Programmation sous Python – 2025-2026

Contrôl continu
26 mai, 2026 — 9 :30-11 :30

À lire. Quelques consignes importantes pour la notation :

- Les scripts doivent être commentés et syntaxiquement corrects : un script qui génère une erreur de syntaxe est systématiquement sanctionné (pas de "petite croix rouge" sous Spyder !)
- Internet est autorisé ainsi que vos travaux en séances. Les échanges de mails et l' utilisation d'intelligences artificielles (type ChatGPT) sont interdits.
- Le non-respect des consignes est aussi pénalisé (si on demande une fonction `somme(n)`, on ne veut pas trouver une fonction `SOMME(n)` ou encore `Somme(N)` comme réponse).
- Votre script `cc2_25-26.py` est à déposer sur Moodle dans l'espace dédié à votre groupe dans `UE 4~: Programmation sous Python`.
- Le barème est sur 26 : **il n'est pas nécessaire de tout faire pour avoir la note maximale**. Les question avec (*) sont plus difficiles et chacune vaut un point.

Exercice 1. On pourra utiliser les librairies, `math`, `numpy` et `mathplotlib.pyplot`.

Nous voulons représenter la multiplication par p modulo n. Dans un premier temps, on pose $p = 2$ et $n = 10$.

1) Tracer sur un même graphe, les n points $M_0, \dots, M_{n-1}$ sur le cercle unité qui ont comme coordonnées :

$$M_k = (\cos(2k\pi/n), \sin(2k\pi/n)).$$

2) Pour chaque k, relier le point M_k avec le point $M_{[2k]_n}$ où $[2k]_n$ est le reste de la division de $2k$ par n. Par exemple pour $n = 10$, on reliera

$$M_1 \text{ avec } M_2, \quad M_2 \text{ avec } M_4, \quad \dots \quad M_9 \text{ avec } M_{[18]_{10}} = M_8.$$

Il faut utiliser `plt.axis('equal')` pour avoir des axes orthonormés.

3) Refaire le point précédent en prenant $p = 3$ (n est toujours 10). Étudier le changement de la figure.

4) Trouver l'inverse de 3 modulo n, c'est-à-dire trouver u tel que $3u \equiv 1 \mod n$.

5*) Dans cette question, on considère p et n généraux. Sous quelle condition p a un inverse modulo n ? Trouver la condition graphique et, dans *un commentaire affiché sur le terminal*,

(a) énoncer cette condition
(b) esquisser une preuve.

Exercice 2. On veut trouver, s'il existe, un nombre entier $N \geq 2$ pour lequel la somme des cubes de ces chiffres (en base 10) soit égale à N.

1) Écrire une fonction python `chiffres` qui prend en argument un entier N et renvoie la liste des chiffres de l'écriture de N en base 10. (*Indication :* On pourrait utiliser la division euclidienne de N par 10.)

2) Écrire une fonction python `laSomme` qui prend en argument un entier N et renvoie la somme des cubes des chiffres de son écriture en base 10.

3) Déterminer les deux premiers nombres entiers $N \geq 2$ tels que `laSomme`$(N) = N$.

4) Refaire les trois points précédents en remplaçant la somme des cubes par la somme des puissances quatrième.

5*) Si on remplace la somme des cubes par la somme des carrés, montrer qu'il n'existe pas de $N \geq 2$ tel que `laSomme`$(N) = N$. On demande un argument mathématique qui justifiera les choix faits dans la réponse `python`. L'argument devra apparaître dans *un commentaire affiché sur le terminal*.

Exercice 3. Une **marche aléatoire** sur $\mathbb{Z}$ est un processus qui produit une suite de positions $(X_0, X_1, \ldots, X_n)$ où :

— $X_0 = 0$ (point de départ),

— à chaque étape k, on avance d'un pas, $X_{k+1} = X_k + 1$, ou on recule d'un pas $X_{k+1} = X_k - 1$ de manière aléatoire équiprobable.

1) Écrire une fonction `marche(n)` qui simule une marche aléatoire de n pas et renvoie la liste des positions successives $(X_0, X_1, \ldots, X_n)$. (*Indication:* `rd.choice([-1, 1])` renvoie -1 ou $+1$ avec équiprobabilité, après avoir chargé la bibliothèque `random` avec `import random as rd`.)

2) Écrire une fonction `afficher_marche(n)` qui trace la trajectoire d'une marche aléatoire de n pas à l'aide de `matplotlib` (en faisant apparaître n dans le titre de l'image). Tester avec $n = 200$ et $n = 1000$.

3) Modifier `afficher_marche` pour superposer 5 trajectoires indépendantes sur le même graphique (avec des couleurs différentes), afin de visualiser la variabilité des marches.

4) On dit que la marche (à n pas) **revient à l'origine** (ou au point de départ) si $X_k = 0$ pour au moins un indice $k \in \{1, 2, \ldots, n\}$.

a) Écrire une fonction `revient_origine(positions)` qui prend en argument la liste des positions d'une marche à n pas et renvoie l'entier r_n, où r_n est **le premier** k pour lequel $X_k = 0$ si la marche revient à l'origine et $n + 1$ si la marche ne revient pas à l'origine.

b) Écrire une fonction `premiers_retours(n, N)` qui effectue N simulations de marches aléatoires de n pas, et renvoie la liste des N entiers r_n donnés par `revient_origine`.

c)* Pour $N = 10\,000$ simulations, calculer la liste P_n donnée par `premiers_retours(n, N)` pour les valeurs suivantes de n :

$$n \in \{10,\ 50,\ 100,\ 500,\ 1000,\ 5000\}.$$

Pour chaque n, tracer un graphique l'histogramme associée à P_n. (*Indication:* Il faut utiliser `pyplot.hist` avec `bins` $= n + 1$.)

5) La théorie prédit que la probabilité de ne jamais revenir à l'origine tend vers 0 lorsque $n \to +\infty$. (On dit que la marche aléatoire sur $\mathbb{Z}$ est *récurrente*.) Vos simulations sont-elles cohérentes avec cette propriété ?

Barème indicatif : 9 — 9 — 8