02 - Structures for et booléens

Avant de commencer ce TD, créer un dossier personnel appelé TD02, où l'on sauvegardera les fichiers.

On rappelle que `print(...)` *va afficher le contenu des* `...` *en console.*

On rappelle (et ceci est à retenir) que `a//b` *contient le quotient de la division euclidienne de* `a` *par* `b` *et* `a%b` *son reste.*

I Les boucles `for`

Exercice 1 Écrire un programme qui affiche :

```
1 km à pied, ça use, ça use... ça use les souliers
2 km à pied, ça use, ça use... ça use les souliers
3 km à pied, ça use, ça use... ça use les souliers
```

jusqu'à 42 km **inclus**

Exercice 2 *...et à l'envers* Écrire un programme qui affiche (on oubliera la faute d'accord sur la dernière ligne) :

`C'est dans 10 ans je m'en irai, j'entends le loup et le renard chanter` ... `C'est dans 1 ans je m'en irai, j'entends le loup et le renard chanter`

Exercice 3 Écrire un programme qui calcule la somme des entiers de 17 à 183

Exercice 4 Écrire un programme qui calcule la sommes des multiples de 3 de 18 à 180 inclus.

Exercice 5 *Un peu d'amusement avec les mathématiques...* On appelle suite de Syracuse la suite définie de la façon suivante :

- $u_0 = N$, où N est un entier quelconque
- $u_{n+1} = \begin{cases} u_n/2 \text{ si } u_n \text{ est pair} \\ 3 * u_n + 1 \text{ si } u_n \text{ est impair} \end{cases}$

5.1 Écrire un programme qui affiche les 30 premiers termes de la suite de Syracuse, et le tester pour diverses valeurs de N. Que remarque-t-on ?

On commence à s'approcher de notre premier projet...

Exercice 6 *Vers le projet 1* On donne la liste suivante :

```
jours=["Dimanche","Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi"]
```

6.1 Que contient `jours[2]` ? A quel position est associé le Dimanche ?

6.2 Écrire un programme qui, à partir d'aujourd'hui (mercredi), affiche la liste (périodique) des 40 jours à venir.

Exercice 7 *Vers le projet 1*

7.1 On suppose que tous les mois ont 30 jours (calendrier égyptien). Écrire un programme qui, à partir du numéro du jour d'aujourd'hui, affiche les 180 prochains numéros de jour.

7.2 On fournit la liste suivante :

`mois=[31,28,31,30,31,30,31,31,30,31,30,31]`.

L'utiliser pour améliorer le programme précédent. On devra également utiliser le numéro du mois.

II Un passage par les booléens

Exercice 8 Que vaut l'expression : `True and False` ?

Exercice 9 Que vaut l'expression : `True or False` ?

Exercice 10 Que vaut l'expression `not (False or True)`

Exercice 11 On se donne deux variables entières `a` et `b`, et un entier `n`. Écrire les expressions booléennes contenant les réponses aux questions suivantes :

- `a` est-il divisible par `b` ?
- `a` est-il divisible par `b` ou `b` est-il divisible par `a`
- `n` est divisible par `a` et par `b`
- `n` est divisible par `a` ou par `b`
- `n` est divisible par `a` mais pas par `b`
- `n` est divisible par l'un des deux mais pas pas l'autre.

Exercice 12 *Vers le projet 1*

12.1 Écrire une expression qui retourne, connaissant une variable entière `a` contenant le numéro d'une année, la réponse à la question : `a` est-elle une année bissextile ?

12.2 Tester en console les expressions `int(True)` et `int(False)`

12.3 Déduire des deux questions précédentes, toujours en se donnant une variable entière `n`, une expression qui retourne le nombre de jour du mois de février selon la valeur de `n`. Ceci revient à écrire un programme python tel que :

```
a=...
nb_jours=...
```

où `nb_jours` devra contenir 28 si l'année n'est pas bissextile (2025 par exemple) et 29 si elle l'est (2024).

12.4 Utiliser ceci pour raffiner (encore) le programme de l'exercice 7.

III La structure `if` (et éventuellement `else`)

Exercice 13 Écrire un programme qui affiche 'Gagné' si une variable `a` est divisible par 7, et `'Perdu'` sinon.

Exercice 14 Écrire un programme qui calcule la somme des entiers de 1 à 2000 divisibles soit par 3, soit par 7.

Exercice 15 Raffiner le programme précédent en ne sommant que les entiers divisibles soit par 3, soit par 7, mais pas par les deux.

Exercice 16 *Amélioration progressive...* On rappelle que l'on peut importer la fonction `random` en écrivant en début de programme `from random import random`. `random()` est une fonction qui retourne un flottant aléatoire dans $[0, 1[$. On va simuler un genre de martingale : un coup consiste en :

1. tirer au sort un flottant `a` dans $[0, 1[$
2. remplacer le score existant, stocké dans une variable `s`, uniquement si `a` est supérieur strictement à `s`

Ecrire un programme qui réalise cela sur 100 coups.

IV Les boucles `while`

Exercice 17 *Un premier entraînement* Ecrire l'équivalent du programme suivant :

```
for i in range(3,19):
    print(i)
```

mais en utilisant une boucle `while` au lieu d'une boucle `for`

Exercice 18 On se donne un entier M. On cherche le plus grand entier p tel que $p^3 \leq M$. Coder un programme réalisant ceci, en choisissant par exemple $M = 4678$.

Exercice 19 *Mille bornes* On se donne la liste d'entiers suivants : `cartes=[25,50,75,100,200]`. On va simuler un mille bornes naïf : on tire au hasard une de ces cartes, et on rajoute la carte tant que la distance `d` parcourue, initialement nulle, est inférieure à 1000. On rappelle qu'on ne peut avancer que si `d`+la valeur de la carte qu'on vient de tirer ne dépasse pas 1000.

On rappelle que l'on peut importer la fonction `randint` via `from random import randint`, à taper en début de programme dans l'éditeur. `randint(a,b)` tire aléatoirement un entier entre `a` et `b` (les deux inclus). On va procéder pas à pas :

1. initialiser une variable `d` à zéro : ce sera la distance parcourue. On peut également initialiser une variable `n` à 0 pour compter le nombre de tours.
2. commencer la boucle `while` en indiquant qu'on va tirer une carte tant que `d` est strictement inférieur à 1000
3. on initialise un `i` à un entier tiré aléatoirement dans les valeurs possibles **d'indices** de la liste `cartes`.
   (a) on utilise `i` pour tirer la valeur `c` correspondante
   (b) on incrémente `n` de 1 (pour compter le nombre de tours).
   (c) on incrémente `d` de `c` **si** `c+d` **reste inférieur à 1000**

... et c'est terminé, on peut afficher `n` à la fin si on veut, ou bien les cartes utilisées au fur et à mesure, etc.

Exercice 20 *(Enfin)* **Trouver les dates des 100 prochains vendredi 13** On va faire tourner tout ceci en utilisant ce que nous avons vu. On va quand même détailler l'algorithme. On part d'une date connue, par exemple le mercredi (jour 2) 9 octobre (mois 10) 2024.

1. on commence par initialiser une liste vide, nommée `l_v` : cela sera la liste de nos vendredis 13.
2. on initialise également `j` à 2, `d` à 9, `m` à 10 et `a` à 2024 : ce sont respectivement les numéros du jour de la semaine, du numéro de la date, du numéro du mois et de celui de l'année.
3. on initialise `mois=[31,29,31,30,31,30,31,31,30,31,30,31]` puisque 2024 est bissextile.
4. on va faire tourner l'algorithme (une boucle donc) tant que `len(l_v)` sera inférieure à 100.
   (a) si on a un vendredi 13 (un booléen est à trouver pour savoir si c'est un vendredi et que la date est bien un 13), on ajoute à `l_v` (voir le TD01 pour savoir comment on ajoute un élément à une liste) la chaîne donnant la date via `str(d)+"/"+str(m)+"/"+str(a)`.
   (b) on incrémente `j` de 1, et on le ramène dans l'intervalle $[0, 6]$ (voir l'exercice **??**)
   (c) on incrémente `d` de 1. Si il dépasse le nombre de jour du mois, on le remet à 1, et on incrémente `m` de 1.
   (d) si `m` dépasse 12, on le ramène à 1 et on incrémente `a` de 1. Lors de cette incrémentation, on modifie la valeur du nombre de jours de février dans `mois` en tenant compte de la bissextilité.

On affichera à la fin `l_v` pour voir. On doit trouver comme derniers éléments le vendredi 13 février 2082 et le vendredi 13 mars 2082 (ce qui est normal, février ayant 28 jour en 2028, comme 28 est multiple de 7, on a les mêmes jours/dates en février et en mars).