

NOTEBOOK : La boucle `for`

September 10, 2024

III. La boucle `for`

Lorsque que dans un programme, certains fragments sont exécutés un grand nombre de fois, on peut utiliser dans le code une instruction pour réaliser ces répétitions, que l'on appelle communément **boucle**.

On s'intéresse ici aux boucles dites *bornées*, aussi appelées boucles `for` en Python, qui permettent de répéter une séquence d'instructions un nombre de fois donnée.

1. Boucles bornées simples

Répétition d'une instruction

Pour exécuter trois fois la même instructions, on peut recopier cette instruction trois fois.

```
print("NSI")
print("NSI")
print("NSI")
```

Dans la plupart des langages de programmation, comme en Python, on peut éviter cette répétition de code en utilisant une boucle `for`.

```
for i in range(3):
    print("NSI")
```

En observant bien la syntaxe de cette construction, on remarque que : - on indique entre parenthèse après `range` un nombre *entier* de répétition à effectuer, - l'instruction à répéter se situe après le symbole `:`, avec un « grand » espace en début de ligne que l'on appelle **indentation**.

```
[3]: #Code à tester :
      # 1. Teste l'exemple ci-dessus.
      # 2. Modifie le code pour que le nombre de répétition puisse être rentré par
      ↪ l'utilisateur du programme.
```

Répétition d'un bloc d'instructions

La répétition n'est pas limitée à une seule instruction, comme le montre l'exemple suivant :

```
a = 1

for i in range(5):
```

```
print("Le double de", a, " est :", a * 2)
a = a * 2
```

On observe ici que :

- Les deux instructions décalées ou « indentées » sont répétées. Cette suite d'instructions regroupées en un **bloc** forme le **corps de la boucle**.
- À chaque nouvelle répétition, ou « tour de boucle », la valeur de la variable **a** est mise à jour en reprenant la valeur qu'elle avait au tour précédent. C'est ce qu'on appelle couramment un **accumulateur**.

```
[25]: #Code à tester :
# 1. Teste l'exemple ci-dessus.
# 2. Écris la trace d'exécution de ce programme.
# 3. Modifie le code pour obtenir les 5 premiers nombres de la table de 3, sans
↳ introduire de nouvelle variable, ni utiliser la variable i.
```

Application : calcul de moyenne

Pour calculer la moyenne en NSI d'un nombre donné d'élèves, on a besoin d'additionner toutes leur note.

Cette addition peut être faite note par note à l'aide d'un accumulateur. 1. Écris un programme qui effectue la moyenne de 5 élèves dont la note va être saisie, l'une après l'autre, par l'utilisateur du programme. 2. Modifie le programme pour effectuer la moyenne d'un nombre quelconque d'élèves, nombre rentrer par l'utilisateur en début de programme.

```
[11]: # Programme à écrire
```

Utilisation de l'indice de boucle

Dans la syntaxe de la boucle `for`, comme `for i in range(10):`, une variable spéciale est introduite, nommée ici `i`, permet de numéroter les tours de boucles. On l'appelle **indice de boucle** ou **compteur de boucle**.

Cette variable est accessible à l'intérieur du corps de boucle, comme le montre l'exemple suivant :

```
for i in range(10):
    print(i)
```

```
[ ]: # 1. Teste le code ci-dessus et observe les valeurs successivement prise par le
↳ compteur i.
# 2. Modifie le code pour afficher tous les entiers naturels inférieurs ou
↳ égale à 100.
```

Pour numéroter les tours de boucle à partir d'un entier donné, on peut utiliser la syntaxe suivante :

```
for k in range(5,13)
    print(k)
```

```
[17]: # 1. Teste le code ci-dessus et observe les valeurs successivement prise par le
      ↪compteur k.
      # 2. Compte de le nombre de répétitions.
      # 3. Écris la formule te permettant d'obtenir le nombre répétition effectuer à
      ↪partir des deux nombres indiqués entre paranthèse après range.
```

Remarques :

- On peut nommer le compteur de boucle comme on le souhaite, cependant il faut veiller à ne pas utiliser un nom de variable déjà existant.
- Rien n'empêche de modifier la valeur du compteur dans le corps de la boucle. Cependant cette modification ne subsiste pas au tour d'après. Cette usage est donc déconseiller.
- Après l'exécution d'une boucle `for`, le compteur subsiste en mémoire et contient la valeur du dernier tour de boucle. Il est là aussi recommander de ne pas utiliser cette variable en dehors du corps de boucle.

2. Boucles imbriquées

Il est tout à fait possible d'inclure une boucle dans une autre boucle. On obtient ainsi « des répétitions de répétition ».

Pour obtenir toutes les coordonnées entières des cases d'un tableau à deux dimensions, de 3 x 3 cases, dont l'origine serait la case en haut à gauche, on peut écrire par exemple :

```
for y in range(3):
    for x in range(3):
        print("(" , x , ";" , y , ")")
```

- On commence par fixer le numéro de ligne, qui correspond ici à la valeur d'ordonnée y du compteur de la 1ère boucle.
- Puis pour chaque valeur de y fixée, on fait varier la valeur d'abscisse qui correspond à la valeur du compteur x de la 2ème boucle.

```
[26]: # 1. Teste le code ci-dessus.
      # 2. Fais sur ton carnet de bord un tableau de 3x3 cases puis construis la
      ↪trace d'exécution de ce programme en l'exécutant pas à pas. Suis le parcours
      ↪du tableau effectué en noicissant chacune des cases.
      # 3. Modifie ce programme pour que la case d'origine serait cette fois-ci en
      ↪bas à gauche.
```

Application : Affichage des tables de multiplication

1. À l'aide d'une boucle `for`, écris un programme qui affiche les 10 premières lignes de la table de multiplication de 2 (ex. $0 \times 2 = 0$, $1 \times 2 = 2$...).
2. Modifie ce programme pour afficher toutes les tables de multiplication de 2 à 9, en imbriquant deux boucles.