TP n° 7 : Boucles conditionnelles

✔ Si par malheur vous écriviez une boucle qui ne terminait pas, on peut redémarrer la console interactive en cliquant sur l'icône 🔄 ou avec le raccourci Ctrl + K.

1 Boucle conditionnelle (ou boucle `while`)

La syntaxe d'une boucle conditionnelle est :

PYTHON
```
while condition:
    # bloc d'instructions
    # exécuté tant que la
    # condition est vraie
```

Comme toujours, un bloc d'instructions est défini par un deux-points et par un même niveau d'indentation. Ce bloc d'instruction est appelé le *corps de boucle* et chaque passage dans ce bloc est appelé une *itération*. La condition doit être une expression à valeurs booléennes.

Exemple : Appartenance d'un élément à une liste ♥

PYTHON
```
def appartient(element, liste):
    """Vérifie si un élément appartient à une liste."""
    n = len(liste)
    i = 0
    while i < n and liste[i] != element:
        i += 1
    return i < n
```

1. Expliquer pourquoi `liste[i]` ne peut pas être un accès à un élément qui n'existe pas, même si la liste est vide.
2. Expliquer pourquoi on renvoie `i < n` à la fin. Quel est le type de la valeur renvoyée ?
3. ♥ Réécrire cette fonction avec une boucle `for` en utilisant le fait que `return` permet de sortir de la boucle dès que l'on a (éventuellement) trouvé l'élément recherché pour ne pas parcourir inutilement la liste en entier.

☞ **Remarque 1**

Bien sûr, si on ne demande pas — explicitement ou implicitement — de ré-implémenter cette fonction, on utilisera directement `element in liste`.

On peut toujours écrire une boucle `for` sur des indices avec une boucle `while`.

PYTHON
```
for i in range(n):
    # instructions
```

peut se réécrire en ⟶

PYTHON
```
i = 0
while i < n:
    # instructions
    i += 1
```

4. Écrire une fonction calculant la somme des éléments d'une liste d'entiers en utilisant une boucle `while` au lieu d'une boucle `for`.

5. Écrire une fonction `est_triee(liste)` qui vérifie si une liste d'entiers est triée par ordre croissant, en utilisant une boucle **`while`**.

Méthode 1

Pour choisir entre une boucle while et une boucle for :

- Lorsque le nombre d'itérations ou la suite des valeurs à parcourir est connu à l'avance, on préfère en général une boucle **`for`** ;
- Lorsque le nombre d'itérations est inconnu, on choisit une boucle **`while`** ;
- Lorsque l'on connaît un majorant du nombre d'itérations ou lorsque l'on est susceptible de s'arrêter avant la fin d'une boucle **`for`**, on utilise soit une boucle **`while`**, soit le fait que **`return`** interrompt la fonction.

6. Écrire une fonction déterminant le plus petit entier n tel que la suite récurrente $u_{n+1} = u_n^2 + 1$ dépasse 10^6 pour un $u_0 > 0$ donné en argument.

Méthode 2

Pour écrire un programme utilisant une boucle while :

- On identifie la condition de la boucle. Il est souvent plus simple de chercher la condition d'arrêt et de calculer sa négation ou d'utiliser l'opérateur `not` ;
- On initialise la ou les variables nécessaires à la condition ;
- On écrit le corps de la boucle, en s'assurant que celui-ci modifiera d'une manière ou d'une autre la condition de la boucle ;
- Éventuellement, on effectue un dernier traitement à la suite de la boucle, sans oublier de revenir au niveau d'indentation du **`while`**.

7. Écrire une fonction qui prend en argument une liste `dons` $= [d_1, d_2, ..., d_n]$ d'entiers strictement positifs et un entier `objectif` qui renvoie le plus petit entier k tel que `objectif` $\leq \sum_{i=1}^{k} d_i$ et -1 si un tel k n'existe pas (attention aux indices).

2 Exercices autour des tranches

EXERCICE 1 *Préfixes d'un mot*

Écrire une fonction qui renvoie la liste des préfixes d'un mot (le mot vide est un préfixe). Par exemple, `prefixes("MPSI")` renvoie la liste des prefixes `['', 'M', 'MP', 'MPS', 'MPSI']`.

Écrire de même une fonction `suffixes`.

EXERCICE 2 *Tranches « à la main »*

Écrire une fonction `tranche(L, i, j)` qui renvoie la sous-liste `L[i:j]` (sans utiliser directement de tranches, évidemment). On pourra supposer que `0 <= i, j <= len(L)`.

Bonus : (une fois tout le TP terminé) : adapter la fonction pour gérer les indices négatifs et/ou un troisième argument pour le pas. On pourra aussi étudier le comportement des tranches lorsqu'il y a dépassement et reproduire le même comportement dans la fonction.

EXERCICE 3 *Segmenter un numéro de téléphone*

On souhaite segmenter une chaîne de caractères à intervalles réguliers par un séparateur donné. Typiquement, à partir d'un numéro de téléphone de la forme `0942371804`, on souhaite obtenir `09-42-37-18-04`.

Écrire une fonction `segmenter(chaine, sep='-', p=2)` qui insère la sous-chaîne `sep` dans une une chaîne de caractères `chaine` entre chaque sous-chaîne de taille `p`. Par exemple, `segmenter("0942371804")` renvoie la chaîne `"09-42-37-18-04"` et `segmenter("20182019", '/', 4)` renvoie la chaîne `"2018/2019"`.

Indication : découper en tranches de longueur `p` avec une tranche de la forme `chaine[k * p : (k + 1) * p]` que l'on place ensuite dans une liste dont on rassemble les éléments tout en insérant le séparateur avec la méthode `join`.

3 Problème : la conjecture de Collatz

Dans le TP n° 5, nous avons présenté la suite de Syracuse (ou suite de Collatz) définie par récurrence par $u_0 \in \mathbb{N}^*$ et

$$u_{n+1} = \begin{cases} n/2 & \text{si } n \text{ est pair} \\ 3n+1 & \text{si } n \text{ est impair} \end{cases}$$

Par exemple, avec $u_0 = 13$, on obtient la suite

$$13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, \ldots$$

Après avoir atteint le nombre 1, la suite de valeurs 1, 4, 2 se répète indéfiniment en un cycle de longueur 3 appelé cycle trivial. La *conjecture de Collatz* est l'hypothèse selon laquelle la suite de Syracuse de n'importe quel entier u_0 strictement positif atteint toujours 1. Bien qu'elle ait été vérifiée pour les 5.7 premiers milliards de milliards d'entiers et en dépit de la simplicité de son énoncé, cette conjecture défie depuis de nombreuses années les mathématiciens.

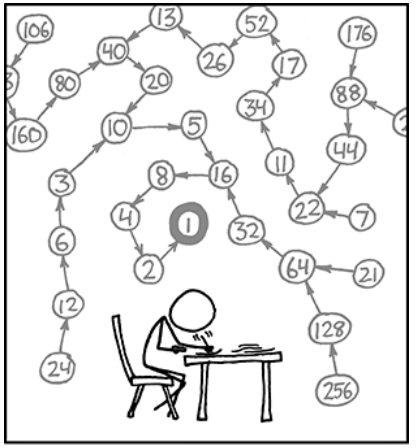
1. Implémenter une fonction `syracuse(u0, n)` qui renvoie la liste des n premiers termes de la suite de Syracuse d'un entier u_0.
2. On appelle orbite de u_0 la liste des termes de la suite de Syracuse de u_0 jusqu'à ce que l'on tombe sur 1 (inclus). Écrire une fonction `orbite(u0)` qui prend en entrée un entier u_0 strictement positif et qui renvoie son orbite sous forme d'une liste. Quelle est l'orbite de 27 ?

Plusieurs caractéristiques d'une orbite peuvent être intéressantes :

- son *temps de vol* correspond au nombre total d'entiers visités sur l'orbite ;
- son *altitude* est donnée par le plus grand entier visité sur l'orbite ;
- son *temps de vol en altitude* correpond au nombre d'étapes avant de passer strictement en dessous du nombre de départ ;
- son *temps de vol avant la chute* correspond au nombre d'étapes minimum après lequel on ne repasse plus au-dessus de la valeur de départ.

Par exemple pour l'orbite du nombre 13, l'altitude vaut 40 (maximum de la suite), le temps de vol vaut 10, le temps de vol en altitude vaut 3 (on passe à $10 < 13$ lors de la 3e étape) et le temps de vol avant la chute vaut 6 (on passe à $8 < 13$ lors de la 6e itération et ensuit on ne repasse jamais au-dessus).

3. Écrire une fonction `temps_de_vol(u0)` qui renvoie le temps de vol correspondant à l'orbite de `u0`.
4. Écrire une fonction `altitude(u0)` qui renvoie l'altitude de l'orbite de `u0`. Pour s'entraîner, on implémentera la recherche du maximum « à la main ».
5. Écrire une fonction `temps_en_altitude(u0)` qui renvoie le temps de vol en altitude correspondant à l'orbite de `u0 > 1`.
6. Écrire une fonction `temps_avant_chute(u0)` qui renvoie le temps de vol avant la chute pour l'orbite de `u0 > 1`.
7. *Bonus* : `https://projecteuler.net/problem=14`. Si votre programme prend trop de temps (plus que quelques secondes) c'est que vous n'avez pas assez réfléchi.



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

https://xkcd.com/710/