Python 9 - Boucles bornées

I – Gestion complète de range()

Revoir Python 1 (4.9 et 4.10) pour le DS.

1) Range avec un seul argument : (« finale »)

On fournit la borne finale exclue, la valeur « presque finale ». On commence à 0, on compte de 1 en 1.

2) Range avec deux paramètres : (depart, « finale »)

On fournit la valeur initiale et la valeur « presque finale ». On compte de 1 en 1.

3) Range avec trois paramètres : (depart, « finale », pas)

On fournit la valeur initiale, la valeur « presque finale » et le pas (combien gagne-t-on à chaque tour ?)

4) Exemples :

```
>>> list(range(5))
[0, 1, 2, 3, 4]

>>> list(range(2, 5))
[2, 3, 4]

>>> list(range(0, 10, 2))
[0, 2, 4, 6, 8]
```

```
>>> list(range(0, 52, 10))
[0, 10, 20, 30, 40, 50]

>>> list(range(5, -1, -1))
[5, 4, 3, 2, 1, 0]

>>> list(range(5, -1, -2))
[5, 3, 1]
```

II – Sommes et concaténations successives

Sommes successives

Nous voudrions faire la somme de plusieurs nombres :
0 + 1 + 2 + 3 + 4 + 5 + 6 ... + jusqu'à un entier n.

On calcule la somme par additions successives, en écrasant la version précédente par la nouvelle version.
Cela reviendrait à faire ceci à la main :

```
  1   somme = 0
  2   somme = somme + 1
  3   somme = somme + 2
 ..   ...
1002  somme = somme + 1000
```

Utilisons plutôt une boucle dont la variable de boucle se nommerait **nombre**.

```
1  somme = 0
2  for nombre in range(1, 1001):
3      somme = somme + nombre
4  print(somme)
```

La phase d'initialisation de la ligne 1 est fondamentale.
Déroulé : L1 – L2(nombre=1) -L3 – L2(nombre=2)-L3 ... L2(nombre=1000)-L3 – L4.

Concaténations successives

Nous voudrions créer cette chaîne :
"0 1 2 3 4 5 6 ... 1000" jusqu'à un entier final.

On crée la chaîne par concaténations successives, en écrasant la version précédente par la nouvelle version.
Cela reviendrait à faire ceci à la main :

```
  1   chaine = ""
  2   chaine = chaine + str(0) + " "
  3   chaine = chaine + str(1) + " "
 ..   ...
1002  chaine = chaine + str(1000) + " "
```

Utilisons plutôt une boucle dont la variable de boucle se nommerait **nombre**.

```
1  chaine = ""
2  for nombre in range(1001):
3      chaine = chaine + str(nombre) + " "
4  print(chaine)
```

La phase d'initialisation de la ligne 1 est fondamentale.
Déroulé : L1 – L2(nombre=1) -L3 – L2(nombre=2)-L3 ... L2(nombre=1000)-L3 – L4.

Remarque: str() permet d'obtenir un string à partir de l'entrée fournie :

```
>>> str(5)
'5'
```

III – Obtenir les indices avec une boucle Pour

1) Rappel fondamental :

- Le conteneur ⇨ **t** (le tableau)
- L'indice d'une case ⇨ **i**
- Le contenu d'une case ⇨ **t[i]**

2) Obtenir un par un les indices d'un string, d'un tableau, d'un tuple

Un conteneur ayant 20 cases possède des indices allant de 0 à 19. Puisqu'on fournit la valeur « presque finale », le -1 est déjà compris dans l'utilisation du range() :

```
1  def somme_tableau(t:list[int]) -> int:
2      """Renvoie la somme des entiers dans le tableau t"""
3
4      s = 0                       # Initialisation d'un compteur
5      for i in range(len(t)):     # Pour chaque indice possible dans t
6          s = s + t[i]            # On incrémente s de la valeur dans la case i de t
7      return s                    # APRES avoir fait tous les tours
```

3) Le tableau est muable

```
1  t = [5, -10, 15, -20]
2  for i in range(len(t)):     # Pour chaque indice possible dans t
3      if t[i] < 0:           # Si le contenu de la case i est négatif
4          t[i] = - t[i]      # On le remplace par son opposé, positif.
```

Après exécution du programme, t référence **5, 10, 15, 20]**

IV – Obtenir les clés avec une boucle Pour

1) Rappel fondamental :

- Le conteneur ⇨ **d** (le dictionnaire)
- La clé d'une case ⇨ **cle**
- Le contenu d'une case ⇨ **d[cle]**

2) Obtenir une par une les clés d'un dictionnaire

```
1  def somme_dict(d:dict[(str, int)]) → int:
2      """Renvoie la somme des entiers dans le dictionnaire d"""
3
4      s = 0                   # Initialisation d'un compteur
5      for cle in d.keys():    # Pour chaque clé du dictionnaire d
6          s = s + d[cle]      # Incrémente s avec le contenu de la case de clé cle
7      return s                # Renvoie s après avoir fait la boucle
```

3) Le dictionnaire est muable

Ce programme rajoute 1 à toutes les valeurs paires contenues dans le tableau.

```
1  d = {'A':10, 'B':9, 'C':18}
2  for cle in d.keys():         # Pour chaque clé possible dans le dictionnaire
3      if (d[cle] % 2) == 0:    # Si la valeur (le contenu de la case) est paire
4          d[cle] = d[cle] + 1  # On rajoute 1 à la case
```

Après exécution du programme, d référence **{'A':11, 'B':9, 'C':19}**

V – Obtenir directement les valeurs avec une boucle Pour

Pour les tableaux, strings et tuples : on peut utiliser de cette façon la boucle for pour obtenir dans la variable de boucle un à un les différents contenus des cases :

```
1  def somme_tableau(t:list[int]) -> int:
2      """Renvoie la somme des entiers dans le tableau t"""
3
4      s = 0                      # Initialisation d'un compteur
5      for valeur in t:           # Pour chaque valeur dans t
6          s = s + valeur         # On incrémente s de la valeur
7      return s                   # APRES avoir fait tous les tours
```

Notez bien qu'on utilise pas la fonction range() ici : on récupère directement un contenu de cases.

Avantage : code plus simple, plus clair.
Désavantage : Uniquement pour lire, ne permet pas la modification des cases puisqu'on a besoin de l'indice de la case pour modifier le contenu.

Pour les dictionnaires : on utilise **`values()`** et pas **`keys()`**.

```
1  def somme_dict(d:dict[(str, int)]) → int:
2      """Renvoie la somme des entiers dans le dictionnaire d"""
3
4      s = 0                      # Initialisation d'un compteur
5      for valeur in d.values():  # Pour chaque valeur contenu dans le dictionnaire d
6          s = s + valeur         # Incrémente s avec cette valeur
7      return s                   # Renvoie s après avoir fait la boucle
```

Avantage : code plus simple, plus clair.

Désavantage : Uniquement pour lire, ne permet pas la modification des cases puisqu'on a besoin de la clé de la case pour modifier le contenu.

Pour le II

```
   1   somme = 0
   2   somme = somme + 1
   3   somme = somme + 2
  ..   ...
1002   somme = somme + 1000
```

```
1  somme = 0
2  for nombre in range(1, 1001):
3      somme = somme + nombre
4  print(somme)
```

```
   1   chaine = ""
   2   chaine = chaine + str(0) + " "
   3   chaine = chaine + str(1) + " "
  ..   ...
1002   chaine = chaine + str(1000) + " "
```

```
1  chaine = ""
2  for nombre in range(1001):
3      chaine = chaine + str(nombre) + " "
4  print(chaine)
```

Pour le III

```
1  def somme_tableau(t:list[int]) -> int:
2      """Renvoie la somme des entiers dans le tableau t"""
3
4      s = 0                       # Initialisation d'un compteur
5      for i in range(len(t)):     # Pour chaque indice possible dans t
6          s = s + t[i]            # On incrémente s de la valeur dans la case i de t
7      return s                    # APRES avoir fait tous les tours
```

```
1  t = [5, -10, 15, -20]
2  for i in range(len(t)):    # Pour chaque indice possible dans t
3      if t[i] < 0:           # Si le contenu de la case i est négatif
4          t[i] = - t[i]      # On le remplace par son opposé, positif.
```

Pour le IV

```
1  def somme_dict(d:dict[(str, int)]) -> int:
2      """Renvoie la somme des entiers dans le dictionnaire d"""
3
4      s = 0                    # Initialisation d'un compteur
5      for cle in d.keys():     # Pour chaque clé du dictionnaire d
6          s = s + d[cle]       # Incrémente s avec le contenu de la case de clé cle
7      return s                 # Renvoie s après avoir fait la boucle
```

```
1  d = {'A':10, 'B':9, 'C':18}
2  for cle in d.keys():          # Pour chaque clé possible dans le dictionnaire
3      if (d[cle] % 2) == 0:     # Si la valeur (le contenu de la case) est paire
4          d[cle] = d[cle] + 1   # On rajoute 1 à la case
```

Pour le V

```
1  def somme_tableau(t:list[int]) -> int:
2      """Renvoie la somme des entiers dans le tableau t"""
3
4      s = 0                       # Initialisation d'un compteur
5      for valeur in t:            # Pour chaque valeur dans t
6          s = s + valeur          # On incrémente s de la valeur
7      return s                    # APRES avoir fait tous les tours
```

```
1  def somme_dict(d:dict[(str, int)]) -> int:
2      """Renvoie la somme des entiers dans le dictionnaire d"""
3
4      s = 0                       # Initialisation d'un compteur
5      for valeur in d.values():   # Pour chaque valeur contenue dans le dictionnaire d
6          s = s + valeur          # Incrémente s avec cette valeur
7      return s                    # Renvoie s après avoir fait la boucle
```