TP 2. Structures de contrôle

Lorsqu'un problème est résolu par un algorithme, pour obtenir sa solution, il convient de suivre les instructions les unes à la suite des autres. C'est d'ailleurs ce que fait Python lorsqu'on exécute un programme : il réalise les instructions les unes après les autres dans l'ordre où elles sont écrites. Ce chemin s'appelle le *flux d'exécution*.

Nous étudions dans ce TP des structures qui vont nous permettre de controler le flux d'exécution.

1 Structure conditionnelle

On appelle *structure conditionnelle* les instructions qui permettent de tester si une condition est vraie ou non.

1.1 Le test *if*

L'instruction `if` est la structure de test la plus basique. Elle permet d'exécuter une série d'instructions si une condition est vraie. La syntaxe de cette expression est la suivante :

`if` *condition* :
    bloc d'instructions ;

La condition est un booléen (type `bool`, cf TP1).

Exemple. En ligne de commande.

```
>>> a=1
>>> if a>0 :
        print('a est positif')

a est positif
```

⚠attention à l'**indentation** (notamment en mode script). Un bloc d'instructions est associé à une ligne d'en-tête : « `if` *condition* ». Les instructions qui dépendent de cette condition doivent être décalées d'un même nombre d'espaces. L'environnement IDLE réalise cette indentation automatiquement lorsque vous passez à la ligne (touche : *Entrée*) : elle est de 4 espaces ou 1 tabulation.

```
if a>0 :
    print(positif)
```

Exercice 1. Réaliser un programme **pair.py** qui teste si une variable `a` (de type entier) est un nombre pair et affiche, le cas échéant, `L'entier` *a* `est pair`.

1.2 La structure conditionnelle avec alternative : `if... elif... else`

L'instruction `else` (*sinon*) permet de programmer une éxecution alternative lorsque la condition suivant `if` n'est pas réalisée. L'instruction `elif` (contraction de *else if*) permet de définir des conditions alternatives.

La structure générale est la suivante :

```
if      condition 1 :
            blocs d'instructions ;
elif    condition 2 :
            blocs d'instructions ;
...
else:
            blocs d'instructions ;
```

Exercice 2. Rédiger et exécuter le programme Python suivant sous le nom de `vab.py`. Tester pour différentes valeurs de la variable `a`. Que réalise-t'il ?

```
a=-12
if a>=0 :
    print(a)
else :
    print(-a)
```

Exercice 3. Compléter le programme `pair.py` précédent en affichant l'alternative `L'entier` a `est impair`.

Exercice 4. Que réalise les deux programmes Python ci-dessous ?

```
a,b=1,2
if a>b :
    print(a)
else :
    print(b)
```

```
a,b=1,2
if a>b :
    m=a
else :
    m=b
print(m)
```

En s'appuyant sur les modèles précédents, réaliser un programme `max3` qui affiche la plus grande valeur parmi trois variables entières a,b et c.
Indication. $\max(a,b,c) = \max(\max(a,b),c)$.

Tester votre programme pour les valeurs : `a,b,c = 1,2,3` ; `a,b,c = 1,3,2` et `a,b,c = 3,1,2`.

Exercice 5. Réaliser un programme Pyhton, nommé `heuresinfo.py` qui demande à un utilisateur (élève de PCSI) d'entrer : un numéro de semaine (calendrier du lycée : entre 0 et 35) puis un jour de la semaine (en minuscule). Le programme affiche le nombre d'heures d'informatique (cours ou TP) dans l'emploi du temps de PCSI de la journée donnée.

```
>>>
Numéro de la semaine ? 3
Jour de la semaine ? dimanche
Vous n'avez aucune heure d'informatique.
```

Pour vous entraîner à domicile :

Exercice 6. Rédiger un programme Python sous le nom de `bissextile.py` qui, pour une variable recevant une valeur entière, indique si l'année correspondante est bissextile en affichant : `"Cette année est bissextile"` ou `"Cette année n'est pas bissextile"` selon le cas.

Rappel. *Les années bissextiles sont celles qui sont divisibles par 4, sauf lorsqu'elles débutent un siècle non multiple de 400. (Par exemple : l'année 2000 était bissextile : 2000 débute un siècle mais 2000/400 = 50 ; l'année 2100 ne sera pas bissextile).*

Tester votre programme avec les années 2000, 2013, et 2100.

Exercice 7. Réaliser un programme `trinome.py` qui demande à l'utilisateur de saisir un trinôme du second degré. On suppose que l'utilisateur rentre successivement les valeurs (nombres à virgule flottante) `a,b` puisc pour désigner le polynôme $P = aX^2 + bX + c$. Le programme retourne le ou les racines réelles du polynôme ou, le cas échéant, un message indiquant que ce polynôme n'a pas de racine réelle. Prévoir le cas $a = 0$.

Exercice 8. Complément à l'exercice 10 du TP 1. Réaliser un programme `heure.py` qui demande à l'utilisateur l'horaire de départ de son train. On suppose que l'utilisateur rentre les horaires sous le format : `XYhZT` (exemples : 19h51, 08h01) où `X, Y, Z` et `T` sont des entiers.

Votre programme affiche un message si l'utilisateur saisi un horaire erroné (exemple : 25h62). Dans l'interpréteur votre résultat ressemble à ça :

```
>>>
Horaire de départ ? 27h51
Cette horaire n'existe pas.
```

2 Structure conditionnelle itérative : boucle `while`

Les structures itératives permettent d'exécuter plusieurs fois une même série d'instructions (itérations). L'instruction `while` (*tant que*) permet d'exécuter les blocs d'instructions qui en dépendent tant que la condition suivant le `while` est vraie. La syntaxe générale est la suivante :

`while` *condition* :
    bloc d'instructions ;

La *condition* est un booléen (type `bool`, cf TP1).

Précisément, le fonctionnement de cette instruction est le suivant :

1. on évalue *condition* ;
2. si elle est vraie, on exécute le *bloc instructions* et on reprend en 1 ;
3. si elle est fausse, le programme passe au bloc d'instructions suivant.

⚠Encore une fois, attention à l'**indentation** qui délimite les blocs d'instructions.

Exercice 9. Rédiger et exécuter les lignes de commandes suivantes. On affiche les dix premiers multiples de 7.

```
>>> c=0
>>> while c<10 :
        print(7*c)
        c=c+1
```

Exercice 10. Voici des commandes soumises à l'interpréteur Python par un élève peu scrupuleux. Qu'en penser ? Tester ces lignes de commandes (?).

Programme A.

```
>>> c=0
>>> while c>=0 :
        print(7*c)
        c=c+1
```

Programme B.

```
>>> c=0
>>> while c<10 :
        print(7*c)
```

Commentaires sur ces deux programmes :

Exercice 11. Calculs de sommes.

1. Dans cette question on se propose d'écrire un programme `somme.py` qui calcule la somme $1+2+3+...+N$ des N premiers entiers où N est un entier naturel [1].
   Voici une proposition de programme. Vous semble-t'elle raisonnable ? Si oui la rédiger ; sinon la corriger. **En parallèle**, écrire les valeurs successivement prises par les variables `indice` et `somme` dans ce programme.

```
N = 5
somme = 0
indice = 1
while indice<=N :
    somme = somme + indice
    indice = indice + 1
print(somme)
```

somme	indice
0	1
1	2
3	

1. On se place dans la position d'un élève peu scrupuleux qui ne connaît pas ses sommes classiques

2. Reprendre le programme précédent en le modifiant de la manière suivante : on définit une variable `Vmax` (tester avec `Vmax = 1000`). Le programme affiche la première valeur de `N` pour laquelle :

$$1 + 2 + 3 + ... + N \geq Vmax$$

3. Reprendre les deux programmes précédent avec la somme $H_N = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N}$ des *inverses* des N premiers entiers. Répondre aux questions suivantes :

$$H_{10^6} \simeq \text{.....} \quad \text{et} \quad H_N \geq 16 \text{ dès que } N \geq \text{.....}$$

Exercice 12. Devine qui c'est ?

La commande `randrange(1,N)` retourne un entier choisi « au hasard » entre 1 et N-1. Il faut l'importer du module `random`.

1. Réaliser un programme `devineKiC.py` dans lequel l'utilisateur doit deviner un entier choisi « au hasard » entre 1 et 100. La machine interroge l'utilisateur tant qu'il n'a pas trouvé le nombre mystère et le félicite cordialement lorsqu'il gagne.
   Voici un exemple de ce que peut donner votre programme à l'exécution :

```
>>>
Devinez le nombre mystère ? 4
Ce n'est pas le nombre mystère.
Quel est le nombre mystère ? 7
Ce n'est pas le nombre mystère.
Quel est le nombre mystère ? 2
Bravo, vous avez trouvé le nombre mystère !
```

2. Compléter votre programme en affichant le nombre de coup joués par l'utilisateur.
3. Compléter votre programme : l'ordinateur indique si le nombre mystère est plus petit ou plus grand que le nombre donné par l'utilisateur.

Exercice 13. La multiplication des lapins.

Vous allez faire l'acquisition d'un couple de bébés lapins. Au bout d'un mois ce couple est adulte. Le mois suivant il donne naissance à un couple de bébés lapins : vous avez maintenant 4 lapins. Puis chaque couple engendre tous les mois un nouveau couple deux mois après sa naissance.

Nous avons le schéma ci-contre :

Légende : m : *bébé lapin* ; M : *lapin adulte*.

- **Mois 0.** m m
- **Mois 1.** M M
- **Mois 2.** M M m m
- **Mois 3.** M M M M m m
- **Mois 4.** M M M M M M m m m m

Notons F_N le nombre de lapins que l'on a au bout du N-ième mois. On convient que : $F_0 = 2$. Nous avons donc $F_1 = 2$ puis $F_2 = 4$ et $F_3 = 6$. Plaçons nous au mois $N + 2$, nous aurons tous les couples de lapins du mois précédent (le mois $N + 1$) et toutes les progénitures des couples de lapins du mois N. Nous avons donc la relation :

$$F_{N+2} = F_{N+1} + F_N$$

⚠Les pythons mangent les lapins.

1. Rédiger un programme Python `lapin.py` qui calcule de manière itérative le nombre de lapins au bout d'un an (le mois $N = 12$).

$$F_{12} = \text{.....}$$

2. Au bout de combien de mois dépasse-t'on le milliard de lapins ?

$$N = \text{.....}$$

3 Structure itérative : boucle `for`

L'instruction `for... in` permet de faire parcourir à une variable l'intégralité d'une structure de données (comme les caractères d'une chaine ou une liste d'entiers) et d'itérer un bloc d'instructions au fil de ce parcours. C'est une structure itérative qui peut représenter une alternative intéressante à l'instruction `while`.

Exercice 14. Voici deux exemples d'utilisation. Les tester.

```
>>> for k in range(1,10) :
        print(k)
```

```
>>> for l in 'Bonjour' :
        print(l)
```

Exercice 15. Écrire une ligne de commande qui affiche tous les multiples de 7 entre 0 et 70.

Exercice 16.

1. Rédiger un programme avec les lignes qui suivent. **En parallèle**, écrire les valeurs successivement prises par les variables `k` et `s` dans ce programme. Que calcule ce programme ?

```
s=0
for k in range(1,6) :
    s=s+k
```

`k`	`s`
-	0
1	
2	

2. Rédiger un programme `factorielle.py` qui calcule le nombre $n!$ où n est une variable de type `int`. Tester votre programme avec $n = 3, 4$ et 12.

Exercice 17. Écrire un programme `voyelles.py` qui affiche le nombre de voyelles d'un mot (écrit en minuscule, sans caractère accentué). Tester votre programme sur le mot `informatique`.

Exercice 18.

1. Écrire un progamme `moyenneAlea.py` qui calcule et affiche la moyenne d'un nombre donné (disons 1000) d'entiers entre 1 et 10 choisis *au hasard* (à l'aide de la fonction `randrange`).
2. Réécrire ce même programme en ne tenant plus compte des occurences du nombre 2.