Le formatage des chaînes et les séquences (Les listes et les Tuples)

Université Virtuelle de Côte d'Ivoire

Table des matières

Objectifs

- Décrire le concept de séquence
- Réaliser des opérations sur les séquences
- Faire des formatages sur les chaînes de caractère
- Écrire des programmes python avancés

Introduction

Une *séquence* est une donnée qui est constituée de plusieurs données ; c'est-à-dire *c'est un type de donnée qui regroupe plusieurs types de données* (par exemple un tableau d'entier est une *séquence* qui contient plusieurs données de type entier). Les données contenues dans une *séquence* peuvent être de même type ou de différents types.

Cette leçon présente d'abord le formatage des chaînes de caractère qui facilite l'usage des chaînes de caractère ce qui est important dans la manipulation des séquences ensuite les séquences de base de python et montre enfin certaines opérations sur une séquence.

Formatage de chaînes de caractère

Objectifs

A la fin de cette activité, l'apprenant sera capable de :

- Écrire des scripts en utilisant le formatage des chaînes
- Présenter l'avantage du formatage de chaîne

La manipulation de chaînes de caractère dans tout langage de programmation est fondamentale pour la mise en place de programme interactif. Python permet au programmeur d'utiliser la méthode *format* de la classe str afin de faciliter l'usage intensive des chaînes. La suite de cette leçon présente la méthode format et ses avantages.

1. Afficher plusieurs variables

Rappel

Pour afficher plusieurs variables à l'écran à l'aide de la fonction print, la syntaxe suivante est utilisée. Le symbole "," est l'outil de concaténation à l'intérieur de la fonction print.

```
1 nom = 'YAO'
2 prenom = "Albert"
3 age = 22
4 print("Votre nom est", nom, ", votre prenom est", prenom, " et vous avez",
  age, "ans")
```

Le résultat de ce code est:

```
Votre nom est YAO, votre prenom est Albert et vous avez 22 ans
```

Remarque

Cette méthode classique de concaténation à l'intérieur de la fonction print devient très complexe à écrire s'il faut gérer l'affichage de plusieurs variables au sein d'une même chaîne. Ceci rend le script difficile à analyser surtout pour un programmeur qui n'est pas à la base de l'écriture du script.

2. La méthode format()

La méthode *format()* peut gérer de manière plus simple l'affichage de plusieurs variables suivant la syntaxe suivante :

`chaine.format()` où chaîne est une chaîne de caractère

Exemple

```
1 nom = "YAO"
2 prenom = "Albert"
3 age = 23
4 print("Votre nom est {}, votre prenom est {} et vous avez {}".format(nom,
  prenom, age))
```

Le résultat de ce code est :

```
Votre nom est YAO, votre prenom est Albert et vous avez 22 ans
```

Explication

Les accolades "{}" seront remplacées par ordre par les noms de variable qui sont inscrit comme paramètre dans la méthode format.

Par exemple la 1ere accolade correspond au paramètre nom, la 2e correspond au paramètre prenom et la 3e au paramètre age.

Remarque

1. Le formatage de la chaîne aurait pu avoir la syntaxe suivante :
   `print("Votre nom est {0}, votre prenom est {1} et vous avez {2}".format(nom, prenom, age))`
   Où {0} représente le 1er paramètre, {1} le 2e paramètre et {2} le 3e paramètre
2. Il est important de rappeler que cette fonctionnalité est bien plus puissante qu'un simple affichage, il est possible de formater une chaîne et de la stocker suivant la syntaxe suivante :
   `nouvelle_chaine = "Je m'appelle {0} {1} et j'ai {2} ans.".format(prenom, nom, age)`

3. Autre syntaxe de la méthode format

On peut également nommer les variables que l'on va afficher, c'est souvent plus intuitif que d'utiliser leur indice. Voici un nouvel exemple :

```
1 # formatage d'une adresse
2 adresse = """
3 rue {no_rue}, {nom_rue}
4  {commune}, {nom_ville} ({pays})
5 """.format(no_rue=12, nom_rue="VGE", commune="Treichville", nom_ville=
  "Abidjan", pays="CI")
6 print(adresse)
```

Le résultat de ce code est:

```
rue 12, VGE
Treichville, Abidjan CI
```

Remarque

Pour les chaînes qui contiennent des sauts de lignes, on triple les guillemets """. On peut aussi faire un saut de ligne en utilisant "\n".

```
>>> chaine = "Abidjan\nBouake"
>>> print(chaine)
Abidjan
Bouake
>>>
```

saut de ligne ou retour à la ligne

```
>>> chaine = """Abidjan
... Bouake
... Daloa"""
>>> print(chaine)
Abidjan
Bouake
Daloa
>>>
```

saut de ligne

4. Exercice

[solution n°1 p.19]

Quel est le résultat de ce script ?

```
1 print("{0}{1}{0}".format("abra", "cad"))
```

5. Exercice

[solution n°2 p.19]

Quel est le résultat de ce script ?

```
1 chaine= "{c},{a},{b}".format(a=5, b=9, c=7)
2 print(chaine)
```

6. Exercice

[solution n°3 p.19]

Complétez le script pour obtenir le même résultat que celui-ci

```
1 print("""1ere ligne
2 2e ligne""")
```

print("1ere ligne 2e ligne")

Les séquences de base

II

Objectifs

A la fin de cette activité, l'apprenant sera capable de :

- Manipuler les séquences de base du langage Python
- Réalisation des opérations sur les séquences de base du langage Python

Cette section introduit les séquences de base. Elle donne une notion générale de séquence et présente ensuite les différents types de séquences de base qui existe sous python (les listes et les tuples).

1. Séquence

Définition

Une *séquence* est un ensemble fini d'élément indexé de `p` à `n-1` où `p` est le 1er élément et `n` le n^{ème} élément. *Généralement* `p` *prend la valeur 0.*

Accès à un élément d'un séquence

Pour accéder à un élément d'une séquence, on utilise la syntaxe suivante :

```
nom_sequence[index]
```

Exemple

Soit la séquence nommée `tableau`:

`tableau[0]` -------> renvoie le 1er élément de la séquence

`tableau[5]` -------> renvoie le 6e élément de la séquence

2. Les tuples

Un *tuple* est une *séquence* utilisée en Python pour stocker des objets (les variables). *Le contenu d'un tuple ainsi que sa taille ne peuvent être modifiés après sa déclaration et son initialisation*. C'est à dire qu'on ne peut plus y ajouter d'objet ou en retirer. On dit que la séquence de type tuple est *immuable*.

Les syntaxes suivantes permettent de déclarer des séquences de type tuple.

Syntaxe

```
1 nom_variable = () #les parenthèses indique à python de créer un tuple vide
2 nom_var = tuple() #permet aussi de créer un tuple vide
3 variable = (element1, element2, ..., elementn)
4 var = element1, element2, ..., elementn
```

"*variable*" et "*var*" sont des séquences de type tuple contenant `n` éléments accessibles avec des index de `0` à `n-1`

Exemple

```
>>> t = ()
>>> type(t)
<class 'tuple'>
>>> t = (3,8,'moyenne',False)
>>> t[2]
'moyenne'
>>> type(t)
<class 'tuple'>
>>> t[3] = True
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    t[3] = True
TypeError: 'tuple' object does not support item assignment
```

manipulation d'une séquence de type tuple

La dernière instruction "t[3] = True" renvoie un message d'erreur car une séquence de type tuple ne peut être modifiée une fois qu'elle est déclarée et initialisée.

Attention

Pour déclarer une séquence de type tuple qui contient un seul élément, il faudrait utiliser l'une des syntaxes suivante :

```
>>>t = (2,)
```

```
>>>t = 2,
```

Sans la virgule, l'interpréteur considérera que les parenthèses représentent juste un délimiteur d'expression. Ainsi si l'on fait `>>>t = (2)` alors l*a variable t ne sera pas un tuple mais un entier*

3. Les listes

Une *liste est une séquence* utilisée en python pour stocker divers objets. *Le contenu d'une liste ainsi que sa taille peuvent être modifiés après sa déclaration et son initialisation*; c'est-à-dire qu'il est possible d'y ajouter et d'en retirer un élément. On dit que la séquence de type liste est *mutable*.

La syntaxe suivante permet de déclarer un élément de type liste.

Syntaxe

```
1 nom_variable = [] # L'usage des crochets permet de créer une liste vide
2 nom_var = list()
3 variable = [element1, element2, ..., elementn]
```

"`nom_variable`" et "`nom_var`" sont une séquence de type liste vide

"`variable`" est une séquence de type liste contenant n éléments accessible avec des index de `0` à `n-1`

Exemple

```
>>> liste = []
>>> type(liste)
<class 'list'>
>>> liste = [12, 'moyenne', 4.5, True]
>>> print(liste)
[12, 'moyenne', 4.5, True]
>>> print(liste[2])
4.5
>>> liste[3] = False
>>> print(liste)
[12, 'moyenne', 4.5, False]
```

création de liste vide et de liste non vide

4. Opérations sur les listes

Ajouter un objet, la méthode append()

La méthode `append()` permet d'ajouter un objet en fin de liste. Contrairement aux méthodes des chaînes de caractère, *la méthode append() modifie l'objet d'origine directement. Par conséquent, elle ne renvoie aucune valeur.* Elle prend en paramètre l'objet à ajouter à la liste

Exemple

```
>>> liste = ["Abel", "Terrain", 12, 0.95, False]
>>>
>>> print(liste)
['Abel', 'Terrain', 12, 0.95, False]
>>>
>>> liste.append("Cafard")
>>> print(liste)
['Abel', 'Terrain', 12, 0.95, False, 'Cafard']
>>>
```

méthode append() sous la console python

Insérer un objet, la méthode insert()

La méthode insert() permet d'insérer un objet à un index défini dans la liste. Cette méthode prend deux paramètres, le 1er paramètre indique l'index auquel doit être inséré l'objet et le 2e paramètre l'objet à insérer

```
>>> liste = ['DAS', 'BDA', 'MMX']
>>>
>>> liste.insert(1, 'RSI')
>>> print(liste)
['DAS', 'RSI', 'BDA', 'MMX']
>>>
```

Méthode insert() sous la console python

Supprimer un objet, la méthode remove()

La méthode remove() permet de supprimer la 1ere occurrence d'un objet dans une liste. Il prend en paramètre la valeur de l'objet à supprimer.

```
>>> filiere_uvci = ['DAS', 'RSI', 'BDA', 'MMX', 'MATH-INFO']
>>>
>>> print(filiere_uvci)
['DAS', 'RSI', 'BDA', 'MMX', 'MATH-INFO']
>>>
>>> filiere_uvci.remove("MATH-INFO")
>>> print(filiere_uvci)
['DAS', 'RSI', 'BDA', 'MMX']
>>>
```

Méthode remove() dans la console python

Remarque

- Pour connaître la longueur d'une séquence, on utilise la fonction len qui prend en paramètre le nom de la séquence. `>>>len(sequence)`
- Les mots clés *in* et *not in* sont aussi applicables aux séquences
- Le mot clé *del* peut aussi être utilisé pour supprimer un élément d'une séquence, il faudrait pour cela lui préciser l'objet à supprimer.

```
del filiere_uvci[0] #supprime l'objet 'DAS' de la liste
del filiere_uvci #supprime l'objet liste filiere_uvci
```

- Il est possible de faire la conversion de type entre les listes et les tuples.

```
t = 2,3,"Med"
l = list(t)  #permet de faire la conversion de tuple à liste

liste = ["A", 2, 5]
tup = tuple(liste)  #permet de faire la conversion de liste à tuple
```

5. Exercice

[solution n°4 p.19]

Quel est le résultat de ce code ?

```
nums = [5,2,4,7]
print(nums[2])
```

6. Exercice

[solution n°5 p.19]

Complétez le script suivant pour créer une liste et afficher le 2e objet de cette liste

liste = 42, 56, 14]

print( )

7. Exercice

[solution n°6 p.20]

Compléter le script suivant pour qu'il affiche le 1er élément de ce tuple si il contient un nombre d'éléments paire

tupl = 1, 2, 3, 4

if (tupl) % 2 ==0

print(tupl[ ])

8. Exercice

[solution n°7 p.20]

Compléter le code suivant ces instructions :

1. créer une liste
2. Ajouter le nom Georges à la fin de cette liste
3. Ajouter le nom Bernard à la 3e position dans la liste
4. Convertir la liste "liste_nom" en tuple
5. Quelle instruction permet de connaître la longueur de ce tuple ?
6. donnez la longueur de ce tuple

```
1. liste_nom =          "Adama", "Wilfried", "Ange", "Pacome"
2.          .          "Georges
3. liste_nom.          (          , "Bernard")
4. liste_nom =          liste_nom
5.          (liste_nom)
6.
```

Parcourir une séquence

III

Objectifs

A la fin de cette activité, l'apprenant sera capable de :

- Écrire des scripts pour parcourir des séquences
- Réaliser des programmes avancés

1. Boucle while

Le code suivant permet de parcourir la liste filiere_uvci

```
>>> filiere_uvci = ['DAS', 'MMX', 'RSI', 'BDA', 'CMD', 'E-COM']
>>> i=0
>>> while i<len(filiere_uvci):
...     print("spécialité {}".format(filiere_uvci[i]))
...     i=i+1
```

Parcours d'une séquence de type liste sous la console Python

La fonction `len(filiere_uvci)` est fournie par Python, elle renvoie un nombre entier qui représente la longueur de la séquence fournie en paramètre.

Ce script affichera le résultat suivant :

spécialité DAS

spécialité MMX

spécialité RSI

spécialité BDA

spécialité CMD

spécialité E-COM

2. Boucle for

Le code suivant permet de parcourir le tuple filiere_uvci

```
>>> filiere_uvci = ('DAS', 'MMX', 'RSI', 'BDA', 'CMD', 'E-COM')
>>> for fil in filiere_uvci:
...     print("spécialité {}".format(fil))
...
spécialité DAS
spécialité MMX
spécialité RSI
spécialité BDA
spécialité CMD
spécialité E-COM
```

Parcours d'une séquence de type tuple avec la boucle for

Remarque

Pour parcourir une séquence, *la boucle for est préférable à la boucle while*; car la boucle while est plus longue à écrire et aussi plus complexe, elle peut conduire à une boucle infinie si le programmeur oublie l'instruction d'incrémentation ou si le prédicat est mal défini.

3. La fonction enumerate

enumerate

La fonction *enumerate* a été mise en place par les programmeurs de Python pour répondre aux faiblesses des boucles for et while. La boucle while pour sa part est complexe à écrire et la boucle for ne fait que parcourir et renvoyer les objets d'une séquence sans fournir leur position exacte dans la séquence.

La fonction *enumerate* prend en paramètre une séquence et retourne chaque objet de la séquence avec sa position exacte ; c'est à dire qu'elle renvoie un tuple composé de deux éléments qui sont la position de l'objet dans la séquence et la valeur de cet objet

Exemple

```
>>> for elt in enumerate(filiere_uvci):
...     print("La valeur {} est de type {}".format(elt, type(elt)))
...
La valeur (0, 'DAS') est de type <class 'tuple'>
La valeur (1, 'MMX') est de type <class 'tuple'>
La valeur (2, 'RSI') est de type <class 'tuple'>
La valeur (3, 'BDA') est de type <class 'tuple'>
La valeur (4, 'CMD') est de type <class 'tuple'>
La valeur (5, 'E-COM') est de type <class 'tuple'>
```

Usage de la fonction enumerate pour parcourir une séquence

On constate l'usage de la boucle for avec un petit changement au niveau de la séquence qui doit être parcourue.

usage optimale de la fonction enumerate

dans l'exemple précédent, on constate que chaque valeur retournée par enumerate est un tuple. Si on voudrait seulement avoir accès à la position d'un objet, il faudrait normalement utilisé la l'instruction suivante :

```
>>> for elt in enumerate(filiere_uvci):
...     print("la position de {} est {}".format(elt[1], elt[0]))
...
la position de DAS est 0
la position de MMX est 1
la position de RSI est 2
la position de BDA est 3
la position de CMD est 4
la position de E-COM est 5
>>>
```

Usage de la fonction enumerate pour parcourir une séquence

Ce script marche bien mais pour une séquence complexe, il ne serait pas trop évident de le mettre en œuvre. Pour cela il existe une autre manière d'avoir, et la position et la valeur de l'objet dans une séquence

```
>>> for indice, valeur in enumerate(filiere_uvci):
...     print("la position de {} est {}".format(valeur, indice))
...
la position de DAS est 0
la position de MMX est 1
la position de RSI est 2
la position de BDA est 3
la position de CMD est 4
la position de E-COM est 5
>>>
```

Autre usage de la fonction enumerate pour parcourir une séquence

entre le `for` et le `in`, il y a 2 noms de variable (indice et valeur) la 1ere variable *indice* permet de récupérer le 1er objet du tuple renvoyé par enumerate et la 2e variable *valeur* permet de récupérer le 2e objet du tuple.

4. Exercice

[solution n°8 p.20]

compléter le code suivant pour qu'il affiche la position de la filière 'MMX' dans le tuple filiere_uvci

= 'DAS', 'MMX', 'RSI', 'BDA', 'CMD', 'E-COM'

for , in enumerate(filiere_uvci) :

    if filiere == 'MMX' :

        print("la filière {} se trouve à la position {}". (filiere, position))

Conclusion

Au terme de ce cours qui a présenté le formatage des chaînes et la manipulation des objets de type séquence, l'apprenant a fait un pas dans l'apprentissage des aspects important du langage Python qui lui permettront de comprendre la prochaine leçon qui traite des dictionnaires.

Solutions des exercices

> Solution n°1

Exercice p. 8

Quel est le résultat de ce script ?

```
1 print("{0}{1}{0}".format("abra", "cad"))
```

abracadabra

> Solution n°2

Exercice p. 8

Quel est le résultat de ce script ?

```
1 chaine= "{c},{a},{b}".format(a=5, b=9, c=7)
2 print(chaine)
```

7,5,9

> Solution n°3

Exercice p. 8

Complétez le script pour obtenir le même résultat que celui-ci

```
1 print("""1ere ligne
2 2e ligne""")
```

print("1ere ligne\n2e ligne")

> Solution n°4

Exercice p. 13

Quel est le résultat de ce code ?

```
1 nums = [5,2,4,7]
2 print(nums[2])
```

4

> Solution n°5

Exercice p. 13

Complétez le script suivant pour créer une liste et afficher le 2e objet de cette liste

liste = [42, 56, 14]

print(liste[1])

> Solution n°6

Exercice p. 13

Compléter le script suivant pour qu'il affiche le 1er élément de ce tuple si il contient un nombre d'éléments paire

tupl = (1, 2, 3, 4)

if len(tupl) % 2 ==0:

  print(tupl[0])

> Solution n°7

Exercice p. 14

Compléter le code suivant ces instructions :

1. créer une liste
2. Ajouter le nom Georges à la fin de cette liste
3. Ajouter le nom Bernard à la 3e position dans la liste
4. Convertir la liste "liste_nom" en tuple
5. Quelle instruction permet de connaître la longueur de ce tuple ?
6. donnez la longueur de ce tuple

```
1. liste_nom = ["Adama", "Wilfried", "Ange", "Pacome"]
2. liste_nom.append("Georges")
3. liste_nom.insert(2, "Bernard")
4. liste_nom = tuple(liste_nom)
5. len(liste_nom)
6. 6
```

> Solution n°8

Exercice p. 17

compléter le code suivant pour qu'il affiche la position de la filière 'MMX' dans le tuple filiere_uvci

filiere_uvci = ('DAS', 'MMX', 'RSI', 'BDA', 'CMD', 'E-COM')

for position, filiere in enumerate(filiere_uvci) :

  if filiere == 'MMX' :

    print("la filière {} se trouve à la position {}".format(filiere, position))