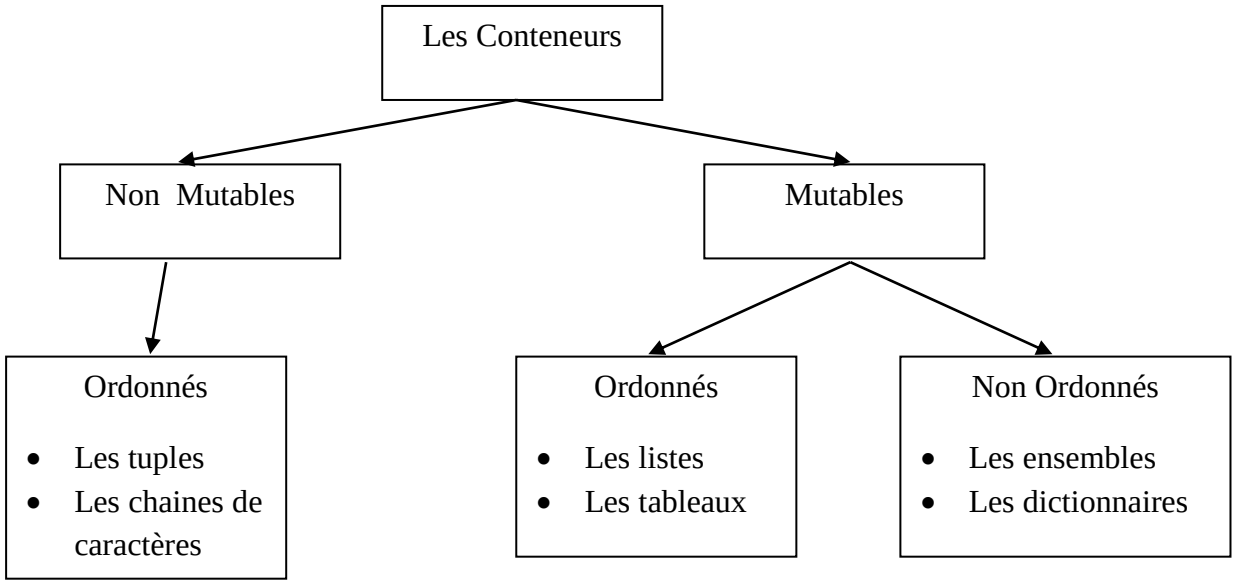
Cours Introduction à Python
Partie 3 : structures de données : Les types mutables

Introduction

Ce sont des données ou objets composites destinés à contenir d'autres objets (une variable dans laquelle on peut mettre plusieurs variables).
On distingue des types modifiables ou mutables et des types non mutables.
1) Mutables et Non Mutables

- Mutable: modification autorisée
- non mutable: modification non autorisée

2) Ordonnés et Non ordonnés



A. Les Listes

Une liste est une séquence ordonnée et modifiable d'objets homogènes ou hétérogènes séparés par des virgules et délimités par des crochets []. Elle supporte la répétition et appartient au type list.

Fonctions	**Syntaxe**	**Exemples**	**Résultats**
Créer une liste vide		>>>L = [ ]	.....
Créer une liste hétérogène		>>>L=[1, 6, 1.3, 'hello', 'bb']	.....
Afficher le contenu de la liste		>>> print (L) #ou >>> L	
Créer une liste de listes	nom_liste = [L1, L2, L3]	>>> enclos1 = ['girafe', 4] >>> enclos2 = ['tigre', 2] >>> enclos3 = ['singe', 5] >>> zoo = [enclos1, enclos2, enclos3]	....
Créer une liste en utilisant la fonction list()	nom_liste = **list**(range(deb,fin,pas))	L=**list**(range(0,1000,200))	....
Ajouter une valeur à la fin de la liste	nom_liste.**append**(valeur)	liste = [] liste.append(1)	....
Ajouter un élément à une position donnée dans une liste	nom_liste.**insert**(pos,valeur)	nb=[11, 14, 17, 2, 17, 3, 6] nb.insert(3,15)	....

Nombre des éléments	**len** (nom_liste)	len (L)	.....
Accèder à un élément de la liste : nom_liste = ['a','b',…,'y','z'] nom_liste [index] avec : index ∈ [0,1,…,len(nom_liste)-1] (de droite à gauche) index ∈ [-len(nom_liste),…,-2,-1] (dans le sens inverse)		L = ["a","d","m"] L [0] L [-1] L [1] = = L [-2]	....
Accès aux éléments d'une liste : **nom_liste [indice début : indice fin : pas]** tels que : *indice début*, *indice fin* et *pas* sont optionnels ; Les valeurs par défaut sont : pas = 1 Si pas > 0 : indice début = 0 indice fin = len (nom_liste) – 1 arrêt = indice fin – 1 (si indice fin existe) Si pas < 0 : indice début = len (nom_liste) - 1 indice fin = 0 arrêt = indice fin + 1 (si indice fin existe)		L = [0,1,2,3,4,5,6,7,8,9,10,11] L [0 **:** len(L)-1 : 1] L [0 **:** len(L)-1] L [:] L [: :] L [: : 1] L [: : 2] L [: : -1] L [: : -3] L [3 : : 2] # fin=11 L [3 : 8 : 2] # fin = 8-1 L [8 : 3 : -2] #fin = 3+1 L [8 : : -2] # fin = 0 L [-8 : : -1] L [-8 : : 1]	....
Les n premiers éléments	nom_liste [: n]	L [: 2]	....
Les n dernières éléments	nom_liste [-n :]	L [-1 :] L [-3 :]	....
Les éléments entre deux index n et m	nom_liste [n : m+1]	L [1 : 3]	....
Inverser les valeurs	nom_liste.**reverse**()	L.reverse()	.....
Nombre d'occurrences d'une entrée	nom_liste.**count**(valeur)	L = ["a","a","a","b","c","c"] L.count("a")	....
Trouver l'index d'une valeur	nom_liste.**index** (valeur)	L.index("b")	....
Supprimer un élément de la liste	del (nom_liste[position]) nom_liste.remove (élément)	del (L[0]) L.remove ("b")	....
Vider la liste		L[:] = []	.....
Supprimer définitivement la liste	del (nom_liste)	del (L)	....
Boucler sur une liste (Les valeurs retournées par la boucle contenant la fonction enumerate sont des **tuples)**		for i in L: print (i) for i in range (len (L)): print (L[i]) for i in **enumerate** (L) : print (i)	....
Copie indépendante d'une liste	import copy nouv_liste = copy.**deepcopy** (ancienne_liste) **ou :** from copy import deepcopy nouv_liste = **deepcopy** (ancienne_liste)	#copie dépendante >>> x = [[1,2], 2] >>> y = x >>> y[1] = [1,2,3] ;x ;y #copie indépendante >>> import copy >>> x = [[1,2], 2] >>> y = copy.deepcopy(x) >>> y[1] = [1,2,3] ;x ;y	....

Transformer une liste en chaine de caractères	"séparateur".**join**(nom_liste) NB : Les éléments de la liste doivent être de type **str**	":".join(L)	…… …… ……
Tester si un élément existe dans une liste	élément **in** nom_liste élément **not in** nom_liste	3 **in** L 11 **not in** L	…… ……
Agrandir une liste ***liste1*** par la liste ***liste2***	Liste1**.extend**(liste2) ou Liste1 = Liste1 + liste2	x = [1, 2, 3, 4] y = [4, 5, 1, 0] x**.extend**(y) x = x + y	…… …… …… ……
Répéter une liste	nom_liste * nombre_entier	x *3 [0] * 5	…… ……
Trier les éléments d'une liste	nom_liste**.sor**t()	L.**sort**()	…… ……
Retourner le maximum, le minimum et la somme des éléments d'une liste		**max**(L), **min**(L), **sum**(L)	…… …… ……
Comparer le contenu de deux listes	list1 = = liste2	L1 = [0, 1, 2, 3, 4, 5, 6] L2 = [4, 5, 6, 7, 8, 9, 5] L1 = = L2	…… …… ……
Les Listes en compréhension : Permet de récupérer une nouvelle liste avec tous les éléments de l'ancienne liste filtrés et sur lesquels on a effectué (ou non) un traitement.	*[expression **for** élément **in** nom_liste **if** prédicat]* Le **prédicat** étant optionnel, la syntaxe peut donc se réduire à ceci : *[expression **for** élément **in** nom_liste]* *expression* : peut-être elle-même une liste en compréhension	L = [i for i in range(10)] L L1 = [i+ 1 for i in L] L1 L2 = [i ** 2 for i in L] L2 L3 = [i for i in L if i % 2 == 0] L3 L4 = [i ** 4 for i in L if i < 7] L4	…… …… …… …… …… …… …… …… …… …… …… ……

Exercices [Listes]

1. Ecrire un programme python qui permet de saisir une liste de 5 entiers strictement positifs

2. Soient les listes suivantes : t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
   t2 = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Aout', 'Septembre', 'Octobre', 'Novembre', 'Décembre'].
   Ecrivez un petit programme qui crée une liste t3. Celle-ci devra contenir tous les éléments des deux listes en les alternant, de telle manière que chaque nom de mois soit suivi du nombre de jours correspondant : t3 = ['Janvier',31, 'Février', 28, 'Mars', 31, etc…]

3. Ecrivez un programme qui affiche « proprement » tous les éléments de la liste. Si on l'appliquait par exemple à la liste t2 de l'exercice ci-dessus, on devrait obtenir :
   Janvier Février Mars Avril Mai Juin Juillet Aout Septembre Octobre Novembre Décembre

4. Ecrivez un programme qui recherche le plus grand élément présent dans une liste donnée. Par exemple, si on l'appliquait à la liste [32, 5, 12, 8, 3, 75, 2, 15], ce programme devrait afficher :
   le plus grand élément de cette liste a la valeur : 75

5. Ecrivez un programme qui analyse un par un tous les éléments d'une liste de nombres (par exemple celle de l'exercice précédent) pour générer deux nouvelles listes. L'une contiendra

seulement les nombres pairs de la liste initiale, et l'autre les nombres impairs. Par exemple, si la liste initiale est celle de l'exercice précédent, le programme devrait construire une liste pairs qui contiendra [32, 12, 8, 2], et un liste impairs qui contiendra [5, 3, 75, 15].

6. Ecrivez un programme qui analyse un par un tous les éléments d'une liste de mots (par exemple ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin']) pour générer deux nouvelles listes. L'une contiendra seulement les mots comportant moins de 6 caractères, l'autre les mots comportant 6 caractères ou d'avantage.

B. Les dictionnaires

Un **dictionnaire** en **python** est une sorte de **liste** mais au lieu d'utiliser des **index**, on utilise des **clés**, c'est à dire des valeurs autres que numériques. C'est un type de données **non ordonné**, **mutable** et appartenant au type **dict** de python et il permet de stocker des couples « **clé : valeur** ».

Les clés de dictionnaire pouvaient être des entiers, des chaînes et «quelques autres types», les tuples sont un de ces types. Ils peuvent être utilisés comme clé dans une chaine de caractères, ce qui n'est pas le cas des listes. Les clés de dictionnaire doivent être non-mutables. Les tuples sont non-mutables mais si vous avez un tuple contenant des listes, il est considéré comme mutable et n'est pas utilisable comme clé de dictionnaire. Seuls les tuples de chaînes, de nombres ou d'autres tuples utilisable comme clé peuvent être utilisés comme clé de dictionnaire.

Fonctions	Syntaxe	Exemples	Résultats
Créer un dictionnaire vide	nom_dict = { }	d = {} type(d)	…… ……
Créer un dictionnaire non vide	nom_dic = {"clé1": "val1", "clé2": val2}	d = {"nom": "Chebl", "age": 12}	…… ……
Ajouter des valeurs sous la forme d'un couple : (clé, valeur)	nom_dic["clé"]="valeur"	d ["prenom"] = "Moutie"	…… ……
	nom_dic.__setitem__(cle, val)	d.__setitem__("age",20)	……
Récupérer une valeur	nom_dic.get ("cle")	d.get("nom")	…… ……
	nom_dic["cle"]	d["nom"]	
Supprimer une entrée	del nom_dic ["cle"]	del d["nom"]	……
Récupérer les clés par une boucle		for clef in d.keys(): print (clef)	…… ……
Récupérer les valeurs par une boucle		for valeur in d.values(): print (valeur)	…… ……
Récupérer les clés et les valeurs par une boucle		for clef,valeur in d.items(): print (clef, valeur)	…… ……
Utiliser des tuples comme clé. La combinaison tuple/dictionnaire qui fait des merveilles dans certains cas comme lors de l'utilisation de coordonnées. (à voir ultérieurement)	nom_dic[tuple] = valeur	b = {} b[(3,2)]=12 b[(4,5)]=13 b	…… …… …… …… ……
Créer une copie indépendante d'un dictionnaire	Comme pour toute variable, vous ne pouvez pas copier un dictionnaire en faisant	#copie dépendante d = {"c1":"v1", "c2":"v2"} e = d	…… …… …… ……

	dic2 = dic1 Pour créer une copie indépendante vous pouvez utiliser la méthode copy dic2 = dic1.copy()	d["c1"] = "X" e #copie indépendante d = {"c1":"v1", "c2":"v2"} e = d.copy() d["c1"] = "X" e	…… …… …… …… …… …… ……
Vider un dictionnaire	nom_dic.clear()	d.clear()	……
Fusionner le deuxième dictionnaire sur le premier	dic1.update(dic2)	e.update({"c3":25})	…… …… ……
Comparer le contenu	dict1 = = dict2	d = = e d != e	…… ……

Exercices [dictionnaires]

1. Ecrivez un script qui crée un mini-système de base de données fonctionnant à l'aide d'un dictionnaire, dans lequel vous mémoriserez les noms d'une série de copains, leur age et leur taille. Votre script devra comporter deux parties : la première pour le remplissage du dictionnaire, et la seconde pour sa consultation. Dans la partie de remplissage, utilisez une boucle pour accepter les données entrées par l'utilisateur. Dans le dictionnaire, le nom de l'élevé servira de clé d'accès, et les valeurs seront constituées de tuples (age, taille), dans lesquels l'âge sera exprimé en années (donnée de type entier), et la taille en mètres (donnée de type réel). La partie de consultation comportera elle aussi une boucle, dans laquelle l'utilisateur pourra fournir un nom quelconque pour obtenir en retour le couple « age, taille » correspondant. Le résultat de la requête devra être une ligne de texte bien formatée, telle par exemple : « Nom : Maher - age : 15 ans - taille : 1.74 m ». Pour obtenir ce résultat, servez-vous du formatage des chaines de caractères.

2. Ecrivez un petit programme utilitaire, qui puisse vous aider à construire facilement et rapidement un nouveau dictionnaire de couleurs, lequel permettrait l'accès technique à une couleur quelconque par l'intermédiaire de son nom usuel en français. Vous savez cependant qu'un ordinateur ne peut traiter que des informations numérisées. Cela implique que la désignation d'une couleur quelconque doit nécessairement tôt ou tard être encodée sous la forme d'un nombre. Il faut bien entendu adopter pour cela une convention, et celle-ci peut varier d'un système à un autre. L'une de ces conventions, parmi les plus courantes, consiste à représenter une couleur à l'aide de trois octets, qui indiqueront respectivement les intensités (entre 0 et 255) des trois composantes rouge, verte et bleue de cette couleur. La couleur sera codée a l'aide d'une chaine de 7 caractères telle que '#00FA4E'. Dans cette chaine, le premier caractère (#) signifie que ce qui suit est une valeur hexadécimale. Les six caractères suivants représentent les 3 valeurs hexadécimales des 3 composantes rouge, vert et bleu ('#RRVVBB').
   Exemple : La couleur verte de code hexadécimal est #00FF00 qui correspond à 02550 en décimal et en binaire égale à 000000001111111100000000.
   Le but du présent exercice est de réaliser un programme qui construit un dictionnaire de couleurs en français qui puisse vous aider à accéder facilement et rapidement au code d'une couleur quelconque. Une fois construit, ce dictionnaire serait donc de la forme :
   {'vert':'#00FF00', 'bleu':'#0000FF', ... etc ...}.
   a. Saisir un entier entre 0 et 255 pour chaque intensité
   b. Convertir ces trois intensités en hexadécimal
   c. Créer la chaine de caractères correspondante à la couleur
   d. Donner un nom à cette couleur et enregistrer la dans le dictionnaire
   Astuce : Vous pouvez utiliser l'utilitaire « Paint » de Windows pour le choix de nom de couleur.

C. Les ensembles

Un ensemble en Python est une collection modifiable d'objets sans répétition et sans ordre (donc sans numérotage). Il appartient au au type **set**. Il est créé par la séquence de ses éléments (valeurs), encadrées par { et }, ou en utilisant le constructeur **set** appliqué sur un itérable.

Fonctions	Syntaxe	Exemples	Résultats
Créer un ensemble vide	nom_ensemble = set()	e = set() type(e)	.....
Créer un ensemble non vide	ensemble = {e1,e2,…} ou ensemble = **set**([e1, e2…])	s ={1,1,3,3,'a','b','ccc' ,2,2,1,'a','a','bb'} s s1 = set([1, 2, 3]) s1	....
Ajouter un élément à un ensemble	nom_ensemble**.add**(valeur)	s.add(9) s	....
Tester l'appartenance d'un élément	valeur **in** nom_ensemble valeur **not in** nom_ensemble	1 in s 1 not in s	....
Supprimer un élément donné	nom_ensemble**.remove**(valeur)	s.remove(1) s	....
Vider un ensemble	nom_ensemble**.clear**()	s.clear() s	....
La taille d'un ensemble	**len**(nom_ensemble)	len(s)	.....
Tester l'égalité ou l'inégalité entre deux ensembles	ens1 = = ens2	s = = s1 s != s1	....

Opérations ensemblistes

Les diverses opérations ensemblistes, définies dans le cours de mathématiques, sont réalisables avec des `set`. Supposons, par exemple, que l'on ait défini les deux ensembles suivants :

```
>>> S1 = set([1, 2, 3])
>>> S2 = set([0, 1])
```

Union : >>> S1.union(S2) ou bien >>> S1 | S2
Intersection : S1.intersection(S2) ou bien >>> S1 & S2
Différence : S1.difference(S2) ou bien >>> S1-S2
Différence symétrique : S1.symmetric_difference(S2) ou bien >>> S1 ^ S2
Inclusion (tester si S1 est inclus dans S2) : >>> S1.issubset(S2) >>> B.issuperset(A)

Exercices [Ensembles]

Définir deux ensembles (*sets*) X = {'*a*', '*b*', '*c*', '*d*'} et Y = {'s', '*b*', '*d*'} puis affichez les résultats suivants:

1. Les ensembles initiaux ;
2. Le test d'appartenance de l'élément 'c' à X ;
3. Le test d'appartenance de l'élément 'a' à Y ;
4. Les ensembles X -Y et Y-X ;
5. L'ensemble $X \cup Y$ (union) ;
6. L'ensemble $X \cap Y$ (intersection).