Cours
Programmation Python

Chapitre 1 : Les Structures de Données

1. Chaines de caractères « String » :

Chaîne de caractères « ou string » signifie une suite finie de caractères. Une chaine de caractères est compris entre **deux guillemets** ou **deux apostrophes**. Le type python est **str**. L'exemple suivant montre comment créer une chaîne de caractères.

```
t = "string = texte"
print(type(t), t)
t = 'string = texte, initialisation avec apostrophes'

print(type(t), t)

t = "morceau 1" \

     "morceau 2" # second morceau ajouté au premier par l'ajout du symbole \,
# il ne doit rien y avoir après le symbole \,
# pas d'espace ni de commentaire

print(t)

t = """première ligne
seconde ligne"""  # chaîne de caractères qui s'étend sur deux lignes

print(t)
x = 5.567
s = str(x)

print(type(s), s) # <type 'str'> 5.567

print(len(s)) # affiche 5

s1 = "C:\\Users\\Dupre\\exemple.txt" # tous les '\' doivent être doublés
```

- La fonction **str** permet de convertir un nombre, un tableau, etc, en chaîne de caractères afin de pouvoir l'afficher.
- Les opérateurs qu'on peut les appliquer sur les chaînes de caractères sont résumés dans le tableau ci-dessous :

opérateur	Signification	Exemple
`+`	concaténation de chaînes de caractères	`t = "abc" + "def"`
`+=`	concaténation puis affectation	`t += "abc"`
`in`, `not in`	une chaîne en contient-elle une autre ?	`"ed" in "med"`
`*`	répétition d'une chaîne de caractères	`t = "abc" * 4`
`[n]`	obtention du enième caractère, le premier caractère a pour indice 0	`t = "abc"; print(t[0]) # donne a`
`[i:j]`	obtention des caractères compris entre les indices `i` et `j-1` inclus.	`t = "abc"; print(t[0:2]) # donne ab`

Les chaînes de caractères sont (outre les opérateurs) aussi manipulées par des fonctions sous la forme : ***res = s.fonction (...)****, où* ***'s'*** *est la chaine,* ***'fonction'*** *est le nom de l'opération et* ***'res'*** *est le résultat de la manipulation.*

`count( sub[, start[, end]])`	Retourne le nombre d'occurrences de la chaîne de caractères `sub`, les paramètres par défaut `start` et `end` permettent de réduire la recherche entre les caractères d'indice `start` et `end` exclu. Par défaut, `start` est nul tandis que `end` correspond à la fin de la chaîne de caractères.
`find( sub[, start[, end]])`	Recherche une chaîne de caractères `sub`, les paramètres par défaut `start` et `end` ont la même signification que ceux de la fonction `count`. Cette fonction retourne -1 si la recherche n'a pas abouti.
`isalpha()`	Retourne `True` si tous les caractères sont des lettres, `False` sinon.
`isdigit()`	Retourne `True` si tous les caractères sont des chiffres, `False` sinon.
`replace( old, new[, count])`	Retourne une copie de la chaîne de caractères en remplaçant toutes les occurrences de la chaîne `old` par `new`. Si le paramètre optionnel`count` est renseigné, alors seules les `count` premières occurrences seront remplacées.
`split( [sep [,maxsplit]])`	Découpe la chaîne de caractères en se servant de la chaîne `sep`comme délimiteur. Si le paramètre `maxsplit` est renseigné, au plus `maxsplit` coupures seront effectuées.
`upper()`	Remplace les minuscules par des majuscules.
`lower()`	Remplace les majuscules par des minuscules.
`join(li)`	`li` est une liste, cette fonction agglutine tous les éléments d'une liste séparés par `sep` dans l'expression `sep.join ( ["un", "deux"])`.
`startswith(prefix[, start[,end]])`	Teste si la chaîne commence par `prefix`.
`endswith(suffix[, start[,end]])`	Teste si la chaîne se termine par `suffix`.

Exemple :

```
st = "langage python"
st = st.upper()                # mise en lettres majuscules
i = st.find("PYTHON")          # on cherche "PYTHON" dans st
print(i)                       # affiche 8
print(st.count("PYTHON"))      # affiche 1
print(st.count("PYTHON", 9))   # affiche 0
s = "un;deux;trois"
mots = s.split(";")            # mots est égal à ['un', 'deux', 'trois']
mots.reverse()                 # retourne la liste, mots devient égal à
#                   ['trois', 'deux', 'un']
s2 = ";".join(mots)            # concaténation des éléments de mots séparés par ";"
print(s2)
```

2. Les tableaux: Array

Les tableaux en Python sont un moyen compact de collecter les types de données de base, toutes les entrées d'un tableau doivent être du même type de données.

Les tableaux sont pris en charge par le module de tableau et doivent être importés avant de commencer à les initialiser et à les utiliser. Les éléments stockés dans un tableau sont contraints dans leur type de données. Le type de données est spécifié lors de la création du tableau et spécifié à l'aide d'un code de type, qui est un caractère unique comme dans l'exemple ci-dessous :

✓ **Exemple :**

```
import array as arr        # importer le module array et l'initialiser dans arr
a1= arr.array("I",[3,6,9])           # creation d'un array 'a1' avec
                                     # un type code 'I': unsigned int
a2= arr.array('l', [1, 2, 3, 4, 5])  # creation d'un array 'a2' avec
                                     # un type code 'l': signed long
a3= arr.array('d', [1.0, 2.0, 3.14]) # creation d'un array 'a3' avec
                                     # un type code 'l': double
```

Remarque : Les tableaux ne sont pas très utilisés en Python

3. Les Listes

- Les listes sont des collections d'objets qui peuvent être de types différents. Elles sont modifiables.
- Une liste apparaît comme une succession d'éléments compris entre crochets et séparés par des virgules.

```
x = [4,5]  # création d'une liste composée de deux entiers

x = ["un",1,"deux",2] # création d'une liste composée de # 2 chaînes de
caractères et 2 entiers

x = [3,] # création d'une liste d'un élément,

x = [ ] # crée une liste vide
x = list ()# crée une liste vide
y = x [0]  # accède au premier élément
y = x [-1]# accède au dernier élément
```

Les opérateurs et les fonctions qu'on peut les appliquer sur les listes sont présentés par la table suivante :

x in l	vrai si x est un des éléments de l
x not in l	réciproque de la ligne précédente
l + t	concaténation de l et t
l * n	concatène n copies de l les unes à la suite des autres
l[i]	retourne l'élément i'ème élément de l,
l[i:j]	retourne une liste contenant les éléments de l d'indices *i* à *j* exclu.
len(l)	nombre d'éléments de l
min(l)	plus petit élément de l, résultat difficile à prévoir lorsque les types des éléments sont différents
max(l)	plus grand élément de l
sum(l)	retourne la somme de tous les éléments
del l [i:j]	supprime les éléments d'indices entre i et j exclu. Cette instruction est équivalente à l [i:j] = [].
list (x)	convertit x en une liste quand cela est possible
l.count (x)	Retourne le nombre d'occurrences de l'élément x.
l.index (x)	Retourne l'indice de la première occurrence de l'élément x dans la liste l.
l.append (x)	Ajoute l'élément x à la fin de la liste l.
l.extend (k)	Ajoute tous les éléments de la liste k à la liste l. La liste l aura autant d'éléments supplémentaires qu'il y en a dans la liste k.

4. Les Tuples

- Un tuple est un tableau d'objets qui peuvent être de types différents. Les tuples ne sont pas modifiables.
- Un tuple apparaît comme une liste d'objets comprise entre parenthèses et séparés par des virgules.

```
x = (4,5)                # création d'un tuple composé de 2 entiers
x = ("un",1,"deux",2)    # création d'un tuple composé de 2 chaînes de caractère
s
                         # et de 2 entiers, l'ordre d'écriture est important
x = (3,)                 # création d'un tuple d'un élément, sans la virgule,
                         # le résultat est un entier
```

- ***Remarque 1*** *: Les tuples ne sont pas modifiables (ou mutable), cela signifie qu'il est impossible de modifier un de leurs éléments.*

```
a = (4,5)
a [0] = 3      # déclenche une erreur d'exécution
```

- ***Remarque 2 :*** Les opérateurs et les fonctions qu'on peut les appliquer sur les tuples sont présentés par la table suivante :

`x in s`	vrai si `x` est un des éléments de `s`
`x not in s`	réciproque de la ligne précédente
`s + t`	concaténation de `s` et `t`
`s * n`	concatène `n` copies de `s` les unes à la suite des autres
`s[i]`	retourne le ième élément de `s`
`s[i:j]`	retourne un tuple contenant une copie des éléments de `s` d'indices *i* à *j* exclu
`s[i:j:k]`	retourne un tuple contenant une copie des éléments de `s` dont les indices sont compris entre *i* et *j* exclu, ces indices sont espacés de *k*.
`len(s)`	nombre d'éléments de `s`
`min(s)`	plus petit élément de `s`, résultat difficile à prévoir lorsque les types des éléments sont différents
`max(s)`	plus grand élément de `s`
`sum(s)`	retourne la somme de tous les éléments

5. Les Dictionnaires :

Les dictionnaires sont des tableaux plus souples que les listes.

- Une liste référence les éléments en leur donnant une position : la liste associe à chaque élément une position entière comprise entre **0** et **n-1** si n est la longueur

de la liste.

- Un dictionnaire permet d'associer à un élément autre chose qu'une position entière : ce peut être un entier, un réel, une chaîne de caractères, un tuple contenant des objets immuables.
- *D'une manière générale, un dictionnaire associe à une valeur ce qu'on appelle une clé de type immuable. Cette clé permettra de retrouver la valeur associée.*

Définition : dictionnaire
Les dictionnaires sont des listes de couples. Chaque couple contient une clé et une valeur. Chaque valeur est indicée par sa clé. La valeur peut-être de tout type, la clé doit être de type immuable, ce ne peut donc être ni une liste, ni un dictionnaire. Chaque clé comme chaque valeur peut avoir un type différent des autres clés ou valeurs.

- *Un dictionnaire apparaît comme une succession de couples d'objets comprise entre accolades et séparés par des virgules. La clé et sa valeur sont séparées par le symbole **':'***

```
x = { "cle1":"valeur1", "cle2":"valeur2" }
y = { }         # crée un dictionnaire vide
z = dict()      # crée aussi un dictionnaire vide
```

```
x = {"cle1": "valeur1", "cle2": "valeur2"}
print(x["cle1"])        # affiche valeur1
```

- La table suivante dresse la liste des opérations sur les dictionnaires.

`x in d`	vrai si `x` est une des clés de `d`
`x not in d`	réciproque de la ligne précédente
`d[i]`	retourne l'élément associé à la clé `i`
`len(d)`	nombre d'éléments de `d`
`min(d)`	plus petite clé

`max(d)`	plus grande clé
`del d [i]`	supprime l'élément associé à la clé `i`
`list (d)`	retourne une liste contenant toutes les clés du dictionnaire `d`
`dict (x)`	convertit `x` en un dictionnaire si cela est possible, `d` est alors égal à `dict ( d.items () )`
`d.copy ()`	Retourne une copie de `d`
`d.items ()`	Retourne un itérateur sur tous les couples (clé, valeur) inclus dans le dictionnaire.
`d.keys ()`	Retourne un itérateur sur toutes les clés du dictionnaire `d`
`d.values ()`	Retourne un itérateur sur toutes les valeurs du dictionnaire `d`
`d.get (k[,x])`	Retourne `d[k]`, si la clé `k` est manquante, alors la valeur `None` est retournée à moins que le paramètre optionnel `x` soit renseigné, auquel cas, ce sera cette valeur qui sera retourné.
`d.clear ()`	Supprime tous les éléments du dictionnaire.
`d.update(d2)`	Le dictionnaire `d` reçoit le contenu de `d2`.
`d.setdefault(k[,x])`	Définit `d[k]` si la clé `k` existe, sinon, affecte `x` à `d[k]`
`d.pop()`	Retourne un élément et le supprime du dictionnaire.

6. Les ensembles: Set

a. set :

Un set est un ensemble de valeurs uniques. Ajouter une valeur déjà dans la liste n'a donc aucun impact. Ce type est dit mutable car il est possible d'ajouter des valeurs. Comme pour les dictionnaires, les valeurs stockées dans un ensemble doivent être immuables.

```
x_set = set('CAKE&COKE')
y set = set('COOKIE')

print(x_set)         # {'A', '&', 'O', 'E', 'C', 'K'}
print(y_set)         # {'I', 'O', 'E', 'C', 'K'}
print(x - y)         # All the elements in x set but not in y set
print(x set - y set) # All the elements in x set but not in y set
print(x_set|y_set)   # Unique elements in x_set or y_set or both
print(x_set & y_set) # Elements in both x_set and y_set
```

b. frozenset : Une version immuable de type **set.** Bien que les éléments d'un **'set'** sont modifiable, les éléments d'un **'frozenset'** restent les mêmes après la création. Pour cette raison, on peut utiliser les **'frozenset'** comme clé dans un dictionnaire.