Structures répétitives

Un programme informatique est souvent amené à répéter des instructions. Le but de ce chapitre est d'étudier les structures qui permettent ce type de fonctionnement.

I. La boucle Pour

La boucle POUR est souvent qualifiée de déterministe. Cela signifie que le programme connaît, dès le départ, le nombre de répétitions qu'il devra effectuer.

Dans l'environnement Python, une première façon d'implémenter une boucle POUR est d'utiliser la fonction range. Cette fonction possède plusieurs modes:

- range(max+1): fournit tous les nombres entiers de 0 à max. Par exemple, range(5) correspond à [0, 1, 2, 3, 4]. On remarquera que le paramètre de la fonction n'est pas le maximum mais l'entier suivant.
- range(min, max+1): fournit tous les nombres entiers de min à max. Par exemple, range(2,10) correspond à [2, 3, 4, 5, 6, 7, 8, 9]. La remarque précédente reste valide.
- range(min, max+1, pas): fournit tous les nombres entiers compris entre min et max séparés par la valeur pas. Par exemple, range(2,10,3) correspond à [2, 5, 8].

La syntaxe de la boucle POUR est la suivante:

```
for variable in collection:
    Bloc d'instructions à effectuer
Suite du programme
```

Voici un exemple de boucle POUR:

```
etendue=range(2,6)                    # 2, 3, 4, 5
for n in etendue:
    print(n)
print("fini")
```

Donnera:

```
2
3
4
5
fini
```

Remarque: si l'on ne souhaite pas créer de façon explicite la collection etendue, il est possible de condenser le programme de la façon suivante:

```
for n in range(2,6):
    print(n)
print("fini")
```

La fonction range est simple à comprendre et à utiliser mais, dans certains cas, elle n'est pas suffisante. Parfois, il est nécessaire de définir la liste des valeurs "manuellement". Pour cela, on utilise la syntaxe suivante: liste=[valeur 1, valeur 2, valeur 3, …. , valeur n].

Remarques:

- on peut obtenir le nombre d'éléments d'une liste à l'aide de la fonction len.
- une liste peut contenir d'autres types de variables que des entiers. On peut, par exemple, définir une liste de chaînes de caractères.

Voici un exemple de boucle POUR utilisant une liste de valeurs définies par l'utilisateur:

```
liste=[2, 3, 5, 7]
for n in liste:
    print(n*n)
print("Nb de valeurs dans la liste:")
print(len(liste))
print("fini")
```

Le résultat à l'écran sera le suivant:

```
4
9
25
49
Nb de valeurs dans la liste:
4
fini
```

II. La boucle Tant Que

Dans certains cas, le nombre de répétitions que le programme doit effectuer n'est pas connu au départ. La boucle POUR devient donc inapplicable. En conséquence, tous les langages informatiques possèdent des boucles exploitant des conditions d'exécution (ou d'arrêt) qui sont réévaluées à chaque itération. La condition est souvent appelée prédicat.

Avec Python, nous exploiterons la boucle Tant Que …. Faire. Son principe de fonctionnement est assez simple: tant qu'une condition est remplie, le programme exécute un bloc d'instructions. Bien sûr, si la condition n'est plus valide, le programme sort de la boucle et passe à la suite.

La syntaxe de la boucle Tant Que est la suivante:

```
while condition:
    Bloc d'instructions
Suite du programme
```

Remarque très importante: la condition doit être évaluée avant le while.

Supposons que l'on souhaite afficher tous les carrés strictement inférieurs à 100. Le code correspondant est le suivant:

```
i=1                         # Initialisation indispensable de i
while i*i<100:              # Le carré de i est-il toujours <100 ?
    print(i*i)
    i=i+1                   # On passe à l'entier suivant
print(« fini »)
```

Remarques:

- 100 n'est pas affiché car l'inégalité est stricte.
- L'initialisation de i est indispensable car cette variable est présente dans le prédicat.
- L'instruction i=i+1 permet de passer à l'entier suivant. En cas d'oubli, le programme « boucle à l'infini ».

Il est possible de stopper l'exécution d'une boucle à l'aide de l'instruction break. Lorsque Python rencontre cette instruction, il passe automatiquement à la suite.

Voici un exemple de boucle qui exploite l'instruction break pour sa sortie:

```
i=1                             # Initialisation indispensable de i
while i<200:                    # i ne doit pas dépasser 199
    print(i*i)
    i=i+1
    if i*i>=100:                # Lorsque le carré est supérieur ou égal à 100,
        break                   # on sort de la boucle
print(« fini »)
```

Ce programme fournit le même résultat que le précédent. C'est le break qui permet de limiter la taille du carré. L'écriture est cependant beaucoup moins lisible.