Plan du chapitre

Un premier exemple

Comment obtenir cet affichage?

```
Pyhtagore?
Pyhtagore?
Pyhtagore?
Pyhtagore?
Pyhtagore?
Thalès, alors ?
```

L'idée consiste à utiliser une structure qui permet de répéter plusieurs fois un bloc d'instructions : une boucle **for**.

▷ **Algorithme** :

pour *i* allant de 1 à 5 **faire**

Afficher `Pythagore ?`

Afficher `Thalès, alors ?`

▷ **Programme** :

```
for i in range(1,6):
        print('Pythagore␣?')
print('Thalès␣alors␣?')
```

I Structure d'une boucle for

I.1 Structure générale

La syntaxe d'une boucle **for** est très simple :

for *element* **in** *iterable* :
    <instr 1>
    ...
    <instr n>
<ensuite>

iterable désigne un objet itérable, c'est-à-dire un objet de type composé dont on peut énumérer les éléments qui le constituent : *iterable* peut donc être un n-uplet, une chaîne de caractères, une liste...

Pour chaque élément de *iterable*, la suite des instructions <instr 1>,...,<instr n> est réalisée. L'indentation marque le bloc exécuté pour chaque élément de *iterable*.

Une fois que tous les éléments de *iterable* ont été passés en revue, Python passe à la suite.

Exemple 1

```
for x in 'Exemple 1':
    print(x)
```

```
E
x
e
m
p
l
e

1
```

Exemple 2

```
for x in [4,8,9]:
    print(2*x)
```

```
8
16
18
```

Attention danger !
Bien faire attention à l'indentation qui marque le bloc d'instructions à réaliser à chaque itération.

```
for i in range(1,6):
        print('Pythagore ?')
        print('Thalès, alors ?')
```

```
Pythagore ?
Thalès, alors ?
Pythagore ?
Thalès, alors ?Allo ?
Pythagore ?
Thalès, alors ?
Pythagore ?
Thalès, alors ?
```

```
Pythagore ?
Thalès, alors ?
```

I.2 La fonction range

Afin de parcourir une suite rangée de nombres, on pourrait penser, de prime abord, qu'il suffit de construire une liste, un tuple ou un chaîne de caractère contenant ces nombres.
Mais les concepteurs de Python ont pensé à tout et, pour éviter la construction de cette liste - qui peut occuper un espace mémoire important si elle est très longue - ont créé l'outil **range** : il permet, comme le ferait une liste, de parcourir les nombres entiers dans l'ordre croissant, avec un pas défini par l'utilisateur mais sans l'inconvénient de l'occupation mémoire.

Sa syntaxe est très simple :

range(start,stop,step)

Dans cette syntaxe :

- **start** désigne le premier nombre de la suite. Il n'est pas indispensable d'indiquer ce nombre, qui est alors pris égal à 0 par défaut.
- **stop** désigne le dernier nombre de la suite **qui ne sera pas itéré**.
- **step** désigne le **pas** avec lequel cette suite sera parcourue. Il n'est pas obligatoire d'indiquer le pas, qui est par défaut pris comme égal à 1.

Ainsi, la fonction **range** peut prendre un seul argument (**stop**), deux arguments (**start**, **stop**), ou trois arguments (**start**, **stop**, **step**).

Ecrivons alors quelques programmes simples :

- 1er programme : afficher les nombres entiers de 0 à 9.

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

- 2nd programme : afficher les nombres pairs compris entre 0 et 20.

```
for i in range(0,21,2):
                print(i)
```

- 3ème programme : écrire la table de multiplication d'un nombre entier quelconque.

```
n=int(input("Entrez un entier positif:"))
for i in range(1,11):
                print(n,"*",i,"=",n*i)
```

I.3 Premiers exercices

Entrainement 1 : calcul de $n!$

Ecrire un programme qui demande à l'utilisateur un entier naturel n et qui calcule $n!$.

```
n=int(input('Un entier naturel svp?'))
A=1
for i in range(1,n+1):
    A=A*i
print('La factorielle de ',n,' vaut: ',A)
```

Savoir-faire : concevoir une boucle for

Pour concevoir une boucle **for** correcte, il faut impérativement :

- bien réfléchir à l'initialisation des variables et au nombre d'itérations de la boucle,
- déterminer le contenu de toutes les variables à l'issue des premières itérations de la boucle.

Dans l'exemple précédent,

- avant d'entrer dans la boucle, la variable A contient 1 ;
- à la fin de la première itération de la boucle (itération où i vaut 1), A contient 1×1 ;
- à la fin de la deuxième itération de la boucle (itération où i vaut 2), A contient $1\times 1\times 2$;
- à la fin de la troisième itération de la boucle (itération où i vaut 3), A contient $1\times 1\times 2\times 3$;

  ...

- à la fin de la $n^{\text{ème}}$ et dernière itération (i vaut alors n), A contient$1\times 1\times 2\times 3\times ...\times n = n!$.

Remarque : si l'utilisateur rentre la valeur 0 pour n, **range(1,n+1)** ne contient aucun élément, la boucle **for** ne réalise aucune itération, et le programme affiche la valeur de **A** égale à 1, ce qui correspond bien à 0!.

Entrainement 2 : une suite récurrente

On considère la suite définie par récurrence par $u_0 = 3$ et $\forall n \in \mathbb{N}, u_{n+1} = 2u_n - 4$.

Ecrire un programme qui demande à l'utilisateur un entier naturel n et qui calcule u_n.

```
n=int(input('Un entier naturel svp?'))
A=3
for i in range(1,n+1):
```

```
    A=2*A-4
print("Le terme de la suite d'indice ",n," vaut :",A)
```

Dans cet exemple,

- avant d'entrer dans la boucle, la variable A contient le terme u_0 ;
- à la fin de la première itération de la boucle (itération où i vaut 1), A contient $2u_0 - 4$, i.e. u_1 ;
- à la fin de la deuxième itération de la boucle (itération où i vaut 2), A contient u_2 ;
- à la fin de la troisième itération de la boucle (itération où i vaut 3), A contient u_3 ;
- ...

Cette boucle ayant en tout n itérations (i vaut n lors de la dernière itération), A contient à la fin de la dernière itération u_n.

II Savoir compter, sommer, et lister des objets

II.1 Compter et sommer des objets

Exemple 3

Comment compter les diviseurs de 1000? Et comment les sommer?

Un algorithme pour compter

$c \leftarrow 0$ # un compteur
pour i allant de 1 à 1000 **faire**
    si i est un diviseur de 1000 **alors**
        $c \leftarrow c + 1$
Résultat : c

Un programme pour compter

```
c=0
for i in range(1,1001) :
    if 1000%i==0 :
        c=c+1
print(c)
```

Un algorithme pour sommer

$s \leftarrow 0$ # un sommeur
pour i allant de 1 à 1000 **faire**
    si i est un diviseur de 1000 **alors**
        $s \leftarrow s + i$
Résultat : s

Un programme pour sommer

```
s=0
for i in range(1,1001) :
    if 1000%i==0 :
        s=s+i
print(s)
```

Entrainement 3 : compter les voyelles

Ecrire un programme qui demande à l'utilisateur une chaîne de caractères et qui affiche le nombre de voyelles de cette chaîne de caractères.

```
A=input('Une chaine de caractères svp:')
c=0   #le compteur
for x in A:
    if x in 'aeiouy': #test d'appartenance
        c+=1
print('Le nombre de voyelles est ',c)
```

Entrainement 4

Ecrire un programme qui permet d'obtenir le nombre d'entiers naturels compris entre 0 et 800 dont l'écriture décimale ne comporte pas le chiffre 4.

```
C=0  #le compteur
for n in range(0,801):
    ch=str(n) #ch est la chaine de caractères qui représente l'entier n
    if '4' not in ch: #test d'appartenance entre chaînes de caractères
        C+=1
print(C)
```

Entrainement 5

Ecrire un programme qui demande à l'utilisateur un entier naturel $n \geqslant 1$ et qui calcule la somme

$$\frac{6}{\pi^2}\sum_{k=1}^{n}\frac{1}{k^2}.$$

Quelle conjecture peut-on faire?

```
n=int(input('Un entier naturel n>=1 svp:'))
S=0 #le sommeur

for k in range(1,n+1):
    S=S+1/k**2

```

```
#A la fin de cette boucle for, S contient 1+1/2**2+...+1/n**2.

import math

print(6/math.pi**2*S)
```

```
Un entier naturel n>=1 svp:10000
0.9999392103293521
```

On peut conjecturer que $\lim\limits_{n\to+\infty}\sum\limits_{k=1}^{n}\dfrac{1}{k^2}=\dfrac{\pi^2}{6}$.

Entrainement 6 : analyse d'un script

Après l'exécution du programme suivant, que contient n ?

```
1 n=0
2 for i in range(1,100):
3     x=0
4     for k in range(1,i+1):
5         x=x+k*k
6     x=x/i
7     if int(x)==x:
8         n=n+1
9 print(n)
```

n joue le rôle d'un compteur, initialisé à 0.

Considérons la boucle **for** extérieure qui commence ligne 2 : elle comporte 99 itérations, i prenant les valeurs successives 1,2,...,99.

Examinons le déroulement de la i-ème itération de cette boucle. A la fin de la boucle **for** intérieure (qui commence ligne 4), le sommeur **x** contient la somme $\sum\limits_{k=1}^{i}k^2$.

L'instruction **x=x/i** remplace le contenu de **x** par $\dfrac{1}{i}\sum\limits_{k=1}^{i}k^2$.

Le booléen **int(x)==x** teste si la valeur de **x** est entière, autrement dit si i divise $\sum\limits_{k=1}^{i}k^2$. Si c'est le cas, le compteur **n** est incrémenté.

Conclusion : le programme affiche le nombre d'entiers i compris (au sens large) entre 1 et 99 pour lequels i divise $\sum\limits_{k=1}^{i}k^2$.

II.2 Lister des objets

Exemple 4

Comment construire la liste des diviseurs de 1000?

Un algorithme pour lister

$L \leftarrow []$ # un listeur
pour i allant de 1 à 1000 **faire**
    si i est un diviseur de 1000 **alors**
        Ajouter i au bout de la liste L
Résultat : L

Un programme pour lister

```
L=[]
for i in range(1,1001) :
    if 1000%i==0 :
        L.append(i)
print(L)
```

La **méthode de liste** (la notion de méthode sera expliquée plus tard) **append** permet d'ajouter un élément à la fin d'une liste. Par exemple :

```
>>> L=[1,2,3]
>>> L.append(456)
>>> L
[1, 2, 3, 456]
```

Entrainement 7

Ecrire un programme qui détermine la liste de tous entiers naturels à quatre chiffres (ne commençant pas par 0), qui sont égaux à la somme des puissances quatrièmes des chiffres les constituant.

```
L=[] # un listeur
for n in range(1000,10000):#la dernière valeur prise par n est 9999.
    ch=str(n) #ch est la chaine de caractères représentant l'entier n.
    a=int(ch[0]) #a est le premier chiffre de l'écriture de n en base 10.
    b=int(ch[1])
    c=int(ch[2])
    d=int(ch[3])
    if n==a**4+b**4+c**4+d**4:
        L.append(n)
print(L)
```

```
[1634, 8208, 9474]
```

III Savoir prouver la correction d'une boucle à l'aide d'un invariant de boucle

Lorsque l'on a écrit un programme, il reste à vérifier qu'il est correct, c'est-à-dire qu'il calcule bien ce que l'on attend ; on peut tester quelques cas significatifs, mais il est beaucoup plus satisfaisant de démontrer qu'il est correct dans tous les cas.

Une des manières les plus efficaces est d'établir un **invariant de boucle**, c'est-à-dire une propriété qui est vérifiée tout au long de l'exécution d'une boucle. Cette démarche est à rapprocher du raisonnement par récurrence : en ne s'intéressant qu'aux valeurs initiales et à leur évolution au cours d'une seule itération, on peut en déduire des propriétés valides quel que soit le nombre d'itérations et donc de connaître la valeur des variables en jeu à la fin de la boucle.

Savoir-faire : démontrer qu'une boucle produit l'effet attendu
On utilise un invariant de boucle, c'est-à-dire une propriété portant sur le contenu des variables et qui :

- est vérifiée avant d'entrer dans la boucle,
- si elle est vérifiée avant une itération de la boucle, est vérifiée après celle-ci,
- lorsqu'elle est vérifiée en sortie de boucle permet d'en déduire que le programme est correct.

Exemple 5

Montrer qu'après l'exécution du script suivant, r_1 et r_2 valent respectivement 841^{42} et $841!$.

```
r1, r2 = 1, 1
for i in range(42):
    r1 = r1*841
for i in range(2,842):
    r2 = r2*i
```

- Occupons-nous d'abord de la première boucle **for**, qui comporte 42 itérations. Pour cela, on montre que la propriété :

$$\mathscr{P}_k : \text{« à la fin de la } k^{\text{ème}} \text{itération de la boucle, la variable } \mathbf{r1} \text{ contient } 841^k \text{ »}$$

est vraie pour tout $k \in [\![0,42]\!]$.

  - ▷ à la fin de l'initialisation, juste avant de rentrer dans la boucle **for**, la variable r_1 contient 1, qui vaut bien 841^0. La propriété $\mathscr{P}_0$ est donc vraie.

▷ Supposons la propriété $\mathscr{P}_k$ vraie pour un entier $k \in [\![0,41]\!]$. Comme **r1** contient 841^k à l'issue de la $k^{\text{ème}}$ itération, **r1** va contenir $841^k \times 841 = 841^{k+1}$ à la fin de l'itération suivante : la propriété $\mathscr{P}_{k+1}$ est par conséquent vraie.

▷ On prouvé :

$$\mathscr{P}_0 \qquad \text{et} \qquad \forall k \in [\![0,41]\!], \mathscr{P}_k \Longrightarrow \mathscr{P}_{k+1}.$$

Par récurrence, la propriété $\mathscr{P}_{42}$ est donc vraie : à la fin de la boucle, **r1** contient bien 841^{42}.

- La seconde boucle **for** comporte 840 itérations. Notons tout de suite que, lors de la $k^{\text{ème}}$ itération, **i** prend la valeur $k+1$.

  Montrons que la propriété :

$$\mathscr{P}_k : \text{« à la fin de la } k^{\text{ème}}\text{itération de la boucle, la variable } \mathbf{r2} \text{ contient } (k+1)! \text{ »}$$

  est vraie pour tout $k \in [\![0,840]\!]$.

  ▷ Lors de l'initialisation, **r2** contient 1 qui est bien la valeur de 1!.

  ▷ Supposons la propriété $\mathscr{P}_k$ vraie pour un entier $k \in [\![0,839]\!]$. Lors de la $(k+1)^{\text{ème}}$ itération de la boucle, i vaut $k+2$, et donc le contenu de **r2** est changé en $(k+1)! \times (k+2) = (k+2)!$: la propriété $\mathscr{P}_{k+1}$ est par conséquent vraie.

  ▷ On prouvé :

$$\mathscr{P}_0 \qquad \text{et} \qquad \forall k \in [\![0,839]\!], \mathscr{P}_k \Longrightarrow \mathscr{P}_{k+1}.$$

  Par récurrence, la propriété $\mathscr{P}_{840}$ est donc vraie : à la fin de la boucle, **r2** contient bien 841!.