Boucles non bornées

1 - Rappel préalable : le module random Voir Variables Python 6.3.6

2 - Boucle non bornée tant que / while

2.1 Rappels Introduction Python 1.4.11

2.2 Bornée ou non bornée

— Si on connait à l'avance le nombre de fois où on va devoir agir, on peut utiliser une boucle POUR.
La boucle POUR est une boucle bornée.

— Si on ne peut pas savoir à l'avance le nombre de fois où on va devoir agir, on utilise une boucle TANT QUE.
C'est la condition de poursuite fournie qui va permettre de faire le choix de continuer ou pas.

— Si l'action peut potentiellement se dérouler à l'infini, il faut utiliser une boucle TANT QUE.
C'est pour cela que la boucle TANT est une boucle non bornée.

Avec l'aléatoire, on a donc souvent du TANT QUE.

2.3 Première évaluation de la condition

Les variables nécessaires à l'évaluation du TANT QUE doivent toutes être définies avant la ligne du TANT QUE.

Si la condition est basée sur une variable qui est calculée dans le bloc de la boucle elle-même, comment faire ? Initialiser cette variable avec une "fausse première valeur" de façon à rentrer dans la boucle au départ.

Exemple : en ligne 7, on place 0 dans total, ce qui permet de valider la condition de poursuite : le total est inférieur à 16.

```
1  import random
2
3  def de(nb) :
4      return random.randint(1, nb)
5
6  total = 0
7
8  while total < 16 :
9      total = de(6) + de(6) + de(6)
10     print(total)
11
12 print(f"Le total est {total}")
```

2.4 Condition d'arrêt ou de poursuite

`False` est la valeur complémentaire de `True`. On passe de l'une à l'autre à l'aide du mot-clé `not`.

A - Lien entre poursuite et arrêt Pour résoudre un problème avec un TANT QUE, on utilise une condition de `poursuite`.

```
while poursuite:
    agir()
```

On tombe parfois plutôt sur une condition d'arrêt de la boucle... Comment faire alors ? C'est facile : on trouve la condition de `poursuite` en inversant la condition d'arrêt avec `not`.

— Si `poursuite` est `True`, c'est que `arret` est `False`.
— Si `poursuite` est `False`, c'est que `arret` est `True`.

B - Deux écritures de la boucle TANT QUE Pour décrire un même TANT QUE, on a donc deux possibilités d'écriture :

— Si vous connaissez la condition de `poursuite` de la boucle :
— Si vous connaissez la condition d'`arret` de la boucle :

```
while poursuite:
    agir()
```

```
while not arret:
    agir()
```

C - Exemple Dire qu'on continue tant qu'on a moins de 2000 euros :

```
# version poursuite
while argent < 2000:
```

```
# version arrêt
while not argent >= 2000:
```

D - Cas typique qui cause des erreurs :

— L'opérateur complémentaire de > est <=
— L'opérateur complémentaire de >= est <

2.5 Boucle infinie

Pour créer une boucle qui ne s'arrête jamais, on choisit une condition de poursuite qui ne pourra jamais être évaluée à **False** : **True** !

```
while True:
    faire_un_truc()
    faire_un_autre_truc()
```

La condition de poursuite étant juste **True**, on va donc effectuer L2-L3 à l'infini.

Ce bouclage peut être voulu (comme sur un système embarqué qui ne doit jamais arrêter son programme) ou involontaire. Si c'est involontaire, c'est le drame car on ne pourra pas prendre la main à moins de stopper le programme.

3 - Exemple d'interaction

```
1  import random
2  MINI = 0
3  MAXI = 100
4
5  def affichage(m, p) :
6      if m  != p :
7          if p > m :
8              print("Raté, trop grand")
9          else :
10             print("Raté, trop petit")
11     else :
12         print("Bien vu !")
13
14 mystere = random.randint(MINI, MAXI)
15 proposition = -1
16 while not proposition == mystere :
17     proposition = input("A votre avis ? ")
18     proposition = int(proposition)
19     affichage(mystere, proposition)
20 print("Vous avez trouvé !")
```

4 - Transformation d'une boucle en l'autre

4.1 Transformer un POUR en TANT QUE

On peut remplacer les **for** par des **while**. La version **while** est plus complexe puisqu'on doit écrire ce que le **for** réalise en sous-main : initialiser la variable **x**, transformer **x in range(n+1)** en condition de poursuite **x < n+1**, incrémenter la variable **x** alors que c'est fait automatiquemnet avec **for**.

```
1  def somme_v1(n :int) -> int :
2      """Renvoie la somme de
3      1 + 2 + 3... jusqu'à n."""
4      reponse = 0
5      for x in range(n+1) : # on a donc x < n+1
6          reponse = reponse + x
7      return reponse
8
9  def somme_v2(n :int) -> int :
10     """Renvoie la somme de
11     1 + 2 + 3... jusqu'à n."""
12     reponse = 0
13     x = 0
14     while x < n+1 :
15         reponse = reponse + x
16         x = x + 1
17     return reponse
```

4.2 Transformer un POUR en TANT QUE

A - Cas des fonctions

Puisqu'on sort définitivement de la fonction une fois qu'on rencontre un **return**, on peut interrompre un POUR à l'intérieur d'une fonction.

On peut donc parfois programmer une boucle non bornée en utilisant un for associé à return plutôt que d'utiliser directement un while.

Exemple : deux versions d'une fonction qui renvoie le premier mot de plus de 5 lettres trouvés dans une phrase.

```
1  def premier_mot_v1(phrase) :
2      """Renvoie le premier mot de
3      plus de 5 lettres de la phrase,
4      un str NON VIDE"""
5      t = phrase.split(" ")
6      i = 0
7      while len(t[i]) < 5 :
8          i = i + 1
9      if  i < len(t) :
10         return t[i]
11
12 def premier_mot_v2(phrase) :
13     """Renvoie le premier mot de
14     plus de 5 lettres de la phrase,
15     un str NON VIDE"""
16     t = phrase.split(" ")
17     for i in range(len(t)) :
18         if len(t[i]) >= 5 :
19             return t[i]
```

B - Est-ce malin de faire cela ?

Tout dépend du point de vue.

AVANTAGE du **for+return** : en terme de programmation, on doit noter moins de choses en utilisant un **for** associé à **return**.

DESAVANTAGE du **for+return** : cela crée une fonction qui possède deux sorties : n'oubliez pas que si Python ne rencontre pas le return de la ligne 8, on sortira de la fonction comme si vous aviez tapé return None. Deux sorties. Algorithmiquement, c'est donc plutôt sale.

C - Pas de remplacement possible pour la boucle infinie

Pour remplacer le **fwhile** par un **for**, il faut qu'il existe un nombre maximal de fois où on réalise la boucle. Si ce nombre n'existe pas, c'est à dire qu'il est possible de tourner à l'infini sans jamais sortir, il est bien entendu impossible de remplacer le TANT par un POUR équivalent.

D - Hors d'une fonction ?

Ne pas utiliser en NSI

A titre d'exemple, voici un programme qui permet de stocker dans **mot** le premier de 5 lettres ou plus, s'il existe.

```
1  phrase = """Phrase permettant
2  de réaliser un exemple"""
3  mot = None
4  t = phrase.split(" ")
5  for i in range(len(t)) :
6      if len(t[i]) >= 5 :
7          mot = t[i]
8          break
```

4.3 Conclusion : BORNEE ou NON BORNEE

Avant de programmer une boucle, il convient de se demander si on a affaire à une boucle bornée ou une boucle non bornée car cela évitera de partir dans la mauvaise direction.

- Boucle bornée : **for**

- Boucle non bornée sans nombre maximum de tours prévisible : **while**

- Boucle non bornée avec un nombre maximum prévisible : **while** ou **for+return**