2 – p-uplet et p-uplet nommé -> (tuples en anglais)

Définition:

- Un objet de type tuple, un p-uplet, est une suite ordonnée d'éléments.
- On parlera indifféremment de p-uplet ou de tuple.
- Chaque éléments est appelés composantes ou encore termes. Chaque terme peut être de nimporte quel type.
- Un p-uplet contenant 2 termes sera appelé un "doublet", 3 termes un "triplet", etc...

Création d'un p-uplet:

- En Python, les termes d'un p-uplet s'écrivent séparés par une virgule, et entre parenthèses (elles ne sont pas obligatoire), mais son grandement conseillé pour faciliter la lisibilité de votre code).
  - t = "a", "b", "c", 3 pour un tuple à 4 éléments

- Un p-uplet ne contenant qu'un seul terme s'écrira aussi avec une virgule, cela permet d'éviter la confusion avec une expression mathématique, qui elle, ne possède pas de virgule.
  - t = "a", pour un tuple à 1 éléments (attention à la virgule)

- Il est également possible de créer un p-uplet vide.
  - t = () pour un tuple à 0 éléments (ici, pas de virgule, mais des parenthèses)

Pour écrire un p-uplet qui contient un n-uplet, l'utilisation de parenthèses est nécessaire. Voici un exemple avec un tuple à 2 éléments dont le second est un tuple:

- t = 3, ("a", "b", "c").

En général, les parenthèses sont obligatoires dès que l'écriture d'un p-uplet est contenue dans une expression plus longue. Dans tous les cas, les parenthèses peuvent améliorer la lisibilité.

Opérations:

Nous avons deux opérateurs de concaténation qui s'utilisent comme avec les chaînes de caractères, ce sont les opérateurs + et *. De nouveaux p-uplets sont créés.

```
>>> t1 = "a", "b"
>>> t2 = "c", "d"
>>> t1 + t2
('a', 'b', 'c', 'd')
>>> 3 * t1
('a', 'b', 'a', 'b', 'a', 'b')
```

Appartenance:

Pour tester l'appartenance d'un élément à un tuple, on utilise l'opérateur in :

```
>>> t = "a", "b", "c"
>>> "a" in t
```

```
True
>>> "d" in t
False
```

Utilisation des indices:

Les indices permettent d'accéder aux différents éléments d'un tuple. Pour accéder à un élément d'indice i d'un tuple t, la syntaxe est t[i]. L'indice i peut prendre les valeurs entières de 0 à n − 1 où n est la longueur du tuple. Cette longueur s'obtient avec la fonction len. Exemple :

```
>>> t = "a", 1, "b", 2, "c", 3
>>> len(t)
6
>>> t[2]
'b'
```

La notation est celle utilisée avec les suites en mathématiques : u0, u1, u2, . . . : les indices commencent à 0 et par exemple le troisième élément a pour indice 2. Le dernier élément d'un tuple t a pour indice len(t)-1. On accède ainsi au dernier élément avec t[len(t)-1] qui peut s'abréger en t[-1].

```
>>> t = "a", 1, "b", 2, "c", 3
>>> t[-1]
3
>>> t[-2]
'c'
```

Exemple avec des tuples emboîtés (un tuple contenant des tuples) :

```
>>> t = ("a", "b"), ("c", "d")
>>> t[1][0]
'c'
```

Explication : t[1] est le tuple ("c", "d") et 'c' est l'élément d'indice 0 de ce tuple.

Rappelons ce qui a été annoncé plus haut : les éléments d'un tuple ne sont pas modifiables par une affectation de la forme t[i]=

```
↪valeur qui provoque une erreur et arrête le programme.
```

Il n'y à pas de .append comme il y aurait pour une liste.

Affectation multiple:

Prenons pour exemple l'affectation a, b, c = 1, 2, 3. Ceci signifie que le tuple (a, b, c) prend pour valeur le tuple (1, 2, 3), autrement dit, les valeurs respectives des variables a, b et c sont 1, 2 et 3.

En particulier, l'instruction a, b = b, a permet d'échanger les valeurs des deux variables a et b.

Les valeurs des éléments d'un tuple peuvent ainsi être stockées dans des variables.

```
>>> t = 1, 2, 3
>>> a, b, c = t
>>> b
2
```

Cette syntaxe s'utilise souvent avec une fonction qui renvoie un tuple.

Voici un exemple avec une fonction qui calcule et renvoie les longueurs des trois côtés d'un triangle ABC. La fonction prend en paramètres trois p-uplets représentant les coordonnées des trois points. On importe au préalable la fonction racine carrée sqrt du module math.

```
from math import sqrt

def longueurs(A, B, C):
    xA, yA = A
    xB, yB = B
    xC, yC = C
    dAB = sqrt((xB - xA) ** 2 + (yB - yA) ** 2)
    dBC = sqrt((xC - xB) ** 2 + (yC - yB) ** 2)
    dAC = sqrt((xC - xA) ** 2 + (yC - yA) ** 2)
    return dAB, dBC, dAC
```

La fonction étant définie, nous l'utilisons dans l'interpréteur :

```
>>> M = (3.4, 7.8)
>>> N = (5, 1.6)
>>> P = (-3.8, 4.3)
>>> dMN, dNP, dMP = longueurs(M, N, P)
>>> dMN
6.4031242374328485
```

Construction en extension, en compréhension:

Les tuples peuvent être construit en extension ou en compréhension.

- En extension :
  - v1 = (1, "test", 3.4)
- En compréhension :
  - v2 = (expr for x in v1)
    - expr est une expression qui dépends généralement de x, et placé dans v2
    - x est l'élément courant qui parcours le tuple v1, il est appelé variable de contrôle du tuple en compréhension
    - v1 est un tuple
    - (), for et in sont obligatoire

v2 serait alors :

```
v2 = (2*x for x in v1)
```

Ceci dit, même si cette écriture est possible et n'est pas fausse, cette expression ne générera pas un tuple en compréhension, mais un generator expression (une expression utilisable dans une boucle for).

Nous pouvons toujours caster l'expression en liste (v2 = list(2*x for x in v1)) mais v2 sera de type list, pas de type tuple. Nous verrons que pour les listes, c'est un peu différent.

P-uplets nommé:

Un p-uplet nommé est un p-uplet, dont les composantes sont appelées via un descripteur au lieu d'un indice. Le principal intérêt de ce type est d'améliorer la lisibilité du code, et partant de réduire les risque d'erreurs. Le type des p-uplets nommés n'existe pas nativement dans Python. On pourrait utiliser le module collection.namedtuple mais le programme invite à utiliser des dictionnaires pour limiter le nombre de syntaxes différentes à utiliser.

Exo:

Se renseigner sur les différentes opérations disponible pour les tuples et créer un tableau (excel ou word) de la forme:

type opération | exemple concret | résultat exemple

créer un tuple | tuple_vide = () | print(tuple_vide) -> ()
créer un tuple | tuple_int = (1, 2) | print(tuple_int) -> (1, 2)
etc...

Créer par affectation le tuple t1, contenant les lettres f, a ,c, e

Créer par affectation le tuple t2, contenant les lettres g, r ,a, m

Ecrire l'instruction qui vérifie si b est dans t1

Ecrire l'instruction qui vérifie si r est dans t2

Que renvoi l'instruction t1 + t2 ?

Que renvoi l'instruction t1 * 2 ?

Que renvoi l'instruction t1[2] ?

Que renvoi l'instruction t1[-1] ?

Ecrire une fonction "multiplier" qui renvoi le double et le triple d'un entier passé en paramètre. Le résultat sera retourné sous forme de tuple (un doublet)

Ecrire une instruction pour stocker le résultat de la fonction "multiplier" dans 2 variables

Ecrire une instruction pour affecter à une variable, la somme des 2 variables créées ci-dessus

Que renvoi l'instruction t1[2] = "s" ?

Que pouvez-vous en conclure ?