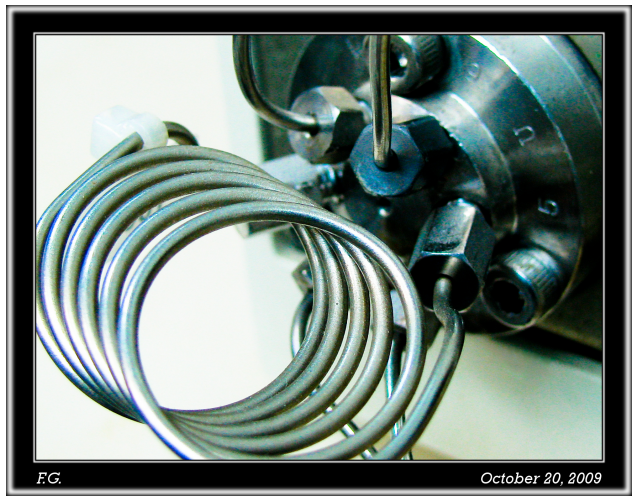
Boucles définies



Skander Zannad et Judicaël Courant

Lycée La Martinière-Monplaisir

2013-09-08

Lire chapitre 3 et 4.

1 Un programme très simple

Écrivons un programme pour saluer la classe (disons les 8 premiers élèves) :

```
#! /usr/bin/python

print('Bonjour␣Doriann')
print('Bonjour␣Silvio')
print('Bonjour␣Corentin')
print('Bonjour␣Brice-Edine')
print('Bonjour␣Lucie')
print('Bonjour␣Jean-Charles')
print('Bonjour␣Ilyas')
print('Bonjour␣Marie')
```

2 Mais peu facile à faire évoluer

Combien de travail faut-il faire si :

- On veut dire bonsoir et non bonjour ?
- On veut dire «Doriann, comment vas-tu ? *etc.*» et non «Bonjour Doriann, *etc.*» ?

Et s'il y a 500 élèves ?

3 Le principe DRY

«**D**on't **R**epeat **Y**ourself»

> «Dans un système, toute connaissance doit avoir une représentation unique, non-ambiguë, faisant autorité ».

«Once and only once»

Ici le programme devrait :

- définir la liste des prénoms auxquels dire bonjour ;
- dire qu'on veut effectuer un même traitement sur tous les prénoms ;

– dire que ce traitement consiste à afficher «Bonjour» suivi du prénom.

3.1 Listes

En Python, on peut définir des listes d'objets.

- de tout type ;
- éventuellement hétérogènes.

Liste de chaînes de caractères :

```
prenoms = [ 'Doriann', 'Silvio',
  'Corentin', 'Brice-Edine',
  'Lucie', 'Jean-Charles',
  'Ilyas', 'Marie' ]
```

Liste hétérogène (entier et chaînes) :

```
liste = [ 1, 'toto', 42 ]
```

3.2 Itération définie

Itérer : répéter une action, un calcul

Itération définie/boucle définie : on itère un traitement sur une liste de valeurs fixée à l'avance.

En python : introduite par la construction

for *variable* **in** *liste* :
␣␣␣␣*instruction 1*
␣␣␣␣*instruction 2*
⋮
␣␣␣␣*instruction* n

```
#! /usr/bin/python

prenoms = [ 'Doriann', 'Silvio',
   'Corentin', 'Brice-Edine',
   'Lucie', 'Jean-Charles',
   'Ilyas', 'Marie' ]

for x in prenoms:
    print('Bonjour ' + x)
```

Combien de travail faut-il faire si :

- On veut dire bonsoir et non bonjour ?

– On veut dire «Doriann, comment vas-tu ? *etc.*» et non «Bonjour Doriann, *etc.*» ?

Et s'il y a 500 élèves ?

4 Itération sur des entiers

Calculer la somme des carrés des entiers de 1 à ~~5000~~ 5 :

```
s = 1**2 + 2**2 + 3**2 + 4**2 + 5**2
```

Peut-on transformer ça en boucle ?

Il suffit de voir que ça s'écrit :

```
s = 0
s += 1 ** 2
s += 2 ** 2
s += 3 ** 2
s += 4 ** 2
s += 5 ** 2
```

D'où la solution :

```
s = 0
for i in [1, 2, 3, 4, 5]:
    s += i ** 2
```

Intervalles d'entiers

Liste des éléments de $[\![a, b[\![$: `range`(a,b)

Attention : intervalle **fermé** à gauche, **ouvert** à droite [1].

Cas particulier : $[\![0, n[\![$: `range`(n)

(voir aide en ligne de range)

```
s = 0
for i in range(1, 6):
    s += i ** 2
```

1. Voir *Why numbering should start at zero*, E. W. Dijkstra, EWD831. Disponible en ligne.

Somme des carrés de 1 à n :

```
s = 0
for i in range(1, n + 1):
    s += i ** 2
```

Marche aussi avec $n = 5000\ldots$

5 Utilisation classique

On définit une suite u par $u_0 = 1$ et

$$\forall n \in \mathbb{N} \quad u_{n+1} = f(u_n)$$

où $f : x \mapsto \sqrt{x^2 + x}$.

On pose $N = 100\,000$. Calculer et afficher une approximation de u_N.

Programme proposé

```
#! /usr/bin/python
from math import sqrt

N = 100000
x = 1
for i in range(N):
    x = sqrt(x ** 2 + x)
print(x)
```

Correction du programme

À la fin, la variable `x` vaut-elle bien u_N ? ou u_{N-1} ? ou u_{N+1} ? ou ...

Cette question :

- Est essentielle ;
- Sera posée par tout lecteur du code.

Il ne suffit pas d'écrire un code, il faut indiquer au lecteur ce qu'il fait et pourquoi il est correct.

Donc on *documente* le code avec un *invariant* de boucle.

Invariant d'un programme : proposition mathématique, liée à un point du programme, toujours vraie lorsque l'exécution en est à ce point du programme.

```
N = 100000
x = 1
for i in range(N):
    # ici x vaut u indice i
    x = sqrt(x ** 2 + x)
    # ici x vaut u indice i+1
```

Reste à vérifier :

1. Vérifier que ces invariants sont corrects ;
2. Vérifier qu'on peut en déduire qu'à la fin, `x` vaut u_N.

Vérifier qu'un invariant de boucle est correct :

- Vérifier que la proposition est vraie au premier tour de boucle ;
- Vérifier que si elle vraie à un tour donné, elle l'est au tour suivant.

Ici : vérifier que ça marche.

En pratique

On ne met que le premier invariant et on documente le programme :

```
# Ce programme calcule et affiche u(N) avec
# u : suite donnée au 2e cours info MPSI LMM
N = 100000


x = 1
for i in range(N):
    # x == u(i)
    x = sqrt(x ** 2 + x)
print(x)
```