Atelier de programmation en python

Kévin “Chewie” Sztern et Christophe “Sagane” Vermorel

October 28, 2015

Contents

Introduction

L'objectif de ce Workshop est de vous montrer quelques bases de programmation dans un langage très simple pour débuter : le Python.
Vous êtes encadrés par plusieurs étudiants d'Épita, en première année du cycle d'ingénieur. N'hésitez pas à nous poser des questions !
Le but de ce Workshop est de vous faire dessiner des formes géométriques à l'aide du langage de programmation **python**[1] et de son module **turtle**[2], un ensemble d'outils destinés aux débutants souhaitant découvrir la programmation. Vous trouverez en annexe des exemples de ce que vous serez capable de réaliser avec ce module et les connaissances acquises au cours de ce Workshop.

L'environnement de développement

Dans un premier temps, nous verrons l'environnement qui sera utilisé pour réaliser ce Workshop.
Un environnement de développement est généralement composé de trois outils :

- Un éditeur : l'espace éditable où vous taperez votre code;
- Un compilateur : un logiciel qui transformera le code que vous aurez tapé de telle sortes à ce qu'il soit réellement compréhensible par l'ordinateur. Pour certains langages, ce compilateur est remplacé par un interpréteur (la différence n'a pas d'importance ici).
- Un debugger : un outil nous permettant d'analyser notre programme afin de corriger les erreurs qui s'y sont glissées.

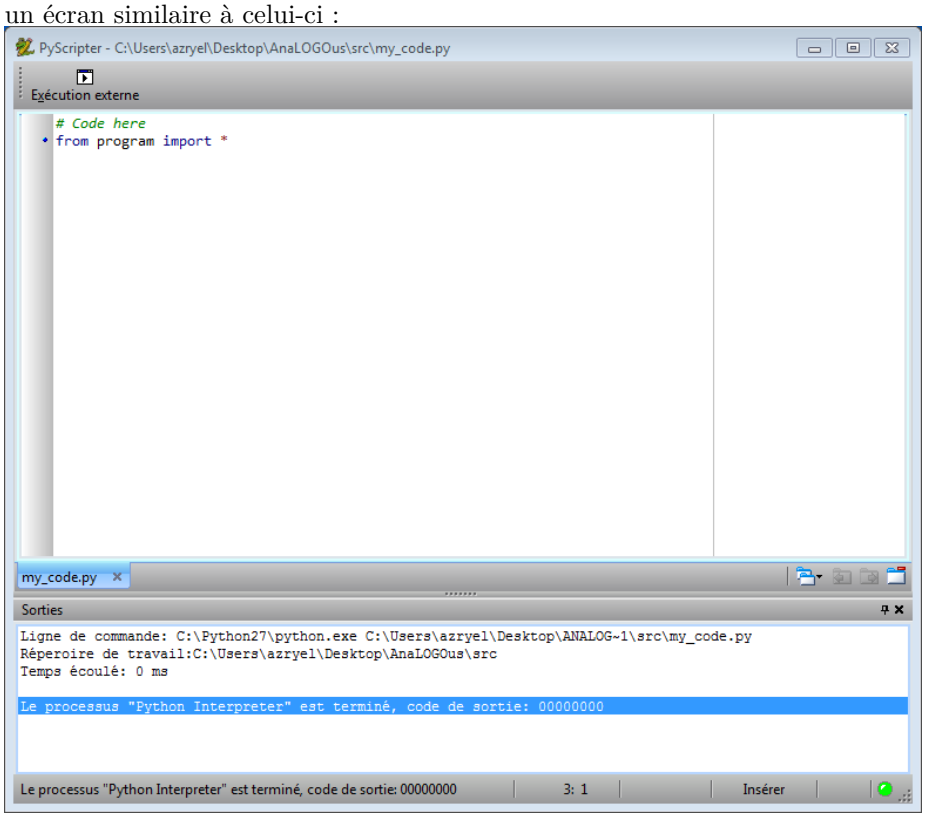
Vous avez de la chance, il existe des logiciel qui combinent *éditeur* et *compilateur/interpréteur*. On les appelle : IDE (Integrated Development Environnement - Environnement de développement intégré).
Nous utiliserons un IDE dédié à la programmation en Python : PyScripter

Démarrer l'IDE

PyScripter se lance très simplement en double cliquant sur l'icône associée se trouvant quelque part sur votre bureau. Si tout se passe bien, vous devriez voir

[1] http://python.org/
[2] http://docs.python.org/2/library/turtle.html

un écran similaire à celui-ci :



Vous l'aurez compris, l'IDE n'affiche que le strict minimum nécessaire pour ce Workshop. Notez que deux lignes de code sont déjà présentes :

- **# insérez votre code ici** est un commentaire, il sera ignoré par l'interpréteur, comme toutes les lignes commençant par #.
- **import turle** permet d'importer l'ensemble des fonctionnalités présentes dqns la bubliothèque turtle.

Au niveau de l'interface de l'IDE, vous utiliserez principalement (de haut en bas) :

- le bouton d'exécution : presser ce bouton exécutera aussi votre code;
- l'espace d'édition;
- une console (onglet *Sortie* en bas d'écran) : c'est ici que s'afficheront les messages d'erreur si votre code n'est pas correct.

Si vous essayez d'exécuter le programme (en appuyant sur le bouton), vous remarquerez qu'une fenêtre s'ouvre, et se ferme juste après. Votre objectif sera de dessiner dans cette fenêtre !

Premiers pas

Pour l'instant, voyons ce qu'il est possible de faire avec quelques lignes de code. Pour bien comprendre ce qu'il se passe : recopiez dans votre fichier ce qui est donné ici, changez les nombres, et voyez quelle œuvre d'art vous êtes en train de créer !

Quelques primitives pour débuter

Une première forme:

```
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
```

Une autre:

```
turtle.forward(100)
turtle.left(120)
turtle.forward(100)
turtle.left(120)
turtle.forward(100)
```

Éviter d'écrire les mêmes instructions : les procédures

Souvent, une forme géométrique est composée de plusieurs formes simples à divers endroits. En fait, nous avons la possibilité d'indiquer à l'ordinateur la procédure à suivre pour dessiner une forme bien précise. Cette procédure pourra ensuite être mise en œuvre afin de dessiner la forme à plusieurs endroits.
En programmation, nous parlons de *procédure* ou *fonction*.
En réalité, vous aviez déjà rencontré cette bestiole : ce sont toutes les lignes que vous tapez et finissez par des parenthèses. En effet, `turtle.forward()`, `turtle.left()`, etc. sont toutes des procédures cachant un code compliqué que

vous appelez plusieurs fois.

Le schéma général de déclaration de procédure est le suivant :

```
def nom_de_ma_super_procedure() :    # on nomme la procedure
    instruction 1                    # tabulation obligatoire
    instruction 2
```

Prenez garde à la tabulation avant chaque instruction : cela permet au compilateur/interpréteur de savoir que les instructions appartiennent bien à la fonction[3].

Une fois définie, vous appelez votre procédure tout simplement comme suit :

```
nom_de_ma_super_procedure()
```

Appeler une procédure revient à exécuter le code qu'elle contient.
Voici quelques procédures que vous pouvez tester pour mieux comprendre :

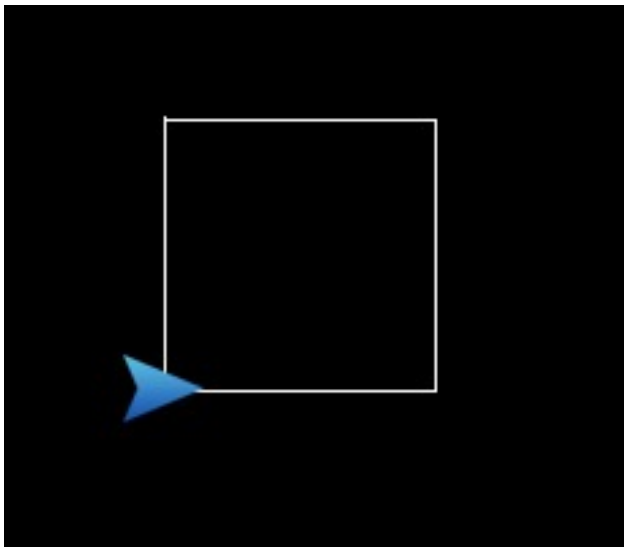


Figure 1: carre()

[3]Certains langages ont une approche différente : le C, C++, Java, et C# par exemple utilisent des accolades pour délimiter le corps des fonctions. Les langages tels que Python ou encore le Haskell délimitent les corps de fonctions par rapport à l'indentation, ce qui rend le code *naturellement* lisible

```
def carre():
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
    turtle.left(90)
    turtle.forward(100)
```

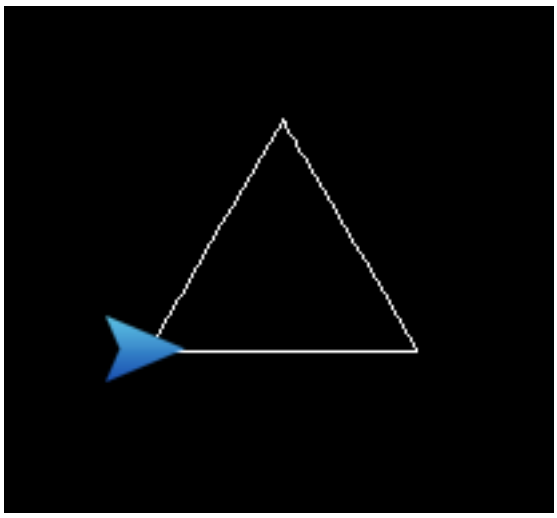


Figure 2: triangle()

```
def triangle():
    turtle.forward(100)
    turtle.left(120)
    turtle.forward(100)
    turtle.left(120)
    turtle.forward(100)
```

Créez vos propre procédures !

Il est maintenant temps de créer vos propres procédure. Créez des procédure pour dessiner :

- une étoile;
- un losange;

- l'approximation d'un cercle (un cercle peut être approximé par une série de segments parcourant son bord). Vous pourrez utiliser les procédures `math.cos(angle)` et `math.sin(angle)`.

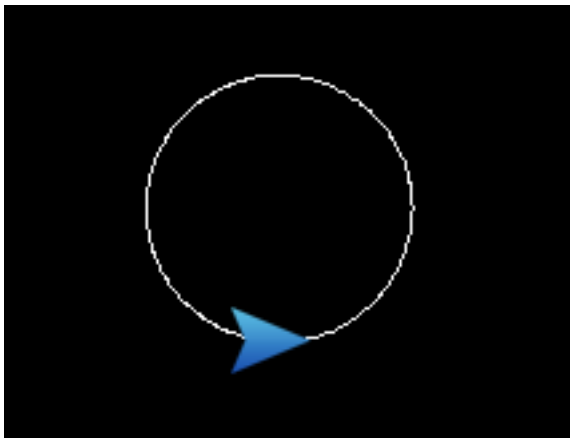


Figure 3: image

Répéter plusieurs fois les mêmes opérations : les boucles

En général, on parviendra à dessiner une forme complexe avec un certain nombre de formes simples.
En fait, jusqu'à présent, vous arrivez déjà à dessiner vos formes à partir d'une autre plus simple : la ligne.

Si vous observez bien, pour dessiner un carré, vous exécutez quatre fois les mêmes instructions :

```
# on dessine un bord
turtle.forward(100)
# on tourne de 90° : la direction du second bord
turtle.right(90)
# et on recommence
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
# les quatre bords apparaissent
```

Pour faire plus court, il serait intéressant de pouvoir dire à l'ordinateur :

```
faire 4 fois :
    avancer de 100 points
    tourner de 90°
```

Il s'avère que cela est possible en Python. C'est un concept que l'on appelle **boucle**.
Dans le cas du carré, on obtient en Python[4]

```
for i in range(4):      # faire 4 fois :
    turtle.forward(100)     # avancer de 100 points
    turtle.right(90)        # tourner de 90°
```

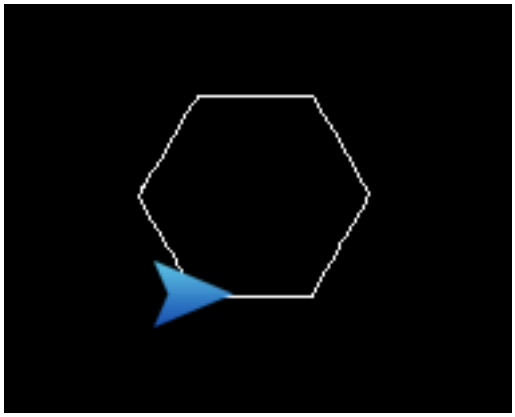
Maintenant réécrivez toutes les formes que vous avez déjà programmées, mais en utilisant les boucles :

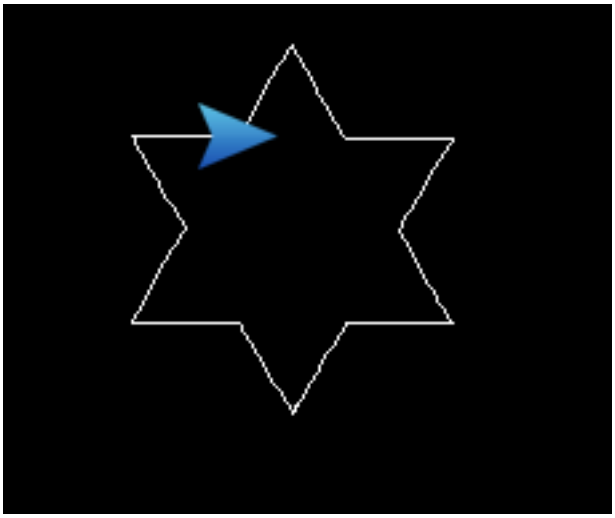
- Le triangle : en 3 lignes de code;
- Le losange : en 3 ligne de codes;
- L'approximation de cercle : en 3 ligne de codes.

[4]Remarquez l'usage de la procédure *range(nombre)*. Elle génèrera une suite de nombres allant de 0 à *nombre* - 1.

Quelques objectifs simples

Cette fois ci, vous n'êtes plus guidés ! Montrez que vous êtes créatifs en réussissant à dessiner les formes suivantes avec le moins de ligne de code possible. N'hésitez pas à appeler à l'aide !





Quelques objectifs impossibles

Les dessins suivants sont bien plus difficiles à réaliser que les autres ! Certains nécessitent même des notions bien plus complexes que celles vues dans ce TP : saut conditionnel, récursion, passage de paramètre aux procédures, etc.

Nous mettons à votre disposition d'autres procédures pour permettre à votre créativité de mieux s'exprimer :

- `turtle.speed(vitesse)` : change la vitesse de tracé du dessin, pour aller le plus vite possible, choisissez "0".
- `turtle.penup()` : les prochains déplacement n'afficheront rien;
- `turtle.pendown()` : a l'effet inverse du lever de crayon;
- `turtle.pensize(taille)` : change la taille des prochains traits dessinés;
- `turtle.pencolor(rouge, vert, bleu)` : change la couleur des prochains traits dessinés. Notez que les couleurs ont trois composantes : vert, rouge et bleu. Chacune peut avoir une valeur allant de 0 (absence de la composante) à 1. Par exemple, du rouge pur correspond à la combinaisons $\{\text{rouge} = 1, \text{vert} = 0, \text{bleu} = 0\}$ et le gris à $\{\text{rouge} = 0.5, \text{vert} = 0.5, = \text{bleu} = 0.5\}$.

Essayez d'obtenir le bon résultat, et appelez un assistant dès que vous bloquez !