Nom :	Prénom :	Classe :

NSI 1re — Tableau indexé (list)

Objectifs :
- Connaître la différence entre un p-uplet (tuple) et un tableau indexé (list), en Python.
- Savoir lire et modifier les éléments d'un tableau grâce à leurs index.
- Savoir itérer sur les valeurs des éléments d'un tableau (parcours par valeur).
- Savoir itérer sur les index (indices) des éléments d'un tableau (parcours par indice).

Type construit (rappel)

Un **type construit** est un type de données qui permet de regrouper plusieurs valeurs dans une seule entité.

Nous avons vu précédemment[1] qu'il existait principalement **trois types de données** appartenant à cette famille nommée « type construit » : p-uplet ou n-uplet (*tuple*, en Python), tableau indexé (*list*, en Python) et p-uplet nommé (*dict*, en Python).

Tableau indexé

Un **tableau indexé** est une structure de données qui permet de stocker **plusieurs valeurs** dans **un seul « conteneur »** et d'y accéder rapidement grâce à un numéro appelé indice (ou *index*).

Comme pour les p-uplet, un tableau indexé peut contenir **autant d'éléments que souhaité**, et chaque élément peut être de n'importe quel **type de donnée** (bool, int, float, etc).

Voici un exemple de tableau indexé :

Élément (valeur)	`'Mon voisin Totoro'`	`'Porco Rosso'`	`'Princesse Mononoké'`
Indice	`0`	`1`	`2`

Que pouvons-nous dire sur ce tableau indexé ?

- Ce tableau indexé est constitué de 3 éléments.
- Chacun de ces éléments est de type chaîne (*str*).
- Chaque élément possède un indice (*index*).

Tableau indexé VS p-uplet

Un **p-uplet (*tuple*)** est une séquence **non modifiable** : une fois créé, on ne peut plus changer, ajouter ou supprimer ses éléments. En Python, on utilise les **parenthèses** pour en créer un : `a = ('one', 'two')`

Un **tableau indexé (list)** est une séquence **modifiable** : on peut modifier un ou plusieurs éléments, insérer, supprimer, trier, etc. En Python, on utilise les **crochets** pour en créer un : `a = ['one', 'two']`

1 Voir le chapitre « Type construit » du cours « p-uplet (tuple) ».

Manipulations

Voici quelques maniements de base d'un tableau indexé :
- Création d'un nouveau tableau.
- Accès à un élément précis.
- Ajout d'un élément.
- Modification d'un élément.
- Suppression d'un élément.

Création d'un nouveau tableau

En Python, un tableau (*list*) est créé en utilisant les symboles crochets `[` et `]`.

```
a = []  # Création d'un tableau vide
```

Un tableau indexé peut contenir autant d'éléments que désirés :

```
b = ['a', ]    # Liste contenant 1 élément str
c = ['a', 'b'] # 2 éléments str

d = [True, False, True]  # 3 éléments bool

e = [3.14, 18]           # 2 éléments float et int
```

Accès à un élément précis d'un tableau

En Python, on **accède** à un élément précis d'un tableau selon la syntaxe suivante :

```
tableau[indice de l'élément auquel accéder]
```

Exemple :

```
a = ['a', 'b', 'c']
b = a[0] # Accède à l'élément d'indice 0

print(b) # Affiche 'a'
```

Comme toutes les valeurs, il n'est pas nécessaire d'utiliser des variables pour les manipuler.
Ainsi il est possible de faire `print( ['a', 'b', 'c'][1] )` ce qui afficherait la valeur `b`.

Ajout d'un élément dans un tableau

En Python, on **ajoute** un élément d'un tableau selon la syntaxe suivante :

```
tableau.append(valeur à ajouter)
```

Exemple :

```
a = ['a', 'b', 'c']
a.append('d')   # Ajoute le nouvel élément 'd' (str) à la fin du tableau stocké dans a

print(a) # Affiche ['a', 'b', 'c', 'd']
```

Il est également possible d'ajouter un élément *à un indice précis*. Dans ce cas, on ne passe plus par *append* mais par *insert* : `tableau.insert(indice, valeur à ajouter)`

Modifier un élément d'un tableau

En Python, on **modifie** un élément d'un tableau selon la syntaxe suivante :

```
tableau[indice de l'élément auquel accéder] = nouvelle valeur
```

Exemple :

```
a = ['a', 'b', 'c']
a[0] = 'x'   # On donne une nouvelle valeur à l'élément d'indice 0

print(a) # Affiche ['x', 'b', 'c']
```

Autre exemple :

```
a = ['a', 'b', 'c']
a[-1] = 'z'   # On donne une nouvelle valeur à l'élément d'indice -1

print(a) # Affiche ['x', 'b', 'z']
```

Supprimer un élément d'un tableau

En Python, on **supprime** un élément d'un tableau selon la syntaxe suivante :

```
tableau.pop(indice de l'élément à supprimer)
```

Exemple :

```
a = ['a', 'b', 'c']
a.pop(0) # Supprime l'élément d'indice 0

print(a) # Affiche ['b', 'c']
```

Autre exemple :

```
a = ['a', 'b', 'c']
a.pop(-1) # Supprime l'élément d'indice -1

print(a) # Affiche ['a', 'b']
```

➤ **Exercice 1** — À vous de jouer !

Déterminez la valeur de `a` à **la fin** de chacun des programmes suivants :

N°	Python	Valeur de `a`
1	`a = [5, 3, 8]` `a[0] = 8`	
2	`a = [5, 6]` `a.append(0)`	
3	`a = [8, 2]` `a[-1] = a[-1] * 2`	
4	`a = [5, 6]` `a.pop(-1)`	
5	`b = [0,1,1,2,3,5]` `a = len(b)`	
6	`a = [3, 6]` `a.insert(1, 9)`	

N°	Python	Valeur de `a`
7	`a = []` `a.append(8)`	
8	`a = []` `[3, 2].append(1)`	
9	`a = [2, 8, 7, 8]` `a[2] = 8`	
10	`a = ['m','d','z']` `a[-1] = 'r'`	

11	`a = []`
	`b = [9, 12, 8]`
	`b.append(3)`
	`a = [ b[1] ]`
	`a.append(13)`	

Algorithmie

Comme pour un p-uplet[2], il est possible de **parcourir les éléments** d'un tableau indexé **par valeur** ou **par indice**. Ces deux types de parcours permettent de réaliser de nombreuses opérations de *traitement par lot*[3] : recherche de l'existence d'un élément, recherche d'un extremum (valeur maximale ou minimale), calcul d'une moyenne, etc.

Parcours par valeur d'un tableau (d'une liste)

Le parcours par valeur est une syntaxe qui permet **d'accéder** successivement à la **valeur** de **chaque élément** présent dans un tableau, en vue de réaliser une opération spécifique (exemples : recherche de l'existence d'un élément, cumul de valeurs, calcul d'une moyenne, recherche d'un extremum, etc).

La syntaxe d'un parcours par valeur est la suivante :

```
for variable in tableau :
    # ...
```

Voici un exemple de parcours par valeur :

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

Cet exemple affichera ceci dans la console de Python :

```
a
b
c
```

➤ Exercice 2

Assignez la liste contenant `'Terrans'`, `'Zergs'` et `'Protoss'` à une variable `a`, puis affichez la valeur de chacun des éléments de cette liste à l'aide du parcours approprié :

2 Voir les chapitres « Parcours par valeur » et « Parcours par indice » du cours « p-uplet (tuple) ».

3 Le traitement par lot (ou *batch processing*) est un mode de traitement où un ensemble d'actions est regroupé puis exécuté automatiquement en une seule fois, sans intervention humaine pendant l'exécution.

Parcours par indice d'un tableau (d'une liste)

Le parcours par indice est, quant à lui, une syntaxe qui permet **d'accéder** successivement à **l'indice** de **chaque élément** présent dans un tableau, en vue de réaliser une opération spécifique (exemples : recherche de l'indice d'un élément précis, recherche de l'indice d'un extremum, etc). En voici la syntaxe :

```
for variable in range(len( tableau )) :
    # ...
```

Voici un exemple de parcours par indice :

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

Cet exemple affichera ceci dans la console de Python :

```
0
1
2
```

➤ Exercice 3

Assignez la liste contenant `'Terrans'`, `'Zergs'` et `'Protoss'` à une variable `a`, puis affichez l'indice de chacun des éléments de cette liste à l'aide du parcours approprié :

➤ Exercice 4

Assignez la liste `[7, 8, 9]` à une variable `a`, puis affichez l'indice **et** la valeur de chacun des éléments de cette liste à l'aide du parcours approprié (un seul parcours autorisé) :