DS 3

Informatique pour tous, première année

Julien Reichert

Exercice 1 : Faire tourner à la main le programme suivant, expliquer précisément ce qu'il fait et pourquoi. [1]

```
def test(s):
    print(s)
    return (len(s) > 1)

if test("a") or test("ab") or test("b"):
    print("c")
else:
    print("d")
```

Exercice 2 : On considère les scripts Python (a), (b), (c), (d), (e) et (f). Déterminer lesquels provoquent une erreur lors de l'évaluation du script ou de l'appel de la fonction et expliquer les erreurs. Déterminer également ce qui apparaît dans la console lors de l'évaluation des scripts. [2]

(a)

```
def f1(n):
    i = 0
    while i < n:
        print(i)
        i = i + 1

f1(42)
```

(b)

```
def f2(a):
    n = len(a)
    for i in range(n-1, 0):
        print(a[i])

f2([1, 2, 3])
```

(c)

```
def f3(n):
    for i in range(n):
        print(i)
return n

f3(42)
```

(d)

```
def f4(l):
    for i in range(l):
        print("l[i] vaut :")
        print(l[i])
        print("\n")
    return i

f4([1, 2, 3])
```

(e)

```
def f5(n):
    while n > 0:
        print(n)
        n - 1
    print("C'est fini !")
    return n

f5(42)
```

(f)

```
def f6(n):
    for i in range(n):
        return i
    if n % 2 == 0:
        print(1/0)
    else:
        print("Ok")

f6(42)
```

1. Adaptation d'une question utilisée par un camarade de l'ENS pour des entretiens d'embauche.
2. Ne pas répondre au hasard, je ne ferai pas de quartier.

Exercice 3 : Écrire en Python la fonction `uniques` qui prend en entrée deux chaînes de caractères et retourne la chaîne composée des caractères figurant dans une seule des chaînes. L'ordre des caractères dans la chaîne à retourner est le suivant : tous les caractères uniques de la première chaîne dans l'ordre, puis tous les caractères uniques de la deuxième chaîne dans l'ordre. Si un caractère apparaît plusieurs fois dans une chaîne et jamais dans l'autre, il apparaîtra autant de fois dans la réponse.

Exercice 3 bis : Déterminer et prouver la complexité asymptotique de la fonction `uniques` en nombre d'opérations élémentaires. On considèrera la comparaison d'un caractère et l'ajout d'un caractère à droite dans une chaîne comme des opérations élémentaires, sans avoir à les distinguer.

Exercice 4 : Écrire en Python la fonction `somme_consecutifs` qui prend en entrée une liste et retourne la liste composée de toutes les sommes d'éléments consécutifs identiques de la liste. La taille de la liste retournée sera donc inférieure ou égale à la taille de la liste en entrée.

Par exemple, l'appel `somme_consecutifs([1, 4, 4, 4, 0, 0, 4, 3, 3, 1])` renverra `[1, 12, 0, 4, 6, 1]`.

Exercice 5 : Écrire en Python la fonction `somme_successifs` qui prend en entrée une liste et retourne la liste composée de toutes les valeurs obtenues en sommant les sept éléments de la liste en partant de l'indice correspondant inclus et en se déplaçant vers la gauche. S'il y a moins de sept éléments en tout, on s'arrête au début de la liste sans déclencher d'erreur.

Par exemple, l'appel `somme_successifs([1, 4, 2, 5, 3, 0, 3, 6, 2, 6, 4, 6])` renverra `[1, 5, 7, 12, 15, 15, 18, 23, 21, 25, 24, 27]`.

Exercice 6 : Écrire en Python la réciproque de la fonction de l'exercice précédent.

Exercice 7 : Écrire en Python la fonction `motif_commun` qui prend en entrée deux listes `l` et `ll` et un entier naturel `k` et qui retourne une liste de taille `k` apparaissant à la fois dans `l` et dans `ll` en tant que tranche. Si plusieurs telles listes existent, on retournera n'importe laquelle.

Exercice 8 : Écrire en Python la fonction `tous_lancers` qui prend en entrée un entier naturel `n` et retourne toutes les listes de `n` entiers entre `0` et `6` possibles. Rappel : il y en a `6` à la puissance `n`. L'ordre n'est pas imposé.