TP6 - 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_2 ». C'est dans ce dossier qu'il faudra mettre tous les exercices du TP.

Chaque exercice va vous demander de créer des formes en utilisant le caractère '*'. Pour chaque exercice, vous devrez utiliser des boucles (`while` ou `for`).

Attention une instruction `for i in range(1):` ne compte pas comme étant une boucle.

EXERCICE 1

Pour cet exercice, il faudra créer un fichier **colonne.py**

Écrire une fonction `colonne` qui prend en argument un nombre (`n`) et qui **affiche** une colonne de `n` étoiles comme suit :

```
>>> colonne(5)
*
*
*
*
*
```

En Python, il est possible de modifier la fonction `print()`. Pour rappel, elle permet d'afficher quelques choses dans la console. Cette fonction possède ce que l'on peut appeler des **paramètres par défaut**, c'est-à-dire des paramètres prédéfinis par Python **MAIS** qui sont modifiables par le programmeur. Un des paramètre de la fonction `print()` est `end` qui permet de définir le caractère de fin de ligne.

EXERCICE 2

Pour cet exercice, il faudra créer un fichier **ligne.py**

1. Recopier le code suivant dans le fichier que vous venez de créer :

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

2. Écrire en commentaire ce qu'il se passe quand vous lancez ce code
3. Sur le même principe que le code précédent, écrire une fonction `ligne` qui prend en argument un nombre (`n`) et qui **affiche** une ligne de `n` étoiles.

```
>>> ligne(5)
*****
```

Rappel : Nous avons vu lors du TP sur les chaînes de caractères qu'il était possible de répéter une chaîne un certain nombre de fois en utilisant l'opérateur « * » (`print("salut"*3)` affiche 3 fois la chaîne `salut`).

EXERCICE 3

Pour cet exercice, il faudra créer un fichier **carre.py**

En vous aidant du rappel ci-dessus et d'**une boucle** `for`, écrire une fonction `carre_plein` qui prend en argument un nombre (`n`) et qui **affiche** un carré plein de `n` étoiles de côté.

```
>>> carre_plein(5)
*****
*****
*****
*****
*****
```

EXERCICE 4

Pour cet exercice, il faudra créer un fichier **triangle.py**

Dans la même optique que la fonction précédente, écrire une fonction `triangle_rectangle_inferieur` qui prend en argument un nombre (`n`) et qui **affiche** un triangle rectangle inférieur de `n` étoiles de haut.

```
>>> triangle_rectangle_inferieur(10)

*
**
***
****
*****
******
*******
********
*********
```

Aide : la solution la plus simple peut se faire avec une boucle `for`

Comme pour les `if`, il est possible en Python de mettre des `for` dans des `for`. Le deuxième s'appellera alors un `for imbriqué`.

EXERCICE 5

Pour cet exercice, il faudra modifier le fichier **triangle.py**

1. Dans le fichier que vous venez de créer, recopier le code suivant :

```
for i in range(5):
    print("i =", i)
    for j in range(5):
        print("j =", j, end="; ")
    print() #retour à la ligne après l'affichage des valeurs de j
    print() #permet de sauter une ligne
```

   Dans le programme précédent, vous pouvez voir que la boucle `for j in range(5):` est la boucle imbriquée. **Attention avec les boucles imbriquées**, la variable de la boucle imbriquée ne peut pas avoir le même nom que la variable de l'autre boucle! Ainsi, dans notre exemple, la variable de la boucle imbriquée ne peut pas être `j`

2. En utilisant une boucle imbriquée, écrire une fonction `triangle_rectangle_superieur` qui prend en argument un nombre (`n`) et qui **affiche** un triangle rectangle supérieur de `n-1` étoiles de haut et `n` étoiles de côté.

```
>>> triangle_rectangle_superieur(7)
 ******
  *****
   ****
    ***
     **
      *
```

Aide : si vous avez du mal, voici une petite aide. Vous allez devoir utiliser une boucle imbriquée (donc une boucle `for` dans une boucle `for`). Voici deux schémas qui représentent la situation :

EXERCICE 6

Pour cet exercice, il faudra modifier le fichier **carre.py**

Écrire une fonction `carre_bordure` qui prend en argument un nombre (`n`) et qui **affiche** la bordure d'un carré de `n` étoiles de côté.

```
>>> carre_bordure(10)
**********
*        *
*        *
*        *
*        *
*        *
*        *
*        *
*        *
**********
```

Aide :

1. Afficher la première ligne d'étoiles

```
>>> carre_bordure(10)
*********
```

2. Afficher la dernière ligne d'étoiles

```
>>> carre_bordure(10)
**********
```

3. Afficher la première colonne d'étoiles

```
>>> carre_bordure(10)
**********
*
*
*
*
*
*
*
*
**********
```

4. Afficher la dernière colonne d'étoiles

```
>>> carre_bordure(10)
**********
*         *
*         *
*         *
*         *
*         *
*         *
*         *
*         *
**********
```

2 Un peu d'`import`

Python est l'un des langage les plus utilisés dans le monde, par conséquent, beaucoup de personnes contribuent pour améliorer le langage. Ces améliorations apportées au langage sont rangées dans ce qu'on appelle des **librairies**. Vous pouvez importer des librairies pour utiliser les fonctions qu'elles contiennent en mettant **tout en haut de votre fichier Python** : `import librairie` où `librairie` est le nom de la librairie à importer. Il est également possible d'importer certains éléments d'une librairie au lieu de toute la libraire entière. La commande qui permet de faire ça est la suivante : `from librairie import module` où `librairie` est le nom de la librairie, `module` est le nom de l'élément (module) à importer. Ces informations sont disponibles dans le récapitulatif Python sur le réseau.

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

Durant le TP noté, vous aviez écrit une fonction permettant de calculer le périmètre et une fonction permettant de calculer l'aire d'un cercle. Ces fonctions sont mises ci-dessous :

```
def perimetre(r):
    """
    calcul le perimetre du disque de rayon r
    r : int ou float
    sortie : float, le perimetre du disque
    """
    if r < 0:
        return None
    return 2*3.14*r


def aire(r):
    """
    calcul l'aire du disque de rayon r
    r : int ou float
    sortie : float, l'aire du disque
    """
    if r < 0:
        return None
    return 3.14*r**2
```

Nous avions mis comme valeur de `pi` 3.14, ce qui n'est pas totalement faux, mais c'est une valeur approchée. En python il existe une variable `pi` dont la valeur est **beaucoup** plus précise que 3.14.

Chercher sur Internet dans quelle librairie Python est la variable `pi`, l'importer dans votre programme et modifier 3.14 par la variable importée.

```
>>> perimetre(5)
31.41592653589793

>>> aire(5)
78.53981633974483
```

EXERCICE 8

Pour cet exercice, il faudra créer un fichier **aleatoire.py**

1. Dans le fichier que vous venez de créer, recopier le code suivant :

```
from random import randint

x = randint(0, 10)
print(x)
```

2. En commentaire de la fonction, répondre aux questions suivantes :
   — De quelle librairie est importée `randint` ?
   — `randint` est une fonction ou une variable ? Pourquoi ?
   — Que permet de faire `randint` ? Si vous n'avez pas trop d'idée ou n'êtes pas sûr, mettez les lignes 3 et 4 dans une boucle `for` pour voir plusieurs exécution
3. Écrire une fonction `genere_nombre` qui prend en argument un nombre. La fonction devra générer un nombre aléatoire en 0 et ce nombre. Le nombre généré devra être retourné pour la fonction.
4. Sous la fonction précédente, écrire une fonction `devine_nombre` qui prend en argument un nombre `n`. Ce nombre sera la valeur maximale de notre jeu. Cette fonction devra :
   (a) appelé la fonction `genere_nombre` et récupérer la valeur retournée dans une variable
   (b) demander une nombre à l'utilisateur
   (c) tant que le nombre généré **est différent du** nombre entré, afficher que le nombre n'est pas celui-ci et donner un indice
      — si le nombre généré est plus grand, alors il faudra afficher « Plus«
      — si le nombre généré est plus petit, alors il faudra afficher « Moins«
   (d) dans la boucle il faudra aussi demander à l'utilisateur de saisir une nouvelle fois un nombre (ceci est important sinon vous aurait une boucle infinie !)
   (e) une fois le nombre trouvé, afficher "Vous avez trouvé le nombre mystère"

Exemple de rendu :

```
>>> jouer(10)
Entrer un nombre : 5
Plus
Entrer un nombre : 7
Plus
Entrer un nombre : 9
Moins
Entrer un nombre : 8
Vous avez trouvé le nombre mystère
```

3 Exercices plus difficiles

EXERCICE 9
Pour cet exercice, il faudra modifier le fichier **triangle.py**

Écrire une fonction `triangle_centre_superieur` qui prend en argument un nombre (`n`) et qui affiche un triangle centré dans la partie supérieur de `n-1` étoiles de côté.

```
>>> triangle_centre_superieur(10)
 *********
  *******
   *****
    ***
     *
```

EXERCICE 10
Pour cet exercice, il faudra modifier le fichier **triangle.py**

Écrire une fonction `triangle_centre_gauche` qui prend en argument un nombre (`n`) et qui affiche un triangle centré dans la partie gauche.

```
>>> triangle_centre_gauche(8)

*
**
***
****
***
**
*
```

EXERCICE 11
Pour cet exercice, il faudra créer un fichier **croix.py**

Écrire une fonction `croix` qui prend en argument un nombre (`n`) et qui affiche une croix de `n` étoiles de côté.

```
>>> croix(10)
*         *
 *       *
  *     *
   *   *
    ** 
    **
   *   *
  *     *
 *       *
*         *
```