Introduction à la programmation (impérative) et dessin avec Turtle !

Thibault PENNING Denis LACROIX

Utilisation de Turtle pour apprendre Python sous un autre angle. Un grand merci à Denis LACROIX de m'avoir fait part de son TP.

Découvrir Turtle

Pour démarrer, ouvrer IDLE de Python et/ou créer un fichier `test.py` *(le nom import peu mais l'extension est importante)*.

Afin de se familiarisez un peu avec Turtle vous pouvez copier-coller le code suivant :

```
from turtle import *                                                    ⑥

reset()                                                                 ⑤
def dessine():                                                          ①
    forward(200)                                                        ②
    left(90)                                                            ③
    forward(50)

dessine()                                                               ④
```

① Ceci est une fonction. Le code de cette dernière sera exécuté quand nous l'appellerons. Le code qui appartient la fonction est défini par l'indentation (les petits espaces ou tabulation au début).
② On demande à la tortue d'avancer de 200 pas.
③ On demande à la tortue de tourner à gauche de 90° (attention dans le repère de la tortue…)
④ On appelle la fonction `dessine` afin que le code soit exécuté. Les parenthèses sont nécessaires.
⑤ Ceci permet d'effacer le dessin. Sans cela le dessin resterait.
⑥ Ceci permet d'importer tout ce qu'il y a dans Turtle. Sans cela le programme ne fonctionnerait pas. Tout programme voulant utiliser Turtle doit contenir cette ligne. Vous pouvez voir l'importation comme un copier-coller d'un autre code (le vrai mécanisme est plus subtile).

Exercice 1 : Familiariser vous avec Turtle

Avec ce code n'hésiter pas à passer du temps à changer les valeurs (300 au lieu de 200, ...), ajouter des lignes, copier/coller. Appeler plusieurs fois la fonction `dessine`, avec ou sans `reset` entre, ...

Observer ce qu'il se passe.

Vous pouvez aussi essayer ces commandes-ci :

```
# Se déplacer
backward(100)
right(30)

speed('normal')

# Intéragir avec le stylo (écrire ou non)
up()
down()

# Pour la feuille de dessin
clearscreen()
home()

# Pour la créativité
color('blue')
width(3)


begin_fill()
# Votre super code !
end_fill()
```

Plus de commande et de description sont disponibles sur la documentation officielle de Turtle.

> ! Important
>
> **Le but est ici de faire des expériences afin d'apprendre. Fait une hypothèse, essayer là en réalisant l'expérience, et tirer les conclusions. C'est cette démarche scientifique qui vous permettra de bien comprendre. N'y passez pas tout le TP, mais ce n'est pas grave de prendre du temps !**

Une courte introduction à Python

Valeurs

En Python, nous pouvons écrire des chiffres de la manière la plus simple possible, en l'écrivant simplement.

```
>>> 1000                                                                    ①
1000

>>> 1.2
1.2                                                                         ②

>>> from math import pi, cos                                                ③
>>> cos(2 * pi)
1.0
```

① Ici on écrit l'entier (dit aussi `int`) représentant 1000
② Ici le flottant (ou `float`) qui représente un nombre à virgule flottante (d'où son nom) 1,2 . La notation est la notation anglo-saxonne, où la virgule est remplacée par un point pour représenter la délimitation de la partie décimal.
③ De nombreuse fonction est constante sont déjà définies si besoin, vous pouvez regarder dans la librairie math dont on importe les éléments ainsi.

> **Note**
>
> On utilise le `+` pour l'addition de nombre, le `*` pour la multiplication, le `-` pour la négation ou la soustraction, le `/` pour la division et le `**` pour la puissance (2^3 s'écrit `2**3`). `//` représente la division euclidienne et `%` représente le reste de la division euclidienne (`19%4` donne `3`). Le reste des fonctions (`sqrt`, `exp`, …) se trouve dans math.

On peux afficher une valeur grâce à la fonction `print`. (Précédemment j'affichais directement la valeur, maintenant nous utiliserons `print`).

```
>>> print(10)
10
```

Pour les chaines de caractère (c'est à dire du texte) on utilise les " ainsi :

```
>>> print("Salut !")
Salut !
```

Variables

Il peut être utile de stocker des valeurs, pour des calculs, où pour stocker une valeur pour plus tard (par exemple afin de réaliser un compteur). On utilise la syntaxe `nom_variable = valeur` comme ceci :

```
>>> a = 1                                                    ①
>>> print(a)                                                 ②
1

>>> b = 2
>>> c = a + b                                                ③
>>> print(c)
3

>>> c = c + 3                                                ④
>>> print(c)
6
```

① Ici nous affectons à la variable `a` la valeur 1. Nous utilisons souvent l'analogie suivante : `a` peut être vu comme un tiroir que l'ont nome ainsi, où l'on peut y placer des choses, les modifier au besoin et y accéder plus tard. C'est cela une variable.

② Nous pouvons par la suite utiliser `a` comme si cette dernière était une valeur. La valeur de `a` sera la dernière valeur que `a` a stocké, ici `1`.

③ Bien évidement, on peut utiliser une variable pour assigner une valeur a une autre variable. Ici la valeur de `c` sera bien celle de `b` plus celle de `a`.

④ Nous pouvons réaffecter les valeurs dans une variable, ici `c` se voit ajouter 3 (on fait donc $3 + 3 = 6$).

Boucle `for`

Il est souvent utile de répéter du code. Pour cela nous pouvons utiliser une boucle `for`. La syntaxe est la suivante :

```
for nom in range(jusque_là):
    # code à exécuter plusieurs fois
```

Lors de l'exécution, la variable se voit modifier a chaque tour, le code a l'intérieur peut donc en profiter pour légèrement varier son comportement. Entre la parenthèse de range se trouve le nombre de fois que l'on répète. La variable modifiée se voit donc prendre la valeur entre 0 et `jusque là-1` (comme l'on commence à 0). Le nombre doit être placé entre `()`. Le `:` à la fin de la ligne est nécessaire et n'est pas cosmétique.

Le code à exécuter est définie par ce que l'on appelle l'indentation. L'indentation est l'espace laissé (ou une tabulation) laisser en début de ligne. Si chaque ligne a le même nombre d'espaces alors, ils appartiennent au même bloc et seront exécuter pareil. La boucle `for` s'attend à trouver du code avec un niveau d'indentation supérieur en dessous de sa ligne, et c'est ce code qui sera exécuter plusieurs fois.

```
for var in range(limite):
    # Code exécuté plusieurs fois car indenté
    # Ici aussi le code est indenté, il sera donc exécuté plusieurs fois, à la suite du préc
# Code non indenté, il ne sera exécuter 1 seule fois.
```

Essayer de comprendre cet exemple :

```
print("Démarrage")
for i in range(10):
    print(i)
print("Fin")
```

Cela donnera :

```
Démarrage
0
1
2
3
4
5
6
7
8
9
Fin
```

> **Note**
>
> Le nom de la variable n'est pas important et le choix est libre (il ne peut y avoir d'espace comme pour les autres variable). Faite attention cependant a ne pas utiliser un nom déjà pris.

Condition

Il peut être utile de définir un comportement baser sur une valeur, afin de différentier des cas. Pour cela on utilise un `if`.

La syntaxe est la suivante :

```
if condition:
    # Seulement si la condition est respectée cette ligne seras exécuté
# Cette ligne est exécuté quoi qu'il arrive
```

Les conditions possiblent sont les suivantes : `==` pour l'égalité (attention un simple `=` affecte, deux `==` compare), `!=` pour différent, `<`, `>`, `<=`, `>=`.

> **Note**
>
> Il est aussi possible d'utiliser `and` et `or` pour combiner plusieurs conditions en une.

On peut utiliser `else` juste après pour exécuter un code uniquement si la condition est fausse.

```
if cond:
    # Si cond vrai
else:
    # Si cond fausse
# Toujours exécuté
```

Exemple :

```
for i in range(5):
    if i <= 2:
        print("Le nombre", i, "est inférieur ou égale à deux")
    else:
        print("Le nombre", i, "est supérieur à deux")
```

Donne :

```
Le nombre 0 est inférieur ou égale à deux
Le nombre 1 est inférieur ou égale à deux
Le nombre 2 est inférieur ou égale à deux
Le nombre 3 est supérieur à deux
Le nombre 4 est supérieur à deux
```

Fonction

Il est possible de stocker du code sous un nom plus court afin de s'en resservir au besoin. On nomme cela une fonction. La syntaxe est la suivante :

```
def nom_fonction():
    # Code dans la fonction
```

On appelle le code à l'intérieur ainsi :

```
nom_fonction()
```

Example :

```
def salutation():
    print("Bonjour tout le monde !")

salutation()
salutation()
```

Donne :

```
Bonjour tout le monde !
Bonjour tout le monde !
```

Il est possible donner une entrée à cette fonction (que l'ont appel argument) afin de rendre chaque exécution unique. Cette entrée se comporte comme une variable dont la valeur change à chaque exécution.

la syntaxe est la suivante :

```
def f(a):
    # Code utilisant la variable a
```

Exemple :

```
def double(nombre):
    print(nombre*2)

double(3)
double(5)
```

Affichera :

```
6
10
```

Exercice 2 : Des dessins plus concrets

Après avoir joué avec Turtle commencez, vous pouvez essayer de dessiner des formes géométriques simples :

- ☐ Carré
- ☐ Triangle
- ☐ Hexagone
- ☐ Pentagone
- ☐ Étoile
- ☐ Sablier
- ☐ Maison
- ☐ Petite ourse (constellation)

(Il est inutile de réaliser toutes les formes. Je vous conseille de faire Carré, Triangle et Étoiles (utiles pour la suite) et une ou deux que vous apprécierez faire.)

> **Astuce**
>
> Essayer de créer une nouvelle fonction à chaque fois pour garder une trace de ce que vous avez fait. Ainsi il suffit d'appeler ou pas la fonction pour dessiner.
> Vous pouvez même utiliser des arguments pour gérer la taille.

```
def carre(taille):
    forward(taille)
    left(90)
    #...
```

Bonus

Essayer de dessiner toutes les formes que vous aviez envie les une à la suite des autres. Faites attention à ce qu'elles ne se touchent pas !

> **Indice**
>
> *Pensez aux fonctions pour interagir avec le stylo. Pour les plus aguerrit, afin de ne pas perdre la tortue dans les dessins, vous pouvez utiliser une variable qui enregistre son orientation par rapport au nord !*

Des frises géométriques

Exercice 3 : Des frises triangulaires

Frise triangulaire simple

Réaliser une frise comme celle-ci :

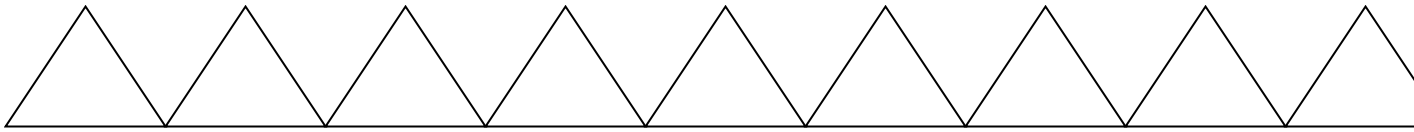

Figure 1: Frise simple à réaliser

Frise triangulaire alterné

Réaliser la frise suivante :

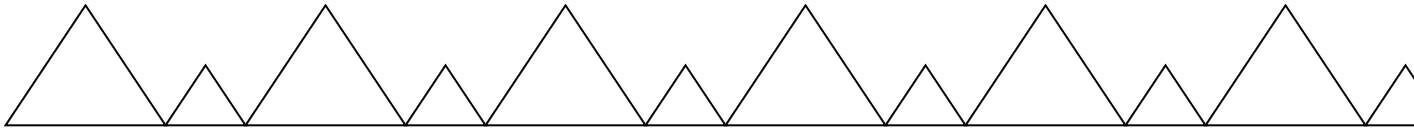

Figure 2: Frise triangulaire alternée

Frise triangulaire grandissante

Réaliser la frise (potentiellement infinie) suivante :

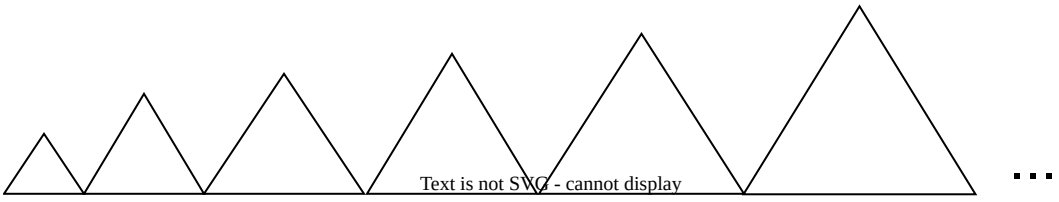

Figure 3: Frise grandissante triangulaire

Exercice 4 : Des frises carrées

Frise carrée simple

Réaliser la frise suivante :

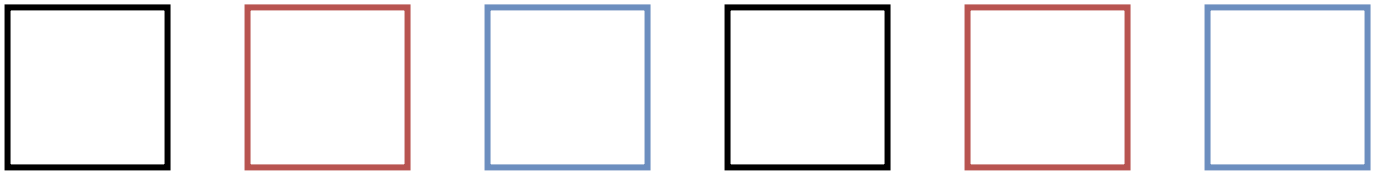

Figure 4: Frise carrée simple

Frise carrée alternante

Réaliser la frise suivante :

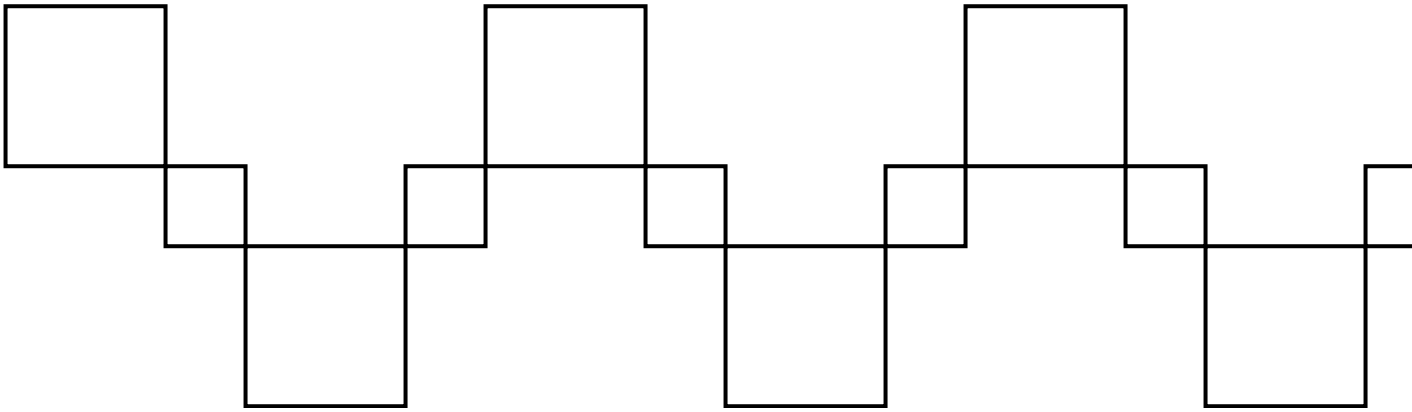

Figure 5: Frise carrée alternante

Exercice 5 : Frise d'étoile

Frise d'étoile simple

Réaliser la frise suivante :

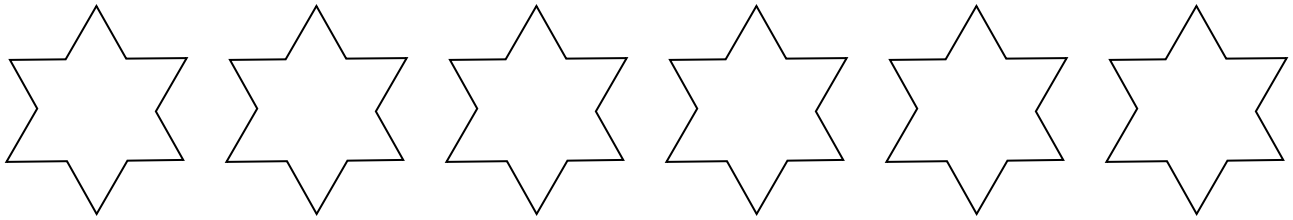

Figure 6: Frise d'étoile simple

Frise d'étoile alternante

Réaliser la frise suivante :

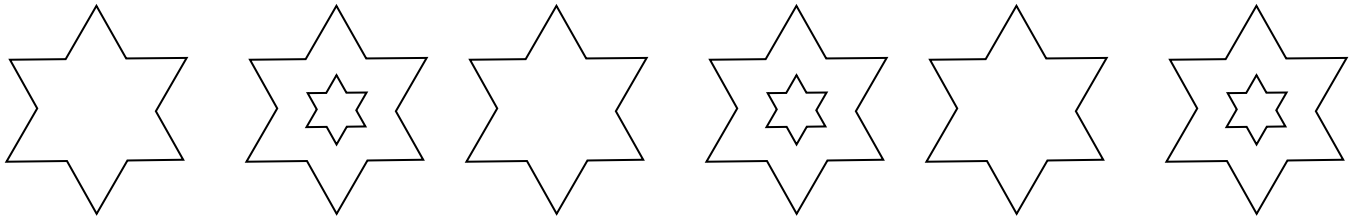

Figure 7: Frise d'étoile alternante