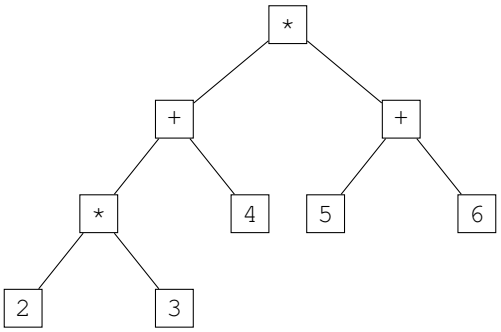
Amérique du sud - septembre 2023 - sujet 2 (corrigé)

Exercice 1 (File, pile et POO)

1. On a l'expression suivante : $((2 \times 3) + (4 \times 5)) \times 2$
2. On a l'arbre suivant :



3. (a) On a la liste suivante : 2 3 × 4 5 × + 2 ×
   (b) Il s'agit de la Proposition 3 car on évalue l'étiquette de l'arbre en dernier.
4. Le code suivant convient :

```
def evalue(arbre):
    f = suffixe(arbre, File())
    p = Pile()
    while not f.est_vide():
        elt = f.defile()
        if elt == '+' or elt == '*':
            d = p.depile()
            g = p.depile()
            p.empile(str(op(elt, d, g)))
        else:
            p.empile(elt)
    return p.depile()
```

Exercice 2 (Routage et bases de données)
Partie A : réseau.

1. On a les chemins suivants :
   - ⋆ Igo – R6 – R4 – serveur – R4 – R5 – R1 – Baduk
   - ⋆ Baduk – R1 – R2 – R4 – serveur – R4 – R6 – Igo
2. La table suivante convient :

R3		
Destination	**Passerelle**	**Métrique**
R1, R5, R6	–	1
R2	R1	2
R4	R6	2

Partie B : base de données.

3. (a) L'erreur produite par l'exécution de cette succession de commandes est une erreur de contrainte de domaine : lorsque l'on insère les données dans la table `Partie`, on donne une valeur à l'attribut `tournoi` qui est une clé étrangère faisant référence à la clé primaire `idtournoi` de la table `Tournoi`, mais cette valeur n'existe pas pour cet attribut (elle est donnée dans l'instruction suivante).
   (b) Il suffit d'intervertir les lignes 9-10 avec les lignes 6-7.
4. (a) On obtient la valeur `3`.
   (b) Cette instruction renvoie le nombre de parties jouées par Kitani Minuro.
5. La requête suivante convient :

```
SELECT nom FROM Tournoi WHERE pays = 'Japon' ORDER BY nom
```

6. La requête suivante convient :

```
SELECT DISTINCT nom
FROM Joueur
JOIN Partie
ON Joueur.idjoueur = Partie.idjnoir
WHERE Partie.jour = '2016-03-15'
```

7. Cette requête renvoie le nom de tous les joueurs ayant participé au tournoi de Meijin (identifiant 3).

Exercice 3 (Tableaux, programmation, récursivité)

1. (a) La valeur `goban[2][5]` est `1`. Il s'agit donc d'une pierre noire.

   (b) La fonction suivante convient :

```
def creer_goban_vide(n):
    assert n == 9 or n == 13 or n == 19, 'valeur de n non permise'
    return [[0 for j in range(n)] for i in range(n)]
```

   (c) La valeur `n = 11` n'est pas une valeur permise. On obtient donc une erreur d'assertion avec le texte alternatif `'valeur de n non permise'`.

2. Le code suivant convient :

```
def libertes_pierre(go, pi, pj):
    libertes = []
    n = len(go)
    for (i, j) in [(pi-1, pj), (pi+1, pj), (pi, pj-1), (pi, pj+1)]:
        if est_valide(i, j, n) and go[i][j] == 0:
            libertes.append((i, j))
    return libertes
```

3. Les instructions suivantes conviennent :

```
if not marquage[i][j]:
    marquage[i][j] = True
    nb_libertes += 1
```

4. La fonction suivante convient :

```
def supprime_prisonniers(go, chaine):
    if nb_liberte_chaine(go, chaine) == 0:
        for i, j in chaine:
            go[i][j] = 0
    return len(chaine)
```

5. Les instructions suivantes conviennent :

```
if est_valide(i, j, n) and go[i][j] == couleur and (i, j) not in chaine:
    cherche_chaine(go, i, j, chaine)
```