Chapitre 2 : Structures algorithmiques en Python

I. Notion d'indentation

En python, la structuration des blocs d'instructions se fait grâce à l'indentation[3] (le décalage visuel avec des espaces) : les lignes consécutives qui ont la même indentation appartiennent au même bloc ; une ligne ayant une indentation moindre termine le bloc d'instructions constitué par les lignes qui la précèdent.

Une instruction composée est formée d'une instruction d'introduction terminée par le caractère deux-points « : », suivi par un bloc d'instructions simples indentées par rapport à cette instruction d'introduction.

```
instruction d'introduction :        (ligne terminée par un caractère deux-points)
    instruction 1                   (bloc d'instructions secondaire,
    instruction 2                    composé de une ou plusieurs
    instruction 3                    lignes d'instructions)
autre instruction                   (autre instruction hors du bloc, entraînant la fin du blocprécédent)
```

Illustration 6: Format d'une instruction composée en Python

Les lignes du bloc secondaire d'instructions sont alignées entre elles et décalées (indentées) par rapport à la ligne d'introduction.

→ Les principales instructions composées sont :

- L'instruction conditionnelle if ;
- L'instruction de boucle for ;
- L'instruction de boucle conditionnelle while.

II. Les conditions

II.1) Synthaxe

L'instruction if est une instruction composée. Elle signifie « si ».

3 *D'autres langages utilisent d'autres méthodes pour structurer les blocs d'instructions : en C, C++, Java, PHP… la structuration se fait grâce à des accolades {}. En Pascal, ADA… la structuration se fait grâce aux mots clés begin et end.*

Sa syntaxe est la suivante :

if condition1 :

bloc1

elif condition2 :

bloc2

elif condition3 :

bloc3

else :

bloc4

Remarque :

Les conditions de test placées entre if et « : » sont appelées *prédicats*.

II.2) Conditions simples if-else

La première ligne de l'instruction *if condition :* est appelée *clause if* et la condition est une expression booléenne évaluée à True ou False.

La ligne suivante, nous avons un bloc d'instructions. Un bloc est simplement un ensemble d'une ou de plusieurs instructions. Lorsqu'un bloc d'instructions est suivi de la clause if, Il est connu comme bloc if. Notons que chaque instruction à l'intérieur du bloc if doit être indentée du même nombre d'espaces.

Exemple :

```
n = int(input("Entrez un nombre: "))

Entrez un nombre: 13

if (n%2)==0:

    print ("C'est un nombre pair")

else:

    print ("C'est un nombre impair")

C'est un nombre impair
```

II.3) Conditions multiples if-elif-else

Comme dans les autres langages de programmation, seule la ligne *if condition1* est obligatoire.

On peut mettre autant de lignes *elif condition_n* que l'on souhaite *(entre 0 et plusieurs)*. On peut mettre au maximum une seule ligne *else*. On ne doit pas préciser de condition après le mot clé *else*.

Exemple :

Si les trois notes d'un étudiant sont : 12, 8, 14 alors on a :

- Moyenne arithmétique : (12+8+14)/3 = 34/3 =11,34 ;
- Moyenne de la mauvaise et de la meilleure : (14+8)/2=22/2=11

On choisira donc la première moyenne.

```
n1 = float(input("saisir la premiere note :"))

n2 = float(input("saisir la deuxieme note :"))

n3 = float(input("saisir la troisieme note :"))

m1 = (n1+n2+n3)/3

m2 = 0

if n1 <= n2 <= n3 or n3 <= n2 <= n1:

    m2 = (n1+n3)/2

elif n2 <= n1 <= n3 or n3 <= n1 <= n2:

    m2 = (n3+n2)/2

else:

    m2 = (n1+n2)/2



if m1 > m2:

    print("la meilleure note est : ", m1)

else:
```

```
print("la meilleure note est : ", m2)
```

III. Les boucles

En Python, on distingue 2 types de boucles : for et while.

III.1) Boucle for

La boucle for permet de répéter un bloc d'instructions un certain nombre de fois.

III.1.A) Format

Le format d'une boucle for est :

```
for compteur in séquence :
    bloc d'instructions
```

III.1.B) Exemples

III.1.B.a) Affichage d'un texte lettre par lettre

Affichage d'un texte lettre par lettre ; le compteur « cpt » permet de parcourir la chaîne « message ».

```
message="Python"
for cpt in message:
    print (cpt)
```

```
P
y
t
h
o
n
```

→ Notons que les doubles côtes ne sont pas pris en compte lors de l'affichage.

III.1.B.b) Afficher les valeurs entre deux bornes

Pour réaliser cette opération, on utilise la fonction *range()* ; cette fonction admet 3 paramètres :

- Valeur de départ (intervalle fermé) ;
- Valeur finale (intervalle ouvert) ;
- Pas (par défaut =1, dans ce cas il est facultatif).

Ainsi, le format de la fonction range est : (valeur_de_départ, valeur_d'arrivée-1, *pas*)

Exemple :

```
for i in range (1,5):
    print (i)
```

On obtient donc :

```
1
2
3
4
```

III.1.B.c) Afficher inversement les valeurs entre deux bornes

Dans ce cas, il faut réaliser deux changements :

- Inverser les bornes de la boucle ;
- Ajouter le pas *(- 1 par exemple).*

Exemple :

```
for i in range (4,0,-1):
    print (i)
```

On obtient :

```
4
3
2
```

```
1
```

III.2) Boucle while

C'est une instruction composée. Elle permet de répéter un bloc d'instructions tant qu'une condition reste vraie.

III.2.A) Format

La syntaxe est :

```
while condition :
    bloc d'instructions
```

III.2.B) Exemple

```
a=2
while a<=25:
    a+=3
    print (a)
5
8
11
14
17
20
23
26
```

→ l'affichage de la valeur 26 s'explique par l'ancienne valeur de la variable a, en fait 23 < 26 donc la variable a sera incrémentée de 3, l'affichage de la nouvelle valeur aura lieu et la boucle s'arrête puisque 26 > 25 !

III.3) Modifier l'exécution d'une boucle

Python fournit des contrôles supplémentaires pour le suivi de l'exécution d'une boucle. Les deux mots clés *break* et *continue* permettent de réaliser ce contrôle.

Ces deux commandes doivent être manipulées avec prudence, car le fait d'abuser ou de mal utiliser ces mots-clés, peut rendre les programmes difficiles à lire et à déboguer.

III.3.A) Le mot clé break

La commande *break* permet l'interruption immédiate d'une boucle.

Exemple 1 :

On interrompt l'exécution de la boucle à la deuxième itération.

```
for i in range(10):
    print("debut de l'iteration", i)
    if i == 2:
        break
    print("fin de l'iteration", i)
    print ("Suite du prgramme")
```

On obtient le résultat suivant :

```
debut de l'iteration 0
fin de l'iteration 0
Suite du prgramme
debut de l'iteration 1
fin de l'iteration 1
Suite du prgramme
debut de l'iteration 2
```

Exemple 2 :

Le code suivant interrompt l'affichage d'un texte dès l'apparition de la lettre « a ».

```
for lettre in "ordinateur":
    if lettre == "a":
        break
    print (lettre)
print("Fin")
```

Résultat :

```
o
r
d
i
n
Fin
```

III.3.B) Le mot clé continue

Contrairement à *break*, la commande *continue* termine une itération.

On reprend les même exemples de la section précédente.

Exemple 1 :

```
for i in range(4):
    print("debut de l'iteration", i)
    if i == 2:
        continue
    print("fin de l'iteration", i)
```

```
        print ("Suite du prgramme")
```

On obtient le résultat suivant :

```
debut de l'iteration 0

fin de l'iteration 0

Suite du prgramme

debut de l'iteration 1

fin de l'iteration 1

Suite du prgramme

debut de l'iteration 2

debut de l'iteration 3

fin de l'iteration 3

Suite du prgramme
```

Ainsi, le message de la fin de l'itération ne sera affiché que lorsque la valeur de i > 2.

Exemple 2 :

```
for lettre in "ordinateur":
    if lettre == "a":
        continue
    print(lettre)
print("Fin")
```

Résultat :

```
o

r

d

i

n

t

e

u

r

Fin
```

IV. Les listes

IV.1) Notion de liste

En Python, le type *list* est un type de données qui permet de former une suite ordonnée d'éléments.

Les éléments d'une même liste peuvent être des données de tous types[4].

On écrit les éléments d'une liste python entre deux crochets, séparés par des virgules :

Exemple :

```
groupe=["abc",51,12,True,"Informatique"]

print (type(groupe))

<class 'list'>
```

4 *La notion de liste en Python est identique à celle dans le langage R, ainsi contrairement au langage C/ C++ ou le Pascal, les listes, appelées tableaux, ne peuvent contenir qu'un seul type de données à la fois.*

IV.2) Opérations sur les listes

IV.2.A) Taille d'une liste

Le nombre d'éléments d'une liste s'appelle sa longueur. Elle est renvoyée par la fonction *len()*.

```
print (len (groupe))

5
```

IV.2.B) Accès

Les éléments d'une liste sont repérés par leur indice. Les indices d'une liste commencent à 0. Python détecte automatiquement l'utilisation d'index invalides et génère une erreur (*exception*).

Exemple :

```
print (groupe [3])

True

print (groupe [8])

IndexError: list index out of range
```

IV.2.C) Modification

Pour modifier un élément, il suffit de préciser son indice.

Exemple :

```
groupe[0]=3.2

print (groupe)

[3.2, 51, 12, True, 'Informatique']
```

IV.2.D) Ajout d'éléments

On peut ajouter un élément *à la fin* d'une liste grâce à la méthode *append()*.

```
groupe.append("classe")

print (groupe)

['abc', 51, 12, True, 'Informatique', 'classe']
```

IV.2.E) Suppression d'un élément

On peut supprimer un élément grâce aux méthodes *pop()* ou *remove()*.

IV.2.E.a) La méthode pop()

Par défaut, elle permet de supprimer le dernier élément d'une liste, sinon, il faut préciser l'indice de l'élément à supprimer.

```
print (groupe.pop())

classe

print (groupe.pop (0)) # Le premier élément sera supprimé

3.2

print (groupe)

[51, 12, True, 'Informatique']
```

IV.2.E.b) La méthode remove()

Cette méthode doit recevoir en paramètres *la valeur de l'élément à supprimer*.

Exemple :

```
groupe.remove (51)

print (groupe)

[12, True, 'Informatique']
```

IV.2.F) Test d'appartenance

On peut tester l'appartenance d'un élément à une liste grâce à l'opérateur *in.*

```
print (36 in groupe)

False
```

IV.2.G) Inverser les éléments d'une liste

Pour inverser les éléments d'une liste, on utilise la fonction *reverse()*.

Exemple :

```
liste = [1,2,3,4]

liste.reverse()

print (liste)

[4, 3, 2, 1]
```

IV.2.H) Itérations

Il y a deux méthodes pour afficher les valeurs d'une liste avec une boucle for :

IV.2.H.a) Itérer sur les éléments de la liste

Exemple :

```
for e in groupe:

 print (e)

12

True

Informatique
```

IV.2.H.b) Itérer sur les indices des éléments de la liste

C'est-à-dire sur une suite d'entiers : On accède à la totalité de la liste ou à un sous-ensemble.

Exemple 1 :

```
 for k in range(3): # 3 c'est la taille de la liste

 print (groupe [k])

12

True

Informatique
```

Exemple 2 :

```
liste = [5,6,7,8]

for x in range(1,3): #Accès aux éléments d'indice 1 et 2

    print (liste[x])

6

7
```