Exercice 2 :

1. Il faut savoir si le score est un multiple de 3 :

```
def possible_avec_penalites_seules(score : int) -> bool:
    return score%3 == 0
```

2. Tableau des solutions possibles pour chaque score de 0 à 10 :

score	liste de solution	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, 3, 10], [0, 5, 10], [0, 7, 10]	3

3. Appliquons $f(n) = f(n-3) + f(n-5) + f(n-7)$ pour n = 10 :
$f(10) = f(7) + f(5) + f(3) = 1 + 1 + 1 = 3$

4. Cas de base de cette fonction récursive :
$f(0) = 1$ $f(1) = 0$ $f(2) = 0$ $f(3) = 1$ $f(4) = 0$ $f(5) = 1$ $f(6) = 1$

5.

```
def nb_solutions(n : int) -> int:
    if n < 0:
        return 0
    elif n in [0, 3, 5, 6]:
        return 1
    elif n in [1, 2, 4]:
        return 0
    else:
        return nb_solutions(n-3) + nb_solutions(n-5) + nb_solutions(n-7)
```

6. Afin de diminuer le nombre d'appels récursifs, il est possible de mémoriser les résultats déjà calculés pour ne pas les refaire. C'est la mémoization.

7. `solutions_possibles(11)` appelle :
`solutions_possibles(8)` `[0, 5, 8], [0, 3, 8]` avec une pénalité supplémentaire
`solutions_possibles(6)` `[0, 3, 6]` avec un essai supplémentaire
`solutions_possibles(4)` `[]` avec un essai transformé supplémentaire
pour trouver les solutions : `[0, 5, 8, 11], [0, 3, 8, 11], [0, 3, 6, 11]`

8.

```
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)
        for solution in liste:
            solution.append(score)
            resultat.append(solution)
    return resultat
```