Exercice 1

Correction — TP(1) Arbres en Python

1.0.1 Thème : La classification du vivant

1.1 Partie 1 — Vocabulaire et exploration

1.1.1 Q1.c — Script à trous

```
print("animalia est une feuille :", animalia.est_feuille())  # donné

lion = panthera.enfants[0]
print("Lion est une feuille :", lion.est_feuille())          # donné

# Trou 1 : tester si mammalia est une feuille
print("mammalia est une feuille :", _____________)

# Trou 2 : afficher les enfants de animalia
print("Enfants de Animalia :", _____________)
```

1.2 Partie 2 — Taille de l'arbre

1.2.1 Algorithme `taille` — Script à trous

Idée récursive : un arbre, c'est un nœud racine + des sous-arbres. La taille d'un sous-arbre = 1 (la racine) + la somme des tailles des enfants.

```
def taille(noeud):
    """
    Renvoie le nombre total de œnuds dans le sous-arbre
    dont noeud est la racine.
    """
    if noeud.est_feuille():
        return ___          # (1) cas de base : que vaut la taille d'
   une feuille ?

    total = 1
    for enfant in noeud.enfants:
        total += ___
        # (2) appel récursif sur chaque enfant
    return total
```

Variante en compréhension :

```
def taille(noeud):
    return 1 + sum(taille(e) for e in noeud.enfants)
```

1.3 Partie 2 — Hauteur de l'arbre

1.3.1 Algorithme `hauteur` — Script à trous

Idée récursive : la hauteur d'un nœud = 1 + la hauteur maximale parmi ses enfants. Une feuille est à hauteur 0 (aucune arête en-dessous).

```
def hauteur(noeud):
    """
    Renvoie la hauteur du sous-arbre dont noeud est la racine.
    """
    if noeud.est_feuille():
        return ___          # (1) cas de base, une feuille est à hauteur 0
    # (2) retourner 1 + le max des hauteurs des enfants
    return 1 + max(___
```

1.3.2 Q2.c — L'arbre est-il équilibré ?

```
for enfant in animalia.enfants:
    print(f"Hauteur de {enfant.valeur} : {hauteur(enfant)}")
```

Résultat :

```
Hauteur de Mammalia : 3
Hauteur de Reptilia : 2
Hauteur de Aves     : 3
```

L'arbre n'est pas parfaitement équilibré (Reptilia est moins profond), mais les écarts restent faibles.

1.4 Partie 3 — Parcours en profondeur (DFS préfixe)

1.4.1 Algorithme `parcours_profondeur` — Script à trous

Idée : on visite et traite le nœud avant ses enfants (ordre préfixe). On descend jusqu'aux feuilles avant de passer au sous-arbre suivant.

```
def parcours_profondeur(noeud):
    """
    Renvoie la liste des valeurs en parcours en profondeur (préfixe).
    """
    resultat = [noeud.valeur]       # on commence par le œnud courant
    for enfant in noeud.enfants:
        # (1) appel récursif : on ajoute le sous-parcours
        resultat += ___
    return resultat
```

1.5 Partie 3 — Parcours en largeur (BFS)

1.5.1 Algorithme `parcours_largeur` — Script à trous

> Idée : on utilise une file (FIFO). On enfile la racine, puis à chaque tour on défile un nœud, on le visite, et on enfile tous ses enfants. Résultat : on visite les nœuds niveau par niveau.

```python
from collections import deque

def parcours_largeur(racine):
    """
    Renvoie la liste des valeurs en parcours en largeur (BFS).
    """
    resultat = []
    file = deque([  ___  ])              # (1) on initialise la file avec la racine
    while file:
        noeud = ___                       # (2) on retire le premier élément de la file
        resultat.append(   ___   )        # (3) on enregistre sa valeur
        for enfant in noeud.enfants:
            file.append(   ___   )        # (4) on enfile chaque enfant

    return resultat
```

1.5.2 Q3.c — Comparaison des parcours

> Q3.c.1 — Les feuilles apparaissent en dernier dans le parcours en _____, car elles sont au niveau le plus bas et sont donc enfilées en dernier.
> Q3.c.2 — Les deux parcours garantissent qu'un nœud est visité avant tous ses descendants

---

1.6 Partie 4 — Recherche dans l'arbre

1.6.1 Algorithme `rechercher` — Script à trous

> Idée : on teste le nœud courant, puis on cherche récursivement dans chaque enfant. Attention : il faut bien retourner le résultat de l'appel récursif si on a trouvé.

```python
def rechercher(noeud, cible):
    """
    Cherche le œnud de valeur 'cible'. Renvoie le œnud ou None.
    """
    if noeud.valeur == ___   :          # (1) cas de base : on a trouvé
        return noeud

    for enfant in noeud.enfants:
        resultat = rechercher(   ___   ,   ___   )   # (2) appel récursif
```

```
        if resultat is not None:
        # (3) on remonte le résultat trouvé
            return  ___

    return None                          # (4) non trouvé dans ce sous-arbre
```

Erreur classique : oublier le `if resultat is not None: return resultat`. Sans cette vérification, la fonction renvoie toujours `None` même si elle a trouvé la cible, car la boucle continue et ignore le résultat.

1.7 Partie 4 — Lister toutes les espèces

1.7.1 Algorithme `lister_especes` — Script à trous

Idée : on collecte les valeurs des feuilles. C'est une variante de `parcours_profondeur` qui ne garde que les feuilles.

```
def lister_especes(noeud):
    """
    Renvoie la liste des valeurs de toutes les feuilles.
    """
    if noeud.est_feuille():
        return ___                  # (1) une feuille : on renvoie sa
   valeur dans une liste

    especes = []
    for enfant in noeud.enfants:
        # (2) appel récursif
        especes += ___
    return especes
```

Variante en compréhension :

```
def lister_especes(noeud):
    if noeud.est_feuille():
        return [noeud.valeur]
    return [e for enfant in noeud.enfants for e in lister_especes(enfant
    )]
```
