Les structures itératives (boucles)

Table des matières

Contexte

Durée : 2h

Environnement de travail : Repl.it

Pré-requis : Aucun

[cf. k3PI73J4]

Il est courant qu'un algorithme répète plusieurs fois la même opération, par exemple sur toute une liste de données. Les boucles permettent d'automatiser cette répétition, et sont de ce fait des structures très utilisées en programmation. Il en existe plusieurs types, avec certaines variations en fonction des langages. Elles sont indispensables pour effectuer des actions plusieurs fois.

Ce module a pour objectif de présenter les boucles et leurs utilisations propres. Il abordera plus particulièrement le principe de l'itération, les boucles « `for` » et les boucles « `while` », ainsi que l'imbrication des boucles.

Principe de l'itération

[cf. 90HfzQ1o]

Objectif

- Comprendre l'utilité de l'itération.

Mise en situation

Lorsque l'on répète plusieurs fois les mêmes actions, on parle d'itération. Une structure d'itération est une structure, dans le code, qui permet de rejouer les mêmes actions, avec d'éventuelles petites différences. Par exemple appliquer une même séquence d'actions à une variable différente à chaque itération. Il existe plusieurs types de structures itératives, mais elles sont généralement communes entre les différents langages.

Itération *Définition*

Une **itération** représente l'exécution d'un bloc d'instructions. Un ensemble d'**itérations** représente donc l'exécution multiple d'un même bloc d'instructions : on dit qu'on **itère sur un bloc d'instructions**.

Structure itérative

Les structures itératives fournissent un moyen d'effectuer des **boucles** sur des instructions : la boucle permet d'exécuter des **itérations**.

Condition de sortie

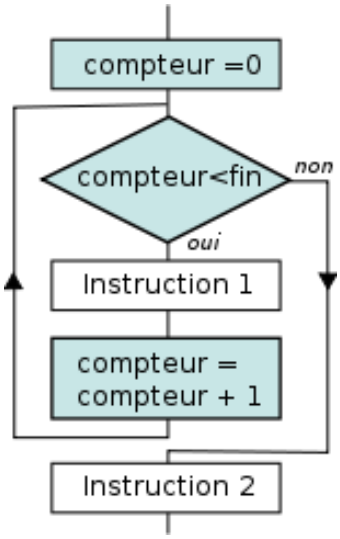
Une boucle s'exécute un certain nombre de fois avant de s'interrompre et que la suite du programme poursuive son exécution. Si une boucle ne s'interrompt jamais, c'est une **boucle infinie** : le programme reste bloqué car la boucle se répète indéfiniment.

Les structures itératives nécessitent donc une **condition de sortie**, c'est-à-dire une condition qui interrompt les itérations dès qu'elle est remplie.

Compteur

Un **compteur** est souvent utilisé à l'intérieur de la boucle : une variable entière, généralement initialisée à 0, est incrémentée à chaque nouvelle itération. Le compteur permet ainsi simplement de compter le nombre d'itérations déjà effectué.

La valeur du compteur est très souvent utilisée dans la **condition de sortie**, pour interrompre la boucle au bout d'un certain nombre d'itérations.



Itération avec un compteur

À retenir

Les structures itératives permettent de répéter plusieurs fois une même suite d'instructions grâce à une boucle.

[cf. Copie d'écran du jeu libre Minetest]

Exercice : Appliquer la notion

Question 1

Qu'est-ce qui est affiché par le programme ?

```
/** JavaScript */
let count = 3

for(let i = 0; i < count; i++) {
  console.log('Pause')
}
```

Indice :

`i` représente un compteur initialisé à 0. Il sera **augmenté** de 1 à chaque **itération**, jusqu'à ce qu'il soit égal à 3.

L'instruction entre les accolades s'exécute à chaque itération.

Question 2

Qu'est-ce qui est affiché par le programme ?

```
/** JavaScript */

for(let i = 3; i >= 0; i--) {
  console.log('Temps restant', i, 'secondes')
}
```

Indice :

`i` représente un compteur initialisé à 3. Il sera **diminué** de 1 à chaque **itération**, jusqu'à ce qu'il soit **inférieur** à 0.

Structure for

[cf. i2VfD8lW]

Objectifs

- Comprendre la structure de la boucle `for` ;
- Savoir initialiser et interrompre une boucle `for`.

Mise en situation

Souvent, il est nécessaire de répéter une action un nombre précis de fois : dans ce cas on utilise souvent des **compteurs**, qui permettent de compter le nombre de répétitions et d'arrêter la boucle au bout d'un certain nombre.

Boucle for

§ Syntaxe

La boucle `for`, ou *« pour »*, permet de réaliser un nombre **connu** d'itérations. Algorithmiquement, la boucle for peut se traduire par : *« Pour compteur de x à y, faire ... »*.

On peut distinguer trois éléments pour paramétrer cette boucle :

- **Initialisation** : le compteur prend sa valeur de départ.
- **Condition de sortie** : si elle est vraie, la boucle continue.
- **Opération** : à appliquer au compteur à chaque itération.

```
for (initialisation; condition; opération) {
  // instructions
}
```

? Exemple

Compteur de 0 à 5 exclus (soit 5 tours).

```
/** JavaScript: arrêt quand i n'est plus strictement inférieur à 5 */
for (let i = 0; i < 5; i++) {
 console.log(i)
}
```

```
"""Python: arrêt quand i n'est plus strictement inférieur à 5."""
for i in range(5):
  print(i)
```

À retenir

La boucle `for` se prête particulièrement aux cas où il est nécessaire d'effectuer un nombre d'itérations connu avant le début de la boucle.

[cf. DWmy2FAd]

Exercice : Appliquer la notion

On dispose d'un programme qui effectue des itérations en incrémentant un compteur.

```
let result = ''
for (let i = 0; i < 4; i++) {
  result = result + 'Toc '
}
console.log(result)

```

Question 1

Quel est le résultat affiché ?

Question 2

Ce programme affiche le même résultat. Quelle opération est faite sur le compteur ?

```
let result = ''
for (let i = 4; i > 0; i--) {
  result = result + 'Toc '
}
console.log(result)

```

Structure while

[cf. rXSEP0zQ]

Objectifs

- Comprendre la structure de la boucle `while` ;
- Savoir utiliser et interrompre une boucle `while`.

Mise en situation

Toutes les situations ne se prêtent pas à une boucle avec un compteur à incrémenter. Prenons par exemple un programme qui demande à l'utilisateur d'entrer son âge. Si l'âge entré n'est pas un nombre valide, le programme redemande de renseigner son âge. Nous avons donc ici un mécanisme de boucle qui va devoir s'exécuter un nombre de fois **inconnu** à l'avance. Ici la condition de notre boucle sera de vérifier la validité de la valeur entrée par l'utilisateur, par exemple en testant le type de la variable.

Boucle while

§ *Syntaxe*

La boucle `while`, ou « *tant que* », permet de réaliser des itérations en fonction de **conditions booléennes** (vrai ou faux). Algorithmiquement, la boucle `while` peut se traduire par : « *Tant que condition vraie, faire ...* ».

La structure de cette boucle est très simple et comporte une condition et un bloc d'instructions.

```
while (condition) {
  // instructions
}
```

Menu des recommandations

? *Exemple*

On affiche un menu avec des options. Seule l'option « *0* » permet de sortir du menu grâce à la condition du `while`.

```
"""Python."""
choice = ''

# Continue tant que 0 n'est pas sélectionné
while choice != '0':
  print('0. Sortir')
  print('1. Afficher le film recommandé')
  print('2. Afficher l\'album recommandé')
  # Attend la réponse
  choice = input('Choix > ')

  if choice == '1': # film
    print('Titanic (1997) - James Cameron\n')
  elif choice == '2': # album
    print('Thriller (1982) - Michael Jackson\n')
```

```javascript
/** JavaScript */
let choice = ''

// continue tant que 0 n'est pas sélectionné
while (choice !== '0') {
  console.log('0. Sortir')
  console.log('1. Afficher le film recommandé')
  console.log('2. Afficher l\'album recommandé')
  // attend la réponse
  choice = prompt('Choix')

  if (choice === '1') { // film
    console.log('Titanic (1997) - James Cameron\n')
  } else if (choice === '2') { // album
    console.log('Thriller (1982) - Michael Jackson\n')
  }
}

```

Roulette ? *Exemple*

Une mise de départ est fixée à 10 €. À chaque tour, la roulette sort un nombre : s'il est pair, on empoche l'argent (moins la mise), sinon la mise double et on continue.

```python
"""Python."""
import random

attempts = 0
gain = 0
bet = 10
loss = 0
winner = False

print('Tu gagnes si la roulette sort un nombre pair. La mise de départ est à 10 euros.')

# continue jusqu'à perdre
while not winner:
  # argent mis en jeu
  loss = loss + bet
  attempts = attempts + 1
  # nombre aléatoire entre 0 et 30
  number = random.randint(0, 30)
  print('Tentative.....', number)

  if number%2 == 1: # nombre impair
    print('Dommage, ta mise double. Retente ta chance')
    bet = bet * 2
  else: # nombre pair
    print('C\'est gagné')
    gain = bet * 2
    winner = True

  input('(Entrée)')

print('Gain de', (gain - loss), ' euros après ', attempts, ' tentative(s)')
```

```javascript
/** JavaScript */
let attempts = 0
let gain = 0
let bet = 10
let loss = 0
```

```
let winner = false

console.log('Tu gagnes si la roulette sort un nombre pair. La mise de départ est à 10 euros.')

// continue jusqu'à perdre
while (!winner) {
  // argent mis en jeu
  loss = loss + bet
  attempts = attempts + 1
  // nombre aléatoire entre 0 et 30
  const number = Math.floor(Math.random() * 31)
  console.log('Tentative.....', number)

  if (number % 2 === 1) { // nombre impair
    console.log('Dommage, ta mise double. Retente ta chance')
    bet = bet * 2
  } else { // nombre pair
    console.log('C\'est gagné')
    gain = bet * 2
    winner = true
  }
  prompt('(Entrée)')
}

console.log('Gain de ' + (gain - loss) + ' euros après ' + attempts + ' tentative(s)')
```

Boucle do...while

Complément

En JavaScript, il existe une structure proche de la boucle `while` : la boucle `do while`. Cette fois ci, le bloc d'instruction est exécuté d'abord, la condition est testée ensuite. Si la condition est remplie, on continue. Il y a donc donc **toujours au moins une itération**.

Algorithmiquement, la boucle `do while` peut se traduire par : *« Faire... Tant que condition vraie »*.

```
do {
  // instructions
} while (condition)
```

```
/** JavaScript: itère une seule fois */
let i = 0
do {
  console.log(i)
} while (i < -2)
```

Compteur avec while

Complément

La boucle `while` peut reproduire le comportement d'un compteur (mais en général, dans ce cas, on préfère une boucle `for`).

```
/** JavaScript: itère 5 fois */
let i = 0
while (i < 5) {
  console.log(i++)
}
```

```
"""Python: itère 5 fois."""
i = 0
while i < 5:
  print(i)
  i+=1
```

À retenir

- La boucle `while` permet des itérations avec des conditions quelconques.
- Elle est principalement utilisée quand on ne connaît pas le nombre d'itérations à faire avant le début de la boucle.

[cf. MVlMaGhR]

Exercice : Appliquer la notion

Un coffre fort ne peut être ouvert qu'avec un mot de passe composé de 4 chiffres. Voici son programme :

```
/** JavaScript */
const secretPassword = '4842'
let answer = ''

console.log('Mot de passe requis')

while (answer !== secretPassword) {
  // Attend une réponse
  answer = prompt('**** ')
}
console.log('Ouverture du coffre')

```

Question

Ce coffre n'est pas très sécurisé. Modifier son programme pour qu'un nombre de mauvaises réponses maximum soit autorisé : au bout de 4 tentatives qui échouent, le coffre affiche *Tentatives dépassées. Blocage de toutes les issues*.

Indice :

Utiliser un compteur incrémenté à chaque tentative et ajouter une condition au `while`.

Indice :

L'affichage des messages se fera une fois que l'on est sorti de la boucle : soit parce que le mot de passe est bon, soit parce que le nombre d'essai maximum est dépassé.

Les boucles imbriquées

[cf. Tmg2FdVB]

Objectif

- Comprendre l'imbrication de plusieurs structures itératives.

Mise en situation

Imaginons un programme qui se charge de modifier tout les pixels d'un écran. Pour cela il effectue une boucle sur chaque pixel, l'un après l'autre, pour lui donner la valeur désirée. Comme vous le savez, un écran est constitué de lignes et de colonnes de pixels. On peut donc dire que notre programme va parcourir l'ensemble des lignes de notre écran, et pour chacune de ces lignes, parcourir l'ensemble des colonnes. Nous avons donc là en fait 2 boucles, l'une imbriquée dans l'autre. Nous allons voir qu'une imbrication de boucle n'est pas compliquée à réaliser, mais que cela nécessite un peu de vigilance.

Imbrication

Une boucle peut en contenir une autre, et plus précisément, être **imbriquée** dans une autre.

Il faut être particulièrement vigilant aux variables qui sont modifiées pendant les itérations d'une boucle imbriquée car celles-ci sont susceptibles d'avoir des répercussions sur l'autre boucle.

En particulier, on veillera à ne pas modifier le compteur de la boucle externe dans la boucle imbriquée.

Double compteur ? *Exemple*

Deux compteurs différents sont déclarés.

Pour chaque itération de la boucle externe, la boucle imbriquée (ou interne) réalise 11 itérations. Le compteur de la boucle imbriquée est remis à zéro à chaque nouvelle itération de la boucle externe.

```javascript
/** JavaScript: affiche les tables de mutliplication de 0 à 10 */
for (let i = 0; i <= 10; i++) {
  for (let j = 0; j <= 10; j++) {
    console.log(i, '*', j, '=', i * j)
  }
}
```

```python
"""Python: affiche les tables de multiplication de 0 à 10."""
for i in range(11):
  for j in range(11):
    print(i, '*', j, '=', i * j)
```

Imbrication mixte

L'imbrication est possible pour tous les types de boucle. On peut même imbriquer des boucles de types différents, comme une boucle `for` dans une boucle `while`.

? *Exemple*

Le programme boucle tant que le jeu continue. À chaque tour, on pioche un nombre aléatoire et on demande sa table de multiplication de 0 à 9.

```python
"""Python."""
import random

end = False
score = 0
maximumScore = 0

# Continue tant que l'utilisateur répond 'o'
while not end:
  # Calcule un nombre aléatoire
  randomNumber = random.randint(0, 9)
  print('Table de', randomNumber)
  # Demande la table de multiplication du nombre aléatoire
  for i in range (10):
    answer = input('{0}*{1} = ? '.format(randomNumber, i))
    # Compte le nombre total de questions
    maximumScore = maximumScore + 1
    # 1 point pour chaque bonne réponse
    if answer == str(randomNumber*i):
      score = score + 1

  print('Vous avez {0}/{1}'.format(score, maximumScore))
  end = input('Continuer le test ? (o/n)') != 'o'
```

```javascript
/** JavaScript */
let end = false
let score = 0
let maximumScore = 0

// Continue tant que l'utilisateur répond 'o'
while (!end) {
  // Calcul un nombre aléatoire
  const randomNumber = Math.floor(Math.random() * 9) + 1
  console.log('Table de ' + randomNumber)
  // Demande la table de multiplication du nombre aléatoire
  for (let i = 0; i < 10; i++) {
    const answer = Number(prompt(randomNumber + '*' + i + ' ?'))
    // Compte le nombre total de questions
    maximumScore = maximumScore + 1
    // 1 point pour chaque bonne réponse
    if (answer === randomNumber * i) {
      score = score + 1
    }
  }
  console.log('Vous avez ' + score + '/' + maximumScore)
  end = prompt('Continuer le test ? (o/n)') !== 'o'
}
```

À retenir

Les boucles imbriquées permettent d'associer des itérations à d'autres et doivent être manipulées avec prudence.

[cf. VIShmtdW]

Exercice : Appliquer la notion

Un étudiant s'amuse à réaliser des programmes permettant de dessiner des formes à partir de caractères « * ». Il a mis au point un programme qui affiche un triangle.

```
/** JavaScript: dessine un triangle */
const heigth = 15
const width = 15

for(let h = 0; h < heigth; h++) {
  let line = ''
  for(let w = 0; w < width - h; w++) {
    line = line + '*' // ajoute une étoile à la ligne à afficher
  }
  console.log(line)
}
```

Question

Il vous demande de modifier son programme de manière à affiche un rectangle 15x15.

Donner le code modifié.

Indice :

Modifier la boucle imbriquée pour que la largeur dessinée soit égale à la longueur.

Essentiel

[cf. dkPaamJH]

Les structures itératives sont très utiles en programmation et permettent de répéter un ensemble d'actions à plusieurs reprises. Elles existent, quasiment à l'identique, dans tous les langages. Il y en a de plusieurs types : certaines basées sur un compteur, d'autres sur l'évaluation d'une condition quelconque. L'utilisation d'un compteur permet de maîtriser précisément le nombre d'itérations, tandis que l'évaluation d'une condition permet de ne répéter la boucle que si c'est nécessaire. En fonction des besoins, il est aussi possible d'utiliser plusieurs boucles imbriquées.

Quiz

Exercice 1 : Quiz - Culture

Exercice

Quelles boucles peuvent utiliser un compteur afin d'effectuer un nombre d'itérations connu à l'avance ?

- ☐ while
- ☐ do while
- ☐ for

Exercice

Quelle boucle est la plus adaptée en général pour utiliser un compteur afin d'effectuer un nombre d'itérations connu à l'avance ?

- ○ while
- ○ do while
- ○ for

Exercice

On peut toujours remplacer une boucle *do while* par une boucle *while* ?

- ○ vrai
- ○ faux

Exercice

On peut toujours remplacer une boucle *for* par une boucle *while* ?

- ○ vrai
- ○ faux

Exercice 6 : Quiz - Méthode

Exercice

Quels éléments sont intégrés dans la première instruction d'une boucle `for` ?

- [ ] Une initialisation
- [ ] Une condition
- [ ] Une opération
- [ ] Une alternative

Exercice

Quand est-il pertinent d'utiliser une boucle `while` ?

- [ ] Quand le nombre d'itérations n'est pas connu avant d'entrer dans la boucle
- [ ] Quand le corps de la boucle est composé de peu d'instructions

Exercice

Pour quelle raison première utilise-t-on une boucle `do while` plutôt que `while` ?

- ○ Lorsque la boucle peut ne réaliser aucune itération.
- ○ Lorsque la condition de sortie ne concerne pas un compteur.
- ○ Lorsque la boucle itère au moins une fois.

Exercice 10 : Quiz - Code

Exercice

Quelle opération est une décrémentation d'un compteur ?

- [ ] `count++`
- [ ] `count = count + 1`
- [ ] `count--`
- [ ] `count = count - 1`

Exercice

Combien de fois s'exécutera l'instruction `line = line + '*'` ?

```
/** JavaScript */
const heigth = 10

for (let h = 0; h < heigth; h++) {
  let line = ''
  for (let w = 0; w < heigth; w++) {
    line = line + '*'
  }
  console.log(line)
}
```

Exercice

Quelle(s) affirmation(s) sont vraie(s) sur la boucle suivante ?

```
/** JavaScript */
let n = 10
while (n >= 0) {
  console.log(n)
  n = n - 1
}

```

- [ ] La boucle effectuera 10 itérations
- [ ] Une boucle `for` serait plus appropriée
- [ ] Ce programme effectue un compte à rebours de 10 à 0.

Crédits des ressources

Itération avec un compteur p. 5

http://creativecommons.org/licenses/zero/4.0/fr/, Wikipedia https://fr.wikipedia.org/wiki/Boucle_forhttps://fr.wikipedia.org/wiki/Boucle_for

Copie d'écran du jeu libre Minetest p. 5

Minetest, https://www.minetest.netminetest.net, licence LGPL 2.0