Partie 1

Boucle bornée

Definition : Une boucle bornée est système d'instructions qui permet de répéter un certain nombre de fois toute une série d'opérations. Les répétitions sont appelées des *itérations*.

La boucle bornée utilise les mots-clés `for` et `in`. La syntaxe de ce système est la suivante :

```
for + variant + in + iterable:
    # instructions du bloc indentée !!
```

Exemple 1 :

```
n = 10
for i in range(n):
    # inst 1
    # inst 2
```

Definition : iteration : c'est un cycle d'execution de la boucle

Definition : itérable : instruction qui définit l'ensemble des valeurs prises par le variant lors de l'execution de la boucle. Il y a 3 exemples, où l'itérable peut être une étendue de valeurs entières, un `str`, ou une `list` :

itérable	nombre de répétitions	séquence itérable
`range(10)`	10	{0, 1, 2, ..., 9}
`"abc"`	3	{"a", "b", "c"}
`[1998, 2018]`	2	{1998, 2018}

Definition : variant de boucle. A chaque *itération*, le variant prend une nouvelle valeurs dans un ensemble de valeurs, défini par l'**itérable**. Ce variant conserve sa valeur pendant l'execution du bloc de code indenté, sous l'instruction `for`. Puis change de valeur jusqu'à la sortie de la boucle.

La fonction `range(n)` établit une liste de valeurs itérables, que prend le variant `i` qui est déclaré dans la boucle : `i` vaut successivement {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, de 0 à `n-1`.

Partie 2

Boucles non bornées

1. Definitions **Definition : Une boucle non bornée** est système d'instructions qui permet de répéter toute une série d'opérations tant que la `<condition d'execution>` est `True`. Le nombre de répétitions est à priori inconnu.

Syntaxe : On écrit l'instruction : `while <condition d execution>:` Puis le bloc de code est indenté sous cette première ligne. Il y a 3 points clés dans la construction de ce systeme d'instructions.

Le variant de boucle : le variable utile pour la condition d'execution.

Souvent, il sera nécessaire de commencer le programme par initier une variable, le variant de boucle. De sorte que cette condition d'execution soit évaluée à `True`, et que le bloc de cette boucle s'execute au moins une fois.

Résumé par l'exemple :

- declaration du variant
- `while + condition`
- bloc avec l'une des instruction qui modifie le variant, le faisant evoluer vers la condition d'arrêt.

```
variant = valeur init
while <condition d execution>:
  instruction 1
  instruction 2
  instruction qui modifie le variant
instruction suivante # suite du programme
```

Lorsque le programme arrive à la dernière ligne de l'indentation, il revient à l'instruction `while`. Il réalise alors une **itération**.

Definition : **condition d'execution** : Le test sur la `<condition>` est réalisé avant chaque nouvelle *itération*. Lorsque cette `<condition>` n'est plus réalisée, le programme passe à l'*instruction* qui suit le bloc. *(sortie de la boucle)*

Definition : **condition d'arrêt** c'est la condition opposée à celle de la condition d'execution. Exemples...

condition d'execution	condition d'arrêt
`i > 0`	`i <=0`
`i <= 0`	`i > 0`
`A==True and B==True`	`A==False or B==False`
`k > 0 and i>j`	`k<=0 or i<=j`

Equivalence `for` et `while`

Les scripts suivants sont équivalents :

```
for i in range(10):
    print(i*2)

i = 0
while i < 10:
  print(i*2)
  i = i + 1
```

2. Détailler un exemple *Exemple 2 : Réaliser un compteur*

```
print("Donner les prénoms des 3 neveux de Donald Duck")
i = 1
```


Exercices

```
while i <= 3:
  nom = input("neveu n°"+str(i)+": ")
  i = i + 1
print("c'est fini")
```

- à la première itération, i vaut 0, donc la condition `i <= 3` est évaluée à `True` et le bloc est executé. l'utilisateur est invité à entrer le premier nom, (*il va certainement entrer Riri*), et i finit avec la valeur 1 (`i = i + 1`)
- La boucle se poursuit jusqu'à ce que i soit égal à 4. Alors `i <= 3` est évaluée à `False` et le programme poursuit APRES la boucle, avec la dernière instruction : affiche `"c'est fini"`

Exemple 2 : Soustractions multiples

```
b = 40
while r >= 3:
  r = r - 3
print('à la fin du programme, r vaut ' + str(r))
```

- A la première itération, `r` vaut 40 donc la `<condition>` `r > 3` est `True`. `r` est diminué de 3 et prend la valeur 37.
- A la dernière itération, `4 > 3` est evalué à `True`. `r` est diminué de 3 et prend la valeur 1
- Puis `r > 3` est evalué à `False`. Le bloc n'est pas executé et le programme s'arrête s'il n'y a pas d'autres instructions (ou poursuit le script sinon).

> Executer ce programme. Comparer la valeur de `r` avec `40%3`. Conclure.

3. Le problème de l'arrêt Pour le script précédent, remarquer que la variable testée est `r`, dans la condition d'execution `r > 3`. Le bloc executé à chaque itération doit alors modifier la valeur de `r`, et rapprocher sa valeur de la condition d'arrêt, `r<3`

C'est le problème avec les boucles non bornées. Celles-ci peuvent ne pas finir, ce qui peut bloquer la machine.

Cet effet de boucle *infini* peut être recherché, par exemple en robotique, où l'on veut que le programme se poursuive indéfiniment. Ainsi, la structure d'un programme *python* sur carte microbit commence par la structure suivante :

```
from microbit import *

while True:
  # instructions
  # ...
```


Exercices