Évaluation — Révisions de début d'année

Terminale NSI — Version simulation — Corrigé détaillé

Exercice 1 — Approcher le nombre d'or

Question 1

```python
def terme_suivant(a: int, b: int) -> int:
    """Renvoie le terme qui suit a et b dans la suite de Fibonacci."""
    return a + b
```

On additionne simplement les deux derniers termes connus — c'est la définition même de la suite.

Question 2

```python
def rapport(a: int, b: int) -> float:
    """Renvoie le rapport b / a."""
    return b / a
```

Question 3

```python
def est_stable(rapport_precedent: float, rapport_actuel: float, tolerance:
float) -> bool:
    """Indique si deux rapports successifs sont suffisamment proches."""
    return abs(rapport_actuel - rapport_precedent) < tolerance
```

Piège fréquent : oublier la valeur absolue `abs(...)`. Sans elle, la fonction pourrait renvoyer `True` alors que le rapport diminue fortement, si la différence négative reste inférieure à `tolerance` par coïncidence de signe.

Question 4

```python
def approx_nombre_or(tolerance: float) -> float:
    """Approxime le nombre d'or par les rapports de Fibonacci
successifs."""
    a, b = 1, 1
    rapport_actuel = rapport(a, b)
    stable = False
    while not stable:
        rapport_precedent = rapport_actuel
        a, b = b, terme_suivant(a, b)
        rapport_actuel = rapport(a, b)
        stable = est_stable(rapport_precedent, rapport_actuel, tolerance)
    return rapport_actuel
```

*Point de vigilance : il faut conserver le rapport du tour précédent **avant** de le recalculer, sinon la comparaison de la question 3 devient impossible. La **valeur exacte** de φ n'est jamais atteinte : la fonction renvoie une **valeur approchée**, d'autant plus précise que `tolerance` est petite.*

Exercice 2 — La caisse du supermarché

Question 1

```
def saisir_ticket() -> tuple:
    """Saisit des prix jusqu'à 0 et renvoie (nombre d'articles, total)."""
    nombre_articles = 0
    total = 0
    prix = float(input("Prix de l'article (0 pour terminer) : "))
    while prix != 0:
        total += prix
        nombre_articles += 1
        prix = float(input("Prix de l'article (0 pour terminer) : "))
    return (nombre_articles, total)
```

*Structure classique de boucle « tant que la sentinelle n'est pas atteinte » : on doit saisir une première valeur **avant** d'entrer dans la boucle, sans quoi la condition ne peut pas être testée.*

Question 2

```
def ticket_avec_remise(seuil: float, taux: float) -> tuple:
    """Idem saisir_ticket, avec une remise si le total dépasse seuil."""
    nombre_articles, total = saisir_ticket()
    if total > seuil:
        total = total * (1 - taux)
    return (nombre_articles, total)
```

On réutilise `saisir_ticket` plutôt que de tout réécrire : c'est le même principe que « reprendre l'exercice précédent » déjà rencontré en cours.

Question 3

```
def grille_tarifs() -> str:
    """Table des totaux pour 1 à 10 articles, prix unitaires de 1 à 5
euros."""
    chaine_retour = ""
    for prix_unitaire in range(1, 6):
        for quantite in range(1, 11):
            chaine_retour += str(prix_unitaire * quantite) + " "
        chaine_retour += "\n"
    return chaine_retour
```

*Même logique que la table de multiplication vue en cours : le `"\n"` se place **après** la boucle interne, jamais à l'intérieur, sous peine de mélanger les lignes.*

Exercice 3 — Suivi de températures

Question 1

```python
def amplitude_thermique(temperatures: list) -> float:
    """Écart entre la température maximale et la température minimale."""
    maxi = temperatures[0]
    mini = temperatures[0]
    for t in temperatures:
        if t > maxi:
            maxi = t
        if t < mini:
            mini = t
    return maxi - mini
```

Question 2

```python
def moyenne_liste(valeurs: list) -> float:
    """Moyenne d'une liste de nombres, sans sum()."""
    total = 0
    for v in valeurs:
        total += v
    return total / len(valeurs)
```

Question 3

```python
def jours_de_gel(temperatures: list) -> list:
    """Indices des jours où la température est strictement négative."""
    indices = []
    for i in range(len(temperatures)):
        if temperatures[i] < 0:
            indices.append(i)
    return indices
```

*On a besoin de l'**indice** (le jour), donc on parcourt avec `range(len(...))` plutôt qu'un simple `for t in temperatures`, qui ne donnerait accès qu'à la valeur.*

Question 4

```python
def ecart_moyen(valeurs: list) -> float:
    """Moyenne des écarts absolus à la moyenne de la liste."""
    m = moyenne_liste(valeurs)
    ecarts = []
    for v in valeurs:
        ecarts.append(abs(v - m))
    return moyenne_liste(ecarts)
```

On réutilise deux fois `moyenne_liste` : une première fois pour obtenir la moyenne de la liste d'origine, une seconde fois pour faire la moyenne des écarts — un même algorithme peut souvent être réinvesti tel quel.