

Boucles et itérations

Initiation à l'algorithmique et à la programmation

Semaine 4 – INALCO

Revekka Kyriakoglou

Objectifs de la séance

- Expliquer une itération et suivre les valeurs des variables.
- Parcourir un texte avec `for`.
- Utiliser `range()` pour parcourir des entiers.
- Écrire une boucle `while` qui se termine.
- Utiliser un compteur ou un accumulateur dans une fonction.

Déroulement : 3 heures

0–15 min	Rappels et motivation
15–45 min	Parcourir un texte avec <code>for</code>
45–70 min	Compteurs, conditions et fonctions
70–80 min	Pause
80–105 min	Répétitions avec <code>range()</code>
105–140 min	Boucle <code>while</code> et arrêt
140–170 min	Exercices et débogage
170–180 min	Vérification des acquis et bilan

Rappel : fonctions et chaînes

Prévoir la valeur affichée.

```
1 def nettoyer_texte(texte):  
2     return texte.strip().lower()  
3  
4 mot = nettoyer_texte("  CHAT  ")  
5 print(mot)  
6 print(len(mot))
```

Résultats : chat et 4.

Une fonction renvoie une valeur que le programme peut réutiliser.

Répéter des instructions

Comment afficher chaque caractère d'un mot ?

```
1 mot = "chat"
2 print(mot[0])
3 print(mot[1])
4 print(mot[2])
5 print(mot[3])
```

Ce code dépend de la longueur du mot.

Comment traiter aussi "bonjour" ou une chaîne vide ?

Boucle et itération

- Une boucle répète un bloc d'instructions.
- Une itération correspond à une exécution de ce bloc.
- `for` parcourt les éléments d'un objet parcourable, par exemple les caractères d'une chaîne.
- `while` répète un bloc tant qu'une condition est vraie.

Nous commencerons par parcourir les caractères d'un texte.

Parcourir une chaîne

À chaque tour, la variable reçoit le caractère suivant.

```
1 mot = "chat"
2
3 for caractere in mot:
4     print(caractere)
5
6 print("Fin")
```

Affichage : c, h, a, t, puis Fin, chacun sur une ligne.

La syntaxe de for

Le nom de la variable de parcours est libre.

```
1 for caractere in "chat":  
2     print(caractere)
```

- `for` introduit la boucle.
- `in` indique l'objet à parcourir.
- Le deux-points et l'indentation délimitent le bloc.
- La boucle s'arrête lorsque le parcours est terminé.

Suivre le parcours

Pour le mot "chat" :

Itération	caractere	Affichage
1	"c"	c
2	"h"	h
3	"a"	a
4	"t"	t

La variable contient un caractère, pas son indice.

On n'a pas besoin de calculer la longueur pour ce parcours.

L'indentation change les répétitions

Comparer les deux programmes.

```
1 for caractere in "oui":
2     print(caractere)
3     print("Tour")
4
5 print("---")
6
7 for caractere in "oui":
8     print(caractere)
9 print("Fin")
```

Tour s'affiche trois fois. Fin s'affiche une seule fois, après la seconde boucle.

Un parcours peut être vide

Combien de fois le corps est-il exécuté ?

```
1  texte = ""
2
3  for caractere in texte:
4      print(caractere)
5
6  print("Fin")
```

Zéro itération. Le programme affiche seulement `Fin`.

Ne supposez pas qu'une boucle s'exécute au moins une fois.

Exercice 1 : lecture de code

Prévoir chaque ligne affichée, puis vérifier (5 min).

```
1 texte = "a b"
2 for caractere in texte:
3     print "[" + caractere + "]"
4 print("Termine")
```

- Combien d'itérations ?
- L'espace fait-il partie du parcours ?
- Que se passe-t-il avec un texte vide ?

Un compteur de caractères

Un compteur mémorise le nombre d'éléments rencontrés.

```
1 nombre = 0
2
3 for caractere in "chat":
4     nombre += 1
5
6 print(nombre)
```

`nombre += 1` signifie `nombre = nombre + 1`.

Le résultat est 4. Ici, on reproduit le résultat de `len()` pour comprendre le mécanisme.

Une condition dans une boucle

Compter uniquement les caractères égaux à "a".

```
1 nombre = 0
2
3 for caractere in "banane":
4     if caractere == "a":
5         nombre += 1
6
7 print(nombre)
```

Six itérations, mais seulement deux augmentations du compteur.

Le `if` est dans la boucle. L'incrément est dans le `if`.

Trace du compteur

Le compteur commence à zéro.

Tour	Caractère	Égal à "a" ?	Compteur après le tour
1	b	Faux	0
2	a	Vrai	1
3	n	Faux	1
4	a	Vrai	2
5	n	Faux	2
6	e	Faux	2

Nombre de tours et nombre de caractères retenus sont deux quantités différentes.

Une fonction de comptage

La boucle traite le texte. La fonction renvoie le total.

```
1 def compter_lettre(texte, lettre):  
2     """Compte les occurrences d'un caractere."""  
3     nombre = 0  
4     for caractere in texte:  
5         if caractere == lettre:  
6             nombre += 1  
7     return nombre  
8  
9 print(compter_lettre("banane", "a"))
```

On fournit un seul caractère comme second argument.
return est après la boucle : il renvoie le compte final.

Exercice 2 : tester une fonction

Prévoir les résultats, puis les vérifier (6 min).

```
1 print(compter_lettre("banane", "a"))
2 print(compter_lettre("banane", "z"))
3 print(compter_lettre("", "a"))
4 print(compter_lettre("Banane", "b"))
```

Comment adapter la fonction pour ignorer la différence entre majuscules et minuscules ?

Utiliser une méthode vue en semaine 3.

Un accumulateur de texte

Un accumulateur conserve un résultat construit progressivement.

```
1 resultat = ""
2
3 for caractere in "chat":
4     resultat = resultat + caractere.upper()
5
6 print(resultat)
```

Valeurs successives : "C", "CH", "CHA", "CHAT".

La chaîne initiale est vide. Chaque affectation conserve un nouveau résultat.

Exercice 3 : retirer les espaces

Écrire une fonction sans utiliser `replace` (8 min).

```
1 def retirer_espaces(texte):  
2     resultat = ""  
3     for caractere in texte:  
4         if ...:  
5             resultat = ...  
6     return resultat
```

Conserver chaque caractère différent de l'espace ordinaire "
".

Tests : "un chat" donne "unchat".

" " et "" donnent une chaîne vide.

Pause

10 minutes

À la reprise : répéter une opération avec une suite d'entiers.

Répéter cinq fois

range fournit ici cinq entiers successifs.

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

Affichage : 0, 1, 2, 3, 4, chacun sur une ligne.
La borne 5 est exclue.

Les formes de range

Les bornes et le pas déterminent les entiers parcourus.

```
1 range(5)           # 0, 1, 2, 3, 4
2 range(2, 6)        # 2, 3, 4, 5
3 range(2, 11, 2)     # 2, 4, 6, 8, 10
4 range(5, 0, -1)     # 5, 4, 3, 2, 1
```

- `range(fin)` commence à zéro, avec un pas de 1.
- `range(debut, fin, pas)` exclut toujours la borne finale.
- Un pas négatif permet de descendre. Le pas ne peut pas être zéro.

Une suite d'entiers peut être vide

La direction du pas doit permettre d'atteindre les valeurs.

```
1 for i in range(5, 0):  
2     print(i)  
3  
4 print("Fin")
```

Le pas vaut 1 par défaut. Ici, la suite est vide.

Le programme affiche seulement `Fin`.

`range(0)` et `range(3, 3)` sont également vides.

Exercice 4 : valeurs et nombre de tours

Pour chaque boucle, noter les valeurs et le nombre d'itérations (5 min).

```
1 for i in range(1, 6):  
2     print(i)  
3  
4 for i in range(2, 11, 2):  
5     print(i)  
6  
7 for i in range(3, 0, -1):  
8     print(i)
```

Répéter un message

La variable de parcours peut ne pas servir dans le corps.

```
1 for i in range(5):  
2     print("Bonjour")
```

Le programme affiche cinq fois `Bonjour`.
Quelle borne choisir pour l'afficher dix fois ?

Un accumulateur numérique

Initialiser le total avant la boucle.

```
1 total = 0
2
3 for nombre in range(1, 6):
4     total += nombre
5
6 print(total)
```

Totaux successifs : 1, 3, 6, 10, 15.

Le total final vaut $1 + 2 + 3 + 4 + 5 = 15$.

Parcourir les indices d'un mot

Lorsque l'on a besoin de la position et du caractère :

```
1 mot = "chat"
2
3 for indice in range(len(mot)) :
4     print(indice, mot[indice])
```

Affichage : 0 c, 1 h, 2 a, 3 t.

Si seule la valeur du caractère est utile, le parcours direct est plus simple.

Répéter tant qu'une condition est vraie

La condition est testée avant chaque exécution du corps.

```
1 i = 1
2
3 while i <= 3:
4     print(i)
5     i += 1
6
7 print("Fin")
```

Initialisation : `i = 1`. Condition : `i <= 3`.

Mise à jour : `i += 1`.

Affichage : 1, 2, 3, puis `Fin`, sur des lignes séparées.

Trace d'une boucle while

La dernière vérification ne déclenche aucune itération.

i avant le test	$i \leq 3$	Action
1	Vrai	Affiche 1, puis i devient 2
2	Vrai	Affiche 2, puis i devient 3
3	Vrai	Affiche 3, puis i devient 4
4	Faux	Sort de la boucle

La valeur finale de i est 4.

Le test est évalué quatre fois, le corps s'exécute trois fois.

Une condition fausse dès le début

Prévoir le résultat.

```
1 i = 5
2
3 while i < 3:
4     print(i)
5     i += 1
6
7 print("Fin", i)
```

Zéro itération. Le programme affiche `Fin 5`.

Comme `for`, `while` peut ne jamais exécuter son corps.

Exercice 5 : suivre une boucle

Prévoir l'affichage et la valeur finale de x (5 min).

```
1 x = 10
2
3 while x > 6:
4     print(x)
5     x -= 1
6
7 print("Fin", x)
```

Combien d'itérations ? Combien d'évaluations de la condition ?

Une boucle infinie

Lire ce programme avant de l'exécuter.

```
1 i = 1
2
3 while i <= 5:
4     print(i)
```

La condition reste vraie : la valeur de i ne change pas.

Ajouter une mise à jour qui permette l'arrêt.

Si une boucle reste active : utiliser le bouton d'arrêt de l'environnement, ou `Ctrl+C` dans un terminal.

Saisir un texte non vide

Le nombre de tentatives dépend de l'utilisateur.

```
1 texte = input("Un mot : ").strip()
2
3 while texte == "":
4     print("La saisie est vide.")
5     texte = input("Reessayez : ").strip()
6
7 print("Mot saisi :", texte)
```

Chaque nouvelle saisie met à jour `texte`.

`strip()` permet aussi de détecter une saisie composée seulement d'espaces.

Une saisie répétée jusqu'à un mot d'arrêt

Le mot "fin" termine le programme.

```
1 mot = input("Un mot (fin pour arreter) : ")
2 mot = mot.strip().lower()
3
4 while mot != "fin":
5     print("Longueur :", len(mot))
6     mot = input("Un autre mot : ").strip().lower()
7
8 print("Termine")
```

Le mot d'arrêt ne reçoit pas de traitement.

Si l'utilisateur saisit `fin` dès le début, il y a zéro itération.

Exercice 6 : compter les saisies

Modifier le programme précédent (7 min).

- Initialiser un compteur avant la boucle.
- L'augmenter une fois pour chaque saisie traitée.
- Afficher le nombre de saisies après la boucle.
- Tester `chat`, `oui`, `fin` : résultat attendu 2.
- Tester `fin` dès le début : résultat attendu 0.

Dans cet exercice, une saisie vide compte aussi comme une saisie traitée.

Choisir entre for et while

Situation	Choix naturel
Parcourir les caractères d'un texte	<code>for</code>
Répéter dix fois une opération	<code>for avec range()</code>
Demander une saisie jusqu'à validation	<code>while condition</code>
Continuer jusqu'à un mot d'arrêt	<code>while condition</code>
Ces choix facilitent l'écriture. Certains problèmes admettent les deux boucles.	

Même affichage, état final différent

Les deux boucles affichent les entiers de 1 à 5.

```
1 i = 1
2 while i <= 5:
3     print(i)
4     i += 1
5 print("Après while :", i)
6
7 for i in range(1, 6):
8     print(i)
9 print("Après for :", i)
```

Après while, i vaut 6. Après for, il vaut 5.

Débogage : une mise à jour mal placée

Le programme doit afficher 1, 2 et 3. Où est l'erreur ?

```
1 i = 1
2 while i <= 3:
3     print(i)
4 i += 1
```

La mise à jour est en dehors de la boucle.

La boucle reste infinie : l'exécution n'atteint jamais `i += 1`.

Corriger l'indentation avant d'exécuter.

Débogage : un compteur réinitialisé

On souhaite compter les trois caractères.

```
1 for caractere in "oui":  
2     nombre = 0  
3     nombre += 1  
4  
5 print(nombre)
```

Le programme affiche 1.

Placer `nombre = 0` avant la boucle.

Ce changement permet aussi de traiter correctement un texte vide.

Débogage : return dans la boucle

Cette fonction doit compter tous les "a".

```
1 def compter_a(texte):  
2     nombre = 0  
3     for caractere in texte:  
4         if caractere == "a":  
5             nombre += 1  
6         return nombre
```

Que renvoie-t-elle pour "banane" ? Et pour "" ?

return termine la fonction dès le premier tour.

Pour renvoyer le total, le placer après la boucle.

Le test d'appartenance

Dans une condition, `in` teste la présence dans une chaîne.

```
1 print("a" in "aeiouy")    # True
2 print("b" in "aeiouy")    # False
3
4 for caractere in "chat":
5     if caractere in "aeiouy":
6         print(caractere)
```

Dans la boucle, `in` indique ce que l'on parcourt.

Dans la condition, `in` produit un booléen.

Ici, chaque caractère a une longueur de 1.

Exercice 7 : compter les voyelles

Écrire une fonction `compter_voyelles(texte)` (12 min).

- Mettre le texte en minuscules.
- Parcourir ses caractères.
- Augmenter un compteur pour chaque voyelle.
- Renvoyer le compteur après le parcours.

Pour cet exercice, on compte uniquement a, e, i, o, u, y.

Une condition possible : `caractere in "aeiouy"`.

Tests : "Bonjour" donne 3, "CHAT" donne 1, "" donne 0.

Vérification des acquis

Sans recopier un exemple (5 min).

- Écrire une fonction `compter_points(texte)` qui compte les ".".
- Prévoir son résultat pour "Bonjour. Au revoir." et "".
- Expliquer pourquoi `return` vient après la boucle.
- Choisir une boucle pour redemander un mot tant que la saisie est vide.

À retenir

- `for` parcourt des valeurs. `range()` fournit des entiers.
- La borne finale de `range()` est exclue.
- `while` teste sa condition avant chaque tour.
- L'indentation détermine les instructions répétées.
- Initialiser compteurs et accumulateurs avant le parcours.
- Placer `return` après la boucle pour renvoyer le résultat final.
- Tester aussi zéro itération et expliquer ce qui permet l'arrêt.

Boucles imbriquées : approfondissement

Chaque tour de la boucle externe déclenche un parcours interne.

```
1 for i in range(3):
2     for j in range(2):
3         print(i, j)
```

Affichage : 0 0, 0 1, 1 0, 1 1, 2 0, 2 1.

L'instruction `print()` s'exécute $3 \times 2 = 6$ fois.

Un rectangle de caractères

end indique ce qui suit la valeur affichée.

```
1 for ligne in range(3):  
2     for colonne in range(5):  
3         print("*", end="")  
4     print()
```

end="" évite le retour à la ligne après chaque étoile.

print() seul termine chaque ligne du rectangle.

Résultat : trois lignes contenant chacune cinq étoiles.

Corrigés : exercices 1 et 2

À consulter après la recherche. **Exercice 1**

`[a]`, `[ ]`, `[b]`, puis `Termine`.

Trois itérations, y compris pour l'espace. Avec une chaîne vide :
seulement `Termine`.

Exercice 2

2, 0, 0, 0.

Pour ignorer la casse, comparer chaque caractère de
`texte.lower()` à `lettre.lower()`.

Corrigé : retirer les espaces

Le résultat se construit dans la boucle.

```
1 def retirer_espaces(texte):  
2     """Retire les espaces ordinaires du texte."""  
3     resultat = ""  
4     for caractere in texte:  
5         if caractere != " "  
6             resultat += caractere  
7     return resultat  
8  
9 print(retirer_espaces("un chat"))
```

Corrigés : exercices 4 et 5

Valeurs parcourues et arrêt. **Exercice 4**

1, 2, 3, 4, 5 : cinq itérations.

2, 4, 6, 8, 10 : cinq itérations.

3, 2, 1 : trois itérations.

Exercice 5

Affiche 10, 9, 8, 7, puis Fin 6.

Quatre itérations et cinq évaluations de la condition.

Corrigé : compter les saisies

Le mot d'arrêt ne compte pas.

```
1 nombre = 0
2 mot = input("Un mot : ").strip().lower()
3
4 while mot != "fin":
5     nombre += 1
6     print("Longueur :", len(mot))
7     mot = input("Un autre mot : ").strip().lower()
8
9 print("Nombre de saisies :", nombre)
```

Corrigé : compter les voyelles

La convention de cet exercice exclut les voyelles accentuées.

```
1 def compter_voyelles(texte):  
2     """Compte les caracteres a, e, i, o, u, y."""  
3     nombre = 0  
4     for caractere in texte.lower():  
5         if caractere in "aeiouy":  
6             nombre += 1  
7     return nombre  
8  
9 print(compter_voyelles("Bonjour")) # 3  
10 print(compter_voyelles("CHAT")) # 1  
11 print(compter_voyelles("")) # 0
```

Corrigé : vérification des acquis

Un compteur avec une condition.

```
1 def compter_points(texte):  
2     """Compte les points dans le texte."""  
3     nombre = 0  
4     for caractere in texte:  
5         if caractere == ".":  
6             nombre += 1  
7     return nombre
```

Les deux tests donnent 2 et 0.

Après la boucle, `return` renvoie le compte complet.

Pour redemander une saisie vide, le choix naturel est `while`.