Exercice 2 Récursivité

1.

```
def possible_avec_penalittes_seules(score):
    return  score % 3 == 0
```

ou (si on a oublié le modulo)

```
def possible_avec_penalittes_seules(score):
    while score > 0:
        score = score - 3
    return score == 0
```

2.

score	liste des solutions	nombre de solutions
0	[0]	1
1	[]	0
2	[]	0
3	[0, 3]	1
4	[]	0
5	[0, 5]	1
6	[0, 3, 6]	1
7	[0, 7]	1
8	[0, 5, 8], [0, 3, 8]	2
9	[0, 3, 6, 9]	1
10	[0, 7, 10], [0, 3, 10], [0, 5, 10]	3

3.

f(10) = f(7) + f(5) + f(3) = 1 + 1 + 1 = 3

4.

n	cas de base
0	1
1	0
2	0
3	1
4	0
5	1
6	1

5.

```
def nb_solutions(n):
    if n == 1 or n == 2 or n == 4 :
        return 0
    if n == 0 or n == 3 or n == 5 or n == 6 :
        return 1
    return nb_solutions(n-3) + nb_solutions(n-5) + nb_solutions(n-7)
```

6. Il s'agit de la programmation dynamique
7. À partir de la ligne 8 : [0,3,8,11], [0,5,8,11]
   À partir de la ligne 6 : [0, 3, 6, 11]
   En théorie, on devrait aussi avoir la ligne 4, mais il est impossible d'avoir un score de 4.
8. Le code à trou tel qu'il est proposé ne peut pas fonctionner

```
def solutions_possibles(score):
    if score < 0:
        resultat = []
    elif score == 0:
        resultat = [[0]]
    else:
        resultat = []
        for coup in [3, 5, 7]:
            liste = solutions_possibles(score - coup)
            solution = []
            for solution in liste:
                solution.append(score)
            if solution != []:
                resultat.append(solution)
    return resultat
```

Exercice 3 SQL

Partie A

1.
```
dossard_pj = participants['PHILIPSEN Jasper']
classement_pj = classement_general[dossard_pj]
dossard_tp = participants['PINOT Thibaut']
temps_tp = temps_etapes[dossard_tp][3]
```
2.
```
def calcul_temps_total(d):
    tab = temps_etapes[d]
    temps =  0
    for t in tab:
        temps = temps + t
    return temps
```

3.
ligne 8 : `element[1] < classement[pos][1]`
ligne 9 : `classement[pos + 1] = classement[pos]`
ligne 14 : `classement_general[classement[i][0]] = i+1`

4.
ligne 16 : `difference_temps = ligne[2] - temps_premier`
ligne 17 : `coureur.append(difference_temps)`
ligne 18 : `tableau_final.append(coureur)`

Partie B

5. La clé primaire doit être unique pour chaque entrée. Dans la table Temps, on aura plusieurs fois le même numéro de dossard, `numDossard` ne peut donc pas jouer le rôle de clé primaire. Même chose pour l'attribut `numEtape`. En revanche, le couple (`numDossard, numEtape`) sera unique pour chaque entrée, c'est donc une clé primaire correcte.
6. La requête renvoie tous les noms des coureurs de l'équipe Cofidis
7.
```
SELECT Date
FROM Etapes
WHERE Type = 'contre-la-montre'
```

8.
```
SELECT directeurSportif
FROM Equipes
JOIN Coureurs ON Equipe = nomEquipe
WHERE nomCoureur = 'BARDET Romain'
```
9. La première requête fait appel à un attribut numEtape = 5, alors que cette étape n'a pas encore été créée dans la table Etapes, ce qui va provoquer une erreur.
10. Il suffit d'inverser l'ordre des requêtes.
11.
```
SELECT SUM(tempsRéalisé)
FROM Temps
JOIN Coureurs ON Coureurs.numDossard = Temps.numDossard
WHERE nomCoureur = 'BARDET Romain'
```

Calculabilité

Éléments de correction sujet 14 (2024)

Exercice 1

1. `i : 7, 8, 9, 10 ; f1(7)` se termine
2. `f1(-2)` se termine, la valeur renvoyée est 10
3. `i : 12 ,13, 14, 15, 16`… ; ne se termine pas (car toujours différent de 10)
4. `f1(n)` se termine pour tous les entiers strictement inférieurs à 11.
5. `f2(4)` se termine et renvoie 6
6. `f2(5)` ne se termine pas, car nous avons les appels récursifs suivants : `f2(3), f2(1), f2(-1), f2(-3)...` on ne rencontre jamais le cas de base (n=0)
7. Tous les entiers pairs supérieurs ou égaux à 0.
8.
```
def infini(n):
    return n + infini(n-1)
```
9. on appelle `infini(42)` et donc `paradoxe(code_paradoxe)` ne termine pas.
10. on exécute return 0, donc `paradoxe(code_paradoxe)` termine.
11. Il y a un paradoxe (la fonction `arret` se termine quand elle ne termine pas), la fonction `arret` ne peut donc pas exister.