

Entrée[]: 1 `import numpy as np`



TP Python 3 : boucles bornées et non bornées



Les boucles en informatique permettent de **répéter une instruction autant de fois que nécessaire**.

Il y a **deux types de boucles** en informatique :

1. les boucles **bornées**, ou **boucles *for***, où l'on précise dès le début **combien de fois il faut répéter l'opération**,
2. **et les boucles non bornées**, ou **boucles *while***, où l'on donne une **condition d'arrêt**.

Partie 1 : Prérequis sur Range et les listes

Une **liste** est une collection **ordonnée** d'éléments (on parle aussi de tableau, dit *array*). On peut coder des listes (ou tableaux) dans Python en utilisant les crochets. Leur type est : `list`.

Entrée[]: 1 `L = [3,2,7]`

Pour a et b deux entiers, `range(a, b)` permet d'obtenir la liste des nombres entre a et $b - 1$ sous forme de liste. `range(b)` est un raccourci désignant `range(0, b)`. On peut aussi utiliser `range(a, b, c)` pour préciser le pas (l'écart entre les valeurs). Ces listes sont fréquemment utilisées et ont un type spécialisé : le type `range`.

Entrée[]: 1 `list(range(2,9))`

Entrée[]: 1 `list(range(9))`

Entrée[]: 1 `list(range(2,15,3))`

Nous détaillons peu les listes ici, car nous y reviendrons dans un futur TP, mais il s'agit d'une structure importante. Quelques commandes sur les listes :

1. `L[k]` renvoie le k -ième élément de la liste `L`

ATTENTION! La numérotation des indices commence à 0 : on dit dans le

jargon que *array starts at 0*. Selon le langage de programmation utilisé, les listes sont initialisées soit à 0, soit à 1.

2. `len(L)` renvoie le nombre d'éléments de la liste `L`.
3. `L.append(a)` rajoute l'élément `a` à la fin de la liste `L`.

```
Entrée[ ]: 1 L = [1,23,7,2]
           2 print(len(L))
           3 L.append(7)
           4 L.append(3)
           5 print(L)
           6 print(len(L))
```

Partie 2 : Boucle bornée *for*

La boucle `for` permet de répéter une commande pour chaque valeur d'une variable muette dans un ensemble, de manière similaire au symbole \sum en mathématique où la commande que l'on répète est une addition. La structure est la suivante :

```
Entrée[ ]: 1 for variable in liste:
           2     instruction1
           3     instruction2
           4     instruction3
```

Exemple :

```
Entrée[1]: 1 for k in range(1,10):
           2     print(k)
```

```
1
2
3
4
5
6
7
8
9
```

Les mots clefs sont `for` et `in`. Comme en mathématique, la variable muette n'a pas forcément à intervenir dans les commandes à répéter. On attire l'attention sur les deux choses suivantes :

1. **Il ne faut pas oublier** : dans le code.
2. **L'indentation est primordiale**. C'est elle qui signifie à Python comment sont organisés les blocs de codes à exécuter et leur agencements entre eux, à la manière des parenthèses pour organiser un calcul en mathématiques.

Exemple :

Pourquoi les codes suivants ne fonctionnent-ils pas comme attendu ?

```
Entrée[2]: 1 # dans cet exemple, on souhaitait augmenter une valeur de 1 cinq  
2  
3 l = 10  
4 for k in range(3,8):  
5     l = l+1  
6     print(l)
```

```
11  
12  
13  
14  
15
```

```
Entrée[ ]: 1 for k in range(3,8)  
2     print(k)
```

```
Entrée[ ]: 1 l = 10  
2 for k in range(7):  
3     a = l+1  
4     a = a+1  
5 l
```

Exemple :

Que renvoie le code suivant ?

```
Entrée[2]: 1 a = 0  
2 for k in range(50):  
3     a += k  
4 a
```

Sortie[2]: 1225

L'ensemble qui suit le mot `in` est donc celui où la variable muette prend ses valeurs. On peut utiliser pour cela la commande `range` comme on a vu, mais on peut aussi utiliser une liste de manière générale.

```
Entrée[ ]: 1 x = 0  
2 for s in [3,5,9,7]:  
3     x = x+s  
4 x
```

```
Entrée[ ]: 1 for j in "test": # un exemple pour constater que cela fonctionn
2         print(j)
```

Partie 3 : Boucle non bornée *while*

Avec une boucle *while*, on répète la ou les commandes associées tant que la condition d'arrêt n'est pas vérifiée.
Donnons deux exemples :

```
Entrée[3]: 1 k = 0
2 while k < 10:
3     print(k)
4     k += 1
5 k
```

```
0
1
2
3
4
5
6
7
8
9
```

Sortie[3]: 10

```
Entrée[4]: 1 k = 2
2 while 3*k <= 14:
3     k = k+1
4 k
```

Sortie[4]: 5

Quelques remarques :

1. Il ne faut pas oublier `:` dans le code.
2. L'indentation est là encore primordiale!
3. **La condition d'arrêt est cruciale.** Si la condition d'arrêt est mal choisie, le code pourrait s'arrêter au mauvais moment ou ne jamais s'arrêter. Dans ce dernier cas, on parle de **boucle infinie**, ce qui fait planter votre machine.

Exemple :

Quel est le plus petit multiple de 73 supérieur à 4000 ?

```
Entrée[ ]: 1 a = 0
           2 while a < 4000:
           3     a = a+73 # on parcourt les multiples de 73
           4 a
```

```
Entrée[ ]: 1 a = 4000
           2 while a%73 != 0: # on part de 4000 et on s'arrête dès qu'on
           3     a = a+1
           4 a
```

Remarque :

On peut utiliser la commande `break` pour sortir d'une boucle, mais nous n'en aurons pas besoin.

Exercices

```
Entrée[ ]: 1 import numpy as np
```

Exercice 1

1. Coder un programme demandant à l'utilisateur la valeur de n (avec ``input``) et qui renvoie en sortie la somme des nombres entiers entre 1 et n .
2. Faire la même chose en utilisant plutôt une fonction informatique

```
Entrée[ ]: 1
```

Exercice 2

Calculer le produit $123 \times 124 \times \dots \times 212 \times 213$.

```
Entrée[ ]: 1
```

Exercice 3

Créer une fonction ``fact(n)`` permettant de calculer $n!$.

```
Entrée[6]: 1
```

```
Sortie[6]: 1
```

Exercice 4

1. Créer une fonction qui prend en entrée un entier n et renvoie :

$$u_n = \prod_{k=2}^n \left(1 - \frac{1}{k^2}\right).$$

2. Tester cette fonction pour des valeurs de n et conjecturer le sens de variation de la suite.
3. Prouver cette conjecture (sans Python).

Entrée[]: 1

Exercice 5

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

1. Écrire une fonction permettant de calculer la valeur du terme d'indice n de cette suite.
2. Modifier la fonction pour qu'elle affiche les 20 premiers termes de la suite.
3. Conjecturer la limite de cette suite puis prouver votre conjecture (sans Python).
4. Écrire une fonction permettant de calculer la somme des n premiers termes de la suite.

Entrée[]: 1

Exercice 6

On considère la suite définie par $u_0 = 1, u_1 = -1$ et :

$$\forall n \in \mathbb{N}, u_{n+2} = 4u_{n+1} - 4u_n.$$

1. Écrire une fonction permettant de calculer la valeur du terme d'indice n de cette suite.
2. Conjecturer la limite de cette suite.
3. Écrire une fonction `S(n)` permettant de calculer la somme $S(n) = \sum_{k=0}^{n-1} u_k$.

Entrée[]: 1

Exercice 7

1. Programmer une fonction `aux(a,b)` prenant en entrée deux entiers naturels a, b avec $a \leq b$ et renvoyant le nombre maximal de fois que l'on peut soustraire a à b sans obtenir une quantité négative.
Par exemple, pour $a = 3$ et $b = 23$, la réponse est 7.
2. Coder une fonction prenant en entrée une somme d'argent n (entière) exprimée en euros, et renvoyant une décomposition de cette somme en un nombre minimal de pièces et de billets.
Par exemple, pour $n = 543$ euros, la décomposition sera : 1 billet de 500, 2 billet de 20, 1 pièces de 2, 1 pièce de 1.

Entrée[]:

1

Exercice 8

On considère la suite définie par $u_0 = 2$ et $u_{n+1} = 2u_n + 1$.
Quel est le rang du premier terme supérieur à 1 000 000 ?

Entrée[]:

1

Exercice 9

Déterminer le rang du dernier terme strictement positif de la suite $(u_n)_{n \in \mathbb{N}}$ définie par $u_0 = 890$ et pour tout $n \in \mathbb{N}$, $u_{n+1} = \frac{u_n}{2} - 3n$.

Entrée[]:

1

Exercice 10

Un appartement vaut 100000 euros. Sa valeur est multiplié par 1.2 chaque année.
Au bout de combien d'années la valeur de l'appartement aura doublé ? (on résoudra le problème informatiquement mais aussi mathématiquement, sans Python)

Entrée[]:

1

Exercice 11

Que fait le programme suivant ?

```
Entrée[ ]: 1 def precision(epsilon):
2           u = 14
3           n = 0
4           while abs(u-4) > epsilon:
5               u = 6-0.5*u
6               n = n+1
7           return n
```

Entrée[]: 1

Exercice 12

Renvoyer la plus petite valeur de $n \in \mathbb{N}$ tel que $1 - \left(\frac{5}{6}\right)^n \geq 0.99$.

Entrée[]: 1

Exercice 13

Écrire une fonction dont l'entrée est un entier positif n et la sortie est le nombre de fois que cet entier est divisible par 2.

Entrée[]: 1

Exercice 14

On considère la suite de Syracuse $(x_n)_{n \in \mathbb{N}}$ définie par la valeur de son premier terme x_0 et par :

$$\forall n \in \mathbb{N}, x_{n+1} = \begin{cases} \frac{x_n}{2} & \text{si } x_n \text{ est pair} \\ 3x_n + 1 & \text{sinon} \end{cases}$$



La conjecture de Syracuse dit : $\forall x_0 \in \mathbb{N}^*, \exists n \in \mathbb{N}, x_n = 1$.

1. A. Écrire une fonction `Syracuse(x0)` qui renvoie le premier indice n tel que $x_n = 1$ en fonction de la valeur de x_0 .
B. Tester votre fonction en $x_0 = 27$.
2. Vérifier la conjecture de Syracuse pour les 20 premiers entiers x_0

Entrée[]: 1

Fin du TP

- **Effectuez** une sauvegarde personnelle du document obtenu :
 - utilisez le menu Fichier / Enregistrer sous...
 - Le fichier sera alors téléchargé dans votre répertoire Téléchargements .

- Charge à vous de le placer dans votre répertoire.
- **Rendez** votre travail pour le **mardi 12 novembre** en utilisant le bouton  **Rendre ce travail** en haut à droite.
- **Fermez** ce document :
 - Cliquez sur le bouton  **Retour** en haut à droite.
 - Ou fermez simplement le navigateur.