CORRIGÉ SUJET N°7

- COCCINELLES & PUCERONS -

Le Dossier ZIP de l'épreuve pratique N°7 se situe à l'adresse suivante :
https://sujets.examens-concours.gouv.fr/delos/api/file/public/69c4f0a48a3f1b2f2dd065ca

Ce corrigé est tiré du site suivant :
https://clem2429.github.io/Correction-Sujets-NSI-2026/

La version du fichier Python corrigé est accessible à l'adresse suivante :
https://clem2429.github.io/Correction-Sujets-NSI-2026/corrige/coccinnelles_corrige.py

- Ce sujet est le N°7 de la banque des sujets pratiques de NSI 2026;
- Ce sujet (corrigé) comporte 4 questions ;
- Chaque réponse à une question se trouve sur une page distincte.

QUESTION 1

On souhaite observer le comportement du modèle sur une courte période.
Créer une population initiale contenant 3 coccinelles (2 femelles et 1 mâle), toutes âgées de 10 jours et ayant un niveau de nutrition de 2. Le nombre initial de pucerons est fixé à 200.
Écrire une séquence d'instructions (utilisant une boucle) permettant de simuler l'évolution de ce petit écosystème sur 5 jours consécutifs, en appelant la fonction evolution.
Afficher le nombre de coccinelles et de pucerons à la fin de chaque journée.

On créera une fonction `init_pop` qui nous servira à initialiser la population pour les 2 premières questions.

```
def init_pop():
    a = Coccinelle("femelle", 10, 2)
    b = Coccinelle("femelle", 10, 2)
    c = Coccinelle("male", 10, 2)
    return [a, b, c]
```

La fonction demandée peut être :

```
def test_evolution():
    pop = init_pop()
    puc = 200
    for k in range(1, 6):
        pop, puc = evolution(pop, puc)
        print("Jour ", k," : \n Coccinelles : ", len(pop), " -
Pucerons : ", puc, "\n")
```

QUESTION 2

Écrire une fonction `simulation_simple(population, nb_proies)` qui automatise ce processus sur une durée maximale de 30 jours.
Cette fonction doit s'interrompre prématurément si la population de coccinelles ou de pucerons tombe à zéro. Elle doit renvoyer un triplet (tuple) contenant : le nombre final de coccinelles, le nombre final de pucerons, et le nombre de jours effectivement simulés. Tester cette fonction avec la même population initiale que la question précédente face à 1000

```
def simulation_simple(population, nb_proies):
    for k in range(1, 31):
        population, nb_proies = evolution(population,
nb_proies)
        res = (len(population), nb_proies, k)
        if len(population) == 0:
            print("Plus de coccinelles !") # Facultatif mais
pas inutile
            return res
        elif nb_proies == 0:
            print("Plus de pucerons !") # Facultatif mais pas
inutile
            return res
    return res
```

QUESTION 3

Écrire la documentation et les commentaires de la méthode `chasser`.

Voici l'une des réponses possibles :

```
def chasser(self, nb_proies, nb_coccinelles):
    '''
        Méthode prenant en paramètre 1 coccinelle de type
Coccinelle,
        un nombre de proies nb_proies et un nombre de
coccinelles nb_coccinelles
        Renvoie le nombre de proies qui reste après que la
coccinelle ait consommé x proies
        Le nombre de consommation est calculé via des moyennes
et du hasard en random
    '''
    # Si le nombre de coccinelles présentes pour chasser est
nul, le nombre de poids reste tel quel
    if nb_coccinelles == 0:
        return nb_proies

    # On calcule le nombre de proies qui sera attribué par
coccinelle
    proies_par_cocci = nb_proies / nb_coccinelles

    # On affecte un nombre de proies consommées selon le nombre
de proies par coccinelles
    if proies_par_cocci > 20:
        consomme = random.randint(12, 20)
    elif proies_par_cocci > 10:
        consomme = random.randint(8, 15)
    else:
        consomme = random.randint(3, 8)

    # On affecte à consomme la valeur minimale entre une donnée
hasardeuse (consomme elle-même)et le nombre de poids
    consomme = min(consomme, nb_proies)

    # On gère le niveau de nutrition d'une coccinelle selon le
nombre de proies consommées
    if consomme >= 10:
        self.niv_nutrition += 1
    else:
        self.niv_nutrition = max(0, self.niv_nutrition - 1)

    # On renvoie le nombre de proies qui reste après que notre
coccinelle en ait consommé x
```

```
        return nb_proies - consomme
```

QUESTION 4

Modifier les méthodes `reproduction` et `a_survecu` de la classe `Coccinelle` afin d'y intégrer ces deux nouvelles règles biologiques (la maturité sexuelle et l'impact mortel du manque de nourriture).
Tester à nouveau la simulation globale pour observer les changements.

Pour rappel des modifications à effctuer :
- Les coccinelles ne peuvent se reproduire qu'à partir de 20 jours de vie ;
- Si le niveau de nutrition d'une coccinelle est nul, elle a 1 chance sur 3 de survivre.

```
# Version corrigée :
def reproduction(self):
    """
    Une femelle avec un niveau de nutrition >= 2 engendre
exactement
    deux descendants : un mâle et une femelle.
    """
    descendants = []
    # On ajoute self.age >= 20 puisque c'est à partir de 20
jours de vie
    if self.sexe == "femelle" and self.niv_nutrition >= 2 and
self.age >= 20:
        descendants.append(Coccinelle("male", 0, 0))
        descendants.append(Coccinelle("femelle", 0, 0))
        self.niv_nutrition = 0

    return descendants
```

```
# Version corrigée :
def a_survecu(self):
        """
        Met à jour l'âge de la coccinelle et indique si elle
est encore en vie.
        """
        # Si niv_nutrition est nulle, et que le hasard donne 3
(ici considérés comme mortel), on indique False
        if self.niv_nutrition == 0:
            if random.randint(1, 3) == 3:
                return False

        self.age = self.age + 1
        return self.age < self.esperance_de_vie
```