TP5 - Boucles et structure de programme

M. Tellene

1 Création de l'environnement de travail

Avant de commencer le TP, vous allez créer votre environnement de travail. Pour ce faire vous allez dans votre dossier personnel, puis « apprentissage_python ». Une fois arrivé, créer un dossier « TP5 ». C'est dans ce dossier qu'il faudra mettre tous les exercices du TP.

2 Rappel sur le `if`

Cette première partie est constituée de quelques exercices qui vont vous faire re-travailler le `if`, `if... else` et `if... elif... else`.

Les fonctions demandées dans cette partie (les 3 prochains exercices) sont à écrire dans un fichier **rappels_if.py** (fichier que vous sauvegarderez dans le dossier « TP5 »)

EXERCICE 1
Écrire une fonction `signe` qui prend en argument un nombre (`n`) et qui renvoie le signe (sous forme de chaîne de caractères) de celui-ci.

Exemple de rendu :

```
>>> signe(0)
nul
>>> signe(121)
positif
>>> signe(-1)
négatif
```

EXERCICE 2
Le lycée veut créer une boutique en ligne afin de vendre des pulls à l'effigie du lycée. L'administration vous a demandé de gérer l'algorithme de calcul de prix. Pour son lancement le lycée fait une offre spéciale :

- une remise de 1% est prévue lorsque le montant de l'achat est compris entre 20 euros (inclus) et 50 euros (exclus)
- une remise de 2% est prévue lorsque le montant de l'achat est supérieur ou égal à 50 euros

Écrire une fonction `calcul_prix` qui prend en argument le montant de l'achat (`montant`) et qui renvoie le montant réel à payer après avoir retiré la remise.

- si le prix est négatif, la fonction devra retourner `None`
- si le prix est compris entre 0 euros (inclus) et 20 euros (exclus), aucune réduction ne sera appliquée

EXERCICE 3

Écrire une fonction `nb_mystere` qui prend un argument un nombre : `x`. Dans un premier temps, nous fixons le nombre mystère à 42. Par conséquent :

- si x est plus petit que 42, la fonction retournera "Trop petit!"
- si x est plus grand que 42, la fonction retournera "Trop grand!"
- si x est égal à 42, la fonction retournera "Bravo, tu as trouvé le nombre mystère!"

3 Faire des boucles

En Python, il est possible répéter des instructions avec une structure appelée **boucle**.

3.1 La boucle `while`

On voudrait améliorer la fonction `nb_mystere` afin qu'il ne soit pas nécessaire de le relancer à chaque essai et qu'il s'arrête automatiquement dès que l'utilisateur le souhaite.

- Dans un nouveau fichier : **nb_mystere.py** réécrire la fonction `nb_mystere`
- Sous votre fonction `nb_mystere`, recopier le code suivant :

```
print("Voici le jeu du nombre mystère amélioré. Amusez-vous bien...")

rejouer='1'

while rejouer == '1':
    nombre=float(input("Donner un nombre : "))
    print(nb_mystere(nombre))
    rejouer=input("Voulez-vous rejouer ? (0=non, 1=oui) ? ")

print("Le jeu est fini. Au revoir")
```

1. Que se passe-t-il quand vous répondez 1 (lorsque le programme vous demande si vous voulez rejouer) ? .....
2. Que se passe-t-il quand vous répondez 0 (lorsque le programme vous demande si vous voulez rejouer) ? .....
3. Quel est le rôle de la variable `rejouer` ? .....
4. Que veut dire `while` en anglais ? .....
5. Quelle est la condition de la boucle `while` ? .....

- Créer un fichier **boucle_infinie.py**
- Recopier le code suivant :

```
n = 1

while n == 1:
    print("Boucle infiniiiiiiie !!!!!!")
```

- Normalement l'éditeur de code n'est pas très content, arrêter le programme de force en appuyant sur :

Ce que vous venez de faire s'appelle **une boucle infinie**, c'est-à-dire une boucle qui ne s'arrête pas.

— Pourquoi cette boucle est-elle infinie ? .....

.....

3.2 A votre tour !

Dans cette partie, vous allez trouver des exercices en rapport avec la boucle `while`.

EXERCICE 4

1. Soit le programme suivant :

```
1  def total():
2      Total = 0
3      continuer = '1'
4      print("Le total actuel est de ",Total)
5      while (continuer == '1'):
6          n = float(input("Donner un nombre à ajouter au total : "))
7          Total = Total + n
8          print("Le total actuel est de",Total)
9          continuer = input("Voulez-vous continuer (oui:1/non:0) ? ")
10     return Total
11
12 T=total()
13 print("Le total atteint est de",T," le jeu est fini.")
```

   (a) Que fait la fonction `total()` ? .....

   (b) Quel est le rôle de la variable `Total` ? .....

   (c) A quoi sert la ligne 7 (`Total = Total + n`) ? .....

   .....

   (d) Quel est le rôle de la variable `continuer` ? .....

   (e) A quoi sert la ligne 10 (`return Total`) ? .....

   (f) A quoi sert la ligne 12 (`T=total()`) ? .....

   .....

2. Créer un fichier **moyenne.py**
3. Sur le même principe que le programme précédent, écrire une fonction `moyenne` qui ne prend pas d'argument. Cette fonction doit demander un certain nombre de notes à l'utilisateur et en calcule la moyenne. Afin de permettre la saisie des notes, il faudra utiliser l'instruction :

```
1 note = float(input("Saisir une note : "))
```

Exemple de rendu :

```
>>> moyenne()
Saisir une note : 2
Voulez-vous continuer (oui:1/non:0) ? 1
Saisir une note : 20
Voulez-vous continuer (oui:1/non:0) ? 1
Saisir une note : 15
Voulez-vous continuer (oui:1/non:0) ? 0
12.333333333333334
```

EXERCICE 5

Pour cet exercice, créer un fichier **nb_entier.py**

1. Dans votre fichier, recopier le programme suivant :

```
i=0 # initialisation de i

while i <=10:
    print(i)
    i=i+1 #on augmente i de 1
```

2. Modifier le programme pour qu'il écrive les nombres entiers jusqu'à 100.
3. Remplacer i+1 par i+5 et écrire en commentaire ce qu'il se passe
4. Mettre le programme en commentaire
5. En vous inspirant du programme précédent, écrire sous celui-ci :
   - une fonction `cent_premiers_carres` qui affiche les 1, 4, 9, 16, 25, ..., 10000, soit les 100 premiers carrés
   - une fonction `somme_cent_premiers_carres` qui calcule 1 + 4 + 9 + 16 + 25 + ... + 10000, soit la somme des 100 premiers carrés (le résultat final devrait être 338350)
   - une fonction `somme_n_premiers_carres` qui calcule la somme de n premiers carrés pour n choisi par l'utilisateur (`n` est donc l'argument de la fonction)

3.3 La boucle `for ... in ...`

`while` permet de répéter une instruction « à l'infini » tant qu'une certaine condition est vérifiée. Cependant, il existe une autre boucle qui, une fois exécutée, répète une instruction un nombre fixe de fois.

1. Créer un fichier **bonjour.py**
2. Recopier le programme suivant :

```
for i in range(5):
    print("bonjour")
```

   (a) Qu'est-ce que fait ce programme ? .....

   (b) Où est décidé le nombre de tour de boucle à faire ? .....

3. Créer un fichier **i.py**
4. Recopier le programme suivant :

```
for i in range(5):
    print(i)
```

(a) Qu'est-ce que fait ce programme ? .....

(b) Dans une boucle `for`, à quoi sert la variable `i` ? .....

5. Mettre en commentaire le programme précédent et écrire sous celui-ci ce programme :

```
for i in range(1,5):
    print(i)
```

(a) Que fait ce programme ? .....

(b) Que peut-on dire de l'instruction `range(x,y)` ? .....

6. Mettre en commentaire le programme précédent et écrire sous celui-ci ce programme :

```
for i in range(1,5,2):
    print(i)
```

(a) Que fait ce programme ? .....

(b) Que peut-on dire de l'instruction `range(x,y,z)` ? .....

Nous allons à présent reprendre notre problème du nombre mystère.

1. Créer un fichier **nb_mystere_for.py**
2. Recopier le programme suivant (il ressemble à la fonction `nb_mystere` que vous avez écrite précédemment) :

```
print("Trouve le nombre mystère :")

for i in range(1):
    a=float(input("Donner un nombre :"))
    if a>42:
        print("Trop grand !")
    if a==42:
        print("Bravo, tu as trouvé le nombre mystère !")
    if a<42:
        print("Trop petit !")
```

3. Faire en sorte (grâce à une boucle `for`) que l'utilisateur ait 5 essais
4. Modifier le programme pour que le numéro de l'essai soit affiché
5. Sous la ligne `print("Bravo, tu as trouvé le nombre mystère !")` mettre `break`. Vous devriez avoir ça :

```
if a==42:
    print("Bravo , tu as  trouvé le  nombre  mystère !")
    break
```

6. A quoi sert l'instruction `break` ? .....

3.4 A votre tour!

Dans cette partie, vous allez trouver des exercices en rapport avec la boucle `for`.

EXERCICE 6
Pour cet exercice, il faudra créer un fichier **table_sept.py**

1. Écrire une fonction `table_sept` qui ne prend pas d'argument et qui **affiche** les 20 premiers termes de la table de multiplication par 7
2. Améliorer votre fonction : les multiples divisibles par 3 devront être signaler avec un « – > »en début de ligne

EXERCICE 7
Pour cet exercice, il faudra créer un fichier **utilisation_for.py**

1. Écrire une fonction `for_normal` qui prend en argument un nombre `x` et qui affiche les $x^{ième}$ premiers nombres (de 0 à $x-1$)
2. Écrire une fonction `for_debut_fin` qui prend en argument 2 nombres `x` et `y` et qui affiche les nombres de `x` à `y`
   (a) Lancer l'appel `for_debut_fin(2,1)`, que se passe-t-il? .....
   (b) A votre avis pourquoi? .....
3. Écrire une fonction `for_deux_en_deux` qui prend en argument un nombre `x` et qui affiche les $x^{ième}$ premiers nombres de 2 en 2
4. Écrire une fonction `for_inverse` qui prend en argument un nombre `x` et qui affiche les $x^{ième}$ premiers nombres de x à 0
5. Écrire une fonction `for_dix_en_dix_inverse` qui prend en argument un nombre `x` et qui affiche les nombres de x à 0 de 10 en 10

EXERCICE 8
Pour cet exercice, il faudra créer un fichier **conversion_monetaire.py**

Écrire une fonction `euros_dollars` qui ne prend pas d'argument et qui affiche les conversions les dizaines d'euro en dollar. On fixe la limite à 100.

1 euro = 1,06 dollar

EXERCICE 9
Pour cet exercice, il faudra créer un fichier **factorielle.py**

Écrire une fonction `factorielle` qui prend en argument un nombre `n` et qui renvoie la factorielle de n. Si n est inférieur ou égal à 0, la fonction devra renvoyer None.

La factorielle d'un nombre :
— s'écrit $n!$
— est égal au produit des nombres de 1 à n
— la factorielle de 1 est égal à 1