Première NSI
Chapitre VII - Les p-uplets

I. Les p-uplets

I.1. Généralités

Définition

Un p-uplet (ou tuple) est une collection ordonnée immuable (non mutable) d'éléments.

En Python

On écrit les éléments d'un p-uplet entre parenthèses séparés par une virgule.
Un p-uplet est de type `tuple`.

Exemple

```
1  >>> a=(2,5,6)
2  >>> type(a)
3  <class 'tuple'>
4  >>> a[2]
5  6
6  >>> a[1]=7
7  Traceback (most recent call last):
8    ...
9  TypeError: 'tuple' object does not support item assignment
```

Les lignes 4 et 5 nous indiquent qu'un p-uplet est une collection ordonnée.
Les lignes 6 et suivantes nous indiquent qu'un p-uplet est immuable.

Attention

Si on veut créer un p-iplet à un élément, en Python on ajoutera une virgule après l'élément du tuple dans les parenthèses.

```
1  >>> a = (4)
2  >>> a
3  4
4  >>> type(a)
5  <class 'int'>
6  >>> b = (4,)
7  >>> b
8  (4,)
9  >>> type(b)
10 <class 'tuple'>
```

Remarque

Comme pour les tableaux, les indices peuvent être négatifs :

```
1  >>> a=(2,5,6)
2  >>> a[-1]
3  6
```

I.2. Opérations sur les p-uplets

Définition

Concaténer deux p-uplets, c'est les mettre bout à bout.
On utilise le symbole `+` pour concaténer deux p-uplets.

Remarque

L'ordre des opérandes autour du symbole `+` est important pour concaténer deux p-uplets car un p-uplet est ordonné.

Exemple

```
>>> (1, 2, 3) + (4, 5)
 (1, 2, 3, 4, 5)
>>> (4, 5) + (1, 2, 3)
 (4, 5, 1, 2, 3)
```

I.3. Construction de p-uplets en compréhension

Propriété

On peut créer des p-uplets en compréhension, c'est-à-dire en utilisant une formule qui permet de calculer chaque éléments, une boucle `for` et éventuellement des instructions conditionnelles.

Exemple

```
>>> tup = tuple(i for i in range(5))
>>> tup
 (0, 1, 2, 3, 4)
>>> tup2 = tuple(3 * chut + 1 for chut in range(3))
>>> tup2
 (1, 4, 7)
>>> tup3 = tuple(5 - i for i in range(5) if i % 2 ==0)
>>> tup3
 (5, 3, 1)
```

I.4. Méthodes et fonctions avec des p-uplets

I.4.a. Méthodes

Remarque

Les p-uplets n'ont pas de méthode.

I.4.b. La fonction len()

Définition

La fonction `len(puplet)` renvoie la **longueur** du p-uplet mis en paramètre, c'est à dire le nombre d'éléments qui le composent.

Exemple

```
>>> tup = (1, 2, 3, 8)
>>> len(tup)
 4
>>> tup3 = ([1, 2, 3], [4, 5, 6], [7, 8, 9])
>>> len(tup3)
 3
>>> tup4 = ()
>>> len(tup4)
 0
```

I.4.c. La fonction tuple()

Définition

La fonction `tuple(collection)` renvoie la collection mise en paramètre sous forme d'un p-uplet.

Exemple

```
>>> chaine = "abcd"
>>> tuple(chaine)
 ('a', 'b', 'c', 'd')
>>> tab = [1, 2, 3, 4]
>>> tuple(tab)
 (1, 2, 3, 4)
```

II. Algorithmes gloutons

Définition

Un algorithme glouton (greedy algorithm en anglais) est un algorithme qui, étape par étape, fait un choix optimum local, en espérant obtenir un résultat optimum global.

Remarque

Il existe deux types d'algorithmes gloutons :

- ceux qui donnent une approximation d'un résultat optimal.
- ceux qui donnent un résultat exact optimal.

Voici la structure d'un algorithme glouton :

Algorithme 1 : algorithme glouton

Entrées : Dépend du problème
candidats ←ensemble des candidats à la solution
soluce ←∅
tant que *soluce n'est pas une solution au problème* **et** *que candidats* $\neq \emptyset$ **faire**
  x ←choix d'un candidat le plus prometteur parmi les candidats
  On enlève x de la liste des candidats
  si *x permet de se rapprocher d'une solution* **alors**
    on ajoute x à soluce
  fin
fin
Résultat : soluce

Remarque

- Les candidats sont souvent ordonnés
- Dans certains cas, le schéma peut être simplifié, dans d'autres cas, il peut être plus complexe.

III. Exercices

III.1. Exercice 1

1. Créer une fonction `nb_impairs(quantite: int) -> tuple` qui renvoie un p-uplet des `quantite` premiers nombres entiers impairs positifs.
2. Créer la fonction `somme(puplet: tuple) -> int` qui renvoie la somme des éléments de `puplet`.
3. Déterminer la somme des
   - 50 premiers nombres entiers impairs positifs.
   - 100 premiers nombres entiers impairs positifs.
   - 1000 premiers nombres entiers impairs positifs.
4. Conjecturer la formule de la somme des n premiers nombres entiers positifs impairs.
5. Démontrer la conjecture.
   On rappelle que la somme des p premiers nombres entiers positifs est donné par la formule $\frac{p(p+1)}{2}$.

III.2. Exercice 2

1. On possède une infinité de pièces de chaque type : 2€, 1€, 50 cents, 20 cents, 10 cents, 5 cents, 2 cents et 1 cent.
   On veut créer une fonction `rendre_monnaie(argent: int) -> tuple` qui renvoie le nombre de pièces de chaque type sera rendu lorsqu'on doit rendre `argent` en cents.

   (a) Proposer un algorithme glouton correspondant au problème.

   (b) Implémenter la fonction `rendre_monnaie(argent: int) -> tuple`.

   (c) S'agit-il d'un algorithme glouton exact ?

2. Que se passe-t-il si on ne possède que des pièces de 2€, 1€, 50 cents, 20 cents, 10 cents, 5 cents et 2 cents.

III.3. Exercice 3

On possède un sac à dos pouvant contenir 15 kg. Devant nous, il y a cinq boîtes ayant une masse et une valeur. Le but de ce problème est de savoir quelles boîtes il faut mettre dans le sac à dos pour obtenir la plus grande valeur totale.
Les boîtes disponibles sont :

Nom	Valeur	Masse
A	10 €	9 kg
B	7€	12 kg
C	3€	7 kg
D	2€	5 kg
E	1€	2kg

1. Créer un tableau `boites_dispo` dont les valeurs sont les boites représentées implémentées par des p-uplets du type `(nom, valeur, masse)`.

2. Créer une fonction `tri_boites(boites)` qui renvoie une liste composée de toutes les boîtes triées par ordre d'intérêt. *On expliquera comment définir l'intérêt d'une boîte.*

3. Créer une fonction `sac_a_dos(boites, masse_totale)` qui renvoie une liste des boites contenues dans le sac à dos afin d'obtenir la valeur maximale. *On utilisera un algorithme glouton.*

4. Répondre au problème grâce à votre implémentation.

5. S'agit-il d'un algorithme glouton exact ?