L'imbrication

1. La notion de blocs d'imbrication

Le branchement d'une instruction `if` et le corps de boucle d'une instruction `for` représentent une notion similaire que nous regroupons sous le terme générique de sous-bloc de code. Les sous-blocs, qu'ils soient issus d'un branchement ou d'un corps de boucle, ont la capacité de contenir d'autres sous-blocs. Ils s'imbriquent alors naturellement comme le font les poupées russes.

La différence entre les poupées russes et le code d'un programme, c'est qu'elles ne permettent qu'une seule inclusion par niveau : chaque poupée ne peut contenir qu'une seule autre poupée de taille inférieure, qui elle-même contient aussi une unique autre poupée.

Imbrication des poupées russes

Dans le langage Python, un bloc correspond à des lignes de code successives ayant le même niveau d'indentation. Peuvent être présentes parmi ces lignes d'autres lignes de code de niveau d'indentation inférieur. Ces lignes décrivent alors des sous-blocs relativement au bloc courant. Les instructions : `print("bonjour")` ou `a = b + 1` ne génèrent pas l'apparition de sous-bloc dans les lignes de code. Pour créer une imbrication, il faut passer par une boucle `for` ou par une condition `if`. Aux derniers niveaux d'inclusion, on trouve des blocs terminaux ne comportant que des instructions simples sans boucle et sans condition.

L'indentation naturelle du langage Python met en évidence le niveau d'inclusion dans lequel se trouve chaque ligne de code. Analysons un exemple pour faire apparaître sa hiérarchie de blocs :

```
for i in range(5) :                    Bloc A : principal - Niv 1
    print ("-"*10)                     Bloc B : Niv 2
    for k in range(10) :
        if i + k == 10 :               Bloc C : Niv 3
            print('#',end="")          Bloc D : Niv 4
        else:
            print(".",end="")          Bloc E : Niv 4
    print()
for i in range(10)
    print("$"*10)                      Bloc F : Niv 2
    for k in range(10) :
        print(k,end='')                Bloc G : Niv 3
    print("toto")

print("terminé")
```

Le bloc principal contient trois instructions dont deux boucles `for` créant deux sous-blocs B et F de niveau 2 et un appel à la fonction `print()`. Les blocs terminaux D, E et H ne contiennent aucun sous-bloc, car ils ne contiennent aucune instruction `if`/`for`. Nous pouvons remarquer qu'ils sont à des niveaux différents. Cela est normal, le niveau d'un bloc terminal dépend des blocs précédents. Ainsi, dans un programme, les blocs terminaux ne sont pas tous au même niveau. Le bloc B contient trois instructions dont une boucle `for` à l'origine du sous-bloc C. Le bloc C contient une instruction `if`/`else` générant deux sous-blocs supplémentaires. Le bloc F contient trois instructions dont une

boucle `for` à l'origine du sous-bloc G. Vous l'aurez compris, un bloc inclut autant de sous-blocs que de mots-clés `if`/`else`/`for`.

Lorsque l'on exécute un programme en mode pas à pas, on se concentre principalement sur le bloc courant. En effet, dès que vous êtes à l'intérieur d'un bloc, le temps semble s'arrêter pour les blocs supérieurs : les indices des boucles `for` supérieures n'évoluent pas et les conditions associées aux instructions `if` supérieures ont été validées. Maintenant que nous savons identifier cette hiérarchie de blocs, nous allons examiner leur dynamique.

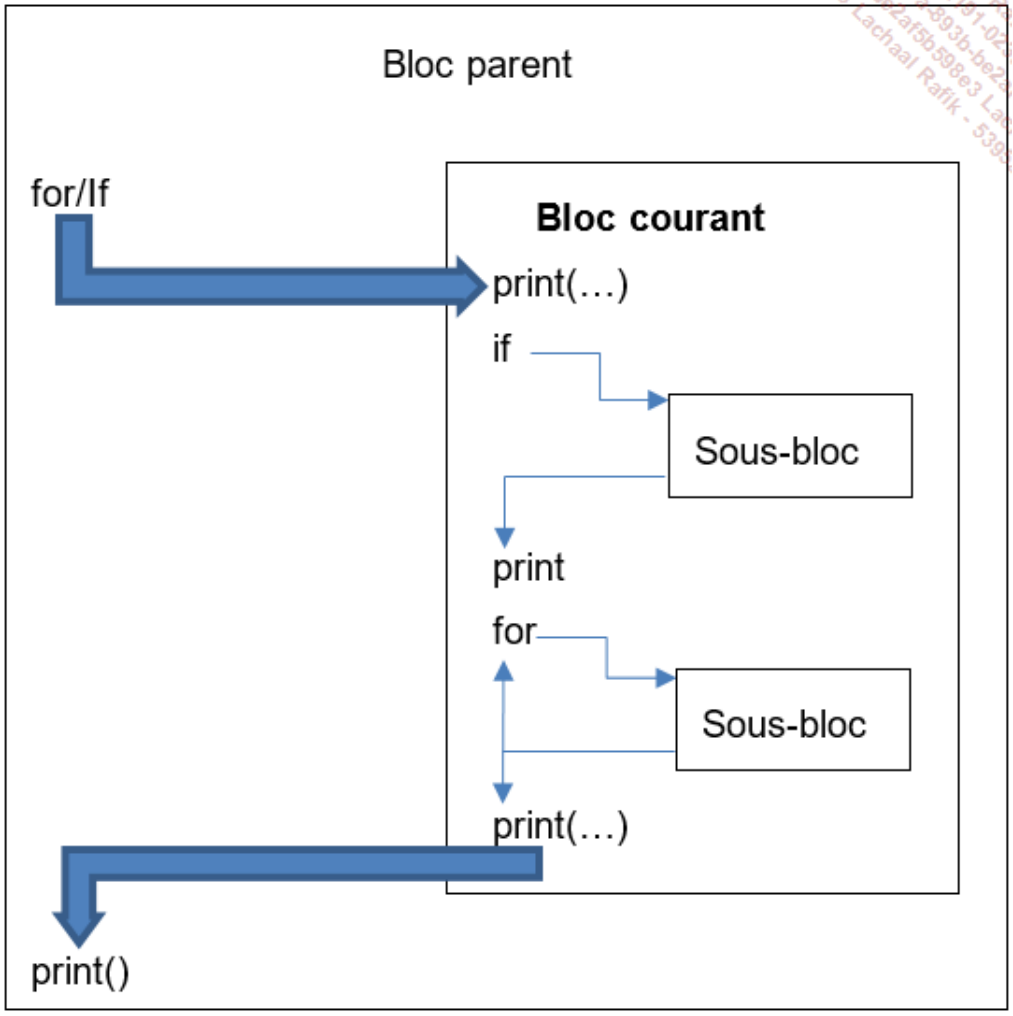
Lorsqu'une instruction `if` est rencontrée, alors :

- Si sa condition est vraie :
  - On entre dans le sous-bloc.
  - On exécute entièrement ce sous-bloc.
  - Une fois le sous-bloc terminé, on revient au niveau de l'instruction `if`.
- Si une instruction se trouve après l'instruction `if` dans le bloc courant :
  - On passe à cette instruction.
  - Sinon on quitte le bloc courant.

Lorsqu'une instruction `for` est rencontrée, alors :

- Tant qu'il reste des indices à parcourir dans la plage d'indices :
  - On entre dans le sous-bloc.
  - On exécute entièrement ce sous-bloc avec cette valeur d'indice.
  - Une fois le sous-bloc terminé, on revient au niveau de l'instruction `for`.
- Si une instruction se trouve après l'instruction `for` dans le bloc courant :
  - On passe à cette instruction.
  - Sinon on quitte le bloc courant.

Vous remarquez que l'interaction de l'instruction `if` et de l'instruction `for` avec leurs sous-blocs et leur bloc parent est quasi identique. La seule différence repose sur le fait que la condition `if` permet d'entrer 0 ou 1 fois dans le sous-bloc alors que l'instruction `for` permet d'y entrer plusieurs fois. Voici un schéma résumant l'interaction du bloc courant avec ses sous-blocs et son bloc parent. Les règles présentées permettent de parcourir un programme dans son intégralité en mode pas à pas. Elles sont communes à tous les autres langages informatiques utilisant les mots-clés `if`/`for` :



2. Imbriquer des conditions

Nous présentons un exemple de conditions `if` imbriquées. Notez que la comparaison entre deux chaînes de caractères se fait avec l'opérateur ==.

```
01 if age >= 18 :                  IF - démarrage branchement 1
02    if genre == "M" :            |     | sous-bloc du if
03         print("Homme")          |     |
04    else:                        |     |
05         print("Femme")          |     v
06    print("Fin sous-partie")     v
07 else :                          ELSE - démarrage branchement 2
08    if age < 12:                 |     | sous-bloc du else
09         print("Enfant")         |     |
10    else:                        |     |
11         print("Adolescent")     v     v
12 print("Fin")
```

À la première ligne se trouve la condition `if age >= 18`. Supposons que la variable `age` vaille 20. Dans ce cas, le résultat de la condition est vraie. L'interpréteur Python rentre alors dans le sous-bloc suivant l'instruction `if`. Le sous-bloc entre les lignes 8 et 11 après l'instruction `else` n'est pas exécuté dans ce cas. Nous passons ainsi à la deuxième ligne où se trouve une deuxième condition `if` sur le genre de la personne. Supposons que la variable

`genre` vaille "F", le résultat de la condition `genre == "M"` est donc faux. C'est le deuxième sous-bloc qui est alors choisi pour l'exécution. La ligne `print("Homme")` est donc ignorée dans ce cas. Nous rentrons donc dans le sous-sous-bloc de la ligne 5 contenant l'instruction `print("Femme")` pour l'exécuter. Après cela, il n'y a plus d'instructions dans ce sous-sous-bloc et nous remontons à l'instruction `if` du bloc supérieur présente à la ligne 2. Comme une instruction de même niveau se trouve à la ligne 6, l'exécution se poursuit dans le sous-bloc courant et affiche ainsi le message : "Fin de sous partie". Il n'y a plus d'instructions dans ce sous-bloc. Ainsi, le sous-bloc courant se termine et on remonte à l'instruction `if` du bloc supérieur à la ligne 1. Nous avons une instruction `print()` en ligne 12 qui se trouve au même niveau que l'instruction `if` de la ligne 1, l'exécution se déplace alors vers cette ligne. Après cela, il n'y a plus d'instructions dans le bloc principal, le programme est donc terminé.

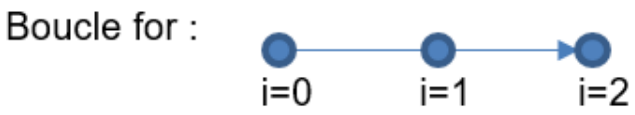
Deuxième scénario. Supposons cette fois que la variable `age` vaille 16. La condition de la ligne 1 vaut donc faux. Comme une instruction `else` est associée à cette instruction `if`, nous allons exécuter le sous-bloc se trouvant après la ligne 7. Supposons que la variable `age` vaille 10. Par conséquent, la condition présente à la ligne 8 est vraie et nous exécutons le sous-bloc présent à la ligne 9. Une fois l'instruction `print("Enfant")` exécutée, ce sous-sous-bloc se termine et nous remontons à l'instruction `if` du bloc supérieur à la ligne 8. Comme dans le bloc courant aucune instruction ne reste à réaliser, nous remontons à l'instruction `if` du bloc supérieur à la ligne 1. Nous avons une instruction `print()` à la ligne 12 qui se trouve au même niveau que l'instruction `if` de la ligne 1. L'exécution se déplace alors vers cette ligne. Après cela, il n'y a plus d'instructions dans le bloc principal, le programme est donc terminé.

3. Imbriquer des boucles et des conditions

Une boucle `for` suivie d'une commande `print()` affichant son indice pose peu de problèmes de compréhension, car sa structure est linéaire, quasi narrative. Tout le monde visualise une progression qui démarre d'un indice 0 et qui avance par pas de 1 jusqu'à atteindre la dernière valeur d'indice. L'utilisation de boucles `for` imbriquées fait apparaître une structure à deux dimensions et cette configuration n'est pas forcément familière aux esprits. Passons maintenant aux exemples :

```
for i in range(0,3) :
  print(i)
```

Nous modélisons la progression de cette boucle par une ligne et des nœuds correspondant à chaque valeur prise par l'indice.

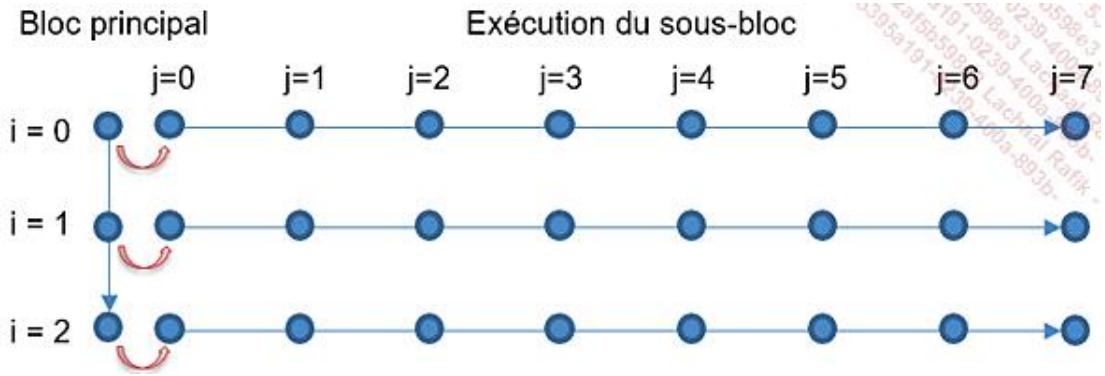


Examinons maintenant une double boucle `for` :

```
for i in range(0,3) :
 print("i :",i, "--> j : ", end="")  #pas de retour à la ligne
 for j in range(8) :
   print("{0:2d}".format(j),end="")
 print()
```

Sachant gérer les appels des sous-blocs, nous savons que pour chaque itération de la boucle `for i`, nous allons déclencher l'exécution d'un sous-bloc contenant une boucle `for j` affichant les valeurs de cet indice. Une fois la boucle `for j` terminée, on effectue un retour à la ligne grâce à la fonction `print()`, puis le bloc étant terminé, on

remonte au bloc parent. Ainsi, on retourne à l'instruction `for i` qui va redémarrer l'exécution du sous-bloc `for j` avec une nouvelle valeur pour l'indice `i`. Ces redémarrages perdurent tant qu'il reste pour l'indice `i` des valeurs à parcourir dans la plage `range(0,3)`. En reprenant le schéma précédent, nous obtenons le parcours à deux dimensions suivant :



En exécutant le code précédent, voici le résultat obtenu :

```
i : 0 --> j : 0 1 2 3 4 5 6 7
i : 1 --> j : 0 1 2 3 4 5 6 7
i : 2 --> j : 0 1 2 3 4 5 6 7
```

Exercice 1 :

```
for i in range(4) :
 for j in range(6) :
   print("{0:3d}".format(i*10+j),end="")
 print()
```

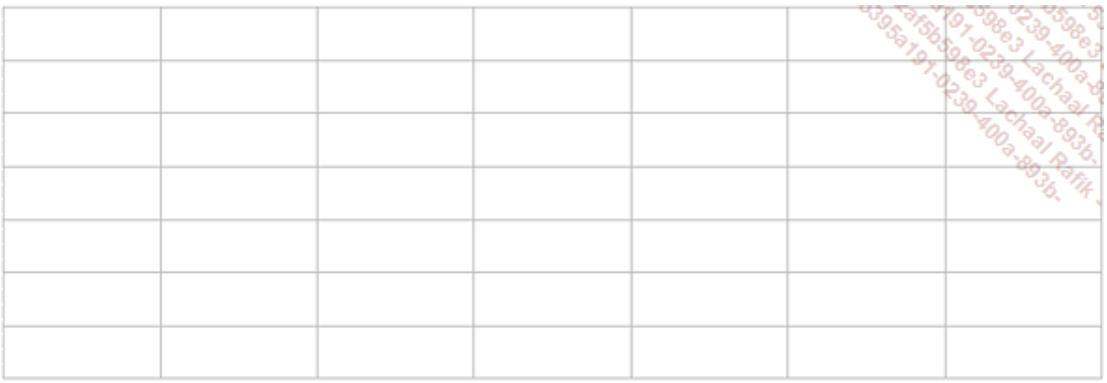
Utilisez le tableau suivant pour donner l'affichage produit par ce code. Des cases peuvent rester vides.

La correction est donnée à la fin de la série d'exercices.

Exercice 2 :

En reprenant la même structure que l'exercice précédent, mais en intervertissant les deux boucles, prédisez la sortie du programme suivant :

```
for j in range(6) :
 for i in range(4) :
   print("{0:3d}".format(i*10+j),end="")
 print()
```



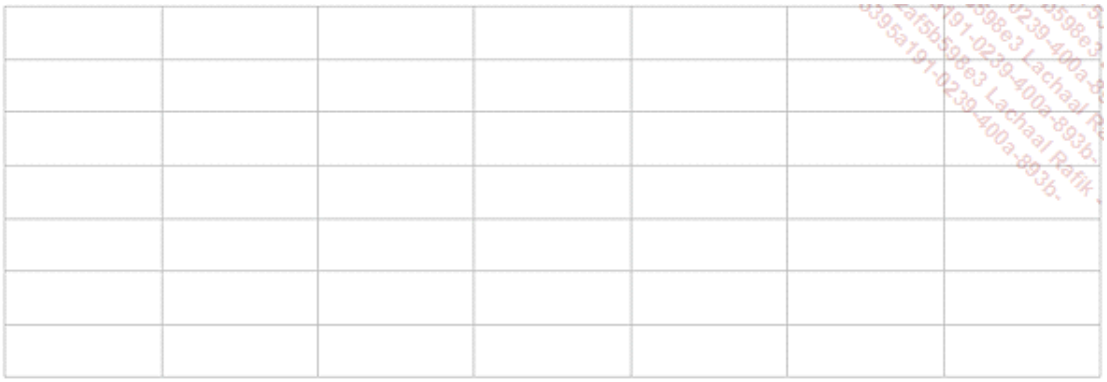
La correction est donnée à la fin de la série d'exercices.

Exercice 3 :

Deux boucles `for` imbriquées restent la difficulté principale d'un programmeur débutant. Les structures imbriquées `for`/`if`, `if`/`for` ou `if`/`if` restent plus accessibles car proches du déroulement linéaire. Nous allons vous proposer maintenant d'étudier l'imbrication `for`/`if`/`for`.

```
for i in range(4) :
 if i % 2 == 0 :
   for j in range(6) :
     print("{0:3d}".format(j*10+i),end="")
 else:
   for j in range(6) :
     print("{0:3d}".format( i),end="")
 print()
```

Utilisez le tableau suivant pour donner l'affichage produit par ce code. Des cases peuvent rester vides.



La correction est donnée à la fin de la série d'exercices.