Type Construits

I. Introduction

On a vu des objets en python de type *int*, *float*, *str* et *bool* et on a souvent manipuler une variable contenant une valeur. On peut facilement voir qu'il serait utile d'avoir des variables contenant plusieurs valeurs : des coordonnées, une liste de noms, une liste de notes, etc...
À partir du moment où on parle d'une collection de valeur il vient le problème de l'organisation de ces valeurs, l'ordre, etc. On a donc **plusieurs types d'objets** pour manipuler une collection de valeur. On les appelle des **types construits**.

II. Tuples

1. Objet modifiable ou persistant

Définition .1.
On dira qu'un objet de type construit est **modifiable** ("mutable" en anglais) si on peut remplacer une des valeurs par une nouvelle valeur, changer la longueur de l'objet, etc. On dit qu'un objet est **persistant** ou **immuable** ("immutable" en anglais) s'il n'est pas modifiable une fois construit.

R **Interêt de l'immuabilité**
- plus rapide d'accès dans la mémoire
- nécessite moins de précautions d'emploi
- l'interpréteur peut gérer l'allocation mémoire de l'objet plus finement, notamment en partageant la représentation mémoire d'objet immuable qui interviennent plusieurs fois

2. Définition des tuples et utilisation en Python

Définition .2.
Un **tuple** ou **p-uplet** ("tuple" en anglais) est une collection immuable de valeur. Cette collection est **indexée**, c'est-à-dire qu'à chaque valeur est attribuée sa position dans la collection.

En Python : Les tuples sont de type *tuple* en python et sont définis en séparant les valeurs par une virgule et en entourant la collection de valeur par des parenthèses.

Exemple de création :

▪ **Exemple .1.**
Exemple de création

```
>>> a=(3,89,-4) # Creation d'un tuple a 3 elements
>>> b=() # Creation d'un tuple vide
>>> c=(2,) # Creation d'un tuple a 1 element (
   attention Ã  la virgule)
```

▪

Opérateur + : Pour concaténer deux tuples, on peut utiliser l'opérateur "+"

```
>>> a=(3,89,-4)
>>> c=(2,)
>>> d = a + c
>>> print(d)
(3,89,-4,2)
```

Opérateur "*" : On peut créer un tuple par répétition d'un tuple avec l'opérateur "*"

```
>>> a=(3,89,-4)
>>> e=a*3
>>> print(c)
(3,89,-4,2,3,89,-4,2,3,89,-4,2)
```

Tester l'appartenance :

```
>>> a=(3,89,-4)
>>> 3 in a
True
>>> 2 in a
False
```

Position et longueur :

```
>>> a=(3,89,-4)
>>> a[0] # Premiere valeur (en position 0)
3
>>> a[2] # Troisieme valeur (en position 2)
-4
>>> len(a) # Longueur de a
3
>>> a[-1] # Avant-derniere valeur (en position "-1")
-4
```

Remarque : Les parenthèses
Lorsqu'il n'y a pas d'ambiguïtés, python reconnaît un tuple sans les parenthèses.

```
>>> a = 3, 4, 5
>>> type(a)
< class 'tuple' >
>>> b = 3,
>>> type(b)
< class 'tuple' >
```

On peut mettre un tuple dans un tuple, il est alors nécessaire d'utiliser les parenthèses

```
>>> t = 3, 4, (2,1), 5
>>> type(t)
< class 'tuple' >
>>> print(t)
(3,4,(2,1),5)
```

Exemple d'utilisation : les affectations multiples

```
def coordFonctionCarre(a):
        return (a, a**2)  # Ici la fonction renvoie un tuple

x,y = coordFonctionCarre(5)
```

■ Exercice .1.

1. Que renvoie les instructions suivantes ? 2

(a)
```
>> a = 3
>> type(a)
```

(b)
```
>> a = "Bonjour"
>> type(a)
```

(c)
```
>> a = 3.2
>> type(a)
```

(d)
```
>> a = (2)
>> type(a)
```

(e)
```
>> a = (2,)
>> type(a)
```

(f)
```
>> a = (2,3,4,90)
>> type(a)
```

(g)
```
>> a = (2, "oui", "bonjour", 23, (3,2), 43)
>> type(a)
```

2. Ecrire la ligne de code permettant d'affecter à la variable "couleur" un tuple contenant le triplet RGB de la couleur rouge.
3. On considère la variable *triplet* qui est un tuple à 3 éléments : (3,22,45)
   (a) Quelle instruction renvoie la deuxième valeur (donc 22) de *triplet* ?
   (b) Peut-on remplacer le 3 par un 2 dans ce tuple ?
   (c) Peut-on ajouter une quatrième valeur dans ce tuple ?

■

3. Cas particulier des chaînes de caractères

Les chaînes de caractère (type "*str*" en Python) possèdent des similarités au tuples
— Une chaîne de caractère est indexée
— On ne peut pas modifier par indexation une chaîne de caractère

Exemples :

1. Qu'affiche les instructions suivantes ?

(a)
```
>> chaine = "Bonjour"
>> type(chaine)
```

(b)
```
>> chaine = "Bonjour"
>> chaine[0]
```

(c)
```
>> chaine = "Bonjour"
>> chaine[-1]
```

(d)
```
>> chaine = "Bonjour"
>> chaine = chaine + " toi"
>> print(chaine)
```

2. On a une variable *mot* qui est une chaîne de caractère "Morengis". Peut-on remplacer le "e" par un "a" dans la variable *mot* ?

III. Listes

1. Généralités

1..1 Définir une liste

Définition .3.
Les listes pythons sont des **variables** dans lesquelles on peut mettre plusieurs **variables**.

▪ **Exemple .2.**
L = [1,2,3,4,5,6] permet de définir une liste dans python. Cette liste comporte 6 éléments. Chacun des éléments sont séparés par des virgules. Le tout est encadré par des crochets.

```
>>> L = [1,2,3,4,5,6]
>>>print(L)
[1,2,3,4,5,6]
```

▪

▪ **Exemple .3.**
L = [] permet de définir une liste vide dans python. Cette liste comporte 0 élément. ▪

R

1. Une liste étant une variable on peut mettre des listes dans des listes. C'est très utile lorsqu'on cherche à faire des tableaux.
2. Une liste est du type 'list'.

2. Ce qu'il y a dans les listes.

2..1 Le premier indice

R Les listes sont indexés. C'est à dire que chaque élément de la liste a un numéro au sein de la liste. Cette indexation nous permet d'avoir accès aux différents éléments de la liste. Le premier élément a pour indice 0.

▪ **Exemple .4.**
Le premier élément de la liste L est donné par L[0]. Le 4e par L[3].

```
>>> L = [1,2,3,4,5,6]
>>> print(L[0],L[3])
1,4
```

▪

2..2 Accéder à des éléments d'une liste

▪ **Exercice .2.**
Li = [[1,"c",3],,7,8,11]

1. Que retourne Li[0] ?
2. Que devez vous taper dans la console pour obtenir l'élément 8 ?
3. Que devez vous taper dans la console pour obtenir l'élément "c" ?

▪

R L[−1] retourne le dernier élément de la liste L. Tester cette commande. ue retourne L[−2] ?

2..3 Modifier une liste

Une liste est un objet mutable. On peut donc modifier les éléments d'une liste. Nous pouvons modifier le 3e élément en faisant :

```
>>> L[2] = "Jai changÃ© !!!"
>>> print(L)
[1,2,"j'ai changÃ© !!!",4,5,6]
```

R Le premier élément a toujours l'indice 0.

2..4 Longueur de liste

Il est possible de connaître le nombre d'éléments d'une liste. Pour cela on utilise la fonction : *len*. len(L) retourne le nombre de variable que contient la liste L.

```
>>> L = [1,2,3,4]
>>> print(len(L))
4
```

3. Manipulation de listes

3..1 Concaténation

Il est possible de concaténer deux listes (*i.e.* les coller l'une après l'autre). Notons L et V deux listes. On peut créer une nouvelle liste W en concaténant les listes L et V. Pour concaténer, on utilise le "+" : W = L + V.

```
>>> L = [1,2,3]
>>> V = [4,6,8]
>>> W = L + V
>>> print(W)
[1,2,3,4,6,8]
```

3..2 Ajouter un élément

Il existe différentes méthodes pour ajouter un élément à une liste.

1. La première méthode (la plus commune) est $append(\cdot)$. Cela permet d'ajouter un élément à droite (à la fin) de la liste. Continuons avec la même liste L = [1,2,3,4,5,6]. Ici L.$append(7)$ va ajouter un 7 à la fin de la liste.

```
>>> L = [1,2,3,4,5,6]
>>> L.append(7)
>>> print(L)
[1,2,3,4,5,6,7]
```

2. La seconde méthode est de concaténer avec une liste à un élément.
3. La méthode $insert(i,\cdot)$ ajoute un élément $\cdot$ à une liste existante à la position i. Tous les éléments d'indice supérieurs à i sont décalés vers la droite.

```
>>> L = [1,2,4]
>>> L.insert(1,4)
>>> print(L)
[1,4,2,4]
```

3..3 Retirer un élément

On a ajouté un élément, modifié un élément, voyons maintenant comment retirer un élément d'une liste.

— **Supprimer une entrée avec un index**

```
>>> L = ["a","b","c"]
>>> del L[1]
>>> print(L)
["a","c"]
```

— On peut supprimer une enrtée de la liste par sa valeur. Mais attention, on ne supprime alors que l'entrée la plus à gauche de la liste avec cette valeur.

```
>>> L = [1,2,3,2,5,2]
>>> L.remove(2)
>>> print(L)
[1,3,2,5,2]
```

R Il nous reste encore la méthode $pop(\cdot)$ mais nous la verrons plus tard.

4. Programmer avec les listes

4..1 Parcourir une liste

Pour essayer de comprendre ce que signifie parcourir une liste, nous allons travailler sur un exemple : une liste de personnage "célèbre". Considérons la liste suivante :
L = ["Asterix","Obélix","Idéfix","Panoramix","James Bond","Harry Potter"]. Faisons afficher ces personnages un à un.

```
>>> L = ["Asterix","Obelix","Idefix","Panoramix","
   James Bond","Harry Potter"]
>>>for perso in L:
        print(perso)
Asterix
Obelix
Idefix
Panoramix
James Bond
Harry Potter
```

▪ **Exercice .3.**
Considérons une seconde liste :
B = ["la marmite de potion magique", "la marmite de jus de citrouille", "l'eau"]. Voici un petit programme qui montre que ces personnages ne sont pas toujours adroits :

```
>>> L = ["Asterix","Obelix","Idefix","Panoramix","James Bond","Harry Potter"]
>>> B = ["la marmite de potion magique", "la marmite de jus de citrouille", "l'eau"]
>>> from random import *
>>>for perso in L:
        print(perso,"est tombÃ© dans ", B[randint(0,2)])
Asterix est tombÃ© dans l'eau
Obelix est tombÃ© dans la marmite de jus de citrouille
Idefix est tombÃ© dans l'eau
Panoramix est tombÃ© dans la marmite de potion magique
James Bond est tombÃ© dans la marmite de jus de citrouille
Harry Potter est tombÃ© dans l'eau
```

Il manque un point à chacune de ces phrases, modifier le programme précédent pour avoir ce point. ■

5. Parcourir une liste avec *range*

Pour parcourir les éléments d'une liste, nous pouvons utiliser l'indexation de la liste. Voici un exemple.

■ **Exemple .5.**

```
>>> L = [1,2,3,2,5,2]
>>> for k in range(len(L)):
        print(L[k])
1
2
3
2
5
2
```

■

■ **Exercice .4.**
Déterminer le nombre d'éléments d'une liste à l'aide d'une boucle for. ■

6. Méthode *pop*

La méthode .$pop()$ supprime et renvoie le dernier élément d'une liste existante. LA méthode .$pop(\cdot)$ avec l'argument optionnel index $\cdot$ supprime et renvoie l'élément à la

position ·.

■ **Exercice .5.**

1. Créer une liste des jours de la semaine.
2. Par deux méthodes différentes retirer le dernier élément de la liste
3. Supprimer et renvoyer le premier élément de la liste.

■

7. Exercice

■ **Exercice .6.**
Écrivez un programme qui réalise les différentes instructions suivantes successivement :
— crée une liste contenant 7 entiers ;
— affiche le deuxième élément de la liste ;
— affecte la valeur 8 au troisième élément de la liste ;
— affecte la moyenne des deux premiers éléments de la liste au dernier élément de la liste ;
— affiche 10 fois de suite l'avant-dernier élément de la liste.

■

■ **Exercice .7.**
Réalisez une fonction Python qui prend une liste en entrée et renvoie en sortie la liste renversée (par exemple, pour l'entrée `[2,7,8,1]`, la fonction renvoie `[1,8,7,2]`). ■

■ **Exercice .8.**
Réalisez une fonction Python qui prend en entrée une liste de nombres et renvoie en sortie une liste contenant deux éléments : le plus petit et le plus grand élément de la liste. ■

■ **Exercice .9.**

1. Réalisez une fonction qui calcule la somme des éléments d'une liste de nombres à l'aide d'une boucle.
2. Réalisez une fonction qui calcule la moyenne des éléments d'une liste à partir de cette liste.

■

IV. Dictionnaire

1. Définition et création

Définition .4.
Un **dictionnaire** en Python est un objet contenant des paires de **clés-valeurs**.
- Les **clés** sont des éléments non modifiables et doivent être uniques. Elles ont pour types des objets immuables comme les entiers, les chaînes de caractères, les tuples.
- Les **valeurs** associées au clés sont elles modifiables et peuvent être de n'importe quel type.

Remarques :

- Un dictionnaire est non ordonné.
- Un dictionnaire est modifiable, on peut modifier ses valeurs, ajouter ou supprimer des éléments après sa création.

Exemples :

1. On crée un dictionnaire vide soit en utilisant la commande dict(), soit en utilisant des accolades {}.

```
dico1=dict() # on crÃ©e un dictionnaire vide.
dico2=\{\} # on crÃ©e un autre dictionnaire vide
```

2. Pour créer un dictionnaire directement avec des paires clés-valeurs, on utilise exclusivement les accolades. Une valeur est associée à une clé selon la syntaxe **clé : valeur** et les différente paires sont séparées par des virgules.

```
pokemon1={'nom':'bulbizarre','type':'plante'}
pokemon2={'nom':'herbizarre','type':'plante'}
pokedex={1:pokemon1,2:pokemon2} # dictionnaire de
   dictionnaires !
```

   On remarquera ici que les clés sont bien uniques au sein de leurs dictionnaires même si des dictionnaires différents les partagent.

▪ **Exercice .10.**
L'objectif de cet exercice et des suivants est de compléter notre podédex.
1. Créer un dictionnaire vide pokemon3.
2. Créer un dictionnaire pokemon4 pour Salamèche avec la clé 'nom' (on s'occupera du type plus tard).

▪

2. Ajout d'un élément à un dictionnaire

On peut **ajouter** un couple clé-valeur à un dictionnaire à condition que la clé soit bien unique. Pour cela, il suffit de faire **dictionnaire[clé]=valeur**.

Exemple :

On ajoute au dictionnaire pokemon3 le couple clé-valeur 'nom'-'florizarre' en faisant :

```
pokemon3['nom']='florizarre'
```

▪ **Exercice .11.**
Ajouter au dictionnaire pokemon3 le type de Florizarre puis le dictionnaire pokemon3 au podédex. ▪

3. Modification d'un élément d'un dictionnaire

On peut **modifier** la valeur associée à une clé mais pas cette. Pour cela, on utilise à nouveau la commande **dictionnaire[clé]=valeur**.

Exemple :

On a crée un dictionnaire pour Carapuce mais celui-ci n'a pas le bon type, on le modifie donc pour corriger cela.

```
pokemon7={'nom':'carapuce','type':'foudre'}
pokemon7['type']='eau'
```

▪ **Exercice .12.**
Modifier le dictionnaire suivant afin de corriger l'erreur sur le type de Reptincel.

```
pokemon5={'nom':'reptincel','type':'psy','exp.'
    :9999}
```

▪

4. Suppression d'un élément d'un dictionnaire

On peut **supprimer** un couple clé-valeur d'un dictionnaire en indiquant la clé du couple à supprimer grâce à la commande **dico1.pop(clé)**.

Exemple :

On a récupéré un dictionnaire pour Dracaufeu mais celui-ci comporte l'élément 'plat favori' dont on ne veut pas, on le supprime donc.

```
pokemon6={'nom':'dracaufeu','type':'feu','plat
    favori':'risotto'}
pokemon6.pop('plat favori')
```

▪ Exercice .13.
Supprimer l'élément 'exp.' du dictionnaire de Reptincel. ▪

5. Fusion de deux dictionnaires

On peut **fusionner** deux dictionnaires grâce à la commande **dico1.update(dico2)**.

Exemple :

On crée le dictionnaire typePlante et on le fusionne avec pokemon3.

```
1 typePlante={'type':'plante'}
2 pokemon3.update(typePlante)
```

▪ Exercice .14.
Créer un dictionnaire typeFeu sur le modèle de typePlante puis le fusionner avec le dictionnaire de Salamèche. ▪

6. Parcours d'un dictionnaire

On peut **parcourir** un dictionnaire selon ses clés, ses valeurs ou ses couples clé-valeur grâce aux commandes suivantes.

Parcours selon	Clés	Valeurs	Couples clé-valeur
Commande	dico.keys()	dico.values()	dico.items()

Exemples :

— L'algorithme suivant permet de parcourir le dictionnaire pokemon1 selon ses clés.

```
1 for cle in pokemon1.key():
2         print(cle)
```

Il renvoie donc

```
1 nom
2 type
```

— L'algorithme suivant permet de parcourir le dictionnaire pokemon1 selon ses valeurs.

```
1 for valeur in pokemon1.values():
2         print(valeur)
```

Il renvoie donc

```
bulbizarre
plante
```

— L'algorithme suivant permet de parcourir le dictionnaire pokemon1 selon ses couples clé-valeur.

```
for cle,valeur in pokemon1.items():
        print(cle,valeur)
```

Il renvoie donc

```
nom bulbizarre
type plante
```

■ **Exercice .15.**

1. Que va renvoyer le dictionnaire pokedex selon qu'il soit parcouru par ses : 3
   (a) clés ?
   (b) valeurs ?
   (c) couples ?
2. Programmer en Python le parcours du pokédex.

■