Mathématiques supérieures — le 27 Septembre 2019

Corrigé du TP n°3 - Mise en pratique : le module `turtle`

Nous avons appris dans le premier Cours-TP à prendre en main l'environnement de travail EduPython et nous avons fait nos premiers pas avec le langage Python. Dans le second Cours-TP, nous avons étudié différentes structures du langage Python : les fonctions, les structures conditionnelles, et les structures itératives.

Le but de ce TP est de mettre en œuvre ces acquis : nous allons utiliser un module (une bibliothèque de fonctions) nommé `turtle` qui va nous permettre, avec des instructions minimalistes, de tracer des dessins dans une fenêtre indépendante.

Commençons par importer le module en question, en tapant dans le shell puis en exécutant :

```
>>>from turtle import *
```

Attention, en cas de redémarrage de l'interpréteur Python, il faudra exécuter de nouveau cette instruction pour continuer à pouvoir utiliser les fonctions contenues dans le module `turtle`.

Maintenant, il nous faut ouvrir la fenêtre dans laquelle nous allons tracer nos dessins : tapons puis exécutons la commande suivante, toujours dans le shell :

```
>>>reset()
```

Ainsi, une nouvelle fenêtre dont le contenu est blanc s'affiche avec un curseur noir en son centre : ce point représente la "tortue" (oui, il faut de l'imagination !) qui va, en se déplaçant sur nos instructions - c'est le cas de le dire - laisser sa "trace". La tortue est située au point de coordonnées $(0, 0)$ de la fenêtre et sa tête est dirigée vers la droite : elle forme un angle de $0°$ par rapport à l'horizontale.

1. Dessinons avec la tortue !

Nous allons écrire des programmes nous permettant de tracer diverses formes dans la fenêtre de dessins ; mais tout d'abord, essayons de comprendre comment se déplace notre tortue.

Testons, DANS LE SHELL, les commandes suivantes, en les exécutant une par une - afin de bien comprendre ce qu'elle produisent au niveau de la tortue :

```
>>>forward(100)
>>>left(120)
>>>forward(100)
>>>left(120)
>>>forward(100)
```

Que produisent ces instructions prises séparément dans la fenêtre de dessin ?

— La fonction `forward` permet de déplacer du nombre de pixels indiqué en argument la tortue dans la direction indiquée par sa tête. Par exemple, `forward(123)` fera avancer la tortue de 123 pixels dans la direction indiquée par la tête au moment de l'exécution.

En indiquant un nombre de pixels négatif en argument de la fonction `forward`, la tortue reculera du nombre de pixel en valeur absolue (toujours dans la direction indiquée par la tête).

— La fonction `left` faire pivoter la tortue de l'angle **en degré** indiqué en argument dans le sens trigonométrique. Par exemple, `left(60)` fera pivoter la tortue de 60° dans le sens trigonométrique. De même, en indiquant un nombre négatif en argument à la fonction `left`, la tortue pivotera de ce nombre de degré en valeur absolue mais dans le sens horaire. Remarque, on peut obtenir le même effet grâce à la fonction... ... `right` !

Pour réinitialiser la fenêtre de dessin et replacer la tortue dans sa position initiale, il suffit d'employer la commande qui nous a permis d'ouvrir la fenêtre de dessin :

```
>>>reset()
```

À partir de ce point, nous allons écrire toutes nos fonctions dans l'éditeur afin de les sauvegarder, et pour les exécuter plus facilement. N'oubliez pas d'ENREGISTRER votre travail régulièrement !!

Maintenant que nous avons compris ces instructions de base, programmons une fonction `triangle` qui prend pour argument un entier `n` et qui réinitialise la fenêtre de dessin, puis qui trace - enfin, qui fait tracer à la tortue - un triangle équilatéral de côté de longueur `n` pixels :

```
def triangle(n):
    """ Trace un triangle équilatéral de côté de longueur n pixels """
    reset()
    forward(n)
    left(120)
    forward(n)
    left(120)
    forward(n)
```

Testons cette fonction dans le shell pour plusieurs longueurs :

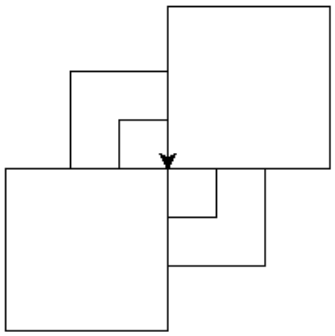
```
>>>triangle(222)

>>>triangle(40)

>>>triangle(-200)
```

Exercice 1.

1. Programmer une fonction `carre` d'argument un entier `n` qui permette de tracer un carré de côté `n` pixels et qui replace la tortue dans sa position précédente (avant le début du tracé du carré) à la fin du tracé. ATTENTION : pour cette fonction, on ne veut pas réinitialiser la fenêtre de dessin avant le tracé du carré, on n'utilisera pas la fonction `reset()` DANS la fonction.
2. Reproduire le dessin suivant en utilisant plusieurs fois la fonction `carre` :



Correction.

1.
```
def carre(n):
    """ Trace un carré de côté de longueur n pixels et replace la tortue en
        position précédente """
    forward(n)
    left(90)
    forward(n)
    left(90)
    forward(n)
    left(90)
    forward(n)
    left(90)
```

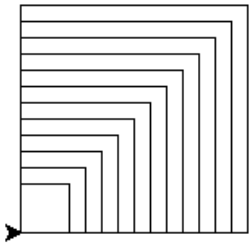
2.
```
reset()
carre(100)
left(90)
carre(30)
carre(60)
left(90)
carre(100)
left(90)
carre(60)
carre(30)
```

Désormais, utilisons ce que nous avons appris à propos des structures itératives (les boucles `for` et `while`).

Exercice 2.

Programmer une fonction `carresemboites` qui prend pour arguments un entier `n` et un entier `nb` plus grand que 1 qui permette de tracer un carré initial de côté `n` puis `nb-1` carrés autour de celui-ci en augmentant à chaque nouveau carré tracé de 10 pixels la longueur du côté. On pourra bien-sûr se servir de la fonction `carre` déjà programmée !

Pour mieux comprendre, voici un exemple de ce que doit produire `carresemboites(30,12)` :



Correction.

```
def carresemboites(n,nb):
    for k in range(nb):
        carre(n+10*k)
```

Remarque 1.

Petite astuce concernant notre pauvre tortue qui est très lente par nature ! On peut lui donner des vitamines afin qu'elle puisse tracer plus vite nos dessins (car il deviendront de plus en plus long à tracer sinon) :

La fonction suivante : `speed` permet de modifier la vitesse de tracé de la tortue. L'argument de la fonction `speed` est un entier allant de 0 à 10.

- 1 correspond à la vitesse minimale possible pour la tortue ;
- 6 correspond à la vitesse normale de la tortue (la vitesse par défaut) ;
- 10 correspond à la vitesse presque maximale de la tortue ; et
- 0 correspond à de la salade radioactive : la vitesse de la tortue est cette fois-ci, vraiment maximale !

On peut bien-sûr moduler la vitesse entre ces valeurs selon nos convenances.

Voici un exemple de changement de vitesse :

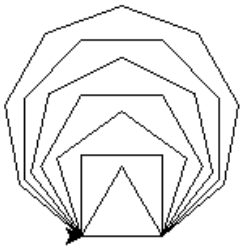
```
>>>speed(1) # la tortue est très lente

>>>speed(0) # la tortue est extrêmement rapide
```

ATTENTION ! La vitesse est réinitialisée après chaque appel de `reset()`. N'oubliez pas de spécifier de nouveau la vitesse qui vous convient après une réinitialisation de la fenêtre de dessin !

Exercice 3.

1. Définir une fonction `polygoneregulier(n,L)` qui trace un polygone régulier à `n` côtés de longueur `L` et qui replace la tortue dans sa position précédant le tracé du polygone.
2. Définir une fonction `polygonesemboites(n,L)` qui trace les polygones réguliers à `k` côtés de longueur `L` pour `k` variant de 3 à `n` emboîtés les uns dans les autres. Voici ce que doit produire `polygonesemboites(10,50)` :



3. Tester `polygoneregulier(150,5)`. Que dire de la forme que l'on semble obtenir sur le dessin ?

Correction.

1.
```
def polygoneregulier(n,L):
    for i in range(n):
        forward(L)
        left(360/n)
```

2.
```
def polygonesemboites(n,L):
    for k in range(3,n):
        polygoneregulier(k,L)
```

3. Waouh, ça ressemble à un cercle !

Mais alors, est-ce que notre tortue sait tracer des cercles d'un rayon donné ? Oui bien-sûr, mais en trichant ! En fait, elle fait comme nous, elle trace un polygone régulier avec beaucoup de côtés pour créer l'illusion d'un cercle.

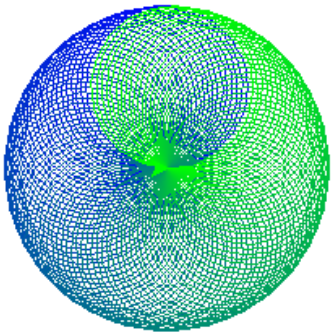
La fonction `circle` permet de tracer un cercle, voici ses 2 arguments :

— Le premier argument de `circle` est le rayon en pixel du cercle à tracer à partir de la tête de la tortue ; si le rayon est positif, le tracé se fait dans le sens trigonométrique, s'il est négatif, dans le sens horaire. Ainsi, la commande `circle(100)` tracera dans le sens trigonométrique, un cercle de rayon 100.

— Le deuxième argument nommé `extent` est facultatif et prend pour valeur par défaut 360. Cette argument permet d'indiquer la portion de cercle à tracer à partir de la tête de la tortue : la commande `circle(100,60)` tracera un arc de cercle de rayon 100 de l'angle 0° à partir de la tête, jusqu'à un angle de 60°.

Exercice 4.

Reproduire la figure ci-dessous (en n'oubliant pas d'augmenter la vitesse : `speed(0)` sinon... ce sera long !)



L'image contient 100 cercles de rayon 50 pixels. Dans un premier temps, on se contentera de tracer ce dessin en noir.

Pour changer la couleur d'un tracé, on utilise la fonction `color(couleur)` où `couleur` est une chaîne de caractère désignant une couleur (en anglais). Par exemple, `color('blue')` indique que la couleur des futurs tracés sera bleue.

On peut également fournir un tuple correspondant à une couleur RVB sachant que `1.0,0,0` correspond au rouge, `0,1.0,0` correspond au vert et `0,0,1.0` correspond au bleu. en modifiant les trois nombres entre 0.0 et 1.0, on obtient des mélanges de ces trois couleurs (de manière plus

ou moins claires). Ainsi, `color(1.0,0.55,0.0)` correspondra à une couleur orangée.

Correction.

```
def supercercle(n):
    for i in range(n):
        color(0.0,1.0*(i/(n-1)),1.0*(n-1-i)/(n-1))
        circle(50)
        left(360/n)
```

Exercice 5.

Tracer le plus joliment possible les lettres "MATHS SUP" dans la fenêtre de dessin.

On pourra utiliser les fonctions `penup()` et `pendown()` qui permettent respectivement d'arrêter ou de reprendre le traçage pendant un mouvement de la tortue.