Chap 10. Python 3 : Tests & boucles

Livre p 25 Chap 2 Boucles for, p 39 Chap 3 Comparaisons et tests & p 87 Chap 6 Boucle while

1. Tests

a. Rappel sur les booléens

Le type booléen (″bool″) en Python ne comprends que deux valeurs : *True* (Vrai) et *False* (Faux)
La négation (*not*) est prioritaire sur la conjonction (*and*) qui est elle-même prioritaire sur la disjonction (*or*), mais il vaut mieux utiliser des parenthèses !
Les opérateurs sont évalués de la gauche vers la droite :

Pour « a *or* b » : si a est vraie, alors l'expression est vraie sans avoir besoin d'évaluer b.
Pour « a *and* b » : si a est faux, alors l'expression est fausse sans avoir besoin d'évaluer a.

Remarque : 0, 0.0, ″″, (), [], {} sont évalué comme *False*.

b. Condition : Si … alors

L'instruction *if* permet de tester une condition, avant de continuer les instructions suivantes.
La condition se termine par des deux-points « : », les instructions à exécuter si la condition est remplie doivent se trouver dans un bloc indenté à l'aide d'une tabulation ou d'espaces.

Exemple
```
if n != 0 :
    print(n, "est non nul")
```
ou
```
if n != 0 : print(n, "est non nul")
```

Remarque : Dans le cas d'une instruction unique à exécuter quand la condition est remplie, on peut placer cette instruction directement après les deux-points « : », sur la même ligne que la condition.

c. Condition : Si … alors … sinon

On peut ajouter, au test précédent, des instructions à exécuter si la condition n'est pas remplie avec l'instruction *else* : comme pour le cas précédent, l'instruction *else* se termine par des deux-points « : » et les instructions doivent se trouver dans un autre bloc indenté.

Exemple
```
if p%2 == 0 :
    print(p, "est pair")
else :
    print(p, "est impair")
```

d. Conditions : Si … alors … sinon-si …

On peut même inclure une autre condition à tester quand la première condition n'est pas remplie, et enchaîner ainsi les conditions avec l'instruction *elif* en gardant toujours le même système de blocs indentés.

Exemple
```
if a > 1 :
    print(a, "est supérieur à 1")
elif a < 0 :
    print(a, "est négatif")
else :
    print(a, "est entre 0 et 1")
```

e. Conditions imbriquées

On peut également inclure des conditions à l'intérieure d'une condition, il faut alors continuer à bien indenter les blocs avec plusieurs tabulations.

f. Les essais et exceptions

Plutôt que d'anticiper les erreurs et autres bugs, on peut les gérer en utilisant les instruction *try* et *except*.

Exemples :

```
a = input()
try :
    print("inverse : ", 1/float(a))
except :
    print("erreur ! ")
```

est plus efficace que :

```
a = input()
if a != "0" :
    print("inverse : ", 1/float(a))
else :
    print("erreur ! ")
```

Le premier exemple traite aussi l'erreur où la variable « a » ne contiendrait pas un nombre…
Remarque : On peut même différencier les erreurs rencontrées (*NameError*, *ZeroDivisionError*, …)
Astuce : L'instruction *pass* permet de ne rien faire dans un bloc (pratique pour un *try … except* quand on ne veut rien faire en cas d'erreur, mais fonctionne aussi avec une condition *if .. else* …)

2. Boucles inconditionnelles

Exemples
for c *in* "bonjour" :
    print(c)
ou
for i *in* (1, 2, 3) : *print*(i)

a. Parcours d'un objet

L'instruction *for* … *in* … permet de répéter les instructions situées dans un bloc indenté pendant qu'une variable décrit les valeurs prises dans une chaîne de caractères, une liste, un tuple, un ensemble ou un dictionnaire.
L'instruction *for* se termine par des deux-points « : », les instructions à répéter doivent se trouver dans un bloc indenté à l'aide d'une tabulation ou d'espaces.
Remarque : Dans le cas d'une instruction unique à répéter, on peut placer cette instruction directement après les deux-points « : », sur la même ligne que la boucle.
Remarque : Si l'objet est vide (″″, (), [], {}), aucune instruction du bloc n'est exécutée.
Attention : Ne pas modifier l'objet que l'on parcourt dans la boucle pendant le parcours, on pourrait créer des erreurs ou même une boucle infinie !

Exemple
for k *in range*(5) :
    print(k)

b. Parcours d'une liste virtuelle

Si l'on souhaite répéter n fois une suite d'instructions avec un compteur allant de 0 à n – 1, on peut utiliser une liste virtuelle avec l'instruction *range*.
Remarque : *range*(n) crée une liste virtuelle de 0 à n – 1.
range(a, b) crée une liste virtuelle de a à b – 1.

c. Compléments

On peut parcourir un objet tout en ayant un compteur en même temps avec l'instruction *enumerate*.
On peut parcourir deux objets de même taille en même temps avec l'instruction *zip*.

Exemple	**Ecrira** :
for i, k *in enumerate*("oui") : *print*(i, k)	0 o 1 u 2 i

Exemple	**Ecrira** :
for i, k *in zip*("non", [4, 5, 6]) : *print*(i, k)	n 4 o 5 n 6

d. Reprise ou sortie prématurée

On peut passer directement à l'élément suivant du parcours de la boucle en ignorant les instructions suivantes avec l'instruction *continue* mais ce n'est pas très utilisé.
On peut sortir prématurément d'une boucle inconditionnelle avant que le parcours soit terminé avec l'instruction *break* mais cela est déconseillé, il vaut mieux dans ce cas utiliser une boucle conditionnelle.
Remarque : On peut ajouter un *else* après une boucle *for* ! Le bloc indenté après le *else* ne sera exécuté que si l'on n'est pas sorti prématurément de la boucle (c'est-à-dire si on a parcouru l'objet en entier).

3. Boucles conditionnelles

Exemple
n = 10
while n**2 < 1000 :
    print(n, "² = ", n**2)
    n = n + 1

a. Boucle tant-que

L'instruction *while* permet de répéter les instructions situées dans un bloc indenté, tant qu'une certaine condition est remplie. La condition est vérifiée avant l'exécution des instructions situées dans le bloc indenté.
Attention : Il y a un risque de créer une boucle infinie si la condition n'est jamais réfutée !

b. Reprise ou sortie prématurée

Comme pour les boucles inconditionnelles, on peut passer directement à la suite avec l'instruction *continue* ou sortir prématurément avec l'instruction *break*.
Certains utilisent l'instruction *break* quand ils ont plusieurs conditions de sortie de boucle en les plaçant dans une boucle infinie « *while True* : »