PROGRAMMATION PYTHON

Chapitre 4: Structures répétitives

Sommaire

Introduction

Les structures **répétitives** ou **boucles** ou **itérations** permettent de répéter x fois certaines instructions ou de parcourir certaines séquences comme les chaînes de caractères (par exemple faire une opération sur tous les caractères d'un mot).

Grâce aux boucles on va pouvoir répéter une suite d'instructions jusqu'à ce qu'une condition spécifique soit atteinte.

Contrairement aux structures de test qui exécutent une suite d'instructions uniquement une fois, la boucle va quant à elle exécuter la suite d'instructions plusieurs fois.

On rencontre régulièrement les boucles, par exemple lors de la saisie de notre mot de passe, l'ordinateur nous demandera de saisir le mot de passe tant que le mot de passe entré ne correspond pas au mot de passe enregistré.

On distingue deux principales catégories de boucles dans la majorité des langages de programmation qui sont: **les répétitions conditionnelles** et **les répétition inconditionnelles.**

Les types de boucles

Les répétitions conditionnelles (ou indéfinies) permettent de répéter une suite d'instructions selon une certaine condition. Ce type de répétition est utilisé pour les mots de passe, on ressaisit le mot de passe tant que celui-ci n'est pas correct. Avec ce type de boucle, on ne peut savoir par avance le nombre de fois que l'utilisateur va saisir son mot de passe (un utilisateur A peut se tromper 2 fois sur son mot de passe alors qu'un autre utilisateur B se trompera 3 fois), c'est pourquoi on parle de **boucle indéfinie.**

Les répétions inconditionnelles (ou avec compteur ou définies) contrairement au précédent, les instructions sont exécutées un nombre de fois donné. On peut les retrouver, par exemple lors de l'affichage de la table de multiplication de 10 d'un nombre donné. Ici, on sait déjà qu'on va devoir afficher n x 0 …n x 10 il va falloir une boucle de 0 à 10 pour calculer et afficher le résultat de la multiplication. Dans cet exemple on fera une répétition inconditionnelle avec un compteur max de 10.

On peut résoudre un problème avec l'une ou l'autre des boucles dans la majorité des cas. Cependant, il plus facile d'utiliser une boucle avec compteur quand on sait le nombre de répétitions sinon utiliser la boucle indéfinie.

La boucle while

La boucle **while** (tant que en français) se retrouve dans la plupart des langages de programmation et permet de répéter un bloc d'instructions tant qu'une condition est vraie. C'est une boucle conditionnelle ou indéfinie.

Pour utiliser la boucle **while** en **Python** il faut utiliser la syntaxe suivante:

```
while condition:
    # Instruction 1
    # Instruction 2
    # ...
    # Instruction n
```

Les deux points à la fin de la condition de boucle sont obligatoires ainsi que l'indentation avant écriture des instructions.

```
"""
    Ce programme demandera le nom d'utilisateur
    tant que celui-ci n'est pas Christophe
    sinon affiche le message de Bienvenue.
"""

# On demande une première fois le nom d'utilisateur
username = input("Entrer votre nom d'utilisateur:")

# On boucle tant que le nom d'utilisateur n'est pas égal à celui attendu
while username != "Christophe":
    # Affichage du message nom d'utilisateur incorrect
    print("Nom d'utilisateur incorrect !")
    # Demander encore le nom d'utilisateur
    username = input("Entrer votre nom d'utilisateur:")

# Instruction qui s'exécutera quand on sortira de la boucle
print("Bienvenue", username) # affichera Bienvenue nom_utilisateur
```

Exécution

```
Entrer votre nom d'utilisateur:toto
Nom d'utilisateur incorrect !
Entrer votre nom d'utilisateur:john
Nom d'utilisateur incorrect !
Entrer votre nom d'utilisateur:test
Nom d'utilisateur incorrect !
Entrer votre nom d'utilisateur:Christophe
Bienvenue Christophe
```

Abus de la boucle while

Quand on utilise la boucle **while** il faut veiller à utiliser des conditions qui peuvent changer dans le temps sinon on parle de boucle infinie. Comme l'exemple ci-dessous:

```python
"""
    Ce programme demandera le nom d'utilisateur
    tant que celui-ci n'est pas Christophe
    sinon affiche le message de Bienvenue.
"""

# On demande une première fois le nom d'utilisateur
username = input("Entrer votre nom d'utilisateur:")

# On boucle tant que le nom d'utilisateur n'est pas égal à celui attendu
while username != "Christophe":
    # Affichage du message nom d'utilisateur incorrect
    print("Nom d'utilisateur incorrect !")

# Instruction qui s'exécutera quand on sortira de la boucle
print("Bienvenue", username) # affichera Bienvenue nom_utilisateur
```

Exécution

```
Entrer votre nom d'utilisateur:toto
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
Nom d'utilisateur incorrect !
```

Ce programme va afficher de manière indéfini « nom d'utilisateur incorrect » dès la première erreur de l'utilisateur. Ce type de boucle est appelé boucle infinie et est à éviter car ici le nom d'utilisateur n'a aucune chance de changer dès qu'on rentre dans la boucle. Pour stopper une boucle infinie il faut taper sur **CTRL + C** dans la fenêtre de l'interpréteur.

La boucle for

La boucle **for** est une boucle inconditionnelle qu'on retrouve dans la plupart des autres langages mais qui n'a pas toujours le même sens que celui de Python. En Python, on utilisera l'instruction **for** pour construire une boucle inconditionnelle qui permet de parcourir des séquences de plusieurs données comme des listes d'éléments, les caractères d'une chaine de caractères (String)..

Pour utiliser la boucle **for** en **Python** il faut utiliser la syntaxe suivante:

```
for element in sequence:
    # Instruction 1
    # Instruction 2
    # ....
    # Instruction n
```

Les deux points à la fin de la condition de boucle sont obligatoires ainsi que l'indentation avant écriture des instructions.

```
"""
    Ce programme affiche la liste des caractères d'un nom.
"""

name = "Christophe"

# Boucle sur la liste des caractères du nom
print("Le nom " + name + " contient les lettres suivantes:\n")
for letter in name:
    print(letter)
```


```
Le nom Christophe contient les lettres suivantes:

C
h
r
i
s
t
o
p
h
e
```

La boucle for

Dans le programme précédent, la variable **lettre** prend successivement chaque lettre contenue dans la chaîne de caractères (C, ensuite h, puis...). On pouvait par exemple appliquer des conditions lors de l'itération comme le programme ci-dessous:

```
"""
    Ce programme affiche la liste des caractères d'un nom.
    Il dira les caractères qui sont des voyelles ou non.
"""

name = "Christophe"

# Boucle sur la liste des caractères du nom
print("Le nom " + name + " contient les lettres suivantes:\n")
for letter in name:
    # Verification si la lettre est une voyelle ou non
    if letter in "AEIOUYaeiouyAEIOUYaeiouy":
        print("La voyelle:", letter)
    else:
        print("La consonne:", letter)
```

Exécution

```
Le nom Christophe contient les lettres suivantes:

La consonne: C
La consonne: h
La consonne: r
La voyelle: i
La consonne: s
La consonne: t
La voyelle: o
La consonne: p
La consonne: h
La voyelle: e
```

Dans programme précédent, en fonction du contenu de la variable **letter** on affichera le type soit une **consonne** ou une **voyelle.**

L'instruction « a in b » renverra True si la variable « b » contient la variable « a » sinon renverra False.

L'instruction range

Python dispose d'une fonction native (fait partie à la base de Python après installation) **range** qui permet de générer une séquence par exemple si on veut itérer une action un certain nombre de fois.

L'instruction **range** utilise la syntaxe suivante: `range(start, stop, step)`

- **start:** un entier qui détermine le premier élément de la séquence par défaut la valeur est 0
- **stop:** l'entier jusqu'auquel on va boucler mais sans l'inclure (comme on commence à compter à partir de 0)
- **step**: un entier qui peut être positif ou négatif appelé **pas** pour définir de combien on va incrémenter ou décrémenter entre chaque élément de la séquence par défaut on incrémente de +1.

```
range(1, 10) #=> equivalent à range(1, 10, 1) pour traiter les séquences de 1 à 9
range(11) #=> equivalent à range(0, 11) ou range(0, 11, 1) pour traiter les séquences de 0 à 10
range(10, 0, -1) #=> pour traiter les séquences de 10 à 1
range(0, 10, 2) #=> pour traiter les séquences de 0, 2, 4, 6, 8
```

range va généralement être associé à la boucle for pour traiter la séquence qu'il génèrera. Par exemple si on veut afficher la liste des nombres de 0 à 100 on utilisera range + for.

Boucle for avec range

Pour traiter la liste des nombres de 0 à 10, on pourra utiliser **range** et **for** pour avoir les programmes suivants:

```
"""
    Programme 1: Nombre de 0 à 10
"""

print("Programme 1: Nombre de 0 à 10:\n")
# On peut ignorer ici le 0 en utilisant range(11)
for n in range(0, 11):
    print(n)

print("\n\n")


"""
    Programme 2: Nombre de 10 à 1
"""

print("Programme 2: Nombre de 10 à 1:\n")
# On peut pas ignorer ici le 0 car c'est le stop
for n in range(10, 0, -1):
    print(n)

print("\n\n")

"""
    Programme 3: Nombre de 0 à 10 en sautant un nombre
"""

print("Programme 3: Nombre de 0 à 10 en sautant un nombre:\n")
# On peut pas ignorer ici le 0
for n in range(0, 10, 2):
    print(n)
```

```
Programme 1: Nombre de 0 à 10:

0
1
2
3
4
5
6
7
8
9
10



Programme 2: Nombre de 10 à 1:

10
9
8
7
6
5
4
3
2
1



Programme 3: Nombre de 0 à 10 en sautant un nombre:

0
2
4
6
8
```

L'instruction continue

Le mot-clé **continue** en Python est utilisé dans les boucles pour continuer la boucle en repartant directement à la ligne de la condition du **while** ou du **for.**

Par exemple si on veut afficher la liste des nombres de 0 à 10 sauf le nombre 2 on aura:

```
"""
 Programme affichant les nombres 0, 1, 3, 4, 5, ...10
"""

# Utilisation de continue dans une boucle for
print("Exemple avec for:")
for n in range(0, 11):
    if n == 2:
        continue
    print(n)

print("\n\n")

# Utilisation de continue dans une boucle while
print("Exemple avec while:")
nb = -1 # On commence ici à partir de -1 pour avoir le 0
while nb < 10:
    nb += 1
    if nb == 2:
        continue
    print(nb)
```


```
Exemple avec for:
0
1
3
4
5
6
7
8
9
10


Exemple avec while:
0
1
3
4
5
6
7
8
9
10
```

Lorsqu'on tombe dans la condition où les variables **n/nb** sont égales 2, le programme n'exécute pas le « **print** » suivant et passe à la valeur suivante de la séquence.

L'instruction break

Contrairement à **l'instruction continue,** ici le mot-clé **break** dans une boucle permet de la stopper complètement. Elle permettra par exemple de sortir des boucles infinies dans un programme.

```
"""
 Programme essayant d'afficher les nombres 0, 1, 3, 4, 5, ...10
"""

# Utilisation de break dans une boucle for
print("Exemple break avec for:")
for n in range(0, 11):
    if n == 2:
        break
    print(n)

print("\n\n")

# Utilisation de break dans une boucle while
print("Exemple break avec while:")
nb = -1 # On commence ici à partir de -1 pour avoir le 0
while nb < 10:
    nb += 1
    if nb == 2:
        break
    print(nb)
```

Exécution

```
Exemple break avec for:
0
1



Exemple break avec while:
0
1
```

Ici les nombres de 2 à 10 ne seront jamais afficher car la boucle est stoppée dès qu'on atteint 2.

Les boucle infinies peuvent être générées avec True ou 1 associé à while.

```
while 1: #=> On peut remplacer 1 par True qui est toujours vrai donc boucle infinie
    reponse = input("Tapez 'Q' pour quitter:")
    if reponse == 'Q':
        print("Fin !")
        break
```

Instructions upper/lower

Python dispose deux fonctions natives pour mettre une chaine de caractères en majuscule ou en minuscule avec la syntaxe suivante:

```
string.upper() #=> pour mettre en majuscule

string.lower() #=> pour mettre en minuscule
```

```
"""
    Afficher le nom Christophe en majuscule et en minuscule
"""

name = "Christophe"

# Convertion en majuscule #=> équivalent à "Christophe".upper()
majuscule = name.upper()
print(name, "en majuscule est:", majuscule)

# Convertion en lower
minuscule = name.lower() #=> équivalent à "Christophe".lower()
print(name, "en minuscule est:", minuscule)
```

```
Christophe en majuscule est: CHRISTOPHE
Christophe en minuscule est: christophe
```

On pourrait par exemple utiliser **upper/lower** pour quitter quand l'utilisateur entre Q ou q.

```
while True:
    reponse = input("Tapez 'Q ou q' pour quitter:")
    if reponse.upper() == 'Q':
        print("Fin !")
        break
```

Equivalent

```
while True:
    reponse = input("Tapez 'Q ou q' pour quitter:")
    if reponse.lower() == 'q':
        print("Fin !")
        break
```

Concaténation en Python

Parfois on a besoin d'afficher une message avec le contenu d'une variable ou pour associer deux ou plusieurs variables chaînes ou une variable chaîne avec un entier, ce type d'association s'appelle de la **concaténation.**

Pour faire de la concaténation en Python on utilisera les caractères suivants le plus (+) ou la virgule (,) avec les syntaxes suivantes: `chaine1 + chaine2 ou chaine1,chaine2`

Il y'a une différence entre les deux types de caractères pour la concaténation:

- **La concaténation avec le plus**: ne met pas d'espace entre les deux variables. De plus, pour concaténer une chaine et un nombre il faut faire une conversion du nombre en chaine.
- **La concaténation avec la virgule:** ne marche que lorsqu'on utilise la fonction **print** ne pas l'utiliser en dehors du **print** car signifie autre chose non pas la concaténation mais la création de type **tuple**. Cependant à l'intérieur du **print** elle mettra un espace entre les éléments concaténés.

```
"""
    Concaténation nom et code postal
"""

# Initialisation des variables
nom = "Christophe"
postal = 29

# Avec le + sans espace
print(nom + "habite le" + str(postal))

# Avec le + avec espace
print(nom + " habite le " + str(postal))

# Avec la virgule
print(nom, "habite le", postal)
```

Exécution

```
Christophehabite le29
Christophe habite le 29
Christophe habite le 29
```

Concaténation avec le formatage

Une autre méthode aussi pour concaténer est d'utiliser le formatage de données de Python Documentation ici

L'avantage de ce type de concaténation est qu'on est pas obligé de convertir la variable et elle peut être réalisée de deux façons différentes:

- Commencer la chaine par un **f** ou **F** en mettant les variables entre accolades {} dans le message.
- Utiliser la méthode **string.format()** en mettant entre accolades {} à l'intérieur de la chaine l'ordre d'affichage des variables qui seront définies dans le format.

```
"""
    Concaténation avec formatage
"""

# Initialisation des variables
nom = "Christophe"
postal = 29

# Avec {}
print(f"Message 1: {nom} habite le {postal}")
print(F"Message 2: {nom} habite le {postal}")

# Avec string.format => l'ordre dans format a de l'importance
print("Message 3: {0} habite le {1}".format(nom, postal))
print("Message 4: {0} habite le {1}".format(postal, nom)) #=> Si on inverse on obtient un autre résultat

# Avec affichage du nom des variable
print(f"Message 5: {nom=} habite le {postal=}")
print(F"Message 6: {nom=} habite le {postal=}")
```

Exécution

```
Message 1: Christophe habite le 29
Message 2: Christophe habite le 29
Message 3: Christophe habite le 29
Message 4: 29 habite le Christophe
Message 5: nom='Christophe' habite le postal=29
Message 6: nom='Christophe' habite le postal=29
```

Pour plus d'informations sur les possibilités avec le formatage en Python lire Documentation

Exercices

Exercice 1

Sachant que **chaine[n]** permet de renvoyer le n-ième caractère de la chaine par exemple **chaine[0]** renverra le premier caractère de chaine pour nom="Toto", nom[0] est T et nom[3] le dernier « o ».
Sachant aussi que **len(chaine)** renvoie la taille d'une chaine exemple **len(nom)** renverra 4 si nom="Toto".

Ecrire un programme python équivalent à celui-ci en utilisant cette fois-ci la boucle **while.**

```
"""
    Ce programme affiche la liste des caractères d'un nom.
"""

name = "Christophe"

# Boucle sur la liste des caractères du nom
print("Le nom " + name + " contient les lettres suivantes:\n")
for letter in name:
    print(letter)
```

Résultat

```
Le nom Christophe contient les lettres suivantes:

C
h
r
i
s
t
o
p
h
e
```

Le script devra donner le même résultat à l'exécution.

Exercices

Exercice 2

Sachant que **chaine[n]** permet de renvoyer le n-ième caractère de la chaine par exemple **chaine[0]** renverra le premier caractère de chaine pour nom="Toto", nom[0] est T et nom[3] le dernier « o ».
Sachant aussi que **len(chaine)** renvoie la taille d'une chaine exemple **len(nom)** renverra 4 si nom="Toto".

Ecrire un programme python équivalent à celui-ci en utilisant cette fois-ci la boucle **while.**

```
"""
    Ce programme affiche la liste des caractères d'un nom.
    Il dira les caractères qui sont des voyelles ou non.
"""

name = "Christophe"

# Boucle sur la liste des caractères du nom
print("Le nom " + name + " contient les lettres suivantes:\n")
for letter in name:
    # Verification si la lettre est une voyelle ou non
    if letter in "AEIOUYaeiouyAEIOUYaeiouy":
        print("La voyelle:", letter)
    else:
        print("La consonne:", letter)
```

Résultat

```
Le nom Christophe contient les lettres suivantes:

La consonne: C
La consonne: h
La consonne: r
La voyelle: i
La consonne: s
La consonne: t
La voyelle: o
La consonne: p
La consonne: h
La voyelle: e
```

Le script devra donner le même résultat à l'exécution.

Exercices

Exercice 3

Ecrire un programme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

Exercice 4

Ecrire un programme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message: « Plus petit ! », et inversement « Plus grand ! » si le nombre est inférieur à 10.

Exercice 5

Ecrire un programme qui demande un nombre de départ à l'utilisateur et qui affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Exercice 6

Ecrire un programme qui demande un nombre de départ à l'utilisateur et qui calcule la somme des entiers jusqu'à ce nombre (inclus). Par exemple, si l'utilisateur entre 5, le programme doit calculer 1 + 2 + 3 + 4 + 5 = 15

Exercices

Exercice 7

Ecrire un algorithme qui demande un nombre à l'utilisateur et qui ensuite écrit la table de multiplication de ce nombre de 0 à 10.

Exercice 8

Ecrire un algorithme qui demande un nombre à l'utilisateur et qui calcule le factoriel de ce nombre. Par exemple si l'utilisateur saisit 8, le factoriel de 8 noté 8! vaut: 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 = 40320. Pour 3 on aura 3! = 3 x 2 x 1 = 6. Sachant aussi que 1! = 1 et 0! = 1.

Exercice 9

Soit un utilisateur de nom d'utilisateur christophe et mot de passe chris1234. Ecrire un programme qui va demander le nom d'utilisateur et le mot de passe tant que le nom d'utilisateur et mot de passe sont incorrects. Sinon Afficher « Bienvenue Christophe ».

Exercice 10

Faire une variance du programme 9 mais cette fois-ci demander le nom d'utilisateur tant que la saisie n'est pas correcte ensuite si correcte demander le mot de passe si la saisie est correcte afficher le message de bienvenue.

Exercices

Exercice 12

Soit un utilisateur dont le code de carte bancaire est 1234. Demander à l'utilisateur de saisir son code secret avec un nombre limite de 3 essais maximum. Au bout de 3 échecs arrêter le programme en affichant « Carte bloquée » sinon afficher « Paiement accepté ».

Exercice 13

Ecrire un programme qui demande une phrase ou mot à l'utilisateur et ensuite affichera la saisie de l'utilisateur en capitalisation avec en lettre majuscule le premier caractère et tout le reste en lettre minuscule.
Exemple l'utilisateur entre « TOTO » ou « toTO » le programme doit afficher « Toto ».

Exercice 14

Ecrire un programme qui demande successivement 20 nombres à l'utilisateur, et qui lui dise ensuite quel était le plus grand parmi ces 20 nombres exemple:
Entrer le nombre numéro 1: 16
…
Entrer le nombre numéro 20: 12
Le plus grand de ces nombres est 16.

Exercices

Exercice 15

Reprendre l'algorithme de l'exercice 14 mais cette fois-ci il faut afficher la position à laquelle le nombre a été saisi.
Par exemple: le plus grand de ces nombres est 15 et c'est le nombre 2 saisi.

Exercice 16

Reprendre l'exercice 15 mais cette fois-ci demander à l'utilisateur la quantité de nombres qu'il veut saisir et ensuite exécuter le programme en fonction de la quantité de nombres que l'utilisateur veut saisir. Attention au fausse saisie sur la quantité par exemple s'il saisit un nombre négatif ou 0 afficher une erreur et redemander combien de nombres il veut saisir.

FIN CHAPITRE 4