Évaluation — Révisions de début d'année

Terminale NSI — Version simulation— Durée : 65 minutes

Consignes — Aucun document ni calculatrice autorisés. Les noms de fonctions et de paramètres demandés doivent être respectés exactement. Si un ordinateur est mis à disposition, testez vos fonctions avec les `assert` fournis avant de rendre votre copie ; sinon, relisez votre code avec soin comme si vous alliez l'exécuter. Le barème est indicatif (60 points au total).

Exercice 1 — Approcher le nombre d'or (20 points)

On souhaite approcher le nombre d'or $\phi \approx 1{,}618$ à l'aide de la suite de Fibonacci : $u_0 = 1$, $u_1 = 1$, et pour $n \geqslant 2$, $u_n = u_{n-1} + u_{n-2}$. Le rapport de deux termes consécutifs u_n / u_{n-1} se rapproche de ϕ lorsque n augmente.

Question 1 (4 points) — Écrire une fonction `terme_suivant(a: int, b: int) -> int` qui, à partir de deux termes consécutifs a et b de la suite (a le plus ancien, b le plus récent), renvoie le terme suivant.

```
assert terme_suivant(1, 1) == 2
assert terme_suivant(3, 5) == 8
```

Question 2 (4 points) — Écrire une fonction `rapport(a: int, b: int) -> float` qui renvoie le rapport `b / a`.

Question 3 (5 points) — Écrire une fonction `est_stable(rapport_precedent: float, rapport_actuel: float, tolerance: float) -> bool` qui renvoie `True` si l'écart absolu entre les deux rapports est strictement inférieur à `tolerance`, `False` sinon.

Question 4 (7 points) — En réutilisant les fonctions précédentes, écrire une fonction `approx_nombre_or(tolerance: float) -> float` qui : part de `a = 1`, `b = 1` ; calcule à chaque tour de boucle le nouveau terme et le nouveau rapport ; s'arrête (boucle `while`) dès que deux rapports successifs sont stables selon `est_stable` ; renvoie le dernier rapport calculé.

Exercice 2 — La caisse du supermarché (20 points)

Une caissière saisit les prix des articles d'un client un par un au clavier ; elle entre `0` pour indiquer la fin du ticket.

Question 1 (6 points) — Écrire une fonction `saisir_ticket() -> tuple` qui demande des prix à l'utilisateur (`input`) tant que le prix saisi n'est pas nul, et renvoie un tuple `(nombre_articles, total)`.

Question 2 (6 points) — Écrire une fonction `ticket_avec_remise(seuil: float, taux: float) -> tuple` qui fonctionne comme `saisir_ticket`, mais applique en plus une remise de `taux` (ex. `0.1` pour 10 %) sur le total si celui-ci dépasse `seuil`. Elle renvoie `(nombre_articles, total_apres_remise)`.

Question 3 (8 points) — Écrire une fonction `grille_tarifs() -> str` qui renvoie, sous forme de chaîne de caractères (une ligne par prix unitaire, séparateur `\n`), le prix total pour 1 à 10 articles, pour chacun des prix unitaires entiers de 1 € à 5 €. Utiliser deux boucles imbriquées.

Exercice 3 — Suivi de températures (20 points)

Une station météo enregistre les températures (en °C) relevées chaque jour d'un mois dans une liste de nombres, positifs ou négatifs.

Question 1 (5 points) — Écrire une fonction `amplitude_thermique(temperatures: list) -> float` qui renvoie l'écart entre la température maximale et la température minimale de la liste.

Question 2 (5 points) — Écrire une fonction `moyenne_liste(valeurs: list) -> float` qui renvoie la moyenne des valeurs d'une liste, **sans utiliser** `sum()` ni le module `statistics`.

Question 3 (5 points) — Écrire une fonction `jours_de_gel(temperatures: list) -> list` qui renvoie la liste des indices des jours où la température est strictement négative.

Question 4 (5 points) — En réutilisant `moyenne_liste`, écrire une fonction `ecart_moyen(valeurs: list) -> float` qui renvoie la moyenne des écarts absolus à la moyenne : pour chaque valeur, on calcule `|valeur - moyenne|`, puis on fait la moyenne de ces écarts.