Chapitre 3 : Types construits en Python

I. Exercices rapides

❐ — Exercice 0 —

- Ecrire une fonction **`poidsTotal`** qui prend en paramètres deux entiers: un nombre de paquets n et le poids d'un paquet poids.
  Si le poids total est strictement inférieur à 105kg, la fonction doit alors renvoyer **`True`**
  Sinon elle doit renvoyer **`False.`**

- Ajoutez le programme principal qui correspond à cette situation :
  On veux faire un premier envoi de 10 paquets de 13kg chacun, puis un second envoi de 12 paquets de 5kg chacun.

  Pour chaque envoi, le programme doit appeler la fonction **`poidsTotal`** et :
  - si elle renvoie True, il affiche : "envoi OK"
  - sinon il affiche : "envoi impossible"

❐ — Exercice 1 —


```python
def fonction1(liste):
    a = -1000
    for i in liste:
        if i > a:
            a = i
    return a

tab = [1, 3, 2, 5, 1, 3, 7, 3, 4]
print(fonction1(tab))
```

En utilisant un compteur, renvoyez la position du maximum en plus de la valeur maximum.

Dans l'exemple ci-contre, `fonction1(tab)` renvoie (7, 6) car 7 est le maximum et 6 est la position de ce maximum.

❐ — Exercice 2 —

Créez une fonction **minimum** qui prend un tableau de nombres en entrée et renvoie le minimum de ce tableau.

— Exercice 3 —

10'

Pensez à l'instruction len(…….) qui vous donne la taille d'une séquence.

- Écrire une fonction **estVide** qui prend une chaine de caractères en paramètre et renvoie **True si il n'y a aucune lettre dans la chaine de caractères** et **False** sinon.
- Écrire une fonction **longueurEgale** qui prend deux chaines de caractères en paramètres et renvoie True si les chaines de caractères sont de même longueur.

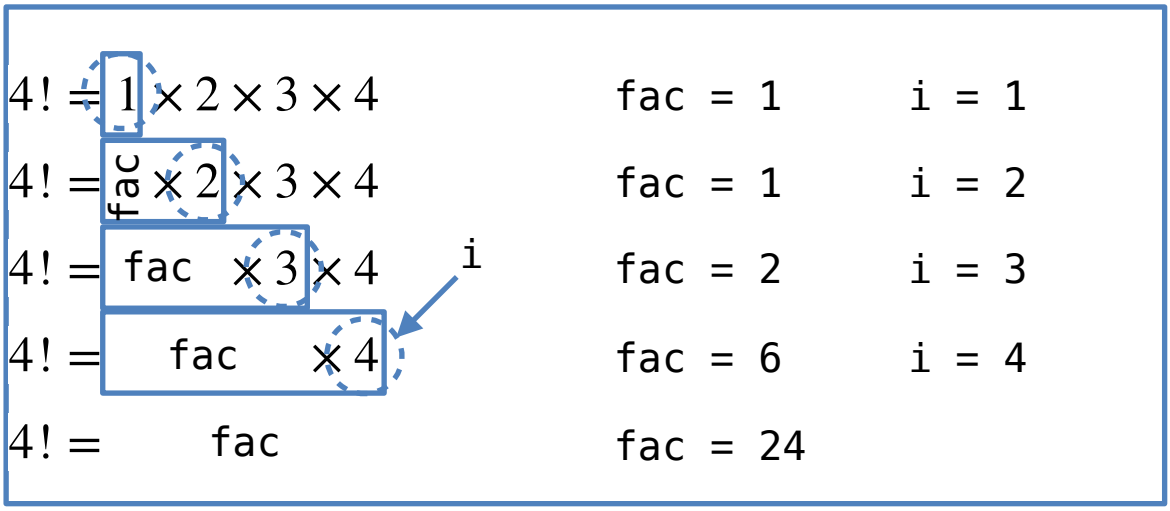
— Exercice 4 —

15'

- Écrire une fonction **factorielle** qui prend un nombre **n** en paramètre et renvoie la valeur de la factorielle de **n.** On donne : $n! = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$ et $0! = 1$.

  Exemple : $4! = 1 \times 2 \times 3 \times 4 = 24$

  Algorithme : Pour calculer $4!$, on va utiliser le schéma suivant (se servir de ce schéma pour code la fonction) :



- On définit la fonction mathématique suivante : $f(n) = 2^{n+1} \times \dfrac{(n!)^2}{(2n+1)!}$.

  En utilisant votre fonction **factorielle**, écrire une fonction Python nommée **terme** qui prend un entier **n** pour paramètre et calcule $f(n)$. Pour rappel, la puissance se note avec **.
- Écrire une fonction **piApprox** qui prend un nombre **n** en paramètre et renvoie la valeur approchée de π selon la formule : $\pi \approx \left(f(0) + f(1) + f(2) + \ldots + f(n)\right)$ avec f la fonction définie précédemment.

II. Tuples : séquences immuables

Pour l'instant, lors de la création et de l'exécution d'un programme, nous n'avons pas sauvegardé nos données. De plus en plus souvent, il va être important de pouvoir sauvegarder des "ensembles" de choses : il va donc falloir utiliser des tableaux.

1) Introduction

Un tuple (ou p-uplet) est une structure de données qui s'initialise ainsi :

monTuple = (5, 8, 6, 9)

La variable **monTuple** référence un tuple, qui est constitué des entiers 5, 8, 6 et 9. Remarquez les parenthèses autour des entiers de **monTuple.**

C'est une séquence donc chaque élément du tuple, en plus de sa valeur, a un indice :
- le premier élément du tuple (l'entier 5) possède l'indice 0 ;
- le deuxième élément du tuple (l'entier 8) possède l'indice 1
- le troisième élément du tuple (l'entier 6) possède l'indice 2
- le quatrième élément du tuple (l'entier 9) possède l'indice 3

On accède à un élément d'indice i grâce à l'instruction : **monTuple[i]**

Exemple : monTuple[0] me permet d'accéder au 0-ème élément donc 5.

❐ — Exercice **5** —

Dans EduPython, écrivez le code suivant :

```
monTuple = (12, -8, 3, 9)
a = monTuple[2]
print(a)
```

Sans tester le code, demandez vous à quelle valeur de la variable vous vous attendez ?
Testez le code : quelle valeur a été affichée ? Pourquoi ?

Attention : dans les séquences, les indices commencent toujours à 0 (le premier élément de la séquence a pour indice 0).

❐ — Exercice **6** —

❖ Complétez le code ci-dessous (en remplaçant les ….) afin qu'après l'exécution de ce programme la variable b référence l'entier 12.

```
monTuple = (5, 8, 6, 9, 12, 22)
b = monTuple[……..]
print(b)
```

Finalement, on peut ajouter des données dans un tuple grâce à l'opérateur + et à des parenthèses bien placées.

❖ Ajoutez **monPetitTuple = (34,)** au précédent tuple, stockez le résultat dans une variable **monGrosTuple** et affichez **monGrosTuple**.

2) Propriétés

Un tuple ne contient pas forcément des nombres entiers, il peut aussi contenir des nombres décimaux, des chaînes de caractères, des booléens ou tout à la fois !

❐ — Exercice **7** —

monTuple = ("bonjour", "le", "monde")
print(monTuple[2] + " " + monTuple[1] + " " + monTuple[0] + "!")

Quel est le résultat attendu après l'exécution de ce programme ?

_____

Vérifiez votre hypothèse en testant ce programme.

Contrairement à une chaine de caractères, un tuple est une séquence qui a une restriction importante :

❐ — Exercice **8** —

monTuple = (2, 4, 8)
monTuple[0] = 16
print(monTuple)

Que cherche-t-on à faire dans ce code ? _____

Quel erreur s'affiche ? Que se passe-t-il ?

_____

On peut donc sauvegarder des données dans un tuple mais on ne peut pas les modifier.

III. Tableaux : séquences mutables

C'est quand même dommage de pouvoir sauvegarder beaucoup de données mais de ne pas pouvoir modifier cette sauvegarde. Les tableaux Python permettent de pallier à ce défaut.

1) Création d'un tableau

Commençons par les bases : créer un tableau. Il y a plusieurs façons d'en créer :

En rentrant les données à la main : Par exemple :

- `mesNombres = [ 1, 4, 9, 3, 1, 2 ]`
- `mesCourses = [ "stylos rouges" , "piles" , "souris pour la salle info" , "claviers" ]`
- `mesCoordonnées = [ (1,1) , (0,1) , (1,6) ]`
- `monBordel = ["un texte", 18, 5.4, True, (1,0)]`

— Exercice **9** —

Dans un fichier appelé chapitre3.py, créez un tableau `maTrousse` indiquant le contenu de votre trousse. Il contiendra des chaines de caractères comme "stylo".
Affichez ensuite `maTrousse[1]` **.**

Le dernier exemple est celui de la liste vide, point de départ pour la création de tableau :

```
monTableauVide = [ ]
```

Rem : on peut aussi créer des tableaux par compréhension où l'on mêle boucle et tableau. Nous en reparlerons dans un prochain chapitre.

2) Ajout et suppressions d'éléments dans un tableau

Les opérations d'ajout ou de suppression d'un élément dans un tableau sont les plus courantes. Il y a deux méthodes qui modifient directement le tableau :

- ajout à la fin du tableau avec l'instruction **`tableau.append(nouvelElt)`**
- suppression du i-ème élément avec l'instruction **`tableau.pop(i)`**.

❐ — Exercice **10** —

Ajouter 42, puis 72 à la liste mesNombres :

```
mesNombres = [1, 3, 5, 7 ]
mesNombres.append(42)
print(mesNombres)
……
print(mesNombres)
```

En quelle position le nombre 72 a-t-il été ajouté ?

À la suite de ce programme, faites **`mesNombres.pop(0)`** puis affichez mesNombres. Que s'est-il passé ?

❐ — Exercice **11** —

Complétez le programme chapitre3.py afin de rajouter des objets dans votre trousse. On utilisera une fonction appelée **`ajouteObjet`** permettant de rajouter des objets, sous forme de chaines de caractères, dans la trousse.

Le programme demandera à l'utilisateur si celui-ci a fini de remplir sa trousse. Tant qu'il n'a pas fini, le programme appellera la fonction et ajoutera à la liste `maTrousse` créée à l'exercice 10 un objet. Pensez à l'instruction **`input`**.

Créez une fonction **jetteContenuTrousse** qui va sortir un à un tous les objets de ma trousse, les afficher et les faire disparaitre du tableau `maTrousse`**.**

3) Opérations sur les listes

Il existe énormément d'opérations sur les listes. Vous les découvrirez au fur et à mesure de vos travaux et projets.

Quelques opérations utiles :

a. `maListe[n]` : permet de récupérer la valeur de l'élément à l'indice n (comme pour un tuple)

❐ — Exercice **13** —

Entrez le code suivant :

```
mesCourses = [ "stylos" , "piles" , "souris" , "claviers" ]
print(mesCourses[1])
print(mesCourses[-1])
```

Que donne ces affichages ? En déduire ce que fait **mesCourses[-1] .**

b. `maListe[a:b]` : récupère la liste des éléments d'indice compris entre **a** et **b-1**

❐ — Exercice **14** —

Entrez le code suivant :

```
mesCourses = [ "stylos" , "piles" , "souris" , "claviers" ]
print(mesCourses[1:3])
print(mesCourses[:3])
print(mesCourses[2:])
```

Que donne ces affichages ? Est-ce attendu ?

c. `maListe[n] = valeur` : permet de modifier la valeur de l'élément d'indice n.
d. `len(liste)` : Donne la longueur de la liste (le nombre d'éléments).
e. `element in maListe` : Renvoie True si `element` est dans la `maListe` et False sinon. Très utile pour faire des tests d'appartenance.

❐ — Exercice **15** —

Entrez le code suivant :

```
mesCourses = [ "stylos" , "piles" , "souris" , "claviers" ]
print("stylos" in mesCourses)
print("stylo" in mesCourses)
```

Que pensez-vous de la réponse dans la deuxième chaine de caractères ? Est-ce normal ?

Rem : Par rapport aux boucles while, if et aux conditionnelles, les listes fonctionnent de la même manière que les chaines de caractères.

Par exemple, si on veut afficher le carré de nombres contenu dans une liste, on peut faire :

```
mesNombres = [1, 3, 7]

for nombre in mesNombres :
    print(nombre**2)
```

```
mesNombres = [1, 3, 7]
longueur = len(mesNombres)
for i in range(longueur) :
    nombre = mesNombres[i]
    print(nombre**2)
```

Vérifions votre compréhension sur les exercices de programmation suivant :

❐ — Exercice **1** —  10'

Écrire une fonction **occurences(v, t)** qui renvoie le nombre d'occurrences de la valeur v dans le tableau t.

Par exemple, pour un tableau tab = [1, 2, 3, 2, 2, 5] , occurrences(2, tab) renvoie 3.

❐ — Exercice **2** —  8'

Écrire une fonction aleatoire(n) qui construit un tableau de n entiers tirés au hasard entre 1 et 1000, puis l'affiche.

Pour tirer des nombres au hasard, on importera la module random :

```
from random import randint
randint(1, 6) #donne un nombre aléatoire entre 1 et 6
```

❐ — Exercice **3** —  10'

Écrire une fonction **fibonacci(n)** qui renvoie un tableau contenant les n premiers termes de la suite de Fibonacci. Cette suite est construite de telle manière à ce qu'un terme soit égal à la somme des deux termes précédents. Les deux premiers termes sont 0 et 1.
Ainsi les premiers termes seront :
0
1
0+1 = 1
1+1 = 2
2+1 = 3
3+2 = 5
5+3 = 8
8+5 = 13

Quelle est la valeur du 30ème terme de la suite ?