5.1 *Définition*

Une chaîne de caractères est une suite ordonnée de caractères quelconques (lettres, chiffres, symboles de ponctuation, etc) délimitée par des guillemets (" ") ou par des apostrophes (' ').

Par exemple, `"TB2 'Le-Chesnoy'"` et `'@ !§a23 m*Et'` sont des chaînes de caractères.

La chaîne vide, qui ne contient aucun caractère, se définit par "" ou ''.

Une chaîne de caractères possède un type propre en Python : le type string (`str`). Il s'agit d'un objet non mutable ce qui signifie qu'on ne peut pas modifier ou supprimer une partie des caractères de la chaîne.

Remarque 1 Les guillemets (ou les apostrophes) sont indispensables pour différencier les chaînes de caractères des noms des variables. Ainsi, `ABC` est le nom d'une variable alors que `"ABC"` est une chaîne de caractères. Dès lors, l'instruction `print(ABC)` affichera la valeur de la variable `ABC` (ou un message d'erreur indiquant : `name 'ABC' is not defined` si cette variable n'a pas été définie précédemment) alors que `print("ABC")` affichera `ABC` (c'est-à-dire la chaîne de caractères formée par les trois lettres A, B et C).

Notons à ce propos que, parmi d'autres, `True` et `False` font exception à cette règle et ne désignent pas des variables mais des booléens. Ainsi, à l'affichage, `print("True")` ou `print(True)` donneront la même chose mais `return "True"` et `return True` ne renvoient pas le même type d'objet. D'ailleurs, dans une instruction conditionnelle, il vaut mieux éviter d'utiliser `return True`. Ainsi, plutôt que d'écrire :

```
def est_positif(x):
    if (x >= 0):
        return True
    else:
        return False
```

il est préférable d'écrire :

```
def est_positif(x):
    return x >= 0
```

5.2 *Longueur et éléments d'une chaîne*

Le nombre de caractères qui composent une chaîne est appelé la longueur de la chaîne. Tous les caractères, y compris les espaces, sont comptés.

La fonction len renvoie la longueur d'une chaîne de caractères (lenght signifiant longueur en anglais). Par exemple, `len("TB2 'Le-Chesnoy'")` renvoie 16.

Les caractères d'une chaîne de longueur n sont numérotés de 0 à $n-1$ de la gauche vers la droite et de -1 à $-n$ de la droite vers la gauche. Pour une chaîne de longueur n, on dit qu'un entier k est un indice valable (ou admissible) si $-n \leqslant k \leqslant n-1$.

Si k est un indice valable pour la chaîne `ch` alors `ch[k]` renvoie le caractère numéroté k dans la chaîne `ch`. Si k n'est pas un indice valable alors `ch[k]` renvoie un message d'erreur en indiquant : `string index out of range`.

Considérons par exemple la chaîne `ch = "chaine_1"`.

ch[0]	ch[1]	ch[2]	ch[3]	ch[4]	ch[5]	ch[6]	ch[7]
↓	↓	↓	↓	↓	↓	↓	↓
c	h	a	i	n	e	-	1
↑	↑	↑	↑	↑	↑	↑	↑
ch[-8]	ch[-7]	ch[-6]	ch[-5]	ch[-4]	ch[-3]	ch[-2]	ch[-1]

5.3 *Opérations sur les chaînes de caractères*

Comme on l'a dit, une chaîne de caractères est un objet non mutable donc on ne peut pas supprimer ou modifier des caractères de la chaîne. En revanche, il est possible de faire certaines opérations sur une ou plusieurs listes :

1) **extraction** ou **slicing** : étant donné une chaîne `ch` et deux indices a et b valables tels que $a < b$ alors
   - `ch[a:b]` renvoie la sous-chaîne de `ch` comprise entre les caractères d'indices a et $b-1$;
   - si $p \in \mathbf{N}^*$, `ch[a:b:p]` renvoie la chaîne formée des caractères de `ch` d'indices compris entre a et $b-1$ avec un pas p, c'est-à-dire la chaîne dont les caractères sont `ch[a]`, `ch[a+p]`, `ch[a+2p]`, ..., `ch[a+kp]` où $a+kp < b \leqslant a+(k+1)p$.

   Ainsi, si `ch = "chaine_1"` alors `ch[2:5]` renvoie la chaîne `"ain"` et `ch[1:5:2]` renvoie la chaîne `"hi"`.

   Remarque 2 Si $a \geqslant b$ alors `ch[a:b]` renvoie la chaîne vide (même si les indices a et b ne sont pas valables).

2) **concaténation** : étant donné deux chaînes de caractères `ch1` et `ch2`, l'instruction `ch1+ch2` concatène les deux chaînes, c'est-à-dire renvoie une nouvelle chaîne constituée des caractères de `ch1` suivi de ceux de `ch2`.

   Ainsi, si `ch1="tic"` et `ch2="tac"` alors `ch1+ch2` renvoie la chaîne `"tictac"`.

   En particulier, on peut utiliser ceci pour modifier une chaîne de caractères en lui ajoutant un caractère à la fin. Ainsi, `ch1 = ch1+"s"` modifie la chaîne `ch1` en `"tics"`. On peut également utiliser la syntaxe `ch1 += "s"`.

3) **répétition** : si on souhaite concaténer plusieurs fois la même chaîne, on peut utiliser l'opérateur * : plutôt que d'écrire `ch + ch + ch + ch`, on peut écrire `ch * 4`.

   Ainsi, `ch1 * 3` renvoie la chaîne `"tictictic"`.

4) **test d'appartenance** : on peut tester si un caractère `c` appartient à une chaine `ch` grâce à l'instruction `c in ch` qui renvoie `True` si le caractère `c` apparaît dans la chaîne `ch` et `False` sinon.

   Ceci fonctionne également pour tester si une chaîne `ch2` est une sous-chaîne d'une chaîne `ch1` en écrivant `ch2 in ch1`.

5.4 *Itération sur les éléments d'une chaîne*

Il est possible d'itérer une action sur les éléments d'une chaîne de caractères à l'aide d'une boucle `for`.

Par exemple, si on souhaite écrire une fonction `presence` qui prend en arguments une chaîne de caractères `ch` et un caractère `c` et qui renvoie `True` si le caractère `c` apparaît dans la chaîne `ch` et `False` sinon, plutôt que d'écrire :

```
def presence (ch , c):
    n = len(ch)
    for k in range(n):
        if ch[k] == c:
            return True
    return False
```

il est préférable d'écrire

```
def presence (ch , c):
    for e in ch:
        if e == c:
            return True
    return False
```

5.5 *Exercices*

♦ **PYT.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'`

♦ **PYT.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`

♦ **PYT.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
```

♦ **PYT.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.

♦ **PYT.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.

♦ **PYT.6** 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.

Modifier ensuite 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`).

♦ **PYT.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.

♦ **PYT.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`.

♦ **PYT.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`.

♦ **PYT.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.

♦ **PYT.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)
```

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'` mais pas la sous-chaîne `'6666'` et qui renvoie `False` sinon.

Compléter la fonction suivante, 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
```

♦ PYT.12

♦ PYT.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 qui renvoie la chaîne obtenue.

b) Écrire une fonction `palindrome2` répondant à la question.