♦ TP2 – Rappels sur le langage Python (II)

Exercice 1. Parmi les syntaxes suivantes, déterminer lesquelles sont correctes en Python (en supposant qu'aucune variable n'a été définie précédemment) et, le cas échéant, déterminer le résultat obtenu.

1) `'4' + 'a'`	2) `4 + 'a'`	3) `'2 - a'`	4) `4 * a`
5) `4 * 'a'`	6) `'a'**3`	7) `'ab' + 'bc'`	8) `'ab' * 'bc'`

Solution.
1) `'4'+'a'` est correct et renvoie la chaîne de caractère `4a`
2) `4+'a'` n'est pas correct car `4` est de type `int` et `'a'` est de type `str`
3) `'2-a'` est correct et renvoie la chaîne de caractères `2-a`
4) `4*a` n'est pas correct car `a` est le nombre d'une variable qui n'est pas définie.
5) `4*'a'` est correct et renvoie la chaîne de caractère `aaaa`
6) `'a'**3` n'est pas correct car `'a'` est une chaîne de caractères.
7) `'ab'+'bc'` est correct et renvoie la chaîne de caractère `abbc`
8) `'ab'*'bc'` n'est pas correct car `'ab'` et `'bc'` sont des chaînes de caractères.

Exercice 2. On considère la chaîne de caractères `a = "Bonjour"`. Dire si les instructions suivantes sont correctes et, le cas échéant, déterminer le résultat obtenu.

1) `a[0]`	2) `a[3]`	3) `len(a)`	4) `a[7]`
5) `a[0] + a[-1]`	6) `a[0] + 4*a[-1]`	7) `"B" in a`	8) `"b" in a`
9) `a[1] == 'o'`	10) `a[1] = 'B'`	11) `a + 'a'`	12) `a + a`

Solution.
1) `a[0]` est correct et renvoie `B`.
2) `a[3]` est correct et renvoie `j`.
3) `len(a)` est correct et renvoie `7`.
4) `a[7]` n'est pas correct car `a` est de longueur 7 donc les indices valables sont les entiers de -7 à 6.
5) `a[0] + a[-1]` est correct et renvoie `Br`.
6) `a[0] + 4*a[-1]` est correct et renvoie `Brrrr`.
7) `"B" in a` et correct et renvoie `True`.
8) `'b' in a` et correct et renvoie `False`.
9) `a[1] == 'o'` est correct et renvoie `True`.
10) `a[1] = 'B'` n'est correct car `a` n'est pas mutable donc on ne peut pas changer l'affectation de `a[1]`.
11) `a + 'a'` est correct et renvoie `Bonjoura`.
12) `a + a` est correct et renvoie `BonjourBonjour`.

Exercice 3. Pour chacun des programmes suivants, déterminer la valeur des différentes variables à la fin de l'exécution.

1)
```
a = '2'
b = 3
a = a + '1'
b = a + 'b'
```

2)
```
A = 'a'
B = 'b'
A = A * 2
B = A + B
```

3)
```
a = 0
b = 1
a = a+b
b = 'a' + 'b'
```

4)
```
a = 'b'
b = 'a'
a = a * 2
b = a + b
```

Solution. Dressons, là aussi, une table d'exécution.

1)

a	b
'2'	
'2'	3
'21'	3
'21'	'21b'

À la fin de l'exécution, `a` contient la chaîne de caractère `21` et `b` la chaîne de caractères `21b`.

2)

A	B
'a'	
'a'	'b'
'aa'	'b'
'aa'	'aab'

À la fin de l'exécution, `A` contient la chaîne de caractère `aa` et `B` la chaîne de caractères `aab`.

3)

a	b
0	
0	1
1	1
1	'ab'

À la fin de l'exécution, `a` contient l'entier `1` et `b` la chaîne de caractères `ab`.

4)

a	b
'b'	
'b'	'a'
'bb'	'a'
'bb'	'bba'

À la fin de l'exécution, `a` contient la chaîne de caractère `bb` et `b` la chaîne de caractères `bba`.

Exercice 4. Écrire une fonction qui prend en arguments un caractère `c` et un entier $n > 0$ et qui renvoie la chaîne de caractère composée du caractère `c` répété n fois.

Solution.

```
def repet(c,n):
  return(c*n)
```

Exercice 5. Écrire une fonction qui prend en arguments deux caractères `c` et `d` et deux entiers $n > 0$ et $m > 0$ et qui renvoie la chaîne de caractères composée du caractère `c` répété n fois suivie du caractère `d` répété m fois. On pourra utiliser la fonction de l'exercice précédent.

Solution.

```
def repet_bis(c,d,n,m):
  return(c*n + d*m)
```

Exercice 6.

1. Écrire une fonction qui prend en arguments une chaîne de caractères `ch` et un entier $n > 0$ et qui affiche n fois la chaîne `ch` sur n lignes successives.
2. Modifier la fonction précédente de telle sorte que les lignes soient de plus numérotées de 1 à n (la première sera donc de la forme : `1. ch`, la seconde de la forme : `2. ch` et ainsi de suite, le numéro de la ligne étant suivi d'un point et le point suivi d'un espace puis de la chaîne `ch`).

Solution.

1.

```
def repet_chaine(ch,n):
  for k in range(n):
    print(ch)
```

2. Pour afficher le numéro de la ligne suivi d'un point, d'un espace et de la chaîne `ch`, il faut transformer l'entier `k` en chaîne de caractère à l'aide de la fonction `str` puis concaténer (à l'aide de `+`) avec `'. '` et avec `ch`.

```
def repet_chaine_bis(ch,n):
  for k in range(1,n+1):
    print(str(k) + '. ' + ch)
```

Exercice 7. Écrire une fonction qui prend en arguments un caractère `c` et un entier $n > 0$ et qui affiche n lignes successives de telle sorte que la première ligne contient `c`, la seconde `cc`, la troisième `ccc` et, de manière générale, la $i-$ème ligne contient la chaîne formée du caractère `c` répété i fois.

Solution.

```
def repet_nb_fois(c,n):
  for k in range(1,n+1):
    print(c*k)
```

Exercice 8. En utilisant l'itération sur les chaînes de caractères, écrire une fonction `premiere_occ` qui prend en arguments une chaîne de caractères `ch` et un caractère `c` et qui renvoie le plus petit indice k tel que `ch[k]` soit égal à `c` si `c` apparaît au moins une fois dans `ch` et qui renvoie `None` sinon. Ainsi, `premiere_occ('aabcada', 'a')` renvoie 0 et `premiere_occ('abcd', 'e')` renvoie `None`.

Solution.

```
def premiere_occ(ch,c):
  k = 0
  for e in ch:
    if e == c:
      return k
    else:
      k += 1
```

Exercice 9. En utilisant l'itération sur les chaînes de caractères, écrire une fonction `nb_occurrence` qui prend en arguments une chaîne de caractères `ch` et un caractère `c` et qui renvoie le nombre de fois où le caractère `c` apparaît dans la chaîne `ch`. Ainsi, `nb_occurrence('aabcada', 'a')` renvoie 4 et `nb_occurrence('abcd', 'e')` renvoie 0.

Solution.

```
def nb_occurrence(ch,c):
  k = 0
  for e in ch:
    if e == c:
      k += 1
  return k
```

Exercice 10. Écrire une fonction `sous_chaine` prenant en arguments deux chaînes `ch1` et `ch2` et qui renvoie `True` si `ch1` est une sous-chaîne de `ch2` ou si `ch2` est une sous-chaîne de `ch1` et `False` sinon.

Solution.

```
def sous_chaine(ch1,ch2):
  return ch1 in ch2 or ch2 in ch1
```

Exercice 11. Déterminer l'affichage obtenu lors de l'exécution des deux programmes suivants.

```
# programme 1
ch = ''
for k in range(1,5):
  if k % 2 == 1:
    ch += 'a'
  else:
    ch += 'b'
print(ch)
```

```
# programme 2
ch = ''
for k in range(4):
  if k % 2 == 1:
    ch += 'a'
  else:
    ch += 'b'
  print(ch)
```

Solution. Pour le programme 1, on part d'une chaîne vide et, pour k allant de 1 à 4, on ajoute la caractère `'a'` si k est impair et le caractère `'b'` si k est pair. On obtient donc la chaîne `abab`.

Pour le programme 2, c'est la même chose mais pour k allant de 0 à 3, on obtient `baba`.

Exercice 12.

1. Écrire une fonction `triple_six` prenant en argument une chaîne `ch` et qui renvoie `True` si le caractère `'6'` apparaît au moins 3 fois consécutivement dans la chaîne `ch` et qui renvoie `False` sinon.
2. On veut écrire une fonction `triple_six_exact` prenant en argument une chaîne `ch` et qui renvoie `True` si `ch` contient la sous-chaîne `'666'` sans que celle-ci soit une sous-chaîne de `'6666'` et qui renvoie `False` sinon.

   Compléter la fonction proposée ci-dessous, qui distingue les cas où les trois 6 consécutifs sont :

   – au début de la chaîne `ch` ;
   – à la fin de la chaîne `ch` ;
   – ni au début ni à la fin de la chaîne `ch`.

```
def triple_six_exact(ch):
  n = len(ch)
  if ch[0:3] == "666" and ..........:
    return True
  if ch[n-3:n] == "666" and ..........:
    return True
  for k in range(1, ......):
    if ch[k:k+3] == "666" and .......... and ..........:
      return True
  return False
```

Solution.

1.

```
def triple_six(ch):
  return '666' in ch
```

2.

```
def triple_six_exact(ch):
  n = len(ch)
  if ch[0:3] == "666" and (n<=3 or ch[3] != '6'):
    return True
  if ch[n-3:n] == "666" and ch[n-4] != '6':
    return True
  for k in range(1,n-3):
    if ch[k:k+3] == "666" and ch[k-1] != '6' and ch[k+3] != '6':
      return True
  return False
```

Exercice 13.

1. On souhaite écrire une fonction `miroir` qui prend en argument une chaîne `ch` et qui renvoie la même chaîne mais écrite à l'envers. Ainsi, `miroir('abcd')` renvoie `'dcba'`.
   Parmi les quatre fonctions suivantes, une seule convient. Déterminer laquelle.

```
def miroir1(ch):
  inv = ''
  for e in ch:
    inv = inv + e
  return inv
```

```
def miroir2(ch):
  inv = ''
  for e in ch:
    inv = e + inv
  return inv
```

```
def miroir3(ch):
  inv = ''
  n = len(ch)
  for k in range(1,n):
    inv = ch[k] + inv
  return inv
```

```
def miroir4(ch):
  inv = ''
  n = len(ch)
  for k in range(1,n):
    inv += ch[k]
  return inv
```

2. Un palindrome est une chaîne de caractère qui donne le même résultat qu'on la lise de la gauche vers le droite ou de la droite vers la gauche. Par exemple, `'non'`, `'kayak'` ou `'etlamarinevaveniramalte'` sont des palindromes.
   Écrire une fonction `palindrome` prenant en argument une chaîne de caractère `ch` et qui renvoie `True` si `ch` est un palindrome et `False` sinon.
3. On souhaite modifier la fonction précédente afin qu'elle ne tienne pas compte des espaces et qu'elle considère que la chaîne `'et la marine va venir a malte'` est aussi un palindrome.
   a. Écrire une fonction `sans_espace` prenant en argument une chaîne de caractères `ch` qui supprime tous les espaces de `ch` et renvoie la chaîne obtenue.
   b. Écrire une fonction `palindrome2` répondant à la question.

Solution.

1. La fonction `miroir1` ne convient pas car elle recopie les caractères de `ch` dans le même ordre et les fonctions `miroir3` et `miroir4` ne conviennent pas car elle omet le caractère `ch[0]`.
   La seule fonction qui convienne est `miroir2`.

2.

```
def palindrome(ch):
  return ch == miroir2(ch)
```

3. a.

```
def sans_espace(ch):
  s = ''
  for e in ch:
    if e != ' ':
      s = s + e
  return s
```

b.

```
def palindorme2(ch):
  chse = sans_espace(ch)
  return chse == miroir2(chse)
```

Exercice 14. On considère la liste suivante :

`L = [1,2.3,4.1,4,2.55,5,23]`

Les instructions suivantes sont-elles correctes et, si oui, que renvoient-elles ?

1. `L[1]`	**2.** `L[-2]`	**3.** `L[7]`	**4.** `len(L)`
5. `L[1.5]`	**6.** `L[1,5]`	**7.** `L[1:5]`	**8.** `L[0:4:2]`
9. `L[0]+L[3]`	**10.** `[L[0]]+[L[3]]`	**11.** `3*L[0]`	**12.** `3*[L[0]]`
13. `L[2] == 2.3`	**14.** `L[-3] == 2.55`	**15.** `4 in L`	**16.** `[4.1,4] in L`

Solution.

1. `L[1]` renvoie `2.3` (le terme de rang 1 est le deuxième terme de la liste).
2. `L[-2]` renvoie `5` (le terme de rang -2 est l'avant-dernier terme de la liste).
3. `L[7]` est incorrecte car il n'y a que 7 termes dans liste et on cherche à afficher le huitième.
4.] `len(L)` renvoie 7 car la liste contient 7 termes.
5. `L[1.5]` est incorrecte.
6. `L[1,5]` est incorrecte.
7. `L[1:5]` renvoie `[2.3,4.1,4,2.55]`.
8. `L[0:4:2]` renvoie `[1,4.1]`.
9. `L[0]+L[3]` renvoie `5`.
10. `[L[0]]+[L[3]]` renvoie `[1,4]`.
11. `3*L[0]` renvoie `3`.
12. `3*[L[0]]` renvoie `[1,1,1]`.
13. `L[0] == 2.3` renvoie `False` (`L[0]` vaut 1).
14. `L[-3] == 2.55` renvoie `True`.

15. `4 in L` renvoie `True`.
16. `[4.1,4] in L` renvoie `False`.

Exercice 15. On considère la liste suivante :

`M = [[1,-2], [0.5,6], [-1,3.45]]`.

Les instructions suivantes sont-elles correctes et, si oui, que renvoient-elles ?

1. `M[0]` **2.** `M[0][1]` **3.** `M[0][1][2]` **4.** `len(M)` **5.** `len(M[0])` **6.** `M[0]+M[2]`.

Solution.

1. `M[0]` renvoie `[1,-2]`.
2. `M[0][1]` renvoie `-2`.
3. `M[0][1][2]` n'est pas correct car `M[0][1]` est un entier.`len(M)` renvoie 3.
4. `len(M)` renvoie 3.
5. `len(M[0])` renvoie 2.
6. `M[0]+M[2]` renvoie `[1,-2,-1,3.45]`.

Exercice 16. On souhaite écrire une fonction `affiche` qui affiche en colonne les entiers de 0 à 3. Parmi les quatre fonctions suivantes, laquelle ou lesquelles convien(nen)t ?

```
def affiche1():
  for i in [0, 3]:
    print(i)
```

```
def affiche2():
  for j in [0, 1, 2, 3]:
    print(j)
```

```
def affiche3():
  for k in range(3):
    print(k)
```

```
def affiche4():
  for m in range(4):
    print(m)
```

Solution. Les fonctions qui conviennent sont `affiche2` et `affiche4`. La fonction `affiche1` n'affiche que `0` et `3` et la fonction `affiche3` n'affiche que les entiers de `0` à `2`.

Exercice 17.

1. Compléter les programmes suivants pour qu'ils affichent la liste `L` de tous les entiers de 1 à 100.

```
# Programme 1
L = []
for k in range(...):
  L += ...
print(L)
```

```
# Programme 2
L = []
for k in range(...):
  L.append(...)
print(L)
```

```
# Programme 3
L = []
L = 100*[0]
for k in range(...):
  L[k] = ...
print(L)
```

2. On considère la liste `L` de la question précédente. Écrire un programme qui multiplie par 2 tous les éléments de la liste `L` et affiche la nouvelle liste obtenue.
3. Écrire une fonction `carre` prenant en argument un entier $n > 0$ et qui renvoie la liste des carrés des entiers de 1 à n. Par exemple, `carre(5)` doit renvoyer `[1, 4, 9, 16, 25]`.

4. En utilisant la définition en compréhension, écrire un programme qui affiche une liste dont les éléments sont les doubles de tous les entiers de 1 à 100 puis une fonction `carre_compr` prenant en argument un entier $n > 0$ et qui renvoie la liste des carrés des entiers de 1 à n.

Solution.

1.

```
# Programme 1
L = []
for k in range(1,101):
  L += [k]
print(L)
```

```
# Programme 2
L = []
for k in range(1,101):
  L.append(k)
print(L)
```

```
# Programme 3
L = []
L = 100*[0]
for k in range(100):
  L[k] = k+1
print(L)
```

2.

```
for k in range(len(L)):
  L[k] *= 2
print(L)
```

3.

```
def carre(n):
  L = []
  for k in range(1,n+1):
    L.append(k**2)
  return L
```

4.

```
L = [2*k for k in range(1,101)]
print(L)
```

```
def carre_compr(n):
  L = [k**2 for k in range(1,n+1)]
  return L
```

Exercice 18. Pour chacune des listes suivantes, donner une syntaxe Python permettant de la définir en compréhension.

1. `[1,3,5,7,9,11,13,15,17,19]`
2. `[0,1,4,9,16,25,36,49,64,81,100]`
3. `[1,2,4,8,16,32,64,128,256,512,1024]`
4. `[3,6,12,15,21,24,30,33,39,42,48,51]`

Solution.

1. `[2*k+1 for k in range(10)]`
2. `[k**2 for k in range(11)]`
3. `[2**k for k in range(11)]`
4. `[3*k for k in range(18) if k%3 != 0]`

Exercice 19. Écrire une fonction `indices` qui prend en argument une chaîne de caractères `ch` et un caractère `c` et qui renvoie la liste, éventuellement vide, des indices de la chaîne `ch` où apparaît le caractère `c`. Ainsi, `indices('abbacccaa','a')` renvoie `[0,3,7,8]` et `indices('abbacccaa','d')` renvoie `[]`.

Solution.

```
def indices(ch,c):
  L = []
  for k in range(len(ch)):
    if ch[k] == c:
      L.append(k)
  return L
```

Exercice 20. Écrire une fonction `sous_liste` prenant en argument deux listes `L1` et `L2` et qui renvoie `True` si `L2` est une sous-liste de `L1` (c'est-à-dire si tous les éléments de `L2` apparaissent consécutivement et dans le même ordre dans `L1`) et qui renvoie `False` sinon.

Solution.

```
def sous_liste(L1,L2):
  n = len(L1)
  m = len(L2)
  for k in range(n-m+1):
    j = 0
    while j < m and L1[k+j] == L2[j]:
      j +=1
    if j == m:
      return True
  return False
```

Exercice 21. Écrire une fonction `tous_positifs` qui prend en argument une liste non vide de nombres `L` et qui renvoie `True` si tous les nombres de la liste sont positifs ou nuls et `False` sinon.

Solution.

```
def tous_positifs(L):
  for e in L:
    if e<0:
      return False
  return True
```

Exercice 22.

1. Écrire une fonction `positifs` prenant en argument une liste non vide d'entiers `L` et qui renvoie le nombre d'éléments de `L` qui sont positifs ou nuls.
   Par exemple, `positifs([2, -1, 3, 4, -6, 0, 6])` doit renvoyer 5.
2. Écrire une fonction `rangs_negatifs` prenant en argument une liste non vide d'entiers `L` et qui renvoie la liste des indices des éléments strictement négatifs de `L`.
   Par exemple, `rangs_negatifs([2, -1, 3, 4, -6, 0, 6])` doit renvoyer `[1, 4]`.
3. Écrire une fonction `motif` prenant en argument une liste non vide d'entiers `L` et qui renvoie `True` si `[0, 0]` est une sous-liste de `L` et `False` sinon.

4. Écrire une fonction `dix_vingt` prenant en argument une liste non vide d'entiers `L` et qui renvoie `True` si tous les éléments de `L` sont compris entre 10 et 20 (avec égalité possible) et `False` sinon.

Solution.

1.
```
def positifs(L):
  c = 0
  for e in L:
    if e >= 0:
      c += 1
  return c
```

2.
```
def rangs_negatifs(L):
  M = []
  for k in range(len(L)):
    if L[k] < 0:
      M.append(k)
  return M
```

3.
```
def motif(L):
  n = len(L)
  for k in range(n-1):
    if L[k] == 0 and L[k+1] == 0:
      return True
  return False
```

4.
```
def dix_vingt(L):
  for e in L:
    if e<0 or e>20:
      return False
  return True
```

Exercice 23.

1. On veut écrire une fonction `croissante` prenant en argument une liste de réels `L` et qui renvoie `True` si chaque élément de la liste est inférieur ou égal au suivant et `False` sinon. Recopier, en la corrigeant, la fonction proposée ci-dessous qui comporte des erreurs.

```
def croissante(L):
  n = len(L)
  for k in range(0, n):
    if L[k] > L[k+1]:
      return False
    else:
      return True
```

2. Écrire une fonction `monotone` prenant en argument une liste de réels `L` et qui renvoie `True` si la suite des termes de `L` est monotone (c'est-à-dire croissante ou décroissante) et `False` sinon.

Solution.

1.

```
def croissante(L):
  n = len(L)
  for k in range(0, n-1):
    if L[k] > L[k+1]:
      return False
  return True
```

2.

```
def monotone(L):
  M = []
  for e in L:
    M = [e] + M
  return croissante(L) or croissante(M)
```

Exercice 24. Expliquer le rôle de la fonction `nb6` suivante :

```
from random import *

def mystere(n):
  c=0
  L=[randint(1,6) for k in range(n)]
  for j in L:
    if (j==2):
      c=c+1
  return c
```

Solution. La fonction `nb6` crée une liste de n nombres entiers entre 1 et 6 puis compte le nombre de 2 contenus dans liste. Cette fonction peut servir à modéliser une variable aléatoire qui compte le nombre de 2 obtenus lorsqu'on lance n fois de suite un dé équilibré.

Exercice 25. Écrire une fonction `suite` qui prend en argument un entier naturel n et qui renvoie la liste des n premières valeurs de la suite (u_n) définie par $u_0 = 1$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = u_n^2 + n$.

Solution.

```
def suite(n):
  L=[1]
  for i in range(n):
    L.append(L[i]**2+i)
  return L
```

Exercice 26. Écrire une fonction `valeur_absolue_liste` qui prend en argument une liste d'entiers `L` et remplace chaque élément de la liste par sa valeur absolue.

Solution.

```
def valeur_absolue(L):
  for i in range(len(L)):
    if L[i]<0:
      L[i]=-L[i]
  return L
```

Exercice 27. Écrire une fonction `E_et_sigma` qui prend en arguments deux listes : la liste des valeurs d'une variable aléatoire X et la liste des probabilités associées (dans le même ordre, évidemment) et qui renvoie l'espérance et l'écart-type de X.

Solution.

```
from math import *

def E_et_sigma(valeurs, proba):
  N=len(valeurs)
  E=0
  for i in range(N):
    E=E+valeurs[i]*proba[i]
  V=0
  for i in range(N):
    V=V+valeurs[i]**2*proba[i]
  V=V-E**2
  S=sqrt(V)
  return(E,S)
```