Partie 2: Tableaux

I Introduction

Dans cette partie, on introduit la notion de tableau et de tuples, qui servent à stoquer plusieurs valeurs dans une même structure.

1 Tableaux

Un tableau sert à stoquer un nombre donné de valeurs du même type. Sa taille est déterminée à la création, et ne peut pas être modifiée. Les valeurs stockées dans le tableau peuvent elles être modifiées.

Warning

En python, on utilisera le type `list` en tant que tableau. On n'utilisera pas les opérations qui modifient la taille (comme `.append()` ou `.insert()`, de manière à les utiliser comme des tableaux plus classiques.

Le type `array` existe aussi, mais est très peu utilisé.

2 Tuples

Un tuple sert aussi à stocker plusieurs valeurs, mais contrairement aux tableaux:

- on **peut** y stocker des valeurs de type différent
- on ne **peut pas** modifier ces valeurs

Pratique lorsque l'on veut retourner plusieurs valeurs.

II Référence

1 Tableaux

1.1 Déclaration

```
my_array: list[int] = [1, 2, 3]
long_array: list[int] = [0] * 100
```

1.2 Accès

```
my_array[0] # accès au 1er élément
my_array[-1] # accès au dernier élément
my_array[4] # IndexError
```

1.3 Opérations

```
my_array[2] = 24 # Assignation d'un élément du tableau
```

```
len(my_array)  #longueur d'un tableau
arr[start : end] # sous-tableau, pour les indices entre start (inclus) et end
(exclus)
arr1 + arr2 # Concaténation de tableaux (du même type)
```

2 Tuples

```
my_tuple : tuple[int, str] = (3, "ok")
my_tuple[0] # accès au 1er élément
```

3 Boucles

L'opérateur `for .. in ..` permet d'itérer sur les valeurs d'un tableau (mais aussi d'un tuple, ou même d'une string).

```
for val in arr:
    ...
```

`val` va prendre tour à tour chacune des valeurs contenues dans le tableau `arr`.

3.1 Itération avec indice

On pourra aussi utiliser les fonctions range et enumerate:

```
for i in range(5):
    # `i` va prendre les valeurs entières de 0 jusqu'à 4 (inclus)
    ...

for i, value in enumerate(my_array):
    # i prends pour valeur l'indice du tableau, et value la valeur dans le tableau
    ...
```

III Exemples

1 Ajout d'un élément

```
def append(array: list[int], value: int) -> list[int]:
  return array + [value]
```

2 Somme des éléments

```
def somme_éléments(array: list[int]) -> int:
  res = 0
  for value in array:
    res = res + value
  return res
```

3 Contains

Ré-implémentation de l'opération `in`:

```
def contains(array: list[int], value: int) -> bool:
  for item in array:
    if item == value:
      return True
  return False
```

4 Modification d'un tableau

Une fonction qui modifie un tableau. On notera que le tableau n'a pas besoin d'être retourné.

```
def add_one(values: list[int], index: int):
  values[index] = values[index] + 1
```

```
mon_tableau: list[int] = [1, 2, 3]
add_one(mon_tableau, 1)
print(mon_tableau)
```

Quelle est la valeur affichée ?

IV Exercices

Notes:
- sauf précision contraire, on ne manipule ici que des tableaux d'entiers.
- les paramètres d'entrée doivent être validés

1 Bases

```python
def base_1(values: list[int]) -> int:
    """
    Pré-condition: au moins un élément
    Sortie: produit du premier et dernier élément
    """
```

```python
def base_2(values: list[int]) -> int:
    """
    Sortie: Somme des éléments du tableau
    """
```

```python
def base_3(n: int) -> None:
    """
    Somme des nombres de 1 à n
    """
```

```python
def base_4(values: list[int]) -> int:
    """
    Sortie: Somme des chaque élément multiplié par l'indice auquel il se trouve
    """
```

```python
def base_5(values: list[int]) -> tuple[int, int]:
    """
    Sortie: tuple
        - somme des éléments dont l'indice est pair
        - somme des éléments dont l'indice est impair
    """
```

```python
def base_6(values: list[int]) -> None:
    """
    Afficher chaque élément du tableau, à l'aide d'une boucle while
    (et non pas une boucle for)
    """
```

2 Correction partiel

2.1 Exercice 2

```python
def max_tableau(values: list[int]) -> int:
    """Sortie: plus grande valeur"""
```

2.2 Exercice 3

```python
def moyenne_tableau(values: list[float]) -> float:
    """Sortie: valeur moyenne"""
```

3 Manipulation de tableaux, et indices

3.1 Reverse

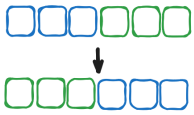
```python
def reverse(values: list[int]) -> list[int]:
    """
    Sortie: tableau renversé
    Ex: [1, 2 , 5, 3] -> [3, 5, 2, 1]
    """
```

3.2 Permutation

```python
def permut(values: list[int], indice: int) -> list[int]:
    """
    Pré-condition: vérifier que l'opération est valide
    Sortie: values, où l'élément désigné par indice, et le suivant, ont été inversés
    """
```

3.3 Permutation par blocs

```python
def bloc_permutation(values: list[int]) -> list[int]:
    """
    Sortie: un tableau, où chaque moitié  a été inversée
    Note: si le nombre d'éléments dans le tableau est impair,
l'élément du milieu ne bouge pas.
    """
```



3.4 Algo inconnu

```python
def algo_inconnu(values: list[int]) -> list[int]:
  res: list[int] = [0] * len(values)
  n: int = len(values)
  for i, value in enumerate(values):
    res[n-1-i] = value
  return res
```

1. Simuler l'exécution de l'algorithme sur de petites entrées.
2. À quoi sert cet algorithme ?
3. Est ce qu'il faudrait rajouter une validation sur les valeurs en entrée ?

4 Autres

4.1 Recherche 1

```python
def compte(values: list[int], x: int) -> int:
    """Sortie: nombre d'éléments égaux à x"""
```

4.2 Recherche 2

```python
def find(values: list[int], x: int) -> int:
    """Sortie: Somme des indices où se trouve la valeur x"""
```

4.3 Recherche 3

```python
def find_all(values: list[int]) -> list[int]:
    """
    On veut compter le nombre de fois que chaque nombre apparaît dans un tableau en entrée.
    Pré-condition: Les éléments du tableaux sont compris entre 0 et 10 (inclus)
    Sortie: Un tableau où l'élément à l'indice i représente le nombre de fois où i apparaît dans le tableau en entrée.
    """
```

4.4 Éléments supérieurs à la moyenne

```python
def find_2(values: list[int]) -> int:
    """
    Pré-condition: au moins un élément
    Sortie: Somme des indices où la valeur est supérieure à la moyennezq
    """
```

4.5 Combined

```python
def find_combined(values: list[int]) -> tuple[int, int, int]:
    """
    Pré-condition: au moins un élément
    Sortie: le max, le min, la moyenne
    """
```

4.6 Recherche séquences 1

```python
def find_sequence(values: list[int]) -> int:
    """Sortie: 1er indice d'une plus longue séquence de 0 (erreur si pas de 0)"""
```

4.7 Recherche séquences 2

```python
def find_sequence_triee(values: list[int]) -> int:
    """Sortie: 1er indice d'une plus longue séquence triée"""
```

4.8 Algo inconnu 2

```python
def algo_inconnu_2(values: list[int]) -> int:
    """Sortie: ???"""
    x: int = values[0]
    y: int = values[0]
    for value in values:
        if value > x:
            x = value
        if value < y:
            y = value
    return x - y
```

1. Éxecuter l'algorithme sur de petites entrées
2. Décrire ce que fait l'algorithme
3. Est ce qu'il faudrait rajouter une validation sur les valeurs en entrée ?

4.9 Sac à dos

On veut remplir un sac avec des objets. Le sac a une contenance maximale, on aimerat le remplir autant qu'on peut en prenant les éléments dans l'ordre.

```python
def remplis_sac(values: list[int], poids_max: int) -> int:
    """Sortie: poids du sac à dos, rempli autant qu'on peut..."""
```

5 Intro tri

Exercies dont on pourra réutiliser les résultats pour implémenter des algorithmes de tri

Note: on supposera que tous les tris se font par ordre croissant.

5.1 Vérifier si un tableau est trié

```python
def verif_tri(values: list[int]) -> bool:
    """Sortie: tableau trié ? """
```

5.2 Tri fusion - 0

```python
def fusionne_tableaux_triés(values1: list[int], values2: list[int]) -> list[int]:
    """
    Pré-condition: les tableaux sont triés
    Sortie: un tableau qui contient l'ensemble des éléments, triés.
    """
```

5.3 Recherche dans un tableau trié

```python
def find(values: list[int], n: int) -> int:
    """
    Pré-condition: values est trié
    Sortie: un indice du tableau où n est présent
    Erreur si n n'est pas présent
    """
```