

Boucles en Python

Table des matières

I. Boucle « while »	3
II. Exercice : Quiz	5
III. Boucle « for »	6
IV. Exercice : Quiz	10
V. Instructions « break » et « continue »	11
VI. Exercice : Quiz	12
VII. Essentiel	13
VIII. Auto-évaluation	13
A. Exercice	13
B. Test	13
Solutions des exercices	14

I. Boucle « while »

Durée : 1 h

Environnement de travail : PC connecté à Internet

Contexte

Les boucles ainsi que les instructions conditionnelles sont les principales bases de l'algorithmique. Sans ces deux choses, on ne peut pas faire grand-chose dans le mode de la programmation. Ces deux notions sont omniprésentes en algorithmique, que ce soit dans les fonctions de recherche ou pour afficher des suites de nombres, par exemple.

Les instructions conditionnelles sont des instructions permettant d'exécuter une suite d'instructions en fonction de la condition posée. En effet, les boucles sont des méthodes qui permettent de répéter une liste d'instructions plusieurs fois selon le besoin afin d'effectuer des traitements.

Dans ce cours, on va voir et entrer un peu plus dans les détails sur les principaux types de boucles notamment les boucles « *while* » et les boucles « *for* » avec le langage de programmation Python.

Définition

Une boucle, ou « *loop* », en anglais est une action qui permet de répéter automatiquement des instructions un certain nombre de fois.

« *While* » signifie « *tant que* ». Cette boucle nous permet d'exécuter des instructions tant qu'une ou des conditions sont vérifiées. Il s'agit donc d'une boucle conditionnelle.

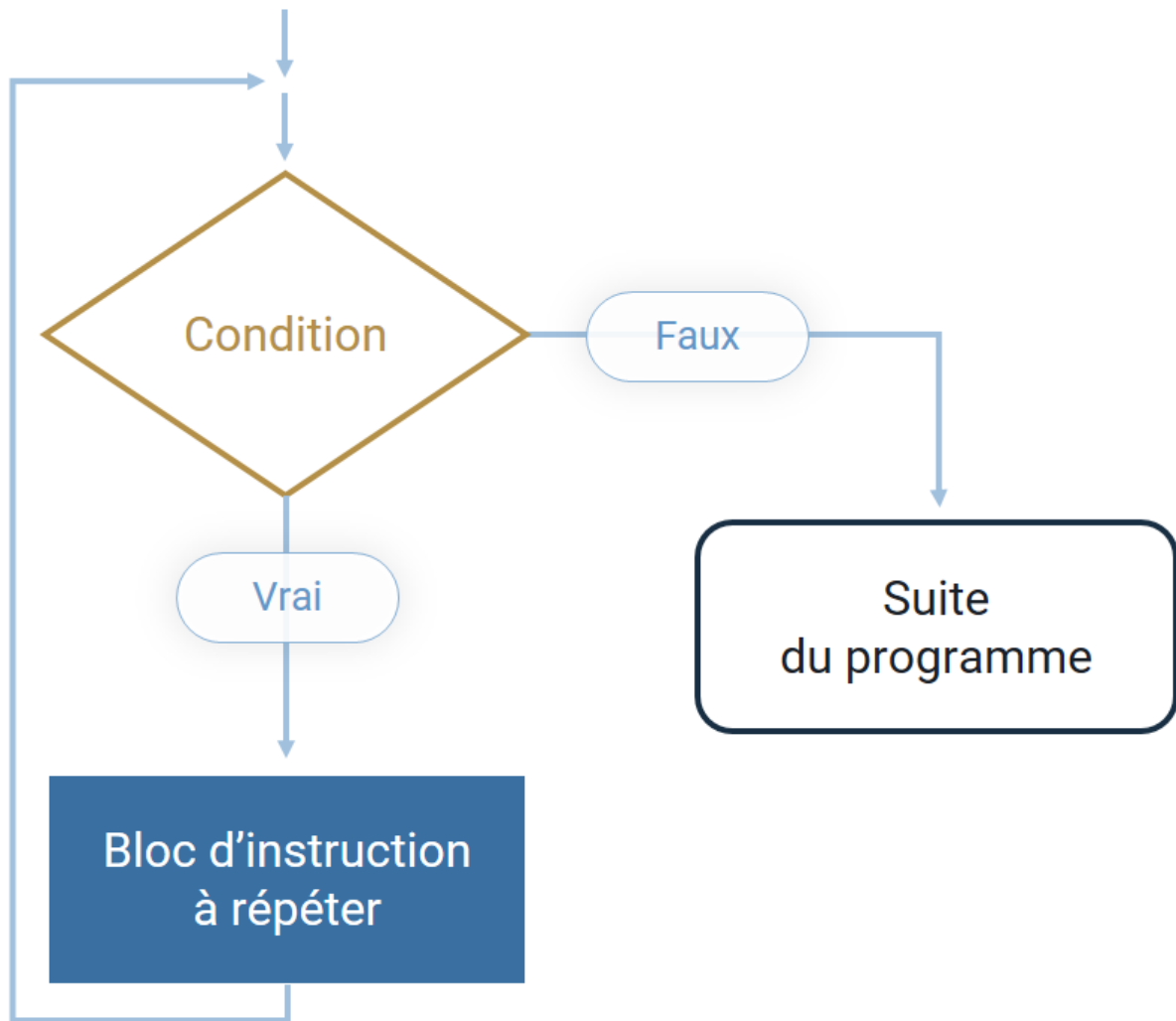
Dans le codage en Python, une boucle *while* se compose toujours des éléments suivants :

- Le mot clé « *while* »,
- Une condition,
- Les deux points (:) juste après la condition pour déclarer le commencement des instructions que le programme va exécuter,
- À partir de la ligne suivante en faisant une indentation, un bloc de code en retrait appelé aussi « *clause while* ».

Une instruction « *while* » ressemble à une instruction « *if* ». La différence se trouve dans leur comportement. À la fin d'une clause *if*, l'exécution se poursuit après l'instruction *if*. Mais à la fin d'une clause *while*, l'exécution du programme revient au début de l'instruction *while*.

Autrement dit, pour *if*, la condition est vérifiée une seule fois tandis que pour *while*, le programme revient au début de l'instruction tant que la condition est respectée.

Dans la boucle *while*, la condition est toujours vérifiée au début de chaque itération (c'est-à-dire à chaque exécution de la boucle). Si la condition est vraie la clause est exécutée puis la condition est vérifiée à nouveau. La première fois que la condition s'avère « *false* », la clause *while* est ignorée.



Fonctionnement de la boucle while

La clause `while` est souvent appelée « *boucle while* ». Regardons une instruction `if` et une boucle `while` qui utilise les mêmes conditions.

Remarque

Avec `if` et `while`, on va vérifier que la valeur d'une variable `i` est inférieure à 7 et imprime un message. Lorsqu'on va exécuter ces deux codes, les résultats seront différents. Pour `if`, la sortie est simplement le message si la condition est vérifiée, mais pour la déclaration `while`, le message est répété tant que la condition n'est pas vérifiée.

Utilisation de `if` :

```

1 i = 0
2 if i < 7 :
3     print ("i est inférieur à 7")
4     i = i + 1
  
```

Interprétation :

Le code avec l'instruction `if` vérifie la condition et imprime le message une seule fois si cette condition est vraie. Il ne va donc pas prendre en compte l'itération qu'on a déclarée dans le bloc d'instruction.

Ce bout de code peut se lire :

Si « `i` » inférieur à 7, alors on affiche « *i est inférieur à 7* ».

Ce code affiche alors la phrase :

```
>> i est inférieur à 7
```

Utilisation de while :

```
1 i = 0
2 while i < 7 :
3     print ("i est inférieur à 7")
4     i = i + 1
```

Interprétation :

Le code avec l'instruction while répètera l'instruction tant que la condition est valide. Par l'incrémementation $i = i + 1$ en bas du code, la valeur de i va être incrémentée de 1 à chaque fois que le programme revient au début de l'instruction.

Ce code peut se lire :

Tant que i est inférieur à 7, alors on affiche « *i est inférieur à 7* ».

Le message est alors affiché 7 fois parce que le programme va tester un par un les valeurs de i à partir de 0. C'est-à-dire qu'il va prendre la valeur de $i = 0$. Si c'est vrai, il affiche le message, il incrémente ensuite de 1 la valeur de i , c'est-à-dire que $i = 0 + 1$ qui est égale à 1, il affichera encore le message si la condition est vraie et cela est répété avant que la condition $i < 7$ ne soit fausse.

Ce code affichera donc le message pour chaque valeur de i inférieur à 7, la sortie est alors :

```
>> i est inférieur à 7
>> i est inférieur à 7
>> i est inférieur à 7
>> i est inférieur à 7
>> i est inférieur à 7
>> i est inférieur à 7
>> i est inférieur à 7
```

Remarque

Si la condition est déjà fausse au départ, les instructions ne seront jamais exécutées. En revanche, si la condition est toujours vraie, les instructions seront répétées d'une manière indéfinie.

Exercice : Quiz

[solution n°1 p.15]

Question 1

Quand est-ce qu'on utilise une boucle ?

- ☐ Pour exécuter des instructions d'une manière répétitive
- ☐ Pour afficher des messages
- ☐ C'est inutile

Question 2

Si une condition reste toujours vraie dans la boucle, quel est le résultat ?

- ☐ Rien ne se passe
- ☐ Cela entraîne une boucle infinie
- ☐ L'exécution du programme affiche une erreur

Question 3

Quel mot-clé utilise-t-on pour définir une boucle conditionnelle ?

- ☐ if
- ☐ def
- ☐ while

Question 4

Combien de fois le message « *Hello world* » dans ce code sera-t-il affiché ?

```
1 i = 0
2 while i < 10
3     print (" Hello world ")
4     i = i + 1
```

- ☐ Une seule fois
- ☐ Le code ne s'exécute pas
- ☐ 10 fois

Question 5

Dans le code suivant, le message « *Salut* » sera affiché 5 fois.

```
1 i = 0
2 while i < 5 :
3     print (" Hello world ")
4     i = i + 1
5 print("Salut")
```

- ☐ Faux
- ☐ Vrai

III. Boucle « for »

Définition

En Python, la boucle for est aussi appelée une boucle itérative, cela nous permet d'effectuer une itération sur une collection d'objets. Une collection peut être une chaîne de caractères, une liste, un tuple ou aussi un dictionnaire. La boucle for est donc utilisée pour parcourir et interagir avec les données de ces collections.

La syntaxe de l'instruction for est la suivante :

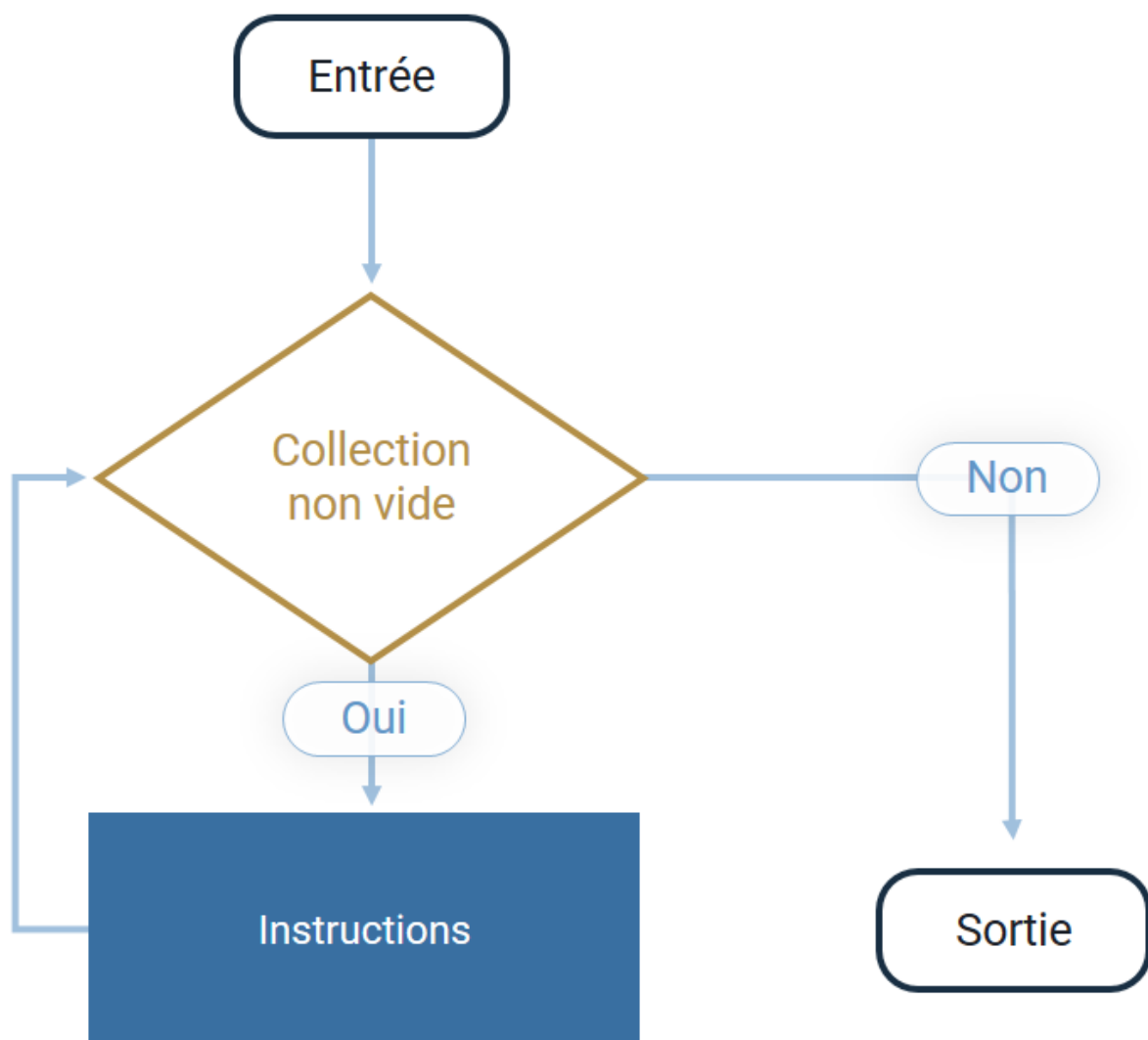
```
1 collections
2
3 for élément in collections:
4     instructions...
```

Ici, le mot « *collections* » déclaré en début représente la séquence à parcourir, il peut être une liste, un tableau, etc. Le mot « *élément* » est aussi une variable, mais celle-ci va recevoir l'élément de la séquence traitée. Et enfin à la place du « *instructions* », on mettra l'instruction à exécuter.

Elle comprend toujours les éléments suivants :

- Le mot-clé for
- Une variable
- Le mot clé-in
- La séquence à parcourir
- Deux points
- À partir de la ligne suivante, un bloc de code en retrait (appelé clause for)

Voici un organigramme montrant le fonctionnement de la boucle for en Python.



**Fonctionnement de la boucle
for**

Dans la boucle for, une collection est parcourue à partir de la première entrée. Le programme exécute ensuite les instructions dans le bloc de code de la boucle pour cet élément. Tant que la séquence contient des données, le programme exécute le code en parcourant l'ensemble des éléments suivant la première entrée.

La boucle ne s'arrête que lorsque chaque élément est traité.

Exemples d'utilisation de la boucle for

Passons maintenant à quelques exemples après que nous avons vu comment la boucle for fonctionne en général.

Utilisation de la boucle for sur une liste

```
1 MaCollection = ["Marie", "Jeanne", "Robert"]
2 for nom in MaCollection :
3     print(nom)
```

Ici, « *MaCollection* » est une liste, elle contient trois chaînes de caractère qui sont : Marie, Jeanne, Robert.

Le but est de parcourir la liste et d'afficher les éléments. La variable « *nom* » après le mot-clé « *for* » va prendre alors la valeur du premier élément de la liste soit « *Marie* ». Elle prend par la suite une par une les autres valeurs et les affiche une après l'autre.

Résultat du code :

```
>> Marie
>> Jeanne
>> Robert
```

Utilisation de la boucle for sur un dictionnaire

Expliquons d'abord ce qu'est un dictionnaire. Il s'agit aussi d'une collection, mais pour lui, chaque objet possède une clé qui est associée à une valeur.

```
1 MonDictionnaire = {"Marie" : 1, "Jeanne" : 2, "Robert" : 3}
2 for nom in MonDictionnaire :
3     print(nom)
```

Sur cet exemple, le code fait le parcours du dictionnaire et affiche ensuite les clés.

Le résultat est alors :

```
>> Marie
>> Jeanne
>> Robert
```

Pour manipuler les valeurs associées aux clés, on peut faire comme suit :

```
1 MonDictionnaire = {"Marie" : 1, "Jeanne" : 2, "Robert" : 3}
2 for nom in MonDictionnaire :
3     print(MonDictionnaire[nom])
```

En utilisant `MonDictionnaire[nom]`, on a accès à la valeur de la clé `nom`.

Le résultat sera alors :

```
>> 1
>> 2
>> 3
```

Et bien sûr, on peut récupérer à la fois la clé et la valeur de chaque élément en utilisant la méthode `items()` . On peut le faire comme suit :

```
1 MonDictionnaire = {"Marie" : 1, "Jeanne" : 2, "Robert" : 3}
2 for clé, valeur in MonDictionnaire.items() :
3     print('La clé est : ' , clé, ', et sa valeur est : ' , valeur )
```

On a déclaré deux variables après le mot-clé « *for* » : la variable « *clé* » pour stocker la clé dans la liste et la variable « *valeur* » pour stocker celle de la valeur.

Le `print` sert pour l'affichage de ces éléments, les mots compris entre les apostrophes sont les mots qu'on veut afficher et ce qui n'est pas dedans sont les variables à appeler.

Le résultat sera alors :

```
>> La clé est : Marie, et sa valeur est : 1
>> La clé est : Jeanne, et sa valeur est : 2
>> La clé est : Robert, et sa valeur est : 3
```

Remarque

On peut parcourir et interagir avec toutes collections d'objets itérables avec la boucle `for`.

Utilisation de la boucle `for` et de la fonction `range()`

On peut utiliser la fonction `range()` à la place d'une collection sur la boucle `for`, une fonction `range()` renvoie une séquence ou suite de chiffres. Elle est donc aussi itérable.

On peut l'utiliser comme suit :

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

Ici, `range()` renvoie une liste de chiffres en commençant par 0. Il va donc afficher une liste de chiffres à partir de 0 jusqu'à 4 un par un. La sortie est donc :

```
>> 0
>> 1
>> 2
>> 3
>> 4
```

Néanmoins, on peut spécifier aussi les points de départ ainsi que la limite dans une fonction `range()`.

Le premier argument sera le point de départ de la variable dans la boucle `for` et le deuxième argument sera sa limite mais sans l'inclure dans le traitement. L'utilisation est comme suit :

```
1 for i in range(5, 10) :
2 print(i)
```

Ici, le point de départ est donc 5 et se termine à 9. Le résultat sera donc :

```
>> 5
>> 6
>> 7
>> 8
>> 9
```

La fonction `range()` peut également être appelée avec trois arguments. Les deux premiers arguments seront les valeurs de début et de fin, et le troisième sera l'argument d'étape, ou aussi la valeur de l'incréméntation. L'étape sera le montant de l'augmentation de la variable après chaque itération.

On peut l'utiliser comme suit :

```
1 for i in range(0, 10; 2) :
2 print(i)
```

Chaque itération sera donc incrémentée de 2 sans inclure l'argument d'arrêt de la boucle. Le résultat sera alors :

```
>> 0
>> 2
```

```
>> 4
>> 6
>> 8
```

La fonction `range()` est flexible dans la séquence de nombres qu'elle produit pour les boucles. Vous pouvez même utiliser avec des nombres négatifs, y compris pour faire un compte à rebours avec la boucle `for`.

Exercice : Quiz

[solution n°2 p.16]

Question 1

Quel mot-clé utilise-t-on pour définir une boucle itérative ?

- ☐ if
- ☐ while
- ☐ for

Question 2

Ce script fonctionne.

```
1 val = 1
2 for num in val :
3     print(num)
```

- ☐ Vrai
- ☐ Faux

Question 3

Ce script fonctionne.

```
1 for lettre in "hello" :
2     print(lettre)
```

- ☐ Faux
- ☐ Vrai

Question 4

On peut mettre des chaînes de caractères dans une fonction `range()`.

- ☐ Vrai
- ☐ Faux

Question 5

« Salut » sera affiché combien de fois dans ce script ?

```
1 for i in range(0, 10, 2) :
2     print("Salut")
```

- ☐ 4
- ☐ 5
- ☐ 10

V. Instructions « break » et « continue »

« *Break* » et « *continue* » sont des instructions utilisées dans une boucle. Ces instructions indiqueront à une boucle quoi faire si une condition est vérifiée ou non.

Instruction break

Il existe un raccourci pour sortir de l'exécution de la clause d'une boucle : **break**. Cette instruction permet de terminer l'itération immédiatement, quel que ce soit le nombre d'itérations effectuées ou restantes à faire lorsqu'une condition est remplie.

Exemple

Une instruction break contient simplement le mot-clé break.

Regardons un exemple avec la syntaxe de l'utilisation de cette instruction.

```
1 Liste=["Salut", "Hello", "Bonjour"]
2 for mot in Liste :
3     if mot == "Hello" :
4         break
5     print(mot)
```

Le but ici est de parcourir la liste par la boucle for, et de sortir de la boucle en utilisant le mot-clé break si une condition est vérifiée. Donc, la boucle commence par le premier élément dans la liste qui est le mot « *Salut* », elle vérifie la condition, si la condition n'est pas vérifiée, la boucle passe à l'élément suivant.

Dans l'exemple, notre condition est que si un mot dans la liste est égal à « *Hello* », on sort de la boucle et on affiche le résultat.

Le résultat de cet exemple est alors :

```
>> Salut
```

Instruction continue

Comme les instructions break, l'instruction **continue** est aussi utilisée dans les boucles. Mais à l'inverse, l'instruction continue quant à elle permet d'interrompre une itération pour passer à la suivante si la condition survient.

Exemple

On utilise l'instruction continue tout simplement en l'écrivant dans la boucle à la ligne suivante de la condition en faisant une indentation.

Prenons encore l'exemple sur l'instruction break mais maintenant on va utiliser l'instruction continue pour voir ce qui se passe.

```
1 Liste=["Salut", "Hello", "Bonjour"]
2 for mot in Liste :
3     if mot == "Hello" :
4         continue
5     print(mot)
```

La liste est parcourue de la même façon, mais au moment où la condition survient, au lieu de terminer l'itération et de sortir de la boucle, elle passe immédiatement à l'élément suivant.

Le résultat de cet exemple est alors :

```
>> Salut
```

```
>> Bonjour
```

Remarque

Les instructions break et continue se trouvent généralement après des instructions conditionnelles.

Exercice : Quiz

[solution n°3 p.17]

Question 1

Quelle instruction utilise-t-on pour stopper l'itération d'une boucle ?

- ☐ pass
- ☐ continue
- ☐ break

Question 2

Quelle instruction utilise-t-on pour interrompre l'itération d'une boucle et passer à la suivante si la condition est vérifiée ?

- ☐ pass
- ☐ continue
- ☐ break

Question 3

Où met-on les instructions break/continue ?

- ☐ En début du code
- ☐ À la fin du code
- ☐ Après l'instruction conditionnelle

Question 4

Quelle est la sortie de ce code ?

```
1 for lettre in "Boucle" :
2     if lettre == "u" :
3         break
4     print(lettre)
```

- ☐ bocle
- ☐ bo

Question 5

Quelle est la sortie de ce code ?

```
1 for lettre in "Boucle" :
2     if lettre == "u" :
3         continue
4     print(lettre)
```

- ☐ bo
- ☐ bocle

VII. Essentiel

Les boucles sont les instructions les plus utilisées dans le monde de la programmation. En effet, dans le langage de programmation Python, on compte deux types de boucles : la boucle « *while* » et la boucle « *for* ». Par définition, qui dit boucle dit répétition. En informatique, une boucle est donc la répétition des exécutions d'une suite d'instructions.

La boucle « *while* » (« *tant que* ») est une boucle conditionnelle. C'est-à-dire qu'elle permet l'exécution d'une suite d'instructions en fonction de la condition fixée. Cette répétition est alors exécutée tant que la condition est vraie. Et au contraire, la boucle s'arrête dès que la condition fixée n'est plus vérifiée.

La boucle « *for* » quant à elle est une boucle itérative et est beaucoup axée sur les collections. Les collections sont des ensembles de données regroupées, comme les tableaux, les listes, les dictionnaires. La boucle *for* parcourt tout ce qui est itératif, même les chaînes de caractères. Cette boucle permet alors l'itération des éléments dans la collection selon leur disposition dans la séquence. Elle ne s'arrête pas tant que ces éléments sont parcourus. Néanmoins, on peut dicter à la boucle ce qu'elle doit faire avec les instructions « *break* » et « *continue* » quand une condition est vérifiée ou pas.

L'instruction *break* permet la possibilité de briser une boucle lorsqu'une condition est vérifiée. Son emplacement est généralement dans le bloc des instructions juste après l'instruction conditionnelle à vérifier. Elle est appelée juste en mettant le mot-clé « *break* ».

Pareil pour l'instruction *continue*, la syntaxe est la même en changeant juste le mot-clé en « *continue* ». En revanche, l'instruction *continue* permet la possibilité de sauter une partie de la boucle et de continuer sur le reste. La différence sur l'utilisation de ces deux instructions est qu'avec « *continue* », l'exécution des instructions se poursuivra malgré les interruptions.

VIII. Auto-évaluation

A. Exercice

On souhaite afficher une table de multiplication à partir d'un nombre saisi par un utilisateur.

Question 1

[solution n°4 p.18]

Écrivez ce programme en utilisant la boucle *for*.

Question 2

[solution n°5 p.18]

Écrivez le même programme, mais en utilisant cette fois la boucle *while*.

B. Test

Exercice 1 : Quiz

[solution n°6 p.19]

Question 1

Lesquels de ces mots-clés sont utilisés pour faire une boucle ?

- ☐ *while*
- ☐ *def*
- ☐ *for*

Question 2

On peut créer une boucle avec la fonction *range()*.

- ☐ Vrai
- ☐ Faux

Question 3

Les instructions break et continue servent à contrôler l'exécution d'une boucle.

- ☐ Vrai
- ☐ Faux

Question 4

Quelle est l'instruction utilisée pour que « n » prenne les valeurs entre 3 et 10 ?

- ☐ for n in range(3, 11)
- ☐ for n in range(3, 10)
- ☐ for n in range(4, 10)

Question 5


Les instructions continue et break fonctionnent sans instruction conditionnelle.

- ☐ Faux
- ☐ Vrai

Solutions des exercices


Exercice p. 5 Solution n°1**Question 1**

Quand est-ce qu'on utilise une boucle ?

- ☒ Pour exécuter des instructions d'une manière répétitive
- ☐ Pour afficher des messages
- ☐ C'est inutile
-  Par définition, une boucle permet de répéter des instructions à l'infinie en fonction des besoins.


Question 2

Si une condition reste toujours vraie dans la boucle, quel est le résultat ?

- ☐ Rien ne se passe
- ☒ Cela entraîne une boucle infinie
- ☐ L'exécution du programme affiche une erreur
-  Si la condition au départ est toujours vraie et reste toujours vraie à chaque incrémentation, cela veut dire que l'instruction sera toujours exécutée sans arrêt puisque la condition est toujours vérifiée.

Question 3


Quel mot-clé utilise-t-on pour définir une boucle conditionnelle ?

- ☐ if
- ☐ def
- ☒ while
-  La boucle while est exécutée selon la condition donc il s'agit d'une boucle conditionnelle.

Question 4

Combien de fois le message « *Hello world* » dans ce code sera-t-il affiché ?

```
1 i = 0
2 while i < 10
3     print (" Hello world ")
4     i = i + 1
```

- ☐ Une seule fois
- ☒ Le code ne s'exécute pas
- ☐ 10 fois
-  Le code ne s'exécutera pas puisqu'il y a une erreur. En effet, l'absence des deux points (:) après la condition va lever une erreur syntaxique et le code ne sera pas exécuté.


Question 5

Dans le code suivant, le message « *Salut* » sera affiché 5 fois.

```
1 i = 0
2 while i < 5 :
3     print (" Hello world ")
4     i = i + 1
5 print("Salut")
```

☒ Faux

☐ Vrai

 Dans ce code, l'instruction `print(« Salut »)` n'est pas indentée. C'est à dire qu'elle n'appartient pas aux instructions à répéter mais sera exécutée une fois la boucle terminée.

Exercice p. 10 Solution n°2


Question 1

Quel mot-clé utilise-t-on pour définir une boucle itérative ?

☐ if

☐ while

☒ for

 La boucle `for` parcourt une collection d'objet par itération des éléments de ce dernier.


Question 2

Ce script fonctionne.

```
1 val = 1
2 for num in val :
3     print(num)
```

☐ Vrai

☒ Faux

 La valeur de `val` est du type entier, ce dernier n'est pas un objet itérable ce qui entraîne une erreur sur l'exécution du script.


Question 3

Ce script fonctionne.

```
1 for lettre in "hello" :
2     print(lettre)
```


☐ Faux

☒ Vrai

 La collection à parcourir ici est une chaîne de caractères, c'est-à-dire que c'est itérable car on peut afficher chaque caractère un par un. Donc ce script marche en utilisant la boucle `for`.

Question 4


On peut mettre des chaînes de caractères dans une fonction `range()`.

- ☐ Vrai
- ☒ Faux
-  Une fonction `range()` ne peut renvoyer que des séquences de chiffres.

Question 5

« *Salut* » sera affiché combien de fois dans ce script ?


```
1 for i in range(0, 10, 2) :  
2 print("Salut")
```

- ☐ 4
- ☒ 5
- ☐ 10
-  À chaque itération, le message « *Salut* » sera affiché. On a mis dans le troisième argument le nombre d'incréméntation donc le message sera affiché pour $i = 0$, $i = 2$, $i = 4$, $i = 6$ et $i = 8$ car l'argument d'arrêt ne sera pas inclus. Donc, le message sera affiché 5 fois.

Exercice p. 12 Solution n°3


Question 1

Quelle instruction utilise-t-on pour stopper l'itération d'une boucle ?

- ☐ `pass`
- ☐ `continue`
- ☒ `break`
-  L'instruction `break` sert à quitter une boucle quand une condition est déclenchée.

Question 2

Quelle instruction utilise-t-on pour interrompre l'itération d'une boucle et passer à la suivante si la condition est vérifiée ?

- ☐ `pass`
- ☒ `continue`
- ☐ `break`
-  L'instruction `continue` permet d'interrompre une itération en fonction de la condition mais elle continuera quand même l'exécution du reste de la boucle.

Question 3

Où met-on les instructions `break/continue` ?

- ☐ En début du code
- ☐ À la fin du code
- ☒ Après l'instruction conditionnelle

Q L'instruction break ou continue est placée juste après les conditions car elle sert à dicter au programme ce qu'il doit faire après la vérification de la condition.

Question 4

Quelle est la sortie de ce code ?

```
1 for lettre in "Boucle" :
2     if lettre == "u" :
3         break
4     print(lettre)
```

☐ bocle

☒ bo

Q En itérant la chaîne de caractères « Boucle », à cause de l'instruction break, la boucle s'arrête lorsqu'elle arrive à la lettre « u » qui est dans la condition.

Question 5

Quelle est la sortie de ce code ?

```
1 for lettre in "Boucle" :
2     if lettre == "u" :
3         continue
4     print(lettre)
```

☐ bo

☒ bocle

Q En itérant la chaîne de caractères « Boucle », à cause de l'instruction continue, la boucle sera interrompue lorsqu'elle arrive à la lettre « u » mais elle continuera à l'itération suivante.

p. 13 Solution n°4

D'abord, on va déclarer une variable pour stocker la valeur du nombre saisi par l'utilisateur. On peut ensuite utiliser la fonction range() pour retourner une séquence de nombres de 1 à 10 pour la table de multiplication.

```
1 x = int(input("Entrer le nombre : "))
2
3 for a in range(1, 11):
4     print("{0} * {1} = {2}".format(x, a, a * x))
5     # L'affiche sera comme {0} * {1} = {2} en formatant {0} par x, {1} par a et {2} par le
    produit de x et a
```

p. 13 Solution n°5

Cette fois-ci, la syntaxe est différente :

```
1 x = int(input("Saisir un nombre : "))
2 a = 1
3 while a <= 10:
4     print("{0} * {1} = {2}".format(x, a, a * x))
5     a = a + 1
6     # L'affiche sera comme {0} * {1} = {2} en formatant {0} par x, {1} par a et {2} par le
    produit de x et a
```


Exercice p. 13 Solution n°6**Question 1**

Lesquels de ces mots-clés sont utilisés pour faire une boucle ?

☒ while

☐ def

☒ for


 Le mot-clé « *while* » permet de créer une boucle pour répéter des instructions selon le besoin, « *for* » permet des itérations sur un élément.

Question 2

On peut créer une boucle avec la fonction range().

☒ Vrai

☐ Faux


 La fonction range renvoie une liste de nombres ce qui permet la création d'une boucle en l'associant avec la boucle for.

Question 3

Les instructions break et continue servent à contrôler l'exécution d'une boucle.

☒ Vrai

☐ Faux

 Break permet de casser une boucle, et continue permet de sauter une partie d'une boucle.


Question 4

Quelle est l'instruction utilisée pour que « *n* » prenne les valeurs entre 3 et 10 ?

☒ for n in range(3, 11)

☐ for n in range(3, 10)

☐ for n in range(4, 10)


 Le premier argument de la fonction range() est son point de départ et le dernier est sa condition d'arrêt sans l'inclure.

Question 5

Les instructions continue et break fonctionnent sans instruction conditionnelle.

☒ Faux

☐ Vrai

 Les instructions continue et break permettent de contrôler une boucle selon la condition posée.