TP5

Partie 1 — Les Tuples

Ex. 1 — Les Tuples

En python, on peut créer des tuples (aussi appelées vecteurs), par exemple le tuple `(12,7)` est une paire, qui contiens 2 valeurs: les entiers `12` et `7`.

Question 1 (console) : Ouvrir la console python en bas à gauche, et y entrer les commandes suivantes:

```
>>> t = (12, 7, 26)
>>> print(t)
>>> type(t)
>>> t[0]
>>> t[1]
>>> t[2]
>>> t[10]
>>> t[3]
>>> print(t)
>>> len(t)
```

Explications: - Il existe un type `tuple`, composé de plusieurs 'cases', chacune contenant une valeur - On peut accéder à ces valeurs avec les crochets (`t[0]`, `t[1]` ...) - `t[i]` renvoie la valeur contenue dans la i-ème case du tuple `t`, mais attention, la numérotation des cases commence à `0` - Si on essaye d'accéder à un numéro de case trop grand, une Erreur est renvoyée - Pour un tuple de `n` éléments, les cases sont donc numérotées de `0` à `n-1`, et on ne peut donc pas accéder à `t[n]` - `len` est une fonction qui prends un tuple en argument et renvoie sa taille (length en anglais)

Question 2 (console) : On continue dans la console:

```
- t = (12, 2.1, "toto")
- print(l[0])
- print(l[1])
- print(l[2])
- print(l)
- t1 = (1, 2)
- t2 = (3, 4)
- print(t1 + t2)
```

Explications: - Les tuples peuvent contenir plusieurs éléments de type différents, içi un `int` (entier), un `float` (nombre réel) et un `str` (chaîne de caractères) - L'addition de deux tuples renvoie un tuple les fusionnant

Question 3 : Créer un tuple `my_tuple` à deux éléments, `0` et `100`

Question 4 : Afficher son deuxième élément en utilisant la fonction `print`

Question 5 : Afficher sa taille avec `len`

Ex. 2 — Fonctions

Question 1 : On souhaite écrire une fonction notes_extremes, qui demande 5 notes (entre 0 et 20) à l'utilisateur, et en extrait la note la plus basse et la note la plus haute. - Le mot-cef `return`, vu précédement, permet à une fonction de renvoyer un résultat, on souhaite içi l'utiliser pour en renvoyer deux - on va donc utiliser un tuple à deux éléments!

Question 2 : Observer l'exemple fourni. Le comportement est similaire, à celui de la question `1`, mais dans ces cas précis on peut ommetre les parenthèses formant le tuple: - `return a, b, c, ...` retourne le tuple `(a, b, c, ...)` - `x, y, ... = <tuple>` permet d'assigner directement aux varibles `x`, `y` … les valeurs de `<tuple>[0]`, `<tuple>[1]` …

Partie 2 — Les Listes

Ex. 3 — Les Listes

En python, on peut créer des listes (aussi appelées tableaux), par exemple la liste `[12,7,26]` contiens 3 valeurs: les entiers `12`, `7` et `26`.

Question 1 (console) : Ouvrir la console python en bas à gauche, et y entrer les commandes suivantes:

```
>>> l = [12, 7, 26]
>>> print(l)
>>> type(l)
>>> l[0]
>>> l[1]
>>> print(l)
>>> len(l)
```

Explications : - Il existe un type `list`, composé de plusieurs 'cases', chacune contenant une valeur - Comme pour les tuples, `l[i]` renvoie la valeur contenue dans la i-ème case de la liste `l`, et la numérotation des cases commence à `0` - On parle d'indice et pas de numéro de case, ainsi, la première case est d'indice `0`, la deuxième d'indice `1`, la dernière d'indice `n-1` - La fonction `len` fonctionne aussi sur les listes

Question 2 (console) : On continue dans la console:

```
>>> l = [12, 7, 26]
>>> print(l[1])
```

```
>>> l[1] = 77
>>> print(l)
```

Explications: - On peut modifier les valeurs contenues dans les cases d'une liste, en utilisant l'assignation (le `=`) - `l[a] = b` replace ainsi la valeur dans la case d'indice `a` par `b` - Cela ne fonctionne pas avec les tuples : les tuples ne sont pas modifiables une fois créés

Question 3 (console) : On continue dans la console:

```
>>> l = [12]
>>> l.append(1)
>>> print(l)
>>> l.append(42)
>>> print(l)
>>> print("taille de l:",len(l))
>>> l.pop()
>>> print(l)
>>> elem = l.pop()
>>> print(elem)
>>> print(l)
```

Explications : - La taille d'une liste peut être modifiée, ainsi, on peut lui ajouter et lui enlever des cases - append permet d'ajouter un élément en bout de liste - pour l'utiliser on fait nom_de_la_liste.append(element_a_inserer) - append n'est pas une fonction comme celles que vous avez vu jusqu'à présent, elle est appelée depuis une variable liste (c'est ce que le variable.append signifie) et agit sur celle-ci, on dit que append est une méthode des listes - Il existe d'autres méthodes, comme 'pop' - faire nom_de_la_liste.pop() à deux effet: il supprime la dernière case de la liste, et return la valeur contenue dans la case supprimée

Question 4 (console) : On continue dans la console:

```
>>> l = []
>>> print(l)
>>> l = [10, 20, 30, 40]
>>> l.insert(0,1)
>>> print(l)
>>> l.insert(2,25)
>>> print(l)
>>> l.pop(2)
>>> print(l)
>>> del l[0]
>>> print(l)
```

Explications : - On peut créer une liste vide, qui ne contiens aucune case avec `[]` - On peut insérer un élément en milieu de liste avec .insert, qui prends en arguments le numéro de case où insérer (son indice), et la valeur à insérer - Si 'l' est une variable liste de taille `n`, insérer une valeur `v` dans la case d'indice `0`

(en faisant `l.insert(0,v)`) ajoute la nouvelle case devant la liste - Insérer en indice `1` insère entre les cases d'indice `0` et `1`, et ainsi de suite - Insérer 'v' en case d'indice `n` a le même effet que faire `l.append(v)` - On a vu précédemment que `l.pop()` supprime la dernière case de la liste et en renvoie la valeur - On peut aussi supprimer des case en milieu de liste, en précisant quel indice de case supprimer (`l.pop(2)` supprime la case d'indice `2`, donc la troisième case puisqu'on commence à `0`) - `del l[i]` permet aussi de supprimer la case d'indice `i`. Contrairement à `.pop`, l'élément supprimé n'est pas renvoyé.

Question 5 (console) On continue dans la console:

```
>>> l1 = [1, 2]
>>> l2 = [3, 4, 5]
>>> l3 = l1 + l2
>>> print(l1,l2,l3)
>>> l1.extend(l2)
>>> print(l1,l2)
```

Explications : - Il y a plusieurs manières de fusionner des listes - Utiliser l'addition permet de le faire en créant une nouvelle liste, sans modifier les listes que l'ont fusionne - La méthode `.extend` permet d'étendre une liste en y ajoutant une autre, cela reviens à itérer le `.append`

Question 6 : Créer une variable `ma_liste` contenant les quatre valeurs `11`, `7`, `2` et `5`

Question 7 : En utilisant `print`, afficher `ma_liste`

Question 8 : Afficher la valeur contenue dans la première case de `ma_liste`

Question 9 : Afficher la valeur contenue dans la deuxième case de `ma_liste` (d'indice `1`)

Question 10 : Afficher la valeur d'indice `3` dans `ma_liste`

Question 11 : Mettre la valeur 1 dans la première case

Question 12 : Ajouter une case en bout de liste, contenant la valeur `57`

Question 13 : Ajouter une case en début de liste, contenant la valeur `-1`

Question 14 : Supprimer la troisième case de la liste

Question 15 : Afficher `ma_liste`

Question 16 : Afficher le nombre de cases dans `ma_liste` (sa taille)

Ex. 4 — Parcours

On a vu, lors des TP précédents, que faire :

```
for i in range(10):
    <code>
```

Exécute `<code>` 10 fois, pour i allant de 0 à 9.

Si `l` est une liste de taille 10, ses cases sont indexées de 0 à 9.

On peut donc itérer sur les indices des cases d'une liste `l`, et le code:

```
for i in range(len(l)):
  print(l[i])
```

affiche le contenu de `l`, une valeur par ligne.

On dit qu'on utilise une boucle for pour parcourir la liste `l`

On peut aussi itérer directement sur les valeurs contenues dans les cases d'une liste `l`:

```
for v in l:
  print(v)
```

Essayez d'utiliser ces deux parcours en console.

Question 1 : Créer une variable `ma_liste` contenant une liste vide

Question 2 : Y ajouter les entiers de 1 à 30, avec une boucle `for` et `.append`

Question 3 : Compter combien d'entiers pairs sont dans `ma_liste`, et mettre le résultat dans une variable `nb_pairs`

Question 4 : Remplacer chaque entier pair dans `ma_liste` par la valeur 1

Question 5 : Afficher `ma_liste`

Ex. 5 — Moyenne

Les listes peuvent être utilisées comme argument de fonction:

```
def fonction(liste):
    <code>
```

Question 1 : Écrire une fonction `moyenne` qui prends en argument une liste (non vide) de nombres, et en `return` la moyenne.

Question 2 : Appeler `moyenne` sur la liste `[12,18,2.5,10]` et en afficher le résultat.

Ex. 6 — Sous-listes

Question 1 (console) : On continue dans la console:

```
>>> l = [12, 7, 26, 2, 56]
>>> print(l[0])
>>> print(l[0:2])
>>> print(l[1:4])
>>> print(l[1:2])
>>> print(l[2:2])
```

Explications : - A partir d'une liste `l`, on peut extraire une sous-liste, en utilisant `l[i:j]` - `l[i:j]` contiens les éléments de `l`, entre les indices `i` et `j-1` - Ainsi, `l[i:i+1]` est équivalent à `l[i]`, et `l[i:i]` est vide - La sous-liste extraite est indépendante

Question 2 (console) : On continue dans la console:

```
>>> l = list(range(22,27))
>>> print(l[:3])
>>> print(l[3:])
```

Explications : - `list(range(...))` permet de créer facilement une liste d'entiers consécutifs - `l[:j]` contiens les éléments de `l`, entre les indices `0` et `j-1` - `l[i:]` contiens les éléments de `l`, entre les indices `i` et la fin de la liste

Question 3 (console) : On continue dans la console:

```
>>> x = list(range(5))
>>> y = x
>>> print(x,y)
>>> y.append(5)
>>> print(x,y)
>>> y = x.copy()
>>> print(x,y)
>>> y.append(5)
>>> print(x,y)
>>> y = x[:2]
>>> y[0] = 10
>>> print(x, y)
```

Explications : - Lorsque l'on fait `y = x`, la variable `y` designe la même liste que la variable `x`. - Si l'une des deux est modifiée, les deux se retrouvent changées. - Si on souhaite éviter ce comportement, on utilise `.copy()` qui met en `y` une copie (indépendante) de la liste qui se trouve dans `x` - Modifier une sous-liste n'affecte pas sa liste parente, une sous-liste est donc une copie (partielle).

Question 4 : Créer une liste `x` contenant les entiers de `0` à `9`

Question 5 :En extraire une sous-liste `y` contenant sa deuxième moitié (les entiers de `5` à `9`) en utilisant les sous-listes

Question 6 : Afficher `x` et `y`

Ex. 7 — Copies de listes

Question 1 : Ecrire une fonction `ma_fonction`, qui prend en argument une liste d'entiers positifs, et a le comportement suivant: - On trouve les indices correspondants a l'élément maximal de la liste (la plus grande valeur, qui peut apparaitre plusieurs fois) - On remplace par `0` le contenu de toutes les cases correspondantes - On affiche la liste obtenue sous la forme: `resultat <liste>`

Question 2 : Suivre les instructions suivantes

```
- Créer une liste `ma_liste`, contenant `[1, 4, 56, 3, 45, 56, -2, 7]`
- Aficher `ma_liste`
- Appeler `ma_fonction` avec comme argument `ma_liste`
- Aficher `ma_liste`
- réaffecter `ma_liste` à `[1, 3, 2]`
- Appeler de nouveau `ma_fonction`, mais sur `ma_liste.copy()` cette fois-ci
- Aficher `ma_liste`
```

Remarque: - On constate que si une liste est modifiée dans une fonction, elle est également modifiée en dehors de la fonction. - La méthode `.copy()` permet d'éviter ce comportement (si l'on souhaite que la liste passée en argument ne soit pas modifiée)

Partie 3 — Les Dictionnaires

Ex. 8 — Les Dictionnaires

En python, on peut aussi créer des dictionnaires (aussi appelées tableaux associatif).

Question 1 (console) : Ouvrir la console python en bas à gauche, et y entrer les commandes suivantes:

```
>>> dico = {"Alice":1.75, "Bob":1.8, "Charlie":1.72}
>>> print(dico)
>>> type(dico)
>>> dico["Alice"]
>>> dico["Bob"]
>>> dico["David"]
>>> len(dico)
```

Explications : - Il existe un type `dict`, qui associe des eléments à d'autres. On parle de clefs et de valeurs. - Par exemple, le dicitionnaire `dico` associe à la clef `"Alice"` la valeur `1.72` - On peut accéder à la valeur d'une clef avec les crochets `dico[...]` - Si on essaye d'accéder à une clef inconnue, une Erreur est renvoyée - `len` fonctionne aussi sur les dictionnaires, et compte le nombre de paires (clef:valeur) contenues

Question 2 (console) : On continue dans la console:

```
- dico = {}
- dico["Alice"] = "Boulanger"
- dico["Bob"] = "Instituteur"
- dico["Charlie"] = "Charpentier"
- print(dico)
- dico["Alice"] = "Comptable"
- del dico["Charlie"]
- dico.pop("Bob")
- print(dico)
```

Explications : - On peut ajouter des clefs et des valeurs en utilisant `dico[clef] = valeur` - Si `clef` est inconnue, une case l'associant à `valeur` est ajoutée au dictionnaire - Si `clef` est déjà une clef de `dico`, l'ancienne valeur est remplacée par la nouvelle - Comme pour les, liste, on peut supprimer des éléments avec `del` - Les dictionnaires possèdent une méthode `.pop(clef)` permettant d'en supprimer une clef et sa valeur. Comme pour les listes, la valeur associée à l'élément supprimé est renvoyé en résultat.

Question 3 : Créer un dictionnaire `my_dict` initialement vide.

Question 4 : Associer à la clef `"Alice"` la valeur `"06 23 45 67 89"`

Question 5 : Associer à la clef `"Bob"` la valeur `"06 98 76 54 32"`

Question 6 : Changer la valeur d'`"Alice"` à `"07 22 44 66 88"`

Question 7 : Afficher `my_dict`

Question 8 : Afficher sa taille

Ex.9 — Parcours

Il y à trois manières d'itérer sur les éléments d'un dictionnaire avec une boulce `for`.

- On peut itérer sur les clefs :

  ```
  for clef in dico.keys():
    print(clef)
  ```

- On peut itérer sur les valeurs associées au clefs :

  ```
  for valeur in dico.values():
    print(valeur)
  ```

- Et enfin on peut itérer sur les deux en même temps :

  ```
  for clef, valeur in dico.items():
    print(clef,":",valeur)
  ```

Essayez d'utiliser ces parcours en console.

Question 1 : Créer une variable `parite` contenant un dictionnaire vide.

On souhaite y stocker des paires clef:valeur sous la forme suivante: Les clefs seront des entiers, les valeurs seront des chaînes de caractères en indiquant la parité (`"pair"` ou `"impair"`).

Question 2 : Y ajouter les clefs de 10 à 20, avec leur valeurs associées.

Question 3 : Afficher le contenu de `parite`, en affichant en console une ligne `"L'entier <i> est <pair/impair>"` par entrée dans le dictionnaire.