Boucle et types construits

1 - Parcours par indices

1.1 str parcours par indice

Voir Type construit 5.2.7

1.2 list parcours par indice

Voir Type construit 5.3.7

1.3 tuple parcours par indice

Voir Type construit 5.4.7

1.4 Boucle POUR : somme d'un tableau

```
def somme_tableau(t :list[int]) -> int :
    """Renvoie la somme des entiers dans le tableau t"""

    s = 0                       # Initialisation d'un compteur
    for i in range(len(t)) :    # Pour chaque indice possible dans t
        s = s + t[i]            # On incrémente s de la valeur dans la case
    return s                    # APRES avoir fait tous les tours
```

1.5 TABLEAU STATIQUE muable en Python (modification possible) : Type construit 5.3.6

2 - Parcours par clés

2.1 dict, parcours par clés : Type construit 5.5.7

2.2 Boucle POUR : somme d'un dictionnaire

```
def somme_dict(d :dict[(str, int)]) -> int :
    """Renvoie la somme des entiers dans le dictionnaire d"""

    s = 0                   # Initialisation d'un compteur
    for cle in d.keys() :   # Pour chaque clé du dictionnaire d
        s = s + d[cle]      # Incrémente la somme avec cette valeur associée
    return s                # Renvoie la somme
```

2.3 Dictionnaire muable en Python (modification possible) Type construit 5.5.6

3 - Parcours par valeurs

3.1 String : parcours par valeurs

Si on désire uniquement LIRE les caractères d'un string, on PEUT utiliser une boucle **`for v in s`** en l'associant directement au nom du string.

La variable de boucle va alors contenir les caractères un par un, et non plus l'indice du caractère.

```
s = "bonjour"

for c in s :    # Pour chaque caractère de s
    print(c)    # Affiche le caractère c
```

Ce programme est équivalent à ceci :

```python
s = "bonjour"

print(s[0])
print(s[1])
print(s[2])
print(s[3])
print(s[4])
print(s[5])
print(s[6])
```

Ils affichent l'un et l'autre ceci dans la console :

```
b
o
n
```

```
j
o
u
```

```
r
```

3.2 tuple : parcours par valeurs

Si on désire uniquement LIRE les valeurs d'un n-uplet, on PEUT utiliser une boucle **`for v in tup`** en l'associant directement au nom du n-uplet.

```python
eleve = ("In Borderland", "Alice", 18, True)

for v in eleve :  # Pour chaque valeur possible dans eleve
    print(v)      # Affiche la valeur
```

Ce programme est équivalent à ceci :

```python
eleve = ("In Bo...", "Alice", 18, True)

print(eleve[0])
print(eleve[1])
print(eleve[2])
print(eleve[3])
```

Ils affichent l'un et l'autre ceci dans la console :

```
'In Borderland'
'Alice'
```

```
18
True
```

3.3 Tableau statique : parcours par valeurs

Syntaxe

Si on désire uniquement LIRE les valeurs d'un tableau, on PEUT utiliser une boucle **`for v in t`** en l'associant directement au nom du tableau.

```python
t = [20, 8, 18, 12]

for note in t :   # Pour chaque valeur possible dans t
    print(note)   # Affiche la valeur
```

Ce programme est équivalent à ceci :

```python
t = [20, 8, 18, 12]

print(t[0])
print(t[1])
print(t[2])
print(t[3])
```

Ils affichent l'un et l'autre ceci dans la console :

```
20
8
```

```
18
12
```

Somme

```python
def somme(t :list[int]) -> int :
    """Renvoie la somme des valeurs contenues dans le tableau"""

    s = 0           # Initialisation de la variable somme

    for v in t :    # Pour chaque valeur du tableau
        s = s + v   # Incrémente s de cette valeur

    return s        # Après avoir fait toute la boucle
```

Impossible de MODIFIER avec cette boucle

Notez bien qu'on ne peut que LIRE. Le code suivant **ne modifie pas** le tableau.

```
for note in t :         # Pour chaque valeur possible dans t
    note = note + 5     # Ne modifie absolument pas le contenu du tableau !
```

Pour modifier, il FAUT passer par l'indice :

```
t = [5, 2, -7, 15, -10]    # Création du tableau t
print("Avant la boucle")
print(t)                   # Affichage sur la console du tableau

for i in range(len(t)) :    # Pour chaque indice possible dans le tableau
    if t[i] < 0 :           # Si le contenu de la case i est négatif
        t[i] = -t[i]        # On remplit la case par son opposé, positif donc

print("Après la boucle")
print(t)                   # Affichage sur la console du tableau
```

On obtient alors ceci :

```
Avant la boucle
[5, 2, -7, 15, -10]
Après la boucle
[5, 2, 7, 15, 10]
```

4 - Création par compréhension

4.1 Création par compréhension à partir d'un autre tableau

A - Principe

La déclaration d'un tableau par compréhension consiste à créer un tableau à l'aide d'une boucle **for**.

```
nt = [ 0  for element in base]
```

Traduction : "Pour chaque élément du tableau initial, créé une case contenant 0 dans le nouveau tableau".

B - Exemple 1 : tableau de même taille

```
>>> base = [1, 2, 5]
>>> nt = [100 for valeur in base]
>>> nt
[100, 100, 100]
```

Traduction : "Pour chaque élément du tableau initial, créé une case contenant 100 dans le nouveau tableau".

C - Exemple 2 : copie peu profonde

```
copie = [v for v in b]
```

Traduction : "Pour chaque élément du tableau b, créé une case contenant la même chose dans le nouveau tableau".

```
>>> b = [1, 2, 5]
>>> copie = [v for v in b]
>>> copie
[1, 2, 5]
```

D - Exemple 3 : copie peu profonde modifiée

On peut indiquer une expression expliquant ce que doit être le contenu de la nouvelle case.

```
copie_mod = [v*100 for v in b]
```

Traduction : "Pour chaque élément du tableau initial, créé une case contenant 100 fois plus dans le nouveau tableau".

```
>>> b = [1, 2, 5]
>>> copie_mod = [v*100 for v in b]
>>> copie_mod
[100, 200, 500]
```

4.2 Création par compréhension sans tableau initial

A - Principe

On peut créer des tableaux contenant un nombre précis d'élements en utilisant **range()**.

```
nt = ["A" for i in range(10)]
```

Traduction : "Crée un tableau de 10 cases contenant toutes A".

B - Exemple 1

```
>>> t = [0 for i in range(10)]
>>> t
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Traduction : "Crée un tableau de 10 cases contenant toutes 0".

C - Exemple 2

```
>>> t = [i*10 for i in range(10)]
>>> t
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

La variable de boucle i varie de 0 à 9, et on place 10 fois plus dans la case correspondante du nouveau tableau.

D - Exemple 3

```
>>> import random
>>> tableau = [random.randint(1, 100) for i in range(10)]
>>> tableau
[42, 41, 96, 35, 58, 94, 16, 21, 84, 51]
```

On crée 10 cases dans lesquelles on place un nombre aléatoire entre 1 et 100.

4.3 Création par compréhension avec condition

A - Principe

On peut rajouter une expression booléenne à la toute fin de la déclaration.

B - Réaliser un filtrage

```
>>> import random
>>> t = [random.randint(0,20) for i in range(10)]
>>> t
[12, 12, 10, 3, 18, 7, 13, 16, 8, 14]
>>> t2 = [note for note in t if note >= 10]
>>> t2
[12, 12, 10, 18, 13, 16, 14]
```

C - Fonction de filtrage

La fonction détermine la valeur à placer lors de la création par compréhension. Imaginons qu'on dispose de cette fonction en mémoire :

```
def pas_plus(valeur :int, seuil :int) ->int :
    """écrête : renvoie la valeur ou le seuil si la valeur est supérieure au seuil"""

    if valeur > seuil :
        return seuil
    return valeur
```

```
>>> import random
>>> t = [random.randint(1, 20) for _ in range(10)]
>>> t
[14, 19, 7, 14, 11, 1, 13, 12, 20, 1]
```

```
>>> t2 = [pas_plus(valeur, 15) for valeur in t]
>>> t2
[14, 15, 7, 14, 11, 1, 13, 12, 15, 1]
```