Python 19 – Indices et strings

I – Connaissances de base

1 – Déclaration d'un string

→ Guillemets doubles en ouverture et fermeture.

→ Guillemets simples en ouverture et fermeture.

→ 3 guillemets simples en ouverture et fermeture pour réaliser un string multiligne.

2 - Nombre d'éléments stockés avec len()

```
>>> s = "bonjour !"   # s pour string
>>> nbr = len(s)
>>> nbr
9
```

On notera que les espaces et les caractères de ponctuation sont bien des caractères.

3 - Accès à l'un des éléments avec [indice]

```
>>> s = "bonjour !"
         012345678

>>> s[0]          >>> s[7]
'b'               ' '
>>> s[1]          >>> s[8]
'o'               '!'

>>> s[9]
IndexError
```

4 - Lecture des éléments par indice

(méthode classique, numérique avec range)

```
01 s = "bonjour !"
02 for i in range( len(s) ):
03     print(s[i], end="-")
```

La variable de boucle **i** prend les valeurs **0,** puis **1**, puis **2…** jusqu'à **8**.

La console affiche alors **: b-o-n-j-o-u-r- -!-**

5 - Lecture directe des éléments un à un

(méthode nominative, sans range)

```
01 s = "bonjour !"
02 for c in s:
03     print(c, end="-")
```

La variable de boucle **c** prend les valeurs **'b',** puis **'o'**, puis **'n'…** jusqu'à **'!'**.

La console affiche alors **: b-o-n-j-o-u-r- -!-**

Si on veut compter les o dans un texte, il suffit de placer un **if** dans le **for.**

```
01 s = "bonjour !"
02 nb = 0
03 for c in s:
04     if c == "o":
05         nb = nb + 1
```

Choix entre les deux types de for

→ si on ne veut que lire : les deux sont possibles.
→ si on a besoin de connaître l'indice : avec range.

6 - Immuabilité des strings EN PYTHON

Mutable ou muable
Donnée qui peut être modifiable après initialisation.

En Python, les strings sont ***immuables***.
On ne peut donc pas modifier les caractères d'un string après sa création.

```
>>> a = "bonjour"
>>> a[0] = "d"
TypeError: 'str' object does not support
item assignment
```

Comment modifier un string alors ?

On ne peut pas !
Il faut juste en créer un nouveau et l'attribuer à une variable qui porte le même nom que la précédente.

```
>>> a = "Bonjour"
>>> id(a)
6160

>>> a = "hello"
>>> id(a)
6216
```

```
Espace des noms          Mémoire
a → 6216                 6160 → "Bonjour"
                         6216 → "hello"
```

Exemple : « changer » les o en O

```
01 s = "bonjour !"
02 tempo = ""
03 for i in range(len(s)):
04     if s[i] == 'o'):
05         tempo = tempo + 'O'
06     else:
07         tempo = tempo + s[i]
08 s = tempo
```

```
01 s = "bonjour !"
02 tempo = ""
03 for c in s:
04     if c == 'o':
05         tempo = tempo + 'O'
06     else:
07         tempo = tempo + c
08 s = tempo
```

II – Rappels et autres exemples sur les strings

1 - Présence d'un caractère dans un string : in

```
s = "bonjour !"   # s pour string
test = 'o' in s or 'O' in s
if test:
    print("Détection d'un o/O")
```

2 - Détecter qu'un caractère est une voyelle

```
voyelles = "AaEeYyUuIiOo"
c = 'o'
est_voyelle = c in voyelles
if est_voyelle:
    print("c est une voyelle")
```

3 - Fonction-prédicat de détection d'une voyelle dans un string

```
def contient_une_voyelle_v1(s):
    '''True si s contient une voyelle'''

    voyelles = "AaEeYyUuIiOo"
    detection = False
    imax = len(s) - 1
    i = 0

    while i <= imax and not detection:
        if s[i] in voyelles:
            detection = True
        i = i + 1

    return detection


def contient_une_voyelle_v2(s):
    '''True si s contient une voyelle'''

    voyelles = "AaEeYyUuIiOo"

    for i in range(len(voyelles):
        if s[i] in voyelles:
            return True

    return False
```

4 - Récupérer uniquement un bout de string

Ce que vous devez savoir faire

```
          0123456
01  s = "Bonjour"
02  a = ""
03  for i in range(2, 5): :
04      a = a + s[i]
```

La variable **a** contient alors **'njo'**

Ce que vous pourriez faire

```
          0123456
01  s = "Bonjour"
02  a = s[2:5]
```

La variable **a** contient alors **'njo'**

5 - Séparer facilement un string : split()

On peut **créer un tableau** contenant des bouts d'un string en utilisant un caractère séparateur .

On utilise la méthode des strings nommée **split()**

```
>>> s = "Bonjour à tous !"
>>> mots = s.split(' ')
>>> mots
['Bonjour', 'à', 'tous', '!']

>>> mots[0]
'Bonjour'
```

On voit qu'avec **s.split(' ')** le caractère de séparation est l'espace.

6 - Dans l'autre sens : join()

On peut **créer un string** à partir d'un tableau dont les éléments sont des strings.

On utilise la méthode des tableaux nommée **join()**

```
>>> mots = ['Bonjour', 'à', 'tous', '!']

>>> s = " ".join(mots)
>>> s
'Bonjour à tous !'

>>> s = "*".join(mots)
>>> s
'Bonjour*à*tous*!'

>>> s = "".join(mots)
>>> s
'Bonjouràtous!'

>>> s = "\n".join(mots)
>>> s
'Bonjour\nà\ntous\n!'

>>> print(s)
 Bonjour
 à
 tous
 !
```

7 - « Modifier » un string : la bonne méthode avec Python

La méthode la plus efficace en terme de coût est

→ de transformer le string en tableau avec **list()**,
→ de modifier les cases du tableau et
→ de transformer le tableau en string avec **join()**.

```
s = "Bonjour"
t = list(s)    # on transforme s en tableau
for i in range(len(t)):  # pour chaque i de t
    if t[i] == 'o':
        t[i] = 'X'
s = ''.join(t)
```

On aura alors un nouveau contenu pour s : **"BXnjXur"**.