TD1 – Fonctions et recherches dans les séquences - correction

Rappel. Une fonction se définit avec `def`. Les paramètres reçoivent les valeurs fournies lors de l'appel ; `return` transmet le résultat et termine la fonction. `print` affiche, mais ne renvoie pas de résultat.

```
def double(x):
    return 2*x

y = double(5)      # y vaut 10
```

1 | Définir et appeler une fonction

Question 1 : Sans exécuter le code, donner les valeurs de `a` et `b` après les instructions suivantes. Expliquer pourquoi `a` ne contient pas le même objet que `b`.

```
def carre(x):
    print(x*x)

def carre_retour(x):
    return x*x

a = carre(4)
b = carre_retour(4)
```

Réponse 1 : `carre(4)` affiche 16 mais renvoie None, donc a vaut None. b vaut 16 car `carre_retour(4)` renvoie cette valeur.

Question 2 : Écrire une fonction `est_pair(n)`, qui renvoie un booléen valant `True` si l'entier `n` est pair. Donner les valeurs renvoyées par les appels `est_pair(8)` et `est_pair(9)`.
Réponse 2 :

```
def est_pair(n):
    return n % 2 == 0

est_pair(8)   # True
est_pair(9)   # False
```

2 | Lire et écrire des fonctions

On suppose que `n` est un entier naturel et que `M` est un nombre positif.

Question 3 : Donner le rôle et un exemple d'appel de chacune des quatre fonctions suivantes. Pour `f4`, préciser ce que devient le résultat si `M = 0`.

```
def f1(x):
    if x >= 0:
        return x
    else:
        return -x

def f2(n):
    s = 0
```

```
    i = 0
    while i < n:
        i += 1
        s += i**2
    return s

def f3(n):
    p = 1
    for i in range(1, n + 1):
        p = p*i
    return p

def f4(M):
    s = 0
    i = 0
    while s < M:
        i += 1
        s += i**2
    return i
```

Réponse 3 : `f1` renvoie la valeur absolue : `f1(-3) == 3`. `f2` renvoie $1^2 + \cdots + n^2$: `f2(3) == 14`. `f3` renvoie $n!$: `f3(4) == 24` et `f3(0) == 1`. `f4` renvoie le plus petit entier $i \geq 0$ tel que $1^2 + \cdots + i^2 \geq M$: `f4(10) == 3` et `f4(0) == 0`.

Question 4 : Écrire une fonction `somme(L)` qui renvoie la somme des nombres de la liste. Que renvoie `somme([])` ?
Réponse 4 :

```
def somme(L):
    total = 0
    for valeur in L:
        total += valeur
    return total
```

`somme([])` renvoie 0, car la boucle n'est jamais exécutée.

Question 5 : Écrire `moyenne(L)` en réutilisant `somme(L)`. On suppose que `L` n'est pas vide. Donner la valeur renvoyée pour `[8, 10, 12]`.
Réponse 5 :

```
def moyenne(L):
    return somme(L) / len(L)
```

On obtient `10.0`. L'hypothèse « liste non vide » évite une division par zéro.

3 Chercher une valeur dans une liste

On considère une liste d'entiers `L`. Pour les questions suivantes, on parcourt explicitement la liste : ne pas utiliser `in`, `index`, `count`, `min`, ni `max`.

Question 6 : Écrire une fonction `contient(L, cible)`, qui renvoie `True` si `cible` apparaît dans `L`, et `False` sinon. *Pour cela, parcourir les éléments de la liste : « pour chaque élément… »*. Déterminer les résultats de `contient([4, -2, 7, -2], -2)` et `contient([], 3)`.
Réponse 6 :

```
def contient(L, cible):
    for valeur in L:
        if valeur == cible:
            return True
    return False
```

```
contient([4, -2, 7, -2], -2)  # True
contient([], 3)               # False
```

Le `return False` est placé après la boucle : on ne peut conclure à l'absence qu'après avoir examiné toutes les valeurs.

Question 7 : Écrire une fonction `premier_indice(L, cible)`, qui renvoie l'indice de la première apparition de `cible`, ou `-1` si la valeur est absente. Prévoir les résultats pour `[4, -2, 7, -2]` avec les cibles `-2`, `5` et pour la liste vide.

Réponse 7 :

```
def premier_indice(L, cible):
    for i in range(len(L)):
        if L[i] == cible:
            return i
    return -1
```

Les trois résultats demandés sont respectivement `1`, `-1` et `-1`.

4 Chercher dans une chaîne de caractères

En tant que séquence, une chaîne se parcourt comme une liste. On cherche ici un **caractère unique** et on distingue majuscules et minuscules. Ne pas utiliser `in`, `find`, ni `count`.

Question 8 : Écrire `nombre_occurrences(texte, caractere)`, qui renvoie le nombre d'apparitions du caractère dans la chaîne. Donner le résultat pour `("banane", "a")` et pour une chaîne vide.

Réponse 8 :

```
def nombre_occurrences(texte, caractere):
    compteur = 0
    for lettre in texte:
        if lettre == caractere:
            compteur += 1
    return compteur
```

On obtient `2` pour `("banane", "a")` et `0` pour la chaîne vide.

Question 9 : Écrire `premiere_position(texte, caractere)`, qui renvoie l'indice de la première apparition du caractère ou `-1` s'il est absent. Préciser les résultats pour `("banane", "n")`, `("banane", "z")` et `("", "a")`.

Réponse 9 :

```
def premiere_position(texte, caractere):
    for i in range(len(texte)):
        if texte[i] == caractere:
            return i
    return -1
```

Les résultats sont `2`, `-1` et `-1`.

Pour aller plus loin. Adapter `premiere_position` pour chercher une sous-chaîne entière, comme `"nane"` dans `"banane"`. Faire apparaître la dernière position de départ possible.

Réponse 9 :

```
def premiere_sous_chaine(texte, motif):
    for i in range(len(texte) - len(motif) + 1):
        if texte[i:i + len(motif)] == motif:
            return i
    return -1
```

Ici, `premiere_sous_chaine("banane", "nane")` renvoie `2`. Avec un motif vide, la fonction renvoie `0`, comme la recherche usuelle en Python.

5 | Manipuler des listes

Question 10 : Pour chacune des listes ci-dessous, écrire une expression qui extrait l'élément demandé : le nombre 2, le nombre 3 dans la sous-liste, puis la lettre `"m"`.

```
L1 = [4, 5, 3, 2, 1]
L2 = [4, "soleil", [1, 3], 0]
L3 = [5.1, 3, "informatique", 89.5]
```

Réponse 10 : Les expressions sont `L1[3]`, `L2[2][1]` et `L3[2][5]`. Dans `"informatique"`, les indices commencent à zéro : `i` est à l'indice 0 et `m` à l'indice 5.

Question 11 : Construire trois listes : les entiers de 0 à 15 inclus ; les entiers pairs de 10 à 100 inclus ; puis les nombres strictement inférieurs à 150 de la forme $3k+2$ avec $k \in \mathbb{N}$. Donner le premier et le dernier élément de chacune.
Réponse 11 :

```
entiers = list(range(16))
pairs = list(range(10, 101, 2))
formes = []
k = 0
while 3*k + 2 < 150:
    formes.append(3*k + 2)
    k += 1
```

Les extrémités sont `(0, 15)`, `(10, 100)` et `(2, 149)`. Pour la dernière liste, `k` va de 0 à 49.

Question 12 : Écrire une suite d'instructions qui affiche une seule fois `True` si `x` figure dans `L`, et `False` sinon.
Réponse 12 :

```
trouve = False
for valeur in L:
    if valeur == x:
        trouve = True
print(trouve)
```

Il faut placer l'affichage après la boucle pour ne produire qu'un booléen.

Question 13 : Les listes `L` et `M` ont la même longueur. Construire une nouvelle liste `N` dont l'élément d'indice `i` est la somme de `L[i]` et `M[i]`. Donner `N` pour `L = [4, 5, 6]` et `M = [2, 3, 4]`.
Réponse 13 :

```
N = []
for i in range(len(L)):
    N.append(L[i] + M[i])
```

On obtient `N = [6, 8, 10]`. Les indices sont nécessaires pour associer les éléments situés à la même position.

6 | Trouver le minimum et le maximum

Dans cette partie, **la liste est supposée non vide**.

Question 14 : Pour `L = [-8, -3, -12]`, expliquer pourquoi le programme ci-dessous donne un résultat faux, puis corriger son initialisation.

```
maximum = 0
for valeur in L:
    if valeur > maximum:
        maximum = valeur
```

Réponse 14 : Toutes les valeurs étant négatives, `maximum` reste à 0, qui n'appartient pas à la liste (0 n'est pas nécessairement une valeur de la liste : il ne convient donc pas comme valeur initiale pour un maximum ou un minimum.). Il faut commencer par `maximum = L[0]` ; le résultat devient -3.

Question 15 : Écrire une fcontion `maximum(L)` qui renvoie la plus grande valeur d'une liste non vide.
Réponse 15 :

```
def maximum(L):
    maxi = L[0]
    for valeur in L:
        if valeur > maxi:
            maxi = valeur
    return maxi
```

Question 16 : Écrire une fonction `extremes(L)` qui renvoie les deux valeurs `(minimum, maximum)` après un seul parcours de la liste. Donner les résultats renvoyés pour `[4, 1, 9, 1]`, `[-8, -3, -12]` et `[6]`.
Réponse 16 :

```
def extremes(L):
    minimum = L[0]
    maximum = L[0]
    for valeur in L:
        if valeur < minimum:
            minimum = valeur
        if valeur > maximum:
            maximum = valeur
    return minimum, maximum

extremes([4, 1, 9, 1])   # (1, 9)
extremes([-8, -3, -12])  # (-12, -3)
extremes([6])            # (6, 6)
```

Question 17 : Écrire une fonction `maximumS(L)` qui renvoie la liste contenant la plus grande valeur et son nombre d'occurrence, *on pourra utiliser la fonction `maximum(L)`*. Donner les résultats renvoyés pour `[4, 5, 6, 1, 6]` et `[-8, -3, -12]`.
Réponse 17 :

```
def maximumS(L):
    maxi = maximum(L)
    compteur = 0
    for valeur in L:
        if valeur == maxi:
            compteur += 1
    return [maxi, compteur]
```

On trouve `maximumS([4, 5, 6, 1, 6]) == [6, 2]`. Pour des valeurs négatives, `maximumS([-8, -3, -12]) == [-3, 1]`. L'initialisation par `L[0]` est indispensable si toutes les valeurs sont négatives.