TP 2
Conditions et boucles

I Rappel des épisodes précédents

Les types vus jusque là :
- `int` : nombres entiers
- `float` : nombres à virgule flottante (virgule `.` et notation scientifique avec `e`)
- `str` : chaines de caractères, entre guillemets doubles `"`
- `bool` : booléens, avec `True` et `False`

Les opérations :
- Les 4 opérations `+`, `-`, `*`, `/`
- Le quotient dans la division euclidienne (en nombres entiers) `//`, le reste `%`
- Les comparaisons `==`, `!=`, `<`, `<=`, `>`, `>=`
- Les opérations sur les booléens `and`, `or`, `not`
- Les opérations sur les chaines : concaténation `+`, multiplication par un nombre entier `*`, accès au `i`-ème caractère de `s` (démarrant à `0`) par `s[i]`

Les fonctions :
- Connaitre le type `type()`
- Conversions de type `int()`, `float()`, `str()`, `bool()`
- Afficher : `print()`, peut prendre une liste d'expressions ou de variables séparées par des virgules
- Demander à l'utilisateur : `input()`, donne un résultat de type `str` qu'il faut éventuellement convertir si on veut demander un entier ou un flottant

Aujourd'hui nous écrivons des programmes avant tout dans le mode script (sur plusieurs lignes) ; le mode interactif sert à faire des tests courts ou à inspecter le contenu ou le type des variables. Séparer les programmes et les exercices par des **cellules** délimitées par une ligne commençant par la double dièse `##` permet de tout garder sur une même page (donc d'enregistrer dans un même fichier `.py`), mais de n'exécuter qu'un morceau à la fois et pas tout depuis le début.

```
## exercice 1
...

## exercice 2
...
```

Dans les exercices, l'expression « écrire un programme qui demande un nombre `a`... » signifie :
- Soit il faut utiliser une syntaxe telle que `a = int(input("Entrez a : "))`,
- Soit il faut que le programme démarre par `a = ...` de façon à ce qu'on puisse en changer la valeur facilement avant de lancer le programme.

II Conditions

II.1 Conditions simples

Le but est d'exécuter un morceau de programme seulement **si** (*if* en anglais) une condition est vérifée. Sinon (*else* en anglais), un autre morceau de programme est exécuté. La syntaxe de base en Python a la forme suivante :

`if` *condition*`:`
    instructions1
`else:`
    instructions2

où :
- *condition* désigne n'importe quelle expression booléenne (formée par exemple avec des comparaisons `==`, `!=`, `<`, `<=`, `>`, `>=`, et les opérations `and`, `or`, `not`) que le programme va tester,

— *instructions1* est du code Python, qui peut être sur plusieurs lignes, qui va être exécuté seulement si la condition a été évaluée à `True`. La partie du programme qui va être exécutée est tout ensemble décalée (on dit aussi **indentée**) vers la droite (en général de 4 espaces, ou en seul caractère tabulation ; le logiciel sert notamment à bien aligner les lignes) et forme tout ensemble un **bloc d'instructions**.
— *instructions2* est un autre bloc d'instructions qui va être exécuté dans le cas contraire.

Si du code n'est pas indenté, alors il ne fait pas partie de la condition et est exécuté quoiqu'il arrive. Le `else` n'est pas obligatoire : si la condition est fausse alors le premier bloc d'instruction n'est pas exécuté et le programme passe directement à la suite.
Voici un exemple de programme qui affiche la valeur absolue de `x` :

```
x = ... # mettre un nombre ici
if x >= 0:
    print(x)
else:
    print(-x)
```

Exercice 1. Écrire un programme qui demande à l'utilisateur son âge, et affiche s'il est majeur ou mineur.

Exercice 2. Écrire un programme qui demande à l'utilisateur deux nombres `a`, `b`, calcule **dans une variable** le nombre `m` qui est le plus grand des deux, et affiche à la fin `le maximum est ...`

Exercice 3. Choisissez un mot de passe ; écrire un programme qui demande à l'utilisateur le mot de passe, et qui lui dit si le mot est correct ou non.

II.2 Plusieurs conditions

Il arrive que l'on veuille tester une condition plus complexe qui ne se traduit pas aussi simplement que « si… alors ».

Exercice 4. Écrire un programme qui demande à l'utilisateur un nombre `x` et affiche si `x` est positif, négatif ou nul. Sans utiliser la syntaxe ci-dessous. Il faut donc nécessairement emboiter un bloc `if` dans un autre (attention à l'indentation !)

Une meilleure possibilité est offerte par le mot-clé `elif`, contraction de *else if* (sinon si).

`if` *condition1* :
    instructions1
`elif` *condition2* :
    instructions2
`elif` *condition3* :
    instructions3
...
`else:`
    instructions

alors lors de l'exécution :
— Le programme teste la *condition1*. Si elle est vraie alors il exécute *instructions1*.
— Sinon, il vérifie la *condition2*. Si elle est vraie alors il exécute *instructions2*.
— Il peut y avoir plusieurs `elif`.
— À la fin, si aucune condition n'a été vérifiée, le programme exécute le bloc d'instructions du `else`. Il n'est pas du tout obligatoire d'avoir un `else`, auquel cas le programme passe simplement à la suite.

Le programme de l'exercice précédent se ré-écrit ainsi :

```
x = ... # mettre un nombre ici
if x > 0:
    print("positif")
elif x < 0:
    print("négatif")
else:
    print("nul")
```

Exercice 5. Écrire un programme qui demande à l'utilisateur sa note au bac (attention, ce n'est pas forcément un nombre entier) et affiche à quelle mention cela correspond.

III Boucles

III.1 La boucle `while`

Une *boucle* permet de répéter des instructions automatiquement. C'est à partir de maintenant que les programmes deviennent vraiment intéressants : ils automatisent des tâches qui seraient bien pénibles pour un humain.
On se concentre sur la boucle `while`. Le bloc d'instructions (**corps** de la boucle) est répété **tant que** (traduction de *while*) une condition est vérifiée. La syntaxe est tout simplement la suivante :

`while` *condition*`:`
    instructions

alors lors de l'exécution :
- Le programme teste si la condition est vraie,
- Si oui, il exécute le bloc d'instructions. Une fois fini, il recommence à évaluer la condition et ainsi de suite.
- Sinon, il sort simplement de la boucle et passe à la suite.

Pour que cela soit intéressant, il faut que la condition puisse varier à chaque passage dans la boucle ! Sinon, si elle est toujours vraie, rien ne l'arrête et on obtient une **boucle infinie**... La méthode de base pour répéter une instructions un certain nombre `n` de fois est de déclarer une variable appelée **compteur**, que l'on va augmenter de 1 (on dit **incrémenter**) à chaque passage dans la boucle, et de tester la condition `i < n` :

```
n = ... # mettre un petit nombre
i = 0
while i < n:
    print("passage dans la boucle avec i = ", i)
    i = i + 1
print("fin de boucle avec i = ", i)
```

Remarque 1. Dans chaque boucle, on porte une attention particulière à :
- La valeur à laquelle le compteur est initialisé (**essayez** avec `i = 1`)
- La comparaison stricte `<` ou large `<=` (**essayez** de remplacer par `<=`)
- L'incrémentation au début ou à la fin de la boucle (**essayez** d'échanger les deux lignes dans le bloc d'instructions),
- Ne pas oublier l'incrémentation (**essayez** de l'enlever !)

L'exemple ci-dessus est le plus standard et est écrit de telle façon à ce qu'il répète exactement `n` fois. Le mode interactif permet de tester facilement la valeur des variables `i`, `n` après l'exécution de la boucle.

Exercice 6. Écrire un programme qui affiche les nombres x^2 pour $1 \leqslant x \leqslant 10$.

Exercice 7. Écrire un programme qui demande à l'utilisateur un nombre `n` et compte à rebours : affiche `n` puis `n-1` puis... jusqu'à `0`. Deux façons possibles (tester **les deux**) :

1. `i` s'incrémente comme ci-dessus,
2. `i` *décrémente*...

III.2 Application aux suites

Pour calculer les termes successifs d'une suite, on se sert en plus d'une variable `u` qui à chaque passage dans la boucle va devenir le terme suivant. L'exemple de base pour les puissances de 2 ressemble à :

```
n = ... # mettre un petit nombre
u = 1
i = 0
while i < n:
    print(u)
    u = 2 * u
    i = i + 1
```

Exercice 8. Écrire un programme qui demande à l'utilisateur un nombre `n` et calcule la somme $1 + 2 + \cdots + n$.

Exercice 9. Écrire un programme qui demande à l'utilisateur un nombre `n` et affiche successivement tous les nombre entiers *pairs* entre 0 et `n`. Deux façons possibles (tester **les deux**) :

1. Changer l'incrémentation,
2. Tester dans la boucle si le nombre est pair.

Exercice 10. Au début de l'an 2022, la population mondiale est estimée à environ 7,9 milliard d'habitants. Elle augmente d'environ 1,12 % chaque année. Écrire un programme qui affiche la population mondiale estimée sur les années futures (afficher à la fois l'année et la population, par exemple `Année 2022 : 7900000000`) si le taux de croissance reste le même, et s'arrête quand elle dépasse 10 milliard.

Exercice 11. La *suite de Syracuse* est la suite $(s_n)_{n\in\mathbb{N}}$ définie par : le nombre $s_0 \geqslant 1$ est à détermminer par l'utilisateur, et ensuite

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

Écrire un programme qui demande à l'utilisateur le nombre s_0 puis affiche tous les termes de la suite jusqu'à ce qu'un terme soit égal à 1 (*pourquoi, que se passe-t-il ensuite ?*)

Exercice 12. La *suite de Fibonacci* est la suite où chaque nombre est égal à la somme des **deux** précédents. C'est donc la suite $(F_n)_{n\in\mathbb{N}}$ telle que $F_0 = 0$, $F_1 = 1$ et $\forall n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$. Écrire un programme qui affiche uns par uns les n premiers termes de la suite (pour un nombre n qu'on peut par exemple demander à l'utilisateur).

III.3 Parcourir une chaine de caractères

La fonction `len()` permet de connaitre la longueur d'une chaine de caractères. Testez-là en mode interactif !

```
len("Bonjour")
```

Pour parcourir une chaine `s` (c'est à dire, récupérer les caractères uns par uns), il faut donc faire une boucle avec un compteur `i` et donc `s[i]` sera le `i`-ième caractère de la chaine. Tester par exemple :

```
s = "..." # mettre des mots
n = len(s)
i = 0
while i < n:
    print(s[i])
    i = i + 1
```

Exercice 13. Écrire un programme qui demande à l'utilisateur une séquence ADN, composée des seules lettres A, T, G, C, et qui compte combien de fois chaque lettre apparait (et l'affiche à la fin).