Travaux pratiques

informatique commune

Graphisme « tortue »

Découverte de l'environnement de travail

Cette première séance de travaux pratiques va être l'occasion de vous familiariser avec l'environnement de travail fourni par la distribution Pyzo. Celle-ci a installé une version récente de Python, les modules scientifiques dont nous aurons besoin plus tard dans l'année, et un environnement de travail nommé IEP (*Interactive Editor for Python*). Débutez votre travail en lançant indifféremment l'application nommée pyzo ou iep (cela revient au même). Vous vous trouvez maintenant face à une fenêtre comportant plusieurs panneaux. S'il n'a pas été lancé automatiquement vous pouvez commencer par lire une courte introduction à IEP par le biais du menu Aide → Guide IEP (disponible en plusieurs langues).
Une fois votre lecture terminée, vous aurez compris que seuls deux composants sont indispensables : l'éditeur et le shell. Fermez les autres panneaux s'ils sont ouverts, et redimensionnez les deux panneaux restants (en utilisant la souris) de manière à partager équitablement l'espace (horizontalement ou verticalement suivant votre préférence).

- Dans l'éditeur, plusieurs fichiers peuvent être ouverts simultanément (étape 2 du guide). Si c'est le cas, fermez ceux qui ne vous appartiennent pas et ouvrez un fichier vierge. À la fin de la séance vous pourrez le sauvegarder (de préférence sur un support amovible vous appartenant si vous utilisez un des ordinateurs du lycée) et poursuivre votre travail chez vous ou lors d'une prochaine séance. Notez bien que seul le contenu de l'éditeur peut être sauvegardé, en aucun cas le contenu du shell !
- C'est dans le shell que s'exécute le code. Plusieurs modes d'exécution sont possibles : directement dans le shell en réponse à l'invite de commande (`>>>` ou `In [1]:` suivant le shell) ou en exécutant tout ou partie de la fenêtre d'édition active (étape 5 du guide).

Le module turtle

Python est un langage destiné à de nombreux usages, c'est pourquoi sa conception est modulaire : au début d'une session, seul un nombre minimal de fonctions indispensables est défini. Chaque utilisateur va ensuite, en fonction de ses besoins, importer un ou plusieurs modules ajoutant aux fonctions de base de nouvelles fonctions destinées à un usage plus spécifique.
Le module que nous allons utiliser aujourd'hui s'appelle turtle ; vous allez importer son contenu en tapant dans le shell, à la suite de l'invite de commande, l'instruction :

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

ou

```
In [1]: from turtle import *
```

Si tout se passe bien il ne doit rien se passer de visible, mais maintenant votre interprète de commande est à même de comprendre et d'exécuter les commandes contenues dans ce module, et dont les principales sont décrites en annexe à ce document.

Ce qu'on appelle aujourd'hui les graphismes « tortue » trouvent leur origine dans le langage Logo, inventé dans les années 1960 dans un but pédagogique. Ce langage comportait un certain nombre d'instructions qui permettaient de commander un robot pouvant tracer des lignes sur une feuille de papier. La forme de ce robot (qui évoquait vaguement une tortue) a donné son nom à cette manière de produire des dessins.
Le module turtle de Python implémente une tortue virtuelle qui reproduit les déplacements de son ancêtre dans une fenêtre de l'écran de votre ordinateur. C'est par son intermédiaire que nous allons découvrir quelques concepts de base de la programmation.
Toujours dans le shell, tapez l'instruction suivante :

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

Une nouvelle fenêtre doit apparaître à l'écran. Elle est peut-être cachée par Pyzo, donc redimensionnez votre espace de travail afin de pouvoir visualiser en même temps l'éditeur, le shell et cette nouvelle fenêtre. Celle-ci représente la feuille sur laquelle la tortue, représentée par défaut par la pointe d'une flèche, va réaliser ses dessins. Initialement la tortue est située au point de coordonnées $(0, 0)$ situé au centre de la feuille et est orientée vers l'est (qui correspond à un angle de 0°).

Un premier dessin

Rappelons-le, Python est un langage *interprété* : chaque instruction écrite dans le shell est traduite en langage machine par l'interprète de commande, puis exécutée. Recopiez dans le shell la succession de commandes suivante, en observant pour chacune d'elles l'effet produit dans la fenêtre graphique :

```
>>> forward(200)
>>> left(120)
>>> forward(200)
>>> left(120)
>>> forward(200)
```

Cette démarche ne permet d'exécuter qu'une instruction Python à la fois. Pour réaliser une succession d'instructions les unes à la suite des autres, il faut utiliser l'éditeur pour réaliser ce qu'on appelle un *script*. Les conventions d'édition du langage sont qu'une seule instruction doit figurer par ligne. Vous allez donc recopier dans l'éditeur le script suivant :

```
reset()
forward(200)
left(120)
forward(200)
left(120)
forward(200)
```

puis l'exécuter (les commandes relatives à l'exécution d'un script sont regroupées dans le menu Exécuter). Vous pouvez exécuter le script entier (commande « Exécuter le fichier ») ou une sélection de celui-ci (commande « Exécuter la sélection ») [1]. Cette fois, la tortue va dessiner le triangle d'un seul tenant.

Exercice 1. Remettre à zéro le contenu de la fenêtre graphique, puis faire dessiner par la tortue un carré.

Le but de l'exercice suivant va être de reproduire le dessin de la figure 1.

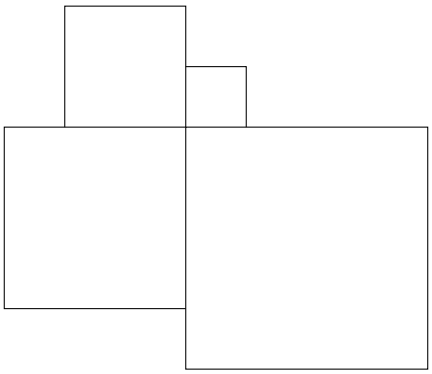

Figure 1 – Un dessin à reproduire dans l'exercice 2.

Il pourrait être utile de disposer d'une fonction dessinant un carré de taille n ; puisqu'il n'existe pas de telle fonction, nous allons la définir.
Pour définir une fonction il faut lui donner un nom, préciser la liste de ses paramètres et enfin décrire les différentes instructions à réaliser. La syntaxe générale est la suivante :

```
def nomdelafcn(liste de paramètres):
    bloc ..........................
    d'instructions .................
    à réaliser .....................
```

Attention à bien respecter la syntaxe de l'instruction `def`. La ligne contenant cette instruction doit obligatoirement se terminer par un double point, et le bloc d'instructions qui suit doit être *indenté* (c'est à dire décalé vers la droite de 4 espaces). Heureusement, si vous n'avez pas fait d'erreur de syntaxe l'éditeur de code se chargera pour vous d'indenter automatiquement votre code [2].

1. Dans la pratique, exécuter une sélection peut se révéler fastidieux à la longue car il faut à chaque fois utiliser la souris pour sélectionner la partie du script à exécuter. Pyzo propose une alternative très pratique, l'exécution d'une *cellule*. Une cellule est tout ce qui se trouve entre deux lignes débutant par `##` (les lignes qui commencent par un caractère `#` sont ignorées par l'interprète de commande, elles servent à commenter une portion de script). Pour exécuter une cellule, il suffit que le curseur soit dans celle-ci. Par la suite, vous pourrez avoir intérêt à créer une cellule par exercice.
2. Cependant, n'oubliez pas de revenir à une indentation normale une fois la définition de la fonction terminée.

Par exemple, pour définir la fonction qui nous intéresse et que nous allons nommer `carre`, on commencera par écrire :

```
def carre(l):
    bloc ..........................
    d'instructions .................
    à réaliser .....................
```

et dans le bloc d'instructions à réaliser il faudra donner les commandes nécessaires pour tracer un carré de côté l.

Exercice 2. Achever la définition de la fonction carré, puis à l'aide de celle-ci rédiger un script pour reproduire le dessin de la figure 1.

Nous allons maintenant chercher à reproduire le dessin de la figure 2.

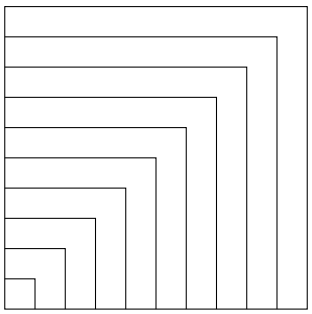

FIGURE 2 – Un dessin à reproduire dans l'exercice 3.

À l'aide de la fonction `carre` le script n'est pas difficile à réaliser mais peut s'avérer fastidieux à écrire. Pour en faciliter l'écriture nous allons introduire la notion d'*énumération*. Si `enumobject` désigne un objet PYTHON énumérable, le script :

```
for x in enumobject:
    bloc ..........................
    d'instructions .................
    à réaliser .....................
```

va successivement attribuer à la variable `x` chacun des éléments de cette énumération et pour chacun d'eux exécuter le bloc d'instructions. C'est un concept important en PYTHON, et de nombreux objets sont énumérables (de manière intuitive le plus souvent). Nous en découvrirons un certain nombre au cours de l'année, mais pour l'instant, un seul nous sera utile durant cette séance : si a, b et c sont trois entiers,

- `range(a)` énumère tous les entiers compris entre 0 (au sens large) et a (au sens strict), autrement dit les entiers 0, 1, 2, … $a-1$;
- `range(a, b)` énumère tous les entiers compris entre a (au sens large) et b (au sens strict), autrement dit les entiers a, $a+1$, $a+2$, … $b-1$;
- `range(a, b, c)` énumère tous les entiers compris entre a (au sens large) et b (au sens strict) avec un pas de c, autrement dit les entiers a, $a+c$, $a+2c$, … $a+nc$ avec $a+nc < b \leqslant a+(n+1)c$.

Exercice 3. À l'aide d'une énumération, reproduire le dessin de la figure 2.

Effacer l'écran, puis augmenter la vitesse de la tortue jusqu'à sa vitesse maximale car le dessin qui va suivre va être un peu long à tracer.

Exercice 4. À l'aide d'une énumération, faire avancer la tortue de 1, puis de 2, 3, 4, …, 500 pas en tournant d'un angle de 91 degrés entre chaque déplacement.

Exercice 5. Tracer un triangle équilatéral, un carré, un pentagone ou un hexagone réguliers ne sont pas des tâches fondamentalement différentes. Définir une fonction `polygone(n, l)` qui trace un polygone régulier à n côtés, chacun de longueur l, puis reproduire à l'aide de cette fonction la figure 3.

Effacer de nouveau l'écran à l'aide de la fonction `reset`, puis réaliser la commande `polygone(100, 5)`. La courbe obtenue vous étonne-t-elle ? Elle peut aussi être obtenue à l'aide d'une fonction prédéfinie du module TURTLE qui se nomme `circle`, mais dont les paramètres sont différents. Pour en connaître les spécifications, nous allons recourir à l'aide contenue dans le module en tapant dans l'interprète de commande l'instruction :

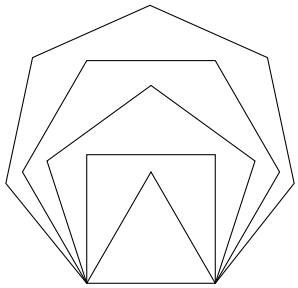

Figure 3 – Un dessin à reproduire dans l'exercice 5.

```
>>> help(circle)
```

Lisez la page qui s'affiche. Vous constaterez que cette fonction possède trois arguments, dont deux optionnels. Le second, `extent` (lorsqu'il est présent) permet de tracer un arc de cercle plutôt qu'un cercle complet ; le troisième, `steps`, permet de tracer des polygones.

Exercice 6. À l'aide de la seule fonction `circle`, reproduire le dessin de la figure 4 (deux lignes suffisent).

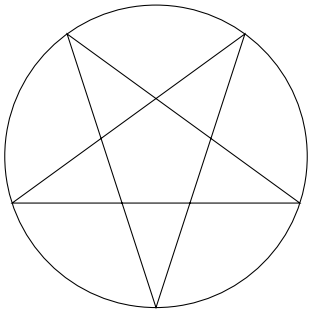

Figure 4 – Un dessin à reproduire dans l'exercice 6.

Exercice 7. Reproduire enfin le dessin de la figure 5. Ce dernier est constitué de 72 demi-cercles régulièrement espacés [3].

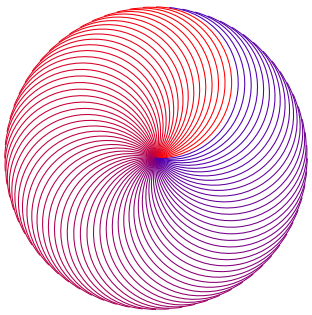

Figure 5 – Un dessin à reproduire dans l'exercice 7.

Avant de tracer cette figure, il pourra être utile d'accélérer la vitesse de la tortue pour éviter une attente trop longue.

Fractales

Le triangle de Sierpiński est une fractale qui s'obtient à partir d'un triangle plein par une infinité de répétitions consistant à diviser par 2 la taille du triangle puis à les accoler en trois exemplaires par leurs sommets pour former un nouveau triangle. On trouvera figure 6 la représentation graphique des trois premières générations de cette transformation.

3. Pour plus d'effet, vous pouvez reproduire cette figure sur fond noir, les différents cercles étant tracés dans des couleurs s'échelonnant entre le bleu et le rouge.

FIGURE 6 – Les trois premières étapes de la construction du triangle de SIERPIŃSKI.

Remarque. Pour colorer l'intérieur d'une courbe fermée, il faut faire précéder le début du tracé par la commande `begin_fill()` et terminer celui-ci par `end_fill()`. Par exemple, le script suivant :

```
begin_fill()
circle(100)
end_fill()
```

crée un disque plein.

Exercice 8.

a) Définir une fonction `sierp1(l)` qui dessine la première étape de la construction du triangle de SIERPIŃSKI, c'est-à-dire un triangle plein de longueur de côté ℓ.

b) À l'aide de la fonction précédente, définir une fonction `sierp2(l)` qui dessine la deuxième étape de la construction du triangle de SIERPIŃSKI.

c) À l'aide de la fonction précédente, définir une fonction `sierp3(l)` qui dessine la troisième étape de la construction du triangle de SIERPIŃSKI.

L'exercice précédent a du vous convaincre qu'à l'exception de la première, la n^e génération du triangle de SIERPIŃSKI s'exprime toujours de la même manière en fonction de la précédente. Pour exploiter cette remarque, nous allons utiliser une particularité des fonctions PYTHON : elles ont la possibilité de faire appel *à elle-mêmes* dans leur définition [4]. Autrement dit, si nous choisissons de passer en paramètre l'ordre n de la génération que nous voulons tracer en définissant une fonction `sierpinski(n, l)` il est possible, au sein de cette définition, de faire appel à la fonction `sierpinski(n-1, l)`. Mais pour cela, il faut distinguer le cas $n = 1$ qui se traite différemment ...

Pour effectuer cette distinction, on utilise l'instruction conditionnelle **`if`** dont la syntaxe est la suivante :

```
if test_à_réaliser:
    bloc..............
    d'instructions 1..
else:
    bloc..............
    d'instructions 2..
```

(Notez bien l'indentation qui permet de délimiter chacun des deux blocs d'instructions.)
Le fonctionnement de cette instruction est le suivant : si le résultat du test est positif, le premier bloc d'instructions est réalisé, dans le cas contraire c'est le second. Notez que l'instruction **`else`** est optionnelle si aucune instruction ne doit être réalisée dans le cas d'un test négatif.
Nous reviendrons en cours sur la nature des tests que l'on peut réaliser ; pour l'instant nous nous contenterons d'expressions simples telles que :

- `x < y` (x est strictement plus petit que y) ;
- `x > y` (x est strictement plus grand que y) ;
- `x <= y` (x est inférieur ou égal à y) ;
- `x >= y` (x est supérieur ou égal à y) ;
- `x == y` (x est égal à y) ;
- `x != y` (x est différent de y).

Exercice 9. Définir la fonction `sierpinski(n, l)` qui dessine la génération d'ordre n du triangle de SIERPIŃSKI avec une longueur de côté égale à l, puis utiliser cette fonction pour effectuer le tracé avec $n = 5$ (ne pas oublier d'accélérer au maximum la vitesse de la tortue car sinon le tracé sera très long).

Exercice 10. D'autres structures fractales peuvent être dessinées par la tortue, à commencer par l'arbre binaire représenté figure 7.
Pour construire l'arbre d'ordre n et de hauteur h, on procède de la manière suivante :

- on trace le tronc de hauteur $h/3$ et d'épaisseur n ;
- puis on trace ses deux branches, qui sont des arbres d'ordre $n-1$ et de hauteur $2h/3$ inclinés d'un angle de 30°.

Définir une fonction `bin_tree(n, h)` qui dessine l'arbre binaire d'ordre n et de hauteur h, puis dessiner l'arbre d'ordre 8.

4. On parle dans ce cas de fonctions *récursives*. Cette notion sera étudiée en détail dans le cours de seconde année.

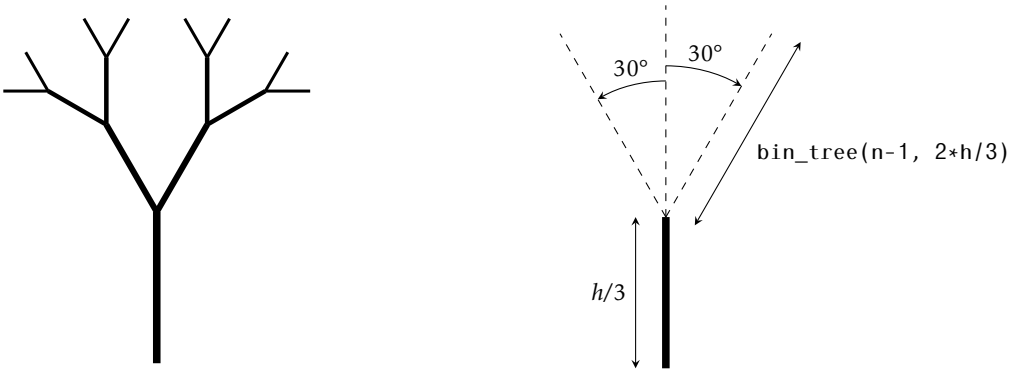

FIGURE 7 – L'arbre d'ordre $n = 4$ et sa règle de construction.

Et pour les plus rapides

Il est possible de gérer plusieurs tortues dans une même fenêtre graphique, mais dans ce cas la syntaxe change un peu. Par exemple, pour créer deux tortues que nous allons baptiser Alice et Bob, on écrit :

```
>>> Alice = Turtle()
>>> Bob = Turtle()
```

Les commandes de chacune des deux tortues ne sont plus alors données par des fonctions mais par des *méthodes*, c'est-à-dire sous la forme de suffixes [5]. Par exemple, pour faire tourner Alice d'un demi-tour et faire avancer Bob de 100 pas on écrira :

```
>>> Alice.left(180)
>>> Bob.forward(100)
```

Exercice 11. Course de tortues

Réaliser une course poursuite entre les deux tortues : placer Bob au point de coordonnées $(-400, -100)$ et lui faire parcourir une trajectoire rectiligne uniforme horizontale tandis qu'Alice, partant du point de coordonnées $(0, 400)$, se dirige invariablement vers Bob, avec une vitesse légèrement supérieure (par exemple, à chaque unité de temps Bob se déplace de 10 pas et Alice de 12).
Terminer la course lorsqu'Alice est suffisamment proche de Bob (leur distance est inférieure à 2 pas).

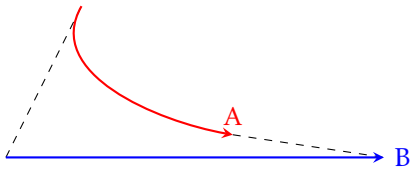

FIGURE 8 – Course poursuite rectiligne.

Faire de même en faisant cette fois parcourir à Bob une trajectoire circulaire uniforme centrée autour de la position initiale d'Alice. Que se passe-t'il lorsque la vitesse d'Alice n'est pas suffisante pour rattraper Bob ?

Réaliser enfin une course poursuite entre trois tortues Alice, Bob et Carole. Initialement ces trois tortues sont placées aux sommets d'un triangle équilatéral, puis A poursuit B qui poursuit C qui poursuit A.

5. Cette manière de procéder correspond à ce que l'on appelle la programmation *orientée-objet*. Plus tard on dira que Alice et Bob sont des *instances* de la classe `Turtle`, à qui on applique respectivement les *méthodes* `left` et `forward`.

Annexe : les principales commandes du module TURTLE

Déplacement de la tortue

- `forward(n)` avance la tortue de n pas dans la direction qui lui fait face ;
- `backward(n)` recule la tortue de n pas ;
- `left(n)` tourne la tortue d'un angle de n degrés dans le sens trigonométrique ;
- `right(n)` tourne la tortue d'un angle de n degrés dans le sens horaire ;
- `speed(n)` modifie la vitesse de déplacement de la tortue de $n = 1$ (lent) à $n = 10$ (rapide) ; $n = 0$ (le plus rapide) supprime toute animation. Par défaut $n = 3$.

Tracé du chemin de la tortue

- `penup()` soulève le crayon de sorte que le déplacement de la tortue ne trace pas de trait ;
- `pendown()` baisse le crayon de sorte que le déplacement de la tortue trace un trait ;
- `pensize(n)` modifie l'épaisseur de la ligne que la tortue trace en se déplaçant (par défaut $n = 1$) ;
- `pencolor(c)` modifie la couleur de la ligne que la tortue trace ;
- `bgcolor(c)` modifie la couleur de la fenêtre où a lieu le tracé.

Pour les deux dernières instructions, la couleur peut être décrite de plusieurs façons, par exemple sous la forme d'une chaîne de caractères associée à une couleur prédéfinie (`'black'`, `'blue'`, `'red'`, `'brown'`, ...) ou encore sous la forme d'un triplet (r, g, b) où r, g et b sont des réels de l'intervalle $[0, 1]$ décrivant les composantes RGB d'une couleur.

Commandes diverses

- `position()` retourne les coordonnées (x, y) de la tortue ;
- `heading()` retourne l'angle de la tortue par rapport à l'horizontale ;
- `setposition(x, y)` déplace la tortue au point de coordonnées (x, y) ;
- `setheading(n)` dirige la tortue dans la direction correspondant à un angle de n degrés ;
- `distance(x, y)` retourne la distance entre la tortue et le point de coordonnées (x, y) [6] ;
- `towards(x, y)` retourne l'angle avec l'horizontale que fait la droite reliant la tortue au point de coordonnées (x, y) [6] ;
- `home()` déplace la tortue au point de coordonnées $(0, 0)$ en la dirigeant vers l'est ;
- `reset()` efface tous les tracés et replace la tortue à sa position initiale.

6. Le couple (x, y) peut être remplacé par une autre instance de la classe `Turtle` (voir l'exercice 11).