

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

Informatique

8

for if while

Cours

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

for if while	3
1.I. Les variables booléennes.....	3
1.II. Implémentation d'un compteur.....	4
1.III. Boucles for-pour	4
1.III.1 Syntaxe	4
1.III.2 Remarques.....	5
1.III.3 Attention, danger : Ne pas modifier ce qui est après « in » dans le for	6
1.III.4 Modifications de la variable i	7
1.IV. Condition if – si.....	8
1.IV.1 Syntaxe	8
1.IV.2 Remarques	8
1.V. Boucle while – tant que.....	10
1.V.1 Syntaxe	10
1.V.2 Remarques	10
1.V.2.a Modification de la condition	10
1.V.2.b for ou while.....	10
1.V.2.c Terminaison	11
1.V.2.d Ordre des conditions	11
1.VI. Exécution de boucles directement dans la console « Shell »	12
1.VII. Remarques sur la syntaxe	12
1.VIII. Le for/else ou while/else.....	13

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

for if while

1.1. Les variables booléennes

L'utilisation des boucles/conditions va nous conduire à devoir effectuer des tests pour obtenir des conditions.

Pour cela, introduisons les variables de type booléennes. Elles valent 1, ou 0, ou plutôt, True (juste) ou False (faux).

Pour obtenir une variable booléenne, il faut effectuer un test :

==	Vérifie l'égalité
!=	Vérifie la différence
>	Comparaison
>=	
<	
<=	
a <= b < c	On peut écrire des inégalités à plusieurs comparaisons
in	'A' in ['A','B'] renvoie True

Les résultats de ces tests sont soit « True », soit « False »

On peut effectuer des opérations entre variables booléennes :

and	Fonction « et »
or	Fonction « ou »
not(Cond)	Donne la condition inverse : not(True)->False not(False)->True

Supposons que True vaut 1 et False vaut 0, on a alors :

and	0	1
0	0	0
1	0	1

or	0	1
0	0	1
1	1	1

Lorsque l'on réalise des opérations booléennes, l'ordre des conditions a de l'importance. Quelques exemples :

False and 1/0	False
1/0 and False	division by zero
True or 1/0	True
1/0 or True	division by zero

On remarque que si le résultat est forcément connu par ce qui a été évalué, les conditions restantes ne seront pas évaluées (ce sera utile dans les while, on en reparlera). Ainsi, **bien retenir que l'ordre des conditions a de l'importance !**

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.II. Implémentation d'un compteur

L'utilisation de boucles va souvent être associée à l'utilisation d'un compteur.

Ce compteur sera toujours initialisé avant la boucle : Compteur = 0

Puis incrémenté, par exemple de 1, à chaque itération, en écrivant au choix :

- Compteur = Compteur + 1
- Compteur += 1

1.III. Boucles for-pour

Le principe est de réaliser une ou plusieurs opérations pour une variable allant d'une valeur à une autre.

1.III.1 Syntaxe

La syntaxe est la suivante :

```
for i in range(n):
    opérations à effectuer
    opérations à effectuer
    opérations à effectuer
    opérations à effectuer
    opérations à effectuer
```

Attention
i varie du séparateur 0 au séparateur n,
soit dans la plage 0, n-1
Il y a bien n étapes

La fin de cette boucle est prévue d'avance !

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.III.2 Remarques

Ecrire « `for i in range(0):` » conduit à la non-exécution du code dans le for !!!

La variable `i` est créée par la boucle, ses valeurs sont imposées par le for et augmentent toutes seules. Inutile donc de l'initialiser avant la boucle, elle sera remplacée :

Code	Résultat
<pre>i = 10 for i in range(5): print(i)</pre>	<pre>0 1 2 3 4</pre>

Il est inutile de l'augmenter :

Code	Résultat
<pre>for i in range(5): print(i) i += 5</pre>	<pre>0 1 2 3 4</pre>

On peut stopper une boucle for avec la commande « `break` » :

Code	Résultat
<pre>for i in range(5): print(i) if i==3: break</pre>	<pre>0 1 2 3</pre>

Si plusieurs boucles sont imbriquées, le `break` met fin à celle dans laquelle il est uniquement.

Enfin, on peut parcourir des termes d'une liste avec « `for t in L` »

Code 1	Code 2
<pre>L = [3,7,9] for i in range(len(L)): terme = L[i] print(terme)</pre>	<pre>L = [3,7,9] for terme in L: print(terme)</pre>

Je vous recommande de ne pas écrire `for i in L` afin de ne pas confondre `i` avec un indice...

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.III.3 Attention, danger : Ne pas modifier ce qui est après « in » dans le for

Quand on utilise « `for t in L:` »... Si on modifie `L` dans la boucle, il y aura des erreurs très compliquées à identifier. Exemple où l'on enlève tous les termes d'une liste :

Code	Résultat
<pre>A = [0,1,2,3,4,5,6,7,8,9,10] for i in range(len(A)): A.remove(i) print("A la fin: A = ",A)</pre>	A la fin: A = []
<pre>A = [0,1,2,3,4,5,6,7,8,9,10] for t in A: A.remove(t) print("A la fin: A = ",A)</pre>	A la fin: A = [1, 3, 5, 7, 9]

On peut remarquer le comportement insoupçonné du second code !

- Ecrire « `for i in range(len(A)):` » fait varier `i` de 0 à la longueur initiale de `A`. On va donc bien demander d'enlever les termes de 0 à 10 dans `A`.
- Ecrire « `for t in A:` » incrémente de 1 un indice caché `j` à chaque itération (`j=j+1`). `t` prend la valeur `A[j]` jusqu'à ce que `j+1` dépasse la taille de `A` modifiée aux itérations précédentes

Quand on utilise « `i in range(len(L))` »... Si on modifie `L` dans la boucle et que l'on appelle un terme de `L` ayant disparu :

Code	Résultat
<pre>L = [1,2,3,4,5,6,7,8,9,10] for i in range(len(L)): L.remove(L[i]) print(L)</pre>	<pre>>>> (executing file "<tmp 1>") [2, 3, 4, 5, 6, 7, 8, 9, 10] [2, 4, 5, 6, 7, 8, 9, 10] [2, 4, 6, 7, 8, 9, 10] [2, 4, 6, 8, 9, 10] [2, 4, 6, 8, 10] Traceback (most recent call last): File "<tmp 1>", line 4, in <module> L.remove(L[i]) IndexError: list index out of range</pre>

On remarque :

- La suppression de termes non successifs...
- L'erreur dès que `i` dépasse la longueur de la liste

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.III.4 Modifications de la variable i

Voyons le comportement de python lorsque l'on modifie la variable i dans la boucle :

<pre>n = 5 for i in range(n): print('Avant: ',i) i = i+2 print('Après: ',i)</pre>	<pre>Avant: 0 Après: 2 Avant: 1 Après: 3 Avant: 2 Après: 4 Avant: 3 Après: 5 Avant: 4 Après: 6</pre>
---	--

On peut remarquer que même si i est modifié dans la boucle, ses valeurs successives sont bien pilotées par le for. Autrement dit, même si i est modifié, la prochaine valeur de i est la valeur de i au début de chaque étape, incrémenté de 1.

Lorsque l'on utilise deux boucles for avec la même variable i, il se passe donc la même chose :

<pre>for i in range(5): print('i1: ',i) for i in range(2): print('i2: ',i)</pre>	<pre>i1: 0 i2: 0 i2: 1 i1: 1 i2: 0 i2: 1 i1: 2 i2: 0 i2: 1 i1: 3 i2: 0 i2: 1 i1: 4 i2: 0 i2: 1</pre>
--	--

Le fait d'utiliser deux fois la lettre i ne pose pas de problèmes à l'exécution du code comme si l'on avait utilisé i et j. Comme précédemment, la modification de i dans la seconde boucle for ne modifie pas l'enchaînement des i de la première boucle.

Lorsque l'on utilise plusieurs boucles for :

- Si elles sont imbriquées, changer le nom de la variable pour chaque boucle : i, puis j, puis k...
- Si elles sont indépendantes, garder i pour chaque boucle

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.IV. Condition if – si

1.IV.1 Syntaxe

Le principe est de réaliser une ou plusieurs opérations si une condition est vraie, et éventuellement d'autres opérations si la condition est fausse, voire même différente.

La syntaxe est la suivante :

```

if Condition_1 :
    opérations à effectuer si Condition_1 vaut True
    opérations à effectuer si Condition_1 vaut True
elif Condition_2 : # Optionnel
    opérations à effectuer si Condition_2 vaut True
    opérations à effectuer si Condition_2 vaut True
else : # Optionnel
    opérations à effectuer si ni Condition_1, ni Condition_2 ne sont
vérifiées
    opérations à effectuer si ni Condition_1, ni Condition_2 ne sont
vérifiées

# Suite du programme

```

1.IV.2 Remarques

- « elif » pour « else if » veut dire « sinon ».
- Ceci n'est pas une boucle, elle est exécutée puis le programme continue.
- Dès qu'une conditions est vérifiée, les opérations associées sont exécutées et on sort du bloc if. Autrement dit :

```

if 1 < 0:
    print("Oui")
elif 1 > 0:
    print("Non")
elif 1 == 1:
    print(1)

```

Non

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

Attention, lorsque vous n'exécutez rien, python n'est pas content. Exemple :

<pre>L = [2,10,9,3,5,6,1,7] LL = [] for i in range(len(L)): Terme = L[i] if Terme > 5: # Ne rien faire else: LL.append(Terme) print(LL)</pre>	<pre>>>> (executing file "<tmp 1>") File "<tmp 1>", line 8 else: ^ IndentationError: expected an indented block</pre>
---	--

Il faut donc mettre quelque chose d'inutile pour que ça passe :

<pre>L = [2,10,9,3,5,6,1,7] LL = [] for i in range(len(L)): Terme = L[i] if Terme > 5: a = 0 # Ne rien faire else: LL.append(Terme) print(LL)</pre>

Ou encore, on peut utiliser l'option « `continue` » :

<pre>L = [2,10,9,3,5,6,1,7] LL = [] for i in range(len(L)): Terme = L[i] if Terme > 5: continue else: LL.append(Terme) print(LL)</pre>
--

Toutefois, lorsque l'on ne veut que le programme n'exécute rien à une condition, il est préférable de transformer la condition en son opposé et de n'exécuter que ce que l'on veut. On pourra donc ne pas mettre de « else » et le programme fonctionnera parfaitement. On préférera donc écrire :

<pre>L = [2,10,9,3,5,6,1,7] for i in range(len(L)): Terme = L[i] LL = [] if Terme <= 5: LL.append(Terme) print(LL)</pre>
--

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.V. Boucle while – tant que

Le principe est de réaliser une ou plusieurs opérations tant qu’une condition est vraie.

1.V.1 Syntaxe

Le while est utilisé lorsque l’on ne connaît pas à priori le nombre d’itérations à effectuer (simulation numérique jusqu’à une valeur d’un paramètre, recherche dans une liste jusqu’à obtenir l’élément voulu etc). Sinon, on privilégie la boucle for ! **Il sera attendu de vous de bien choisir entre for et while**

La syntaxe est la suivante :

```
Initialisation de la condition
while Condition :
    opérations à effectuer
    opérations à effectuer
    opérations à effectuer
    opérations à effectuer
    opérations à effectuer
    Modification de la condition

# Suite du programme
```

1.V.2 Remarques

1.V.2.a Modification de la condition

Lorsque l’on rentre dans cette boucle, la condition est vérifiée. Si on ne modifie pas cette condition dans la boucle, le programme y restera indéfiniment ! Pour en sortir, casser le programme avec la commande « Ctrl+C ». Il faudra donc que la condition soit modifiée lors de la lecture de la boucle jusqu’à ce qu’elle corresponde à ce qui est attendu.

1.V.2.b for ou while

Les deux codes suivants sont identiques, mais on n’acceptera pas celui avec un while puisque l’on connaît à l’avance le nombre d’itérations :

```
L = [1,5,3,2,5,9,2]
Taille = len(L)
for i in range(Taille):
    Terme = L[i]
    print(Terme)
```

```
L = [1,5,3,2,5,9,2]
Taille = len(L)
i = 0
while i < Taille:
    Terme = L[i]
    print(Terme)
    i += 1
```

Ce n’est certes pas très grave, mais faite attention

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.V.2.c Terminaison

Le while est sujet à l'étude de la terminaison des algorithmes, il peut en effet ne jamais se terminer :

Avec fin	Sans fin
<pre>a = 100 while a != 1: a = a/10 print(a)</pre>	<pre>a = 99 while a != 1: a = a/10 print(a)</pre>

Le code est stoppé lorsque la condition devient fausse :

Code	Résultat
<pre>i = 5 while i != 0: i -= 1 print(i)</pre>	<pre>4 3 2 1 0</pre>

Ainsi, quand i=0, il ne rentre pas dans la boucle !

On peut faire une boucle infinie en écrivant « `while True :` », et on peut stopper une boucle while avec la commande « `break` » :

Code	Résultat
<pre>i = 0 while True: i += 1 print(i) if i==3: break</pre>	<pre>1 2 3</pre>

Si plusieurs boucles sont imbriquées, le `break` met fin à celle dans laquelle il est uniquement.

1.V.2.d Ordre des conditions

Regardez attentivement les codes suivants, vous verrez que ce sont pratiquement les mêmes

<pre>L = [0,1,2,3] i = 0 while L[i]==i and i<len(L): i += 1</pre>	<p>Ce premier code ne fonctionne pas ! En effet, 4 fois de suite, les conditions « <code>L[i]==i</code> » et « <code>i<len(L)</code> » sont vérifiées. On arrive à i=4, et là il y a un problème car <code>L[4]</code> n'existe pas</p> <p>« list index out of range »</p>
<pre>L = [0,1,2,3] i = 0 while i<len(L) and L[i]==i: i += 1</pre>	<p>Si l'on inverse les tests, le code fonctionne. On remarque donc que les conditions sont lues dans un ordre. Sans même évaluer « <code>L[i]==i</code> », le code sait que la première condition « <code>i<len(L)</code> » n'étant pas satisfaite, le résultat sera false. Tant mieux !</p>

Conclusions : comme vu en introduisant les opérations booléennes, attention à l'ordre des conditions dans un while

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.VI. Exécution de boucles directement dans la console « Shell »

Il est possible d'exécuter des boucles dans la console. Il suffit :

- Soit de la copier dans le code et de la coller dans la console puis d'appuyer 2 fois sur « Entrée ».
- Soit d'écrire la première ligne (ex : `for i in range(10):`) puis de valider avec « Entrée ». Puis on écrit les lignes de code les une après les autres en validant à chaque fois avec « Entrée ». Pour l'exécuter, appuyer 2 fois sur « Entrée », ou plutôt, valider deux lignes vides

1.VII. Remarques sur la syntaxe

Il est obligatoire de respecter la présence des « : » d'une part, et « l'indentation » d'autre part. En effet, c'est ce décalage des instructions qui va permettre de dire si l'instruction est dans la boucle ou en dehors.

Indenter d'un étage correspond à l'appui une fois sur la touche Tabulation, ou 4 fois la barre espace. Lorsque l'indentation est mauvaise dans les boucles et conditions, on peut le voir en surlignant le code :

<pre> 1 n = 10 2 Res = 0 3 for i in range(n): 4 Res = Res + i 5 print(Res) </pre>	<p>Ligne 4, avant Res, on voit un trait vertical blanc qui correspond à l'indentation qu'il devrait y avoir, on voit qu'il y a un espace en trop ici. On pourra aussi remarque que le print est décalé d'un espace en trop ligne 5.</p>
--	---

Dans d'autres langages de programmation, l'indentation peut ne pas avoir d'effet, mais dans ce cas, la boucle contient un terme, par exemple :

<pre> if a == 1 then print('Fin') endif </pre>	<pre> if a == 1 then print('Fin') ifend </pre>
--	--

Dernière mise à jour	Informatique	Denis DEFAUCHY
19/11/2021	8 – for if while	Résumé

1.VIII. Le for/else ou while/else

Il est possible d'écrire les codes suivants :

<pre>for i in range(10): if i==10: break else: print("Pas de solution")</pre>	On passe par là
<pre>while i<10: if i==10: break i += 1 else: print("Pas de solution")</pre>	

Le principe est le suivant :

- Si aucun **break** n'est lu et si la boucle **for** ou le **while** se terminent, ce qu'il y a dans le **else** sera exécuté.
- Si un **break** est lu, ce qui est dans le **else** ne sera pas exécuté

Un petit exemple où c'est pratique : insertion d'un terme à la bonne place dans une liste triée :

Avec else	Sans else
<pre>L = [1,2,3,4,5,6,7] t = 8 for i in range(len(L)): if t<L[i]: L.insert(i,t) break else: L.append(t) print(L)</pre>	<pre>L = [1,2,3,4,5,6,7] t = 8 Inser = 0 for i in range(len(L)): if t<L[i]: L.insert(i,t) Inser = 1 break if Inser == 0: L.append(t) print(L)</pre>