Chap 4

Les séquences

Enseignante: Mme Lamia MANSOURI

Année universitaire: 2020-2021

1. introduction

Une séquence est une suite d'éléments ordonnés séquentiellement et accessibles via des indices.

Python dispose de trois types prédéfinis de séquences :

- les chaînes;
- les listes;
- les tuples

2. Les chaines de caractères

Les chaînes de caractères sont un type de données **non modifiable** .

Non modifiable signifie qu'une donnée, une fois créée en mémoire, ne pourra plus être changée, toute transformation résultera en la création d'une nouvelle valeur distincte.

2.1 Opérations sur les chaines

- ***Longueur :***

```
s = "abcde"
len(s) # affiche 5
```

- ***Concaténation :***

```
s1 = "abc"
s2 = "defg"
s3 = s1 + s2 # s3 sera égale à 'abcdefg'
```

- ***Répétition :***

```
s4 = "bien! "
 s5 = s4 * 3
s5= 'bien! bien! bien! '
```

2. Les chaines de caractères

2.2 Fonctions vs méthodes

On peut agir sur une chaîne en utilisant des fonctions (notion procédurale) communes à tous les types séquences ou conteneurs, ou bien des méthodes (notion objet) spécifiques aux chaînes.

• **Exemple d'utilisation de la fonction len() :**

```
lng = len("abc")
print(lng)      # affiche 3
```

• **Exemple d'utilisation de la méthode upper() :**

```
ch1 = "python"
ch2 = ch1.upper()
```

2. Les chaines de caractères

2.3 Quelques méthodes sur les chaines

méthode	description	Exemple

lower()	Mets la chaîne en minuscules	`prenom = "LAMIA"`
		`prenom_m = prenom.lower()`
		`print(prenom_m)# affiche lamia`

upper()	permet de mettre une chaîne de caractères en majuscules	`prenom = "lamia"`
		`prenom_m = prenom.upper()`
		`print(prenom_m)# affiche LAMIA`

capitalize()	Mets la 1ere lettre de la chaine en majuscules	`prenom = "lamia"`
		`prenom_m = prenom.capitalize()`
		`print(prenom_m)# affiche Lamia`

replace(old, new)	Remplace une sous-chaine old par une sous-chaine new	`mot1 = "crime"`
		`mot2 = mot.replace('im', 'is')`
		`print(mot2)#affiche crise`

strip(), lstrip(), rstrip()	**strip()** supprime les espaces éventuels en début et en fin de chaîne. **lstrip()** enlève les espaces éventuels seulement en début de chaîne tandis que **rstrip()** enlève les espaces éventuels en fin de chaîne.	`titre1 = "    Python    "`
		`titre2 = titre.strip()`
		`print(titre2)`

2. Les chaines de caractères

méthode	description	Exemple

split()	Transforme une chaine en une liste de sous-chaines. Le séparateur par défaut est un espace mais il est possible de lui donner en argument n'importe quel autre séparateur.	`phrase = "Python est un langage."`
		`liste_mots = phrase.split()`
		`print(liste_mots)`
		`#affiche ['Python','est','un','langage.']`
		`Exemple 2`
		`phrase = "J'ai trouvé un porte-clé."`
		`liste_mots = phrase.split("-")`
		`print(liste_mots)`
		`#affiche ['J'ai trouvé un porte','clé.']`

join(liste)	Opération inverse de split()	`liste_mots =['Python','est','un','langage.']`
		`phrase = " ".join(liste_mots)`
		`print(phrase)`
		`# affiche 'Python est un langage.'`

index(y)	retourne l'indice de la première occurrence de la chaîne passée en argument.	`phrase ="J'ai trouvé une chaine."`
		`mot = 'une'`
		`indice = phrase.index(mot)`
		`print(indice)#affiche 12`

find(x)	Cette méthode fait exactement la même chose que index(y) sauf que si elle ne trouve pas la sous-chaîne, elle retourne -1	`phrase ="J'ai trouvé une chaine."`
		`mot = 'porte'`
		`indice = phrase.find(mot)`
		`print(indice)`

count()	Cette méthode retourne le nombre de fois où une sous-chaîne apparait dans une chaîne.	`phrase = "J'ai trouvé un porte-clé."`
		`car = 'é'`
		`number = phrase.count(car)`
		`print(number)# affiche 2`

2. Les chaines de caractères

méthode	description	Exemple

startswith()	retourne True si la chaîne commence par la sous-chaîne passée en argument. Sinon, elle retourne False.	`phrase = "Python est un langage."`
		`mot = phrase.startswith("Python")`
		`print(mot)#affiche True`

endswith()	retourne True si la chaîne se termine par la sous-chaîne passée en argument. Sinon, elle retourne False.	`phrase = "Python est un langage."`
		`mot = phrase.endswith("lapin")`
		`print(mot)# affiche False`

isupper() **islower()**	retournent True si la chaîne ne contient respectivement que des majuscules/minuscules	`s = "cHAise basSe"` `s.isupper() # False`
istitle()	retourne True si seule la première lettre de chaque mot de la chaîne est en majuscule	`s = "cHAise BasSe"` `s.istitle() #False`

isalnum(),
isalpha(),
isdigit()
isspace()
	retournent True si la chaîne ne contient respectivement que des caractères alphanumériques, alphabétiques, numériques ou des espaces	`S="cHAise BasSe"`
		`s.isalpha() #False`
		`s.isdigit() False#`

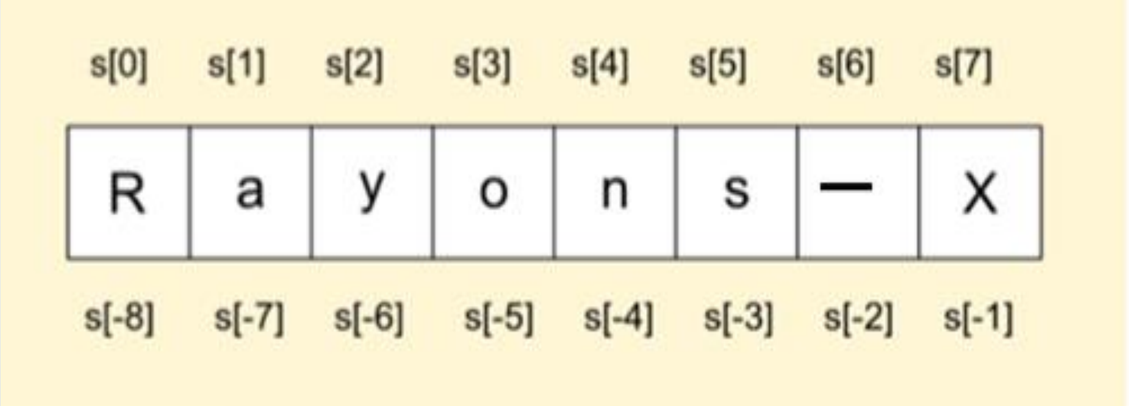
2. Les chaines de caractères

2.4 Indexation simple

Pour indexer une chaîne, on utilise l'opérateur [ ] dans lequel l'index, un entier signé qui commence à 0 indique la position d'un caractère :

```
s = "Rayons-X"          # len(s) ==> 8

    s[0] # donne 'R'
    s[2] # donne 'y'
    s[-1]# donne 'X'
    s[-3] # donne ‘s'
```



2. Les chaines de caractères

2.5 Extraction de sous-chaînes (le slicing)

- Le slicing de **séquences** consiste à extraire une sous-séquence à partir d'une séquence.
- Le slicing fonctionne de manière similaire aux intervalles mathématiques : **[début:fin[**
- La borne de fin ne fait pas partie de l'intervalle sélectionné.

La syntaxe générale est **L[i:j:k]**, où :

- i = indice de début
- j = indice de fin, le premier élément qui n'est pas sélectionné
- k = le "pas" ou intervalle (s'il est omis alors il vaut 1)

- La sous-liste sera donc composée de tous les éléments de l'indice **i** jusqu'à l'indice **j-1**, par pas de **k**.
- La sous-liste est un nouvel objet.

2. Les chaines de caractères

2.5 Extraction de sous-chaînes (le slicing)

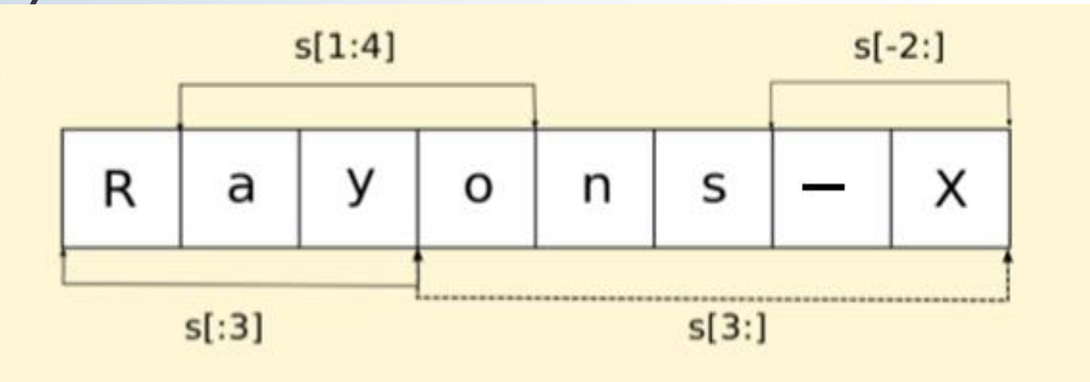
Dans le sens normal (le pas k est positif) :

- Si i est omis alors il vaut 0.
- Si j est omis alors il vaut len(L).

Dans le sens inverse (le pas k est négatif) :

- Si i est omis alors il vaut -1 .
- Si j est omis alors il vaut -len(L)-1

Exemple :




```
print(s[1:4])  # de l'index 1 compris à 4 non compris : 'ayo'
print(s[-2:] ) # de l'index -2 compris à la fin :  '-X'
print(s[:3])  # du début à l'index 3 non compris :  'Ray'
print(s[3:])   # de l'index 3 compris à la fin : 'ons-X'
print(s[::2])  # du début à la fin, de 2 en 2 : 'Ryn-'
```

3. Les listes

3.1 Déclaration

Une liste est une collection ordonnée , modifiable et dynamique d'éléments éventuellement hétérogènes.

Syntaxe :

Nom-liste =[élement1, élement2, élément3,..]

Exemples :

```
couleurs = ['blanc','rouge','noir','vert']
print(couleurs)
couleurs[1] = 14
print(couleurs)     # ['blanc', 14, 'noir', 'vert']
list1 = ['a', 'b']
list2 = [4, 2.718]
L1 = [12, 15, 16, 11, 19, 17]
L2 = [13, 20, 32]
print(L1+ L2)     # + permet de concaténer deux listes
print(L1 is L2)   # 'is' qui permet de tester si deux noms de variables désignent le même objet,
                  # retourne ici : False
```

3.2 Différence entre 'is' et '=='

M et L désignent le même objet, au même emplacement mémoire. Il s'agit de deux noms différents pour un même objet.
X et L désignent deux listes différentes, ayant les mêmes contenus.

```
L = [2,5,8]
M = L
X = [2,5,8]
print("Est-ce que M et L désignent le même objet ? ", M is L)
print("Est-ce que X et L désignent le même objet ? ", X is L)
print("avec ==")
print("Est-ce que les contenus de M et L sont les mêmes :", M == L)
print("Est-ce que les contenus de X et L sont les mêmes :", X == L)
```


résultat

```
Est-ce que M et L désignent le même objet ? True
Est-ce que X et L désignent le même objet ? False
avec ==
Est-ce que les contenus de M et L sont les mêmes :True
Est-ce que les contenus de X et L sont les mêmes : True
```

3.3 Changer de type

Il pourra parfois être utile de fabriquer une liste à partir d'un objet qui n'en est pas un.

Exemple

```
t = 'Azerty’
P=123
print(t)
t = list(t)
print(t)
print (P)
```

L'affichage obtenu :

```
Azerty
['A', 'z', 'e', 'r', 't', 'y']
```

3.4 Quelques méthodes/fonctions sur les listes

- **Ajout d'un élément à une liste :** Ajout en fin de liste avec la méthode append

L.append(élément) ajoute l'élément donné entre parenthèses à la fin de la liste L.

Exemple :

```
fruit=[]
fruit.append("orange")
print(fruit)                    # affiche ['orange']
fruit.append('pomme')
print(fruit)                    #affiche ['orange','pomme']
```

- **Ajout d'un élément à un emplacement de la liste : L.insert(pos,e)** ajoute l'élément e à la liste de telle sorte que l'index de e soit pos.

Exemple :

```
fruit.insert(2,"cerise")
print(fruit)                    #affiche['orange','pomme','cerise']
```

- **Ajout des éléments d'une liste M à une liste L : extend()**

Exemple :

```
L=['ba', 'bo', 'be', 'bi', 'bu', 'by']
M=[ 4.8, 9.3, 7.2]
L.extend(M)
print(L)                        #affiche  ['ba', 'bo', 'be', 'bi', 'bu', 'by', 4.8, 9.3, 7.2
```

- **Suppression d'un élément à l'indice i et retourner sa valeur : L.pop(i)** efface & retourne l'élément L[i], tandis que L.pop() efface & retourne le dernier élément

- **Suppression d'un élément d'une liste connaissant son index : del L[i]** supprime l'élément d'index i de la liste L.

  Exemple : del fruit[2]

- **Suppression d'un élément de liste connaissant sa valeur :L.remove(e)** supprime le premier élément de la liste L qui a la même valeur que e. Si aucun élément n'est trouvé, une erreur est retournée.

  Exemple : Fruit.remove[2]

- **Index d'un élément de la liste :** L.index(e) renvoie l'index du premier élément dont la valeur est e. Une erreur est renvoyée si e n'est pas dans la liste.

  Exemple : L=[2,3,5,8,7]
  print(L.index(8)) #affiche 3

- **Nombre d'occurrences de l'élément e:** L.count(e) indique le nombre d'occurrences de l'élément e.

  Exemple : L=[2,3,2,8,2,7]
  print(L.count(2)) #affiche 3

- **Longueur d'une liste :** len(L) renvoie la longueur de la liste L

  Exemple : L=[2,3,2,8,2,7,8]
  print(len(L)) #affiche 7

- **Trier une liste numérique**

- Dans l'ordre croissant

L.sort() trie les éléments de L (liste numérique) dans l'ordre croissant.

Exemple : L=[2,3,1,8,5,7,8]
L.sort()
Print(L) # affiche [1, 2, 3, 5, 7, 8, 8]

- Dans l'ordre décroissant **L.reverse()** permute les éléments de la liste L de telle sorte que le premier élément devient le dernier, le deuxième l'avant-dernier, etc..

Exemple :

```
L=[2,3,1,8,5,7,8]
L.sort()
L.reverse()
print(L) # affiche [8,8, 7, 5, 3, 2, 1]
```

Pour trier une liste dans l'ordre décroissant il suffit donc d'appliquer successivement les méthodes sort() puis reverse() à la liste L

- ***Exemples de quelques manipulations des listes:***

```
liste = [1, 10, 100, 25, 50]
print(liste[-1]) # Cherche la dernière occurrence et affiche 50
print(liste[-4:]) # Affiche les 4 dernières occurrences   [10, 100,25, 50]
print(liste[:]) # Affiche toutes les occurrences [1, 10, 100, 25, 50]
liste[2:4] = [69, 70]
print(liste) #affiche [1, 10, 69, 70, 50]
liste[:] = [] # vide la liste
```

- ***Exemple de* Test d'appartenance d'un élément dans une liste**

```
l=['tunis','algérie','maroc','mauritanie']
if 'tunis' in l:
    print('existant')
```

3.5 Liste de listes

Exemple : Considérons le tableau suivant :

5	**57**	**12**
3	**2**	**1**

On peut stocker les éléments de ce tableau dans une liste de la manière suivante : tableau=[[5,7,12],[3,2,1]] L'élément *3* de ce tableau est l'élément d'index 0 de l'élément d'index 1 du tableau. tableau[1][0] renvoie l'élément d'index *0* de l'élément d'index 1 du tableau (l'élément d'index 1 du tableau est la liste [3,2,1]) c'est-à-dire 3.

Exemples :

```
tab=[[5,57,12],[3,2,1]]

print(tab[0])       #affiche [5, 57, 12]
print(tab[1])       # affiche [3, 2, 1]
# print(tab[2])     # erreur ,on n'a pas l'indice 2
print(tab[1][0])    # affiche 3
```

3.6 Liste en compréhension

Les listes en compréhension sont des listes dont le contenu est défini par filtrage du contenu d'une autre liste selon un principe analogue à celui de la définition en compréhension de la théorie des ensembles.

Exemples :

1. **Copie de liste**

```
l1 = [1, 2, 3]
l2=[x for x in l1]
```

2. **Filtrage avec un test**

```
from math import sqrt
entiers=[1,-2,3,4,-5,6,7,8,-9,10]
positifs=[x for x in entiers if x>=0]
print(positifs)
racine=[sqrt(x) for x in positifs]
print(racine)
```

3.

```
animaux=['chien','lion','renard', 'carani','chats','poule']
pluriels=[nom +"s" for nom in animaux]
print(pluriels)
animaux_c=[nom for nom in animaux if nom[0]=="c"]
print(animaux_c)
```

4. Les tuples

Un tuple est un ensemble d'éléments comparable aux listes mais qui, une fois déclaré, ne peut plus être modifié.(Il s'agit donc d'une séquence **immuable** d'objets indicés qui peuvent être des nombres entiers ou décimaux, des chaînes de caractères, des listes, des tuples etc…

4.1 Syntaxe et déclaration

Un tuple est un ensemble d'éléments séparés par des virgules. Cet ensemble d'éléments est entouré de parenthèses mais ce n'est pas une obligation. Cela permet toutefois d'améliorer la lisibilité du code.

Exemples :

```
tuple1=(1, "python",2.5)
c=(5,)
print(type(c))     # affiche <class 'tuple'>
c=(5)
print(type(c))     #affiche <class 'int'>
```

4.2 Utilité d'un tuple

Au niveau des fonctions
Le tuple permet de renvoyer plusieurs valeurs lors d'un appel de fonction.

Un tuple est « protégé en écriture »

L'avantage d'un tuple, c'est qu'il s'agit d'une séquence non modifiable (immuable) donc **protégée en écriture**. Nous sommes sûrs que les données transmises par la fonction **input()** ne seront pas modifiées en cours d'exécution par un autre programme. Dans le code ci-dessus, on n'a aucun moyen pour modifier les informations transmises.

Un tuple est moins gourmand en ressources système qu'une liste.
Il a besoin de moins de mémoire et il s'exécute plus rapidement.

4.3 Opérations sur les tuples

Il n'existe pas de méthodes associées aux tuples.
Les tuples sont des séquences non modifiables donc il n'est pas possible d'utiliser les méthodes remove() ou append() par exemple.

Il n'est pas possible également d'utiliser l'opérateur [ ] pour insérer ou remplacer un élément :

```
tuple_1 = (5, 2, 25, 56)
tuple_1[2] = 23
print(tuple_1)
```

TypeError: 'tuple' object does not support item assignment.

- **L'instruction in**

Mais il est possible d'utiliser l'instruction in pour faire un test d'appartenance

```
tuple_1 = (5, 2, 25, 56)
print(2 in tuple_1) # affiche True
```

- **La fonction len()**

Il est également possible d'utiliser la fonction intégrée len() pour déterminer la longueur d'un tuple.

```
tuple_1 = (5, 2, 25, 56)
print(len(tuple_1))
```

4.4. Tuples comme arguments d'une fonction

Pour avoir un nombre variable d'arguments on utilise l'emballage de tuple : un seul argument préfixé par *****

Exemple

```
def moyenne(*nombres):
    m=sum(nombres)/len(nombres)
    return m

print(moyenne(2))          #affiche 2.0
print(moyenne(2, 4))       #affiche 3.0
print(moyenne(2, 4, 6))  #affiche 4.0
```

Dans l'exemple, args deviens un tuple qui contient tous les arguments passés à moyenne().

4.5 . Tuples comme valeurs de retour

Au sens strict, une fonction ne peut renvoyer qu'une seule valeur, mais si la valeur est un tuple l'effet est le même que de renvoyer des valeurs multiples.

Exemple

```
def min_max(t):
    return min(t), max(t)
```

5. Opérations en commun entre les séquences

Les types prédéfinis de séquences Python (chaîne, liste et tuple) ont en commun les opérations résumées dans le tableau suivant où **s** et **t** désignent deux séquences du même type et **i, j** et **k** des entiers :

L'opération	Description
x in s	True si s contient x , False sinon
x not in s	True si s ne contient pas x, False sinon
s + t	concaténation de s et t
s * n, n * s	n copies (superficielles) concaténées de s
s[i]	ieme élément de s (à partir de 0)
s[i:j]	tranche de s de i (inclus) à j (exclu)
s[i:j:k]	tranche de s de i à j avec un pas de k
len(s)	longueur de s
max(s), min(s)	plus grand, plus petit élément de s
s.index(i)	indice de la 1re occurrence de i dans s
s.count(i)	nombre d'occurrences de i dans s

6. Le parcours d'une séquence s

1ière approche :

```
for membre in s:
    instruction
    instruction
      . . .
    instruction
```

A chaque itération, *membre* contient un élément de la séquence s; le bloc d'instructions est exécuté avec l'élément *membre.*

Parcours d'une chaîne à l'aide de l'instruction while

```
mot = "papa"
i = 0
while i < len(mot):
  if mot[i] == "p" :
    print( "m")
  else:
    print(mot[i])
  i = i + 1
```

ou encore

```
mot = "papa"
for caractere in mot:
  if caractere == "p":
    print ("m")
  else:
    print (caractere)
```

Exemple :Déterminer le nombre de voyelles dans une chaîne.

2ième approche :

```
for indice in range(len(s)):
    instruction
    instruction
      . . .
    instruction
```

A chaque itération, indice contient l'indice d'un élément de la séquence s;
le bloc d'instructions est exécuté avec *l'indice d'un membre*.

Note : **range(len(s)) désigne la liste des indices des éléments de s.**

```
mot = "saperlipopette"
Nombre_de_voyelles = 0
Voyelles = "aeiouyAEIOUY"
for indice in range(len(mot)):
        if (mot[indice] in Voyelles):
            Nombre_de_voyelles = Nombre_de_voyelles + 1
print (Nombre_de_voyelles)
```

La fonction **range()** dispose de plusieurs syntaxes :

- **range(fin)** : retourne une liste de nombres de 0 jusqu'à fin - 1.

```
>>> range(5)
[0, 1, 2, 3, 4]
```

- **range(debut, fin)** : retourne une liste de nombres de début jusqu'à fin - 1.

```
>>> range(3, 8)
[3, 4, 5, 6, 7]
```

- **range(debut, fin, pas)** : retourne une liste de nombres de début jusqu'à fin – 1 par incréments de pas.

```
>>> range(3, 13, 4)
[3, 7, 11]
```

3ième approche :

On peut aussi parcourir séquentiellement une séquence s par élément et par indice.

```
for indice, membre in enumerate(s):
    instruction
    instruction
      . . .
    instruction
```

A chaque itération, indice et membre réfèrent à un élément de la séquence s;
le bloc d'instructions est exécuté avec l'élément *membre et l'indice de ce membre*.

```
mot = "saperlipopette"
Nombre_de_voyelles = 0
Voyelles = "aeiouyAEIOUY"
for indice, caractere in enumerate(mot):
if(caractere in Voyelles):
    Nombre_de_voyelles = Nombre_de_voyelles + 1
    print (indice, caractere)
1 a
3 e
6 i
8 o
10 e
13 e
```