

TD : le module `turtle`

Planche de TD pour un cours dispensé à l'université de Rennes 2

Romain Tavenard

Le but de cette séance est de continuer à vous habituer à la programmation en Python et notamment aux notions de :

- structures conditionnelles ;
- boucles ;
- fonction.

Pour cela, nous manipulerons le module `turtle` dont le principe est de suivre dans une fenêtre l'évolution d'une tortue (symbolisée par une flèche) qui suivra vos instructions à la lettre.

1 Organisation de votre code

Pour ce TD, vous créerez un nouveau fichier `td3.py` dans le répertoire que vous avez créé à la première séance. Dans ce fichier, votre code sera organisé de la manière suivante :

```
def immeuble(n_etages, n_fenetres):  
    [...]
```

```
# Tests  
immeuble(n_etages=3, n_fenetres=2)  
[...]
```

Notamment, vous définirez vos fonctions en début de fichier et les appels seront listés en fin de fichier. De cette manière, vous pourrez, d'une question à l'autre, réutiliser les fonctions déjà codées au besoin.

Si vous souhaitez, à un certain moment, ne plus ré-exécuter vos tests de début de TD, il suffira de commenter les appels de fonction correspondants (et pas les fonctions elles-mêmes !).

2 Échauffement

1. Quels sont les prototypes (liste d'arguments et leur type, liste de valeurs de retours et leur type) des fonctions `rectangle`, `carre` et `immeuble` présentées dans la section [L'immeuble](#) ci-dessous ?

3 Présentation du module `turtle`

Lors de cette séance de TD, vous serez amenés à faire appel aux fonctions de base du module `turtle` suivantes :

- `turtle.forward(dist)` : faire avancer la tortue de `dist` unités ;
- `turtle.left(alpha)` : faire tourner la tortue sur la gauche d'un angle de `alpha` degrés ;
- `turtle.right(alpha)` : faire tourner la tortue sur la droite d'un angle de `alpha` degrés ;
- `turtle.up()` : faire léviter la tortue (sa trace ne s'écrit donc plus à l'écran) ;
- `turtle.down()` : stopper la lévitation de la tortue ;
- `turtle.goto(x, y)` : faire se déplacer la tortue jusqu'à la position `(x, y)`.

4 Lecture d'un programme

2. Sans l'exécuter pour le moment, tentez de comprendre ce que fait le code suivant et de deviner ce qu'il affichera à l'écran :

```
import turtle

for i in range(100):
    turtle.forward(10 + i)
    turtle.left(90)

turtle.exitonclick() # Attend un clic avant de fermer la fenetre
```

3. Vérifiez votre prédiction en exécutant ce code.

5 L'étoile à 5 branches

4. Dessinez à l'écran une étoile à 5 branches similaire à celle-ci :



6 L'immeuble

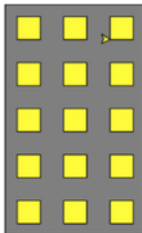
Pour cet exercice, votre code final devra ressembler à :

```
import turtle

# [...]

n_etages = 5
n_fenetres = 3
immeuble(n_etages, n_fenetres)
turtle.exitonclick() # Attend un clic avant de fermer la fenetre
```

L'exécution de ce code devra faire dérouler à l'écran une animation se terminant sur le dessin suivant :



Pour cela, vous définirez 4 fonctions (à l'emplacement des points de suspension dans l'extrait de code ci-dessus) :

- **position** permettra de placer la tortue à la position spécifiée, sans que celle-ci ne laisse de trace à l'écran ;
- **rectangle** permettra de dessiner un rectangle à l'écran ;
- **carre** permettra de dessiner un carré à l'écran ;
- **immeuble** permettra de dessiner un immeuble tel que celui représenté ci-dessus à l'écran.

6.1 La fonction rectangle

5. Écrivez une fonction `rectangle` qui permette de tracer à l'écran un rectangle de taille et de position spécifiées lors de l'appel de la fonction. Pour coder

cette fonction `rectangle`, vous devrez au passage également implémenter la fonction `position` décrite ci-dessus.

6.2 La fonction `carre`

6. Écrivez une fonction `carre` qui permette de tracer à l'écran un carré de taille et de position spécifiées lors de l'appel de la fonction. Est-il nécessaire de réécrire la fonction dans son ensemble (avec les appels successifs aux fonctions du module `turtle`) ou est-ce possible de s'en sortir en une ligne de code ?

6.3 La fonction `immeuble`

7. Écrivez une fonction `immeuble` qui permette de tracer un immeuble à l'écran, connaissant son nombre d'étages et le nombre de fenêtres par étage. N'hésitez pas à faire un schéma de l'immeuble sur papier pour vous rendre compte des dimensions à utiliser. Vous définirez au sein de la fonction `immeuble` les grandeurs caractéristiques nécessaires (taille des fenêtres, espace inter-fenêtres, *etc.*).

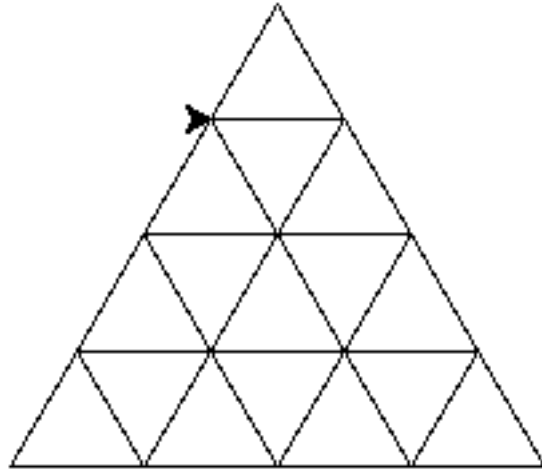
6.4 Un peu de *tuning*

8. Ajoutez aux fonctions nécessaires un paramètre facultatif qui permette de spécifier la couleur de remplissage des formes géométriques tracées. Utilisez ce paramètre facultatif pour demander de tracer l'immeuble en gris ("`grey`") et les fenêtres en jaune ("`yellow`"). Pour cela, vous aurez besoin des fonctions `turtle.fillcolor(couleur)`, `turtle.begin_fill()` et `turtle.end_fill()` qui s'utilisent comme suit :

```
turtle.fillcolor(couleur)
turtle.begin_fill()
# Ici, tracer le polygone
turtle.end_fill()
```

7 Pour aller plus loin : le château de cartes

Dans cet exercice, vous allez tenter de dessiner à l'écran un château de cartes (fait de triangles superposés) similaire à celui-ci :



Pour cela, vous devrez tout d'abord être capable de tracer un triangle équilatéral à une position donnée.

9. Écrivez une fonction qui prenne en entrée une position (sous la forme de deux entiers x et y) et une taille c et trace à l'écran un triangle équilatéral de côté c ayant son bord inférieur gauche situé à la position (x, y) .
10. Avant de passer à la question suivante, importez le module `math` de Python qui permet d'accéder à des fonctions usuelles telles que le [sinus](#), le [cosinus](#) ou la [racine carrée](#). À quel endroit dans votre fichier Python devez-vous écrire cette instruction d'import ?
11. Écrivez une fonction qui prenne en entrée un nombre n et trace à l'écran un château de cartes dont la base est constituée de n triangles. Vous définirez au sein de cette fonction une variable correspondant à la taille des triangles du château de cartes.