111, 9232) pour 27. Vérifier les premières étapes pour 7.
- c) Parmi $1 \leq n < 100$, l'entier qui demande le plus d'étapes est 97. Quel est le

problème mathématique posé par la terminaison de ce programme ?

d) Proposer une modification garantissant la terminaison.

Exercice 14 – Nombres parfaits

Un entier $n \geq 2$ est **parfait** s'il est égal à la somme de ses diviseurs stricts ($6 = 1 + 2 + 3$).

a) Écrire `somme_diviseurs(n)` en parcourant les $d \leq \sqrt{n}$ et en ajoutant d et $\frac{n}{d}$.

b) Pourquoi faut-il traiter à part le cas $d = \frac{n}{d}$?

c) Le programme trouve $[6, 28, 496, 8128]$ jusqu'à 10 000. Vérifier que 28 est parfait et que ces nombres s'écrivent $2^{p-1}(2^p - 1)$.

d) Comparer le coût de la recherche jusqu'à N avec et sans l'astuce $d \leq \sqrt{n}$.

Exercice 15 – Synthèse : vérifier la conjecture de Goldbach

La conjecture de Goldbach affirme que tout entier pair $n \geq 4$ est somme de deux nombres premiers.

a) En utilisant `crible(100)` (exercice 2), écrire `decompositions(n)` renvoyant la liste des couples (p, q) premiers, $p \leq q$, avec $p + q = n$.

b) Le programme renvoie pour 100 : $[(3, 97), (11, 89), (17, 83), (29, 71), (41, 59), (47, 53)]$. Vérifier deux couples et expliquer pourquoi p parcourt seulement les premiers $\leq \frac{n}{2}$.

c) Écrire une ligne vérifiant la conjecture pour tous les pairs de 4 à 100 (le programme affiche `True`).

d) Pourquoi ce programme ne **démontre** pas la conjecture ? Que prouve-t-il exactement ?