Nom :	Prénom :	Classe :

NSI 1re — p-uplet (tuple)

Objectifs :
- Savoir définir un type "construit".
- Savoir définir un p-uplet et en connaître les principales caractéristiques.
- Savoir créer un (tuple) p-uplet en Python. Savoir accéder à un élément précis d'un p-uplet.
- Savoir parcourir un (tuple) p-uplet par valeur et par indice.

Type de base

L'appellation « **type de base** » comprend tous les types de données « élémentaires » croisés jusqu'à maintenant[1]. Ces types de données élémentaires servent à représenter des **valeurs simples** telles que des entiers, des flottants, des chaînes de caractère ou des booléens.

```
1  a = 123      # int (entier)
2  b = 12.3     # float (flottant)
3  c = 'chaîne' # str (chaîne de caractère)
4  d = True     # bool (booléen)
```

Limite d'un type de base

Comment faire pour stocker *plusieurs* valeurs, dans *une seule* variable ?

Prenons un exemple : nous souhaitons stocker les valeurs "vert", "jaune" et "rouge" dans une seule variable.

Première tentative « naïve » :

```
1  couleur_1 = 'vert'
2  couleur_2 = 'jaune'
3  couleur_3 = 'rouge'
```

Dans cette première tentative, nous ne respectons pas l'énoncé (utilisation d'une seule variable).

Et, d'autre part, comment ajouter d'autres couleurs ? Faudrait-il créer et gérer une multitude de nouvelles variables (couleur_4, couleur_5, etc.) dans notre programme ? Ce serait vite *ingérable*, compliqué.

Deuxième tentative « naïve » :

```
1  couleurs = 'vert, jaune, rouge'
```

Dans cette deuxième tentative, nous stockons bien toutes nos valeurs dans une seule variable. Mais comment accéder à l'une de ces valeurs ? Par exemple comment accéder *uniquement* à la deuxième couleur (jaune) ?

N'existerait-il pas *un autre type* permettant de gérer facilement plusieurs données ? *Réponse page suivante ...*

1 Voir le cours « Type de donnée - entier, flottant, booléen, chaîne de caractères ».

Type construit

Contrairement à un *type de base*, un **type construit** est un type de données qui permet de regrouper **plusieurs valeurs** dans **une seule entité**.

Ainsi, nous pourrions regrouper nos trois valeurs (vert, jaune et rouge) dans une seule entité.

Il existe principalement trois types de données appartenant au groupe « type construit » :

- Le type de donnée « p-uplet » (également appelé « n-uplet »).
- Le type de donnée « tableau indexé ».
- Le type de donnée « p-uplet nommé ».

p-uplet, tableau indexé, p-uplet nommé : chacun de ces trois types de données permet de regrouper plusieurs valeurs dans une seule entité.

Le présent cours se penche sur le type de donnée p-uplet.

p-uplet

Un ***p-uplet*** (également appelé « n-uplet ») est une **suite ordonnée d'éléments** dont les types peuvent éventuellement être différents. Par exemple, un p-uplet peut contenir à la fois des entiers, des chaînes et des booléens.

```
1  a = ()                    # p-uplet vide
2  b = ('Emma', 'Enzo')      # p-uplet contenant 2 éléments
3  c = ('Emma', )            # p-uplet contenant 1 élément (pensez à la virgule)
4  d = (14, 'Juillet', 1789) # p-uplet contenant 3 éléments
```

En Python, la syntaxe d'un p-uplet est la suivante :

```
( élément 1, élément 2, ..., élément n )
```

Un p-uplet est formé d'une **parenthèse ouvrante**, suivi d'un **groupe d'éléments séparés par des virgules**, et se termine par une **parenthèse fermante**.

Un p-uplet peut contenir autant d'éléments que désirés ❤️.

➤ **Exercice 1** — À vous de jouer !

Dans le tableau suivant, rayez les éléments qui ne sont pas des p-uplets :

`('mario', 'luigi')`	~~`'bleu, orange'`~~	`a = (1,)`	`(True, )`
`b = ()`	~~`(False and True)`~~	~~`'(1, 2, 3)'`~~	~~`('élément')`~~

➤ Exercice 2

Affectez à une variable `couleurs` le p-uplet constitué des 3 éléments suivants : vert, jaune et rouge.

```
couleurs = ( 'vert', 'jaune', 'rouge')
```

Caractéristiques d'un p-uplet

Un p-uplet est un type construit qui possède un certain nombre de caractéristiques, notamment :

- Chaque **élément** d'un p-uplet possède un **indice**.
- Les **éléments** d'un p-uplet peuvent être de **types différents**, on dit qu'il a une *structure hétérogène*.
- En Python, un p-uplet est également *immuable*, c'est à dire qu'il est impossible de modifier un des éléments contenu dans un p-uplet après création.

Un tuple ?

En Python, un p-uplet se nomme un `tuple`.

En voici la démonstration :

```
1  a = ('Emma', 'Enzo')
2
3  print( type(a) )       # Affiche <class 'tuple'>
```

Obtenir la longueur (length) d'un tuple

Comme pour une chaîne, il est possible d'obtenir la longueur (*length*, en anglais) d'un tuple, c'est à dire d'obtenir le nombre d'éléments qu'il contient, grâce à la commande `len()` :

```
1  a = ('Emma', 'Enzo')
2
3  print( len(a) )       # Affiche 2
```

➤ Exercice 3

Écrivez à droite de chaque code, la valeur de la variable `a` :

`a = len( (3,) )`	1	`a = len( ('333') )`	3 # ce n'est pas un p-uplet !
`a = len( ('3',) )`	1	`a = len((True, 1))`	2

Accès à un des éléments d'un p-uplet

Comme pour les chaînes, il est possible d'accéder à un des éléments d'un p-uplet.

Prenons par exemple le p-uplet constitué des trois éléments suivants : `('Soolking', 'Mozart', 'MHD')`

Chacun de ces éléments possède son propre **indice** (sa propre **position**) :

Élément (valeur)	`'Soolking'`	`'Mozart'`	`'MHD'`
Indice	`0`	`1`	`2`

Dès lors, il devient possible d'accéder à un élément précis, en utilisant son indice :

```
1  a = ('Soolking', 'Mozart', 'MHD')
2
3  print( a[0] )  # Affiche 'Soolking'
4  print( a[1] )  # Affiche 'Mozart'
5  print( a[2] )  # Affiche 'MHD'
6  print( a[-1] ) # Affiche 'MHD'
```

Notons qu'il est possible d'utiliser un indice négatif, exactement comme pour une chaîne[2].

➤ Exercice 4

Que vont afficher les codes Python suivants ?

N°	Python	Qu'est-ce qui sera affiché ? (à compléter)
1	`b = ('Emma', 'Enzo')` `print( b[1] )`	Enzo
2	`j = (3, -45, 10)` `print( j[2] )`	10
3	`k = ('mario', 'luigi')` `print( len(k) )`	2
4	`a = ('hi', 'salut', 'ciao')` `print( a[-1] )`	ciao

2 Voir le chapitre « Indice négatif » du cours « Manipulation de chaîne de caractères ».

Slices

Il est possible également d'effectuer des découpes (ou tranches ou *slices*, en anglais), selon le format suivant :

```
tuple[indice_début_inclus:indice_fin_exclus]
```

Exemples :

```
1  a = ('Jan', 'Fev', 'Mar', 'Avr')
2
3  print( a[0:1] )   # Affiche ('Jan',)
4  print( a[0:2] )   # Affiche ('Jan', 'Fev')
5  print( a[:2] )    # Affiche ('Jan', 'Fev')
6  print( a[1:2] )   # Affiche ('Fev', )
7  print( a[1:] )    # Affiche ('Fev', 'Mar', 'Avr')
8  print( a[:-1] )   # Affiche ('Jan', 'Fev', 'Mar')
```

➤ Exercice 5

À votre avis, que vont afficher les codes suivants ?

N°	Python	Qu'est-ce qui sera affiché ? (à compléter)
1	`print( ('Emma', 'Enzo')[1:] )`	('Enzo',)
2	`a = ('Mario Kart', 'FC26', 'Zelda')` `print( a[1] )`	FC26
3	`a = ('Mario Kart', 'FC26', 'Zelda')` `print( a[:-1] )`	('Mario Kart', 'FC26')
4	`b = (True, 1789, 'Bob')` `print( b[1:] )`	(1789, 'Bob')

Concaténation

La concaténation de p-uplets en Python utilise l'opérateur `+` pour créer un **nouveau** p-uplet contenant tous les éléments des p-uplets originaux, sans modifier ceux-ci.

Exemple :

```
a = ('Mario Kart',)
b = ('FC26',)

c = a + b

print(c)    # Affiche ('Mario Kart', 'FC26')
```

Lors de la **création d'un** `tuple` ne contenant **qu'un seul élément**, pensez bien à ajouter une virgule après ce premier élément, sinon Python considérera votre `tuple` comme un `str` ...

➤ Exercice 6

Écrivez un programme Python à partir de cette description :

- Affectez à une variable `film_1` un tuple contenant la chaîne `'Ne Zha 2'`.
- Affectez à une variable `film_2` un tuple contenant la chaîne `'Arco'`.
- Affectez à une variable `films` le tuple résultant de la concaténation de `film_1` et de `film_2`.
- Affichez la valeur de `films` dans la console.

À vous de coder :

```
film_1 = ( 'Ne Zha 2' , )
film_2 = ( 'Arco' , )
films = film_1 + film_2
print( films )
```

Parcours par valeur

Un parcours de tuple par valeur, c'est le fait de parcourir directement les éléments contenus dans un tuple, sans utiliser leurs indices.

Exemple en Python :

```
1  a = ('a', 'b', 'c', 'd')
2  for v in a:
3      print(v)
```

Dans cet exemple, la variable `v` stocke successivement chaque élément (chaque valeur) du tuple `a`.

➤ **Exercice 7** — Prédire

À votre avis, que va afficher le programme de l'exemple précédent ?

```
'a'
'b'
'c'
'd'
```

Parcours par indice

Un parcours de tuple par indice, c'est le fait de parcourir le tuple en utilisant les positions (indices) des éléments, et non les éléments eux-mêmes.

Exemple en Python :

```
1  a = ('a', 'b', 'c', 'd')
2  for i in range(len(a)):
3      print(i)
```

Dans cet exemple, la variable `i` stocke successivement chaque indice du tuple `a`.

➤ **Exercice 8** — Prédire

À votre avis, que va afficher le programme de l'exemple précédent ?

```
0
1
2
3
```

Ce qu'il faut retenir

Vous retrouverez ici l'essentiel du cours.
Astuce : transformez ces points en questions/réponses pour vos flashcards (Anki).

- Les valeurs simples telles que les entiers `int`, les flottants `float`, les chaînes `str` et les booléens `bool` font parti de la famille des **types de base**.
- Les types de bases ne permettant pas de regrouper plusieurs valeurs dans une seule entité, les programmeuses et programmeurs ont inventé la famille des **types construits**.
- Le **p-uplet** — également appelé n-uplet — fait parti de la famille des types construits.
- Le p-uplet est appelé `tuple` en Python.
- Un p-uplet est une suite ordonnée d'éléments : chaque élément possède une position, un **indice**.
- Un p-uplet peut contenir des éléments de différents types.
- Un p-uplet est **immuable** : il est impossible de modifier un des éléments contenu dans un p-uplet après création.
- Un p-uplet peut être mesuré avec len().
- On peut **accéder à un élément précis** d'un p-uplet en utilisant l'indice de cet élément.
- On peut **trancher** (*slicer*) un p-uplet pour en créer un autre.
- On peut **concaténer** des p-uplets pour en créer un autre.
- On peut parcourir un p-uplets par **valeur** et par **indice**.

Pour aller plus loin

Il est impossible de modifier un des éléments contenu dans un p-uplet après création, en voici la preuve :

```
1  >>> c = ('Vert', 'Jaune', 'Rouge')
2  >>> c[0] = 'Bleu'

   TypeError: 'tuple' object does not support item assignment
```

Heureusement, nous découvrirons un autre type de donnée construite qui nous donnera plus de libertés ...