Nom :……...

Le code de Sierpiński

En 1915, pendant la première guerre mondiale, le cryptographe polonais Waclaw Sierpiński a inventé des règles de codage cachées dans ces dessins :

Fig. 3.

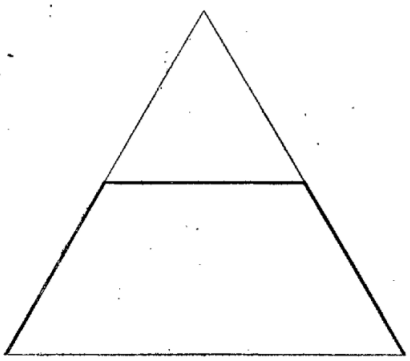


Fig. 4.

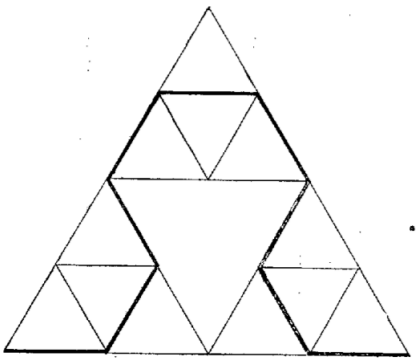


Fig. 5.

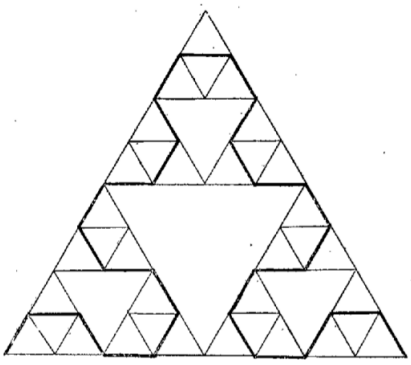
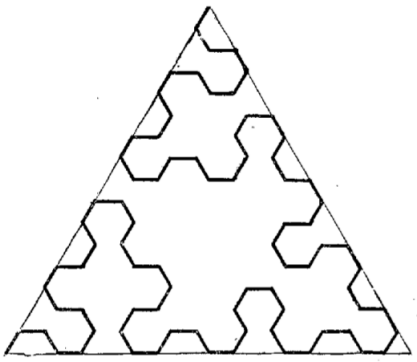


Fig. 6.



La comparaison des figures 3, 4, 5 et 6 de Sierpiński suggère qu'il y a un procédé (algorithmique) pour construire chaque étape à partir de la précédente. Par exemple entre les figures 3 et 4 on voit que des motifs (rectilignes sur la figure 3 et polygonaux sur la figure 4) se correspondent :

Fig. 3.

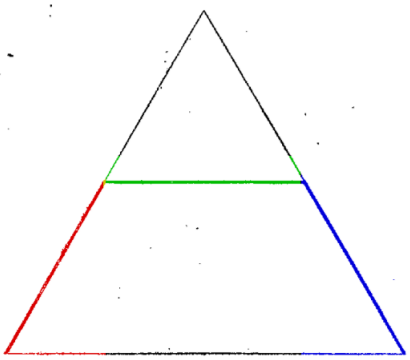
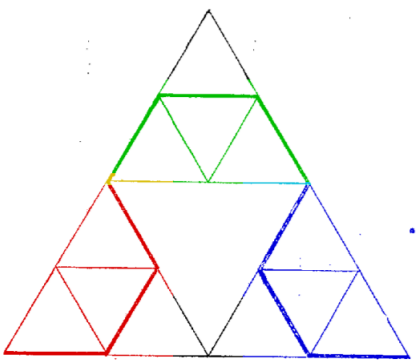


Fig. 4.

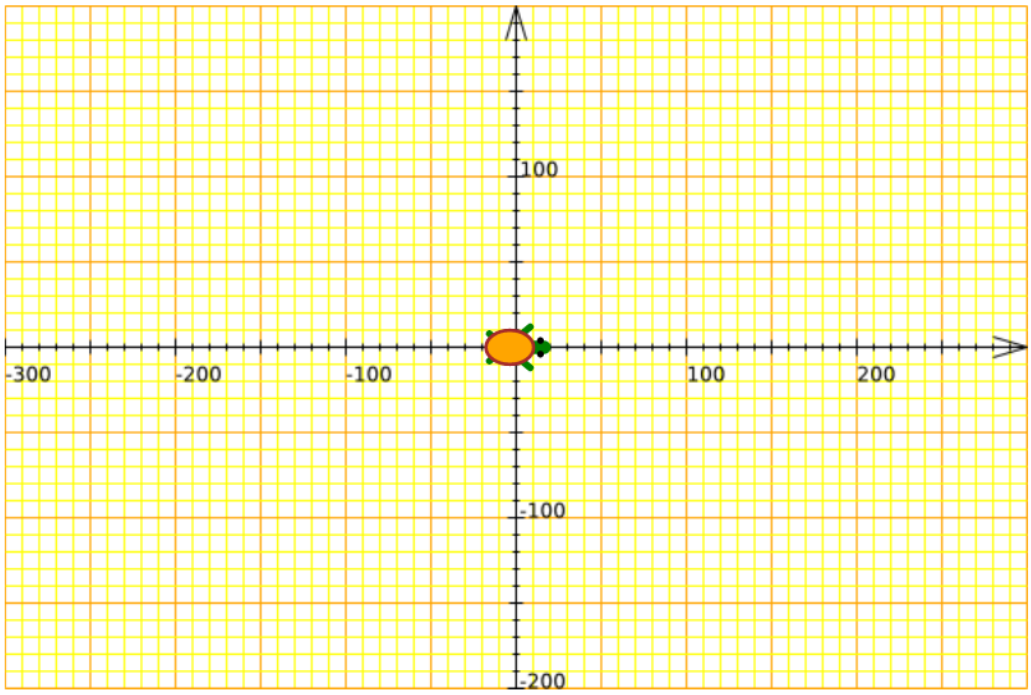


Le but de ce TP est d'essayer de deviner le codage de Sierpiński en s'aidant d'un robot appelé « tortue » (programmable en Python avec `from turtle import *`) et qui peut dessiner des

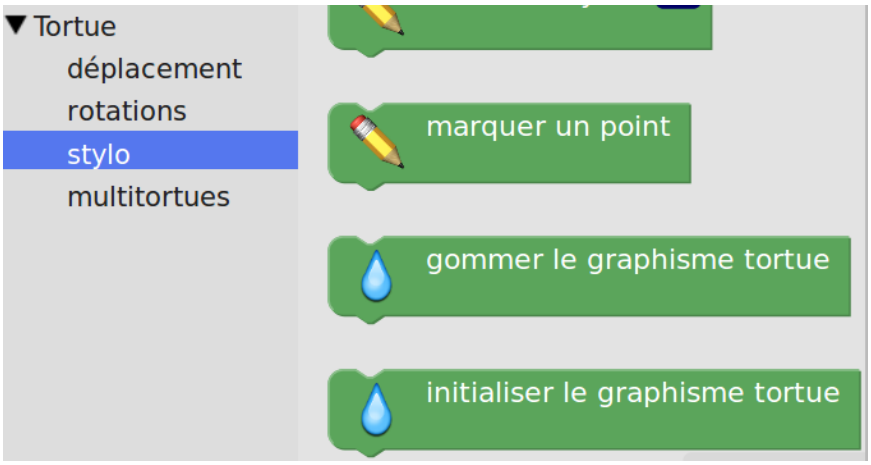
segments sur l'écran. On propose la version en ligne, accessible ici : https://alainbusser.github.io/SofusPy974/

ou là : http://zotweb.re/irem/SofusPy974/

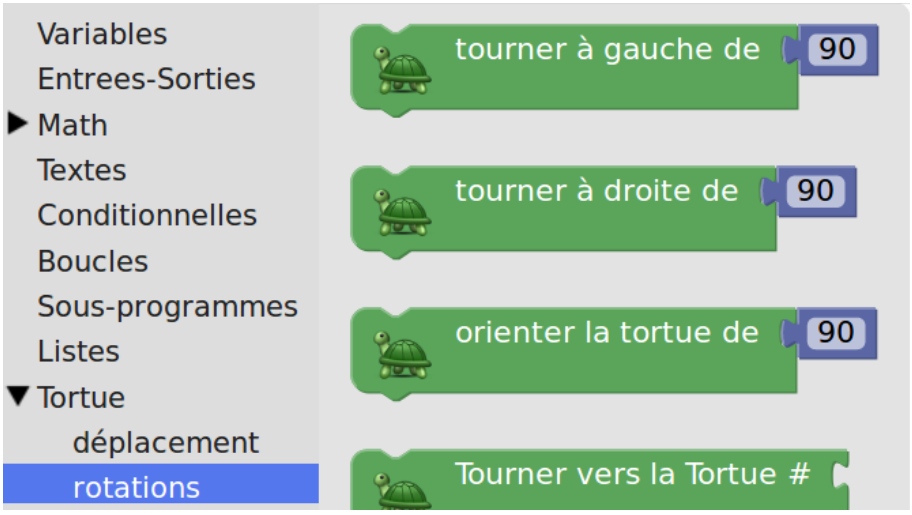
La tortue est visible en bas de la page web, placée au départ à l'origine d'un repère orthonormal, sur l'axe des abscisses et dirigée vers les abscisses positives :



Au-dessus de ce terrain de jeu du robot, se trouve l'espace de programmation. Il est en blanc, mais sa marge de gauche (grise) comprend des outils de programmation. Ceux concernant les déplacements du robot sont dans la catégorie « tortue » et parmi eux, dans la sous-catégorie « stylo », l'outil permettant de rétablir la situation initiale :

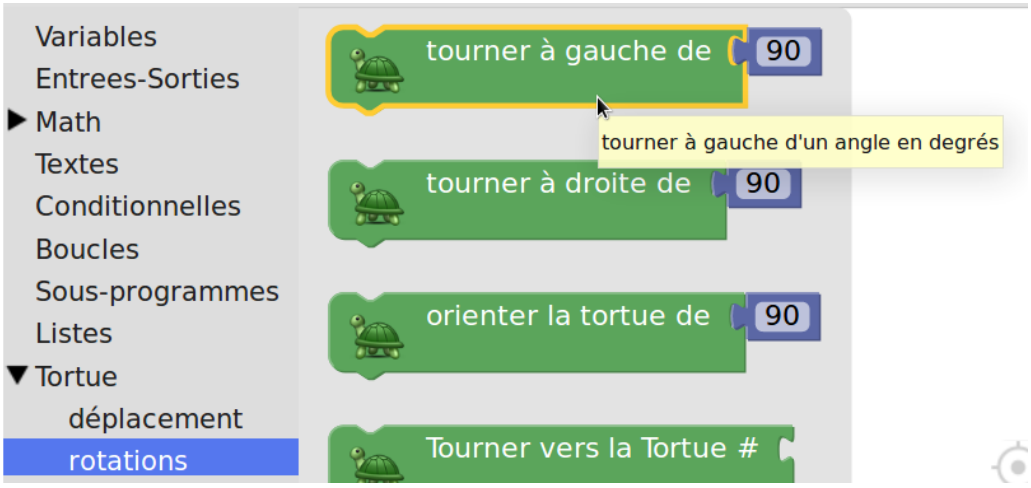


Deux autres sous-catégories seront intéressantes pour que le robot effectue une des trajectoires de Sierpiński : celle des rotations (tourner à gauche ou à droite)

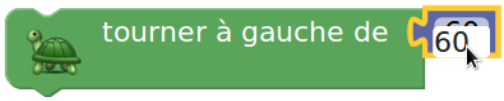


et celle des déplacements (faire avancer le robot) :

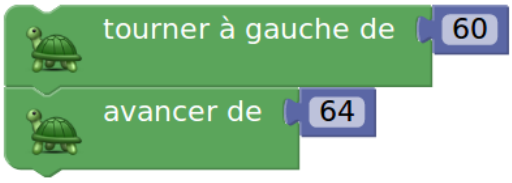
Par exemple, pour que la tortue trace un trait oblique, elle doit tourner puis avancer tout droit. Pour programmer la rotation (de 60° : on constate que dans les dessins de Sierpiński, tous les angles sont multiples d'un sixième de tour), on navigue dans la boîte à outils de SofusPy974, en allant dans la catégorie « tortue », sous-catégorie « rotation » et on fait glisser sur l'espace de travail le bloc « tourner à gauche » :



Une fois placé sur le plan de travail, il ne reste plus qu'à remplacer l'angle de 90° (prédéfini) par un angle de 60°. Ceci se fait au clavier :

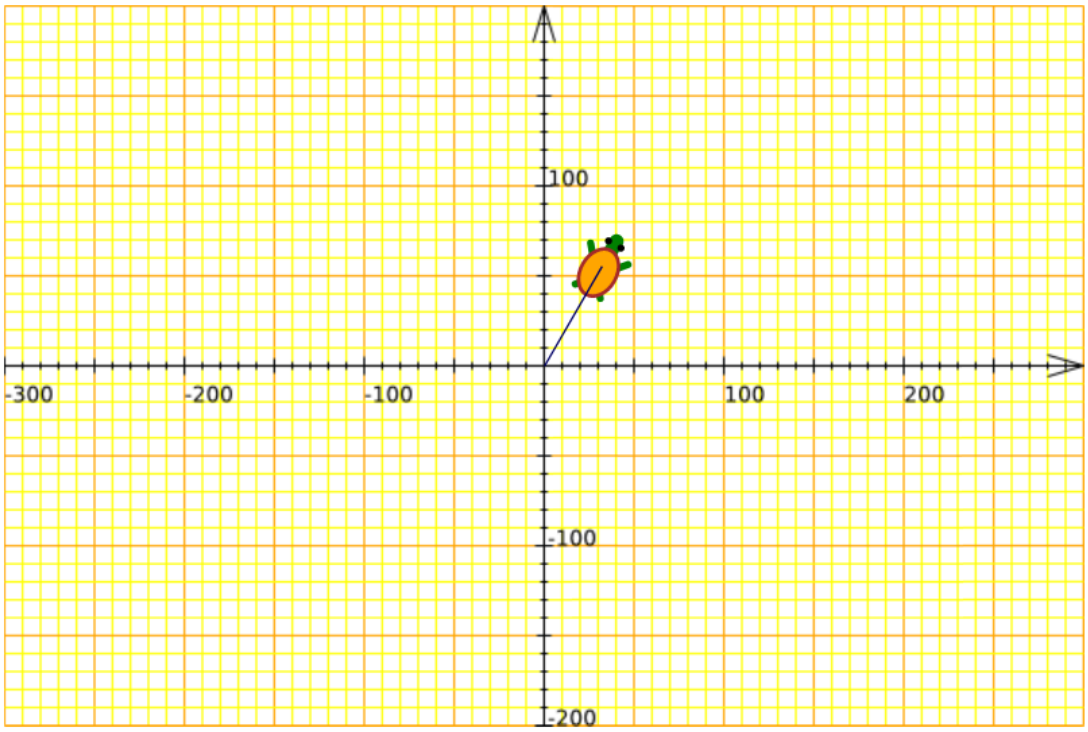


Puis, de même, on va dans la sous-catégorie « déplacements » pour saisir le bloc « avancer » et remplacer la longueur prédéfinie par 64 pixels. Pour écrire un programme de déplacement du robot, on colle le bloc de déplacement juste en-dessous de celui de rotation :

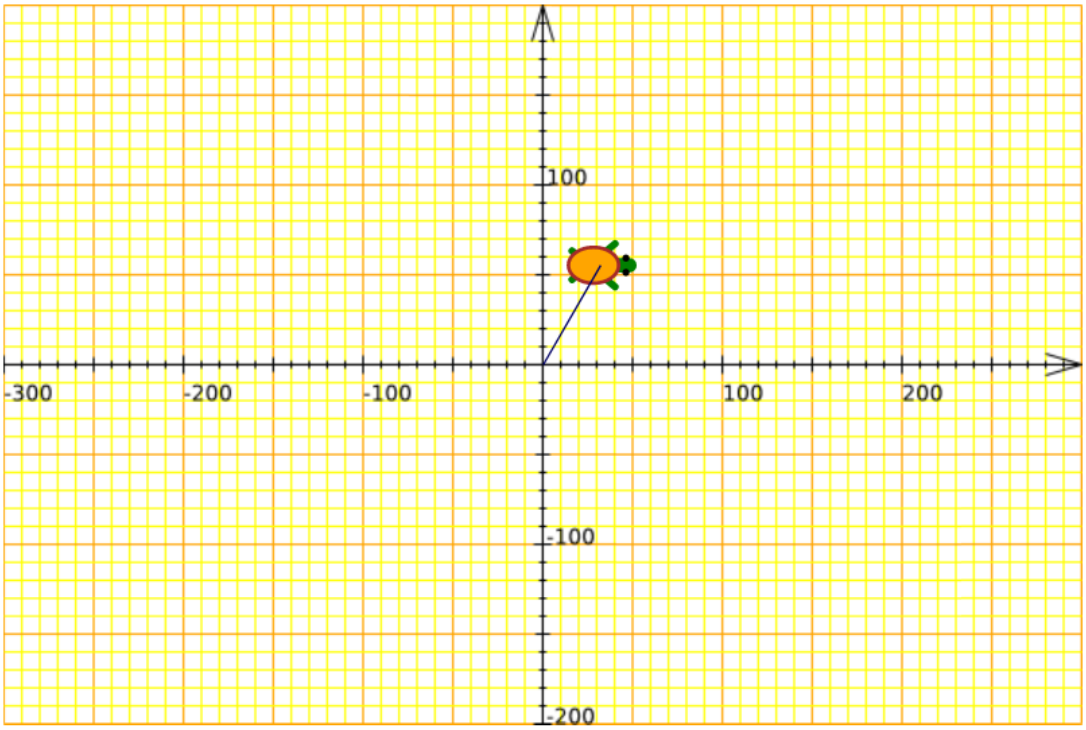


Pour que le robot exécute réellement les instructions qu'on lui donne, il ne reste plus, juste en-dessous de l'espace de travail, qu'à cliquer sur le bouton

Une fois que c'est fait, on constate que le robot a effectivement bougé, laissant une trace de son passage dans le repère :



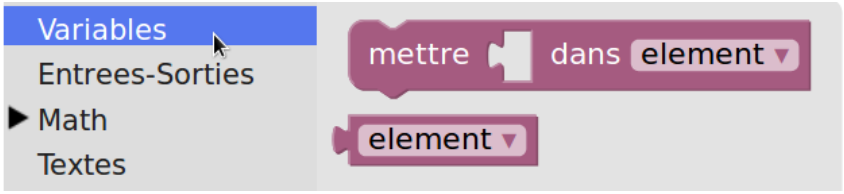
Exercice : Quelle instruction doit-on ajouter au programme précédent `gauche(60);avancer(64)` pour que la tortue soit prête à avancer ensuite parallèlement à l'axe des abscisses, comme ci-dessous ?



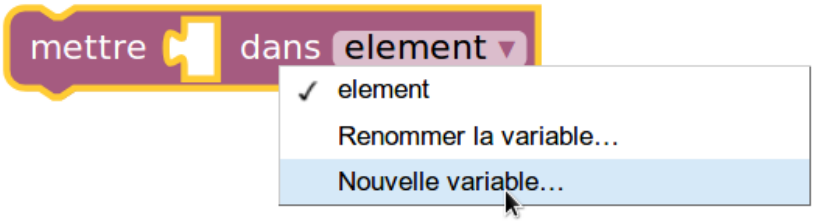
Réponse :…….

Notion de variable

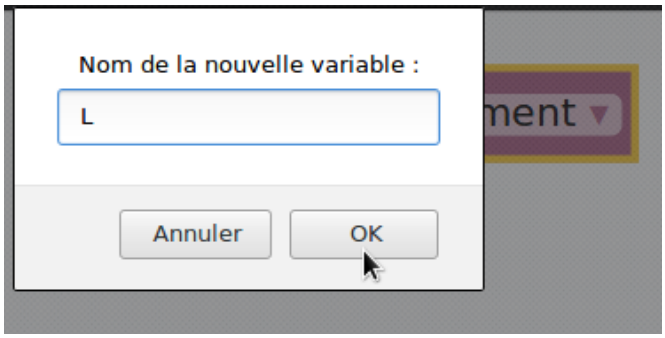
Le polygone de la figure 6 de Sierpiński ayant 81 côtés de longueur 16 pixels chacun, il peut s'avérer vite fastidieux de modifier les 80 pixels du bloc « avancer » pour les remplacer par 16. On propose alors de renommer le nombre 16 par le nom L qui en fait une variable (pour la figure 3, 128 pixels sont plus adaptés que 16 pixels, pour la figure 4 c'est plutôt 64 pixels, pour la figure 5 c'est 32 pixels – à écrire 27 fois). Dans SofusPy974 les variables sont groupées dans une catégorie appelée « variables » :



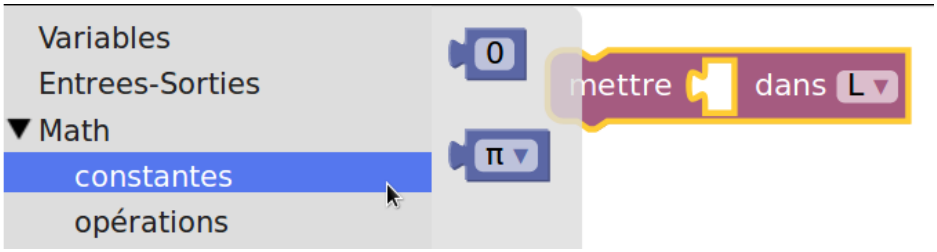
Pour créer (et initialiser) une variable, il faut mettre quelque chose (une valeur initiale) dedans, mais également nommer la variable. Par défaut la variable est nommée « element » mais on peut la renommer en cliquant sur le petit triangle à droite, et en choisissant l'option « nouvelle variable » :



C'est alors au clavier qu'on entre le nom de cette nouvelle variable (par exemple L) :

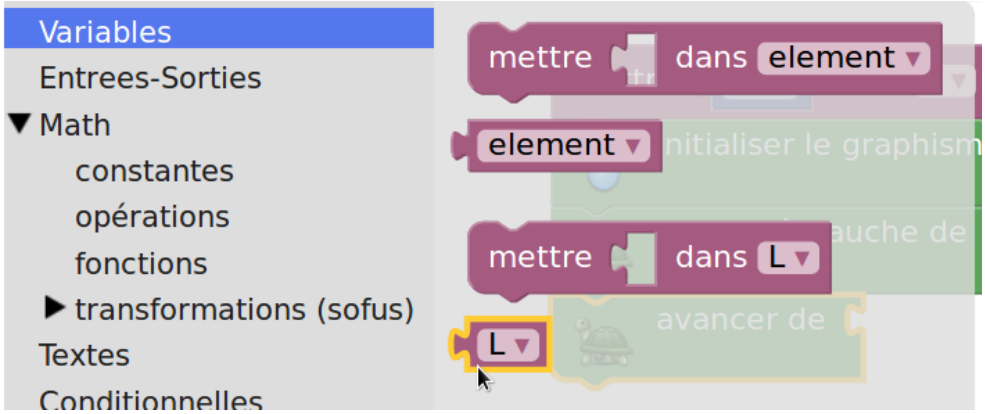


La valeur actuelle d'une variable est un nombre (une constante). On le trouve donc dans la catégorie « mathématiques », sous-catégorie « constantes ».

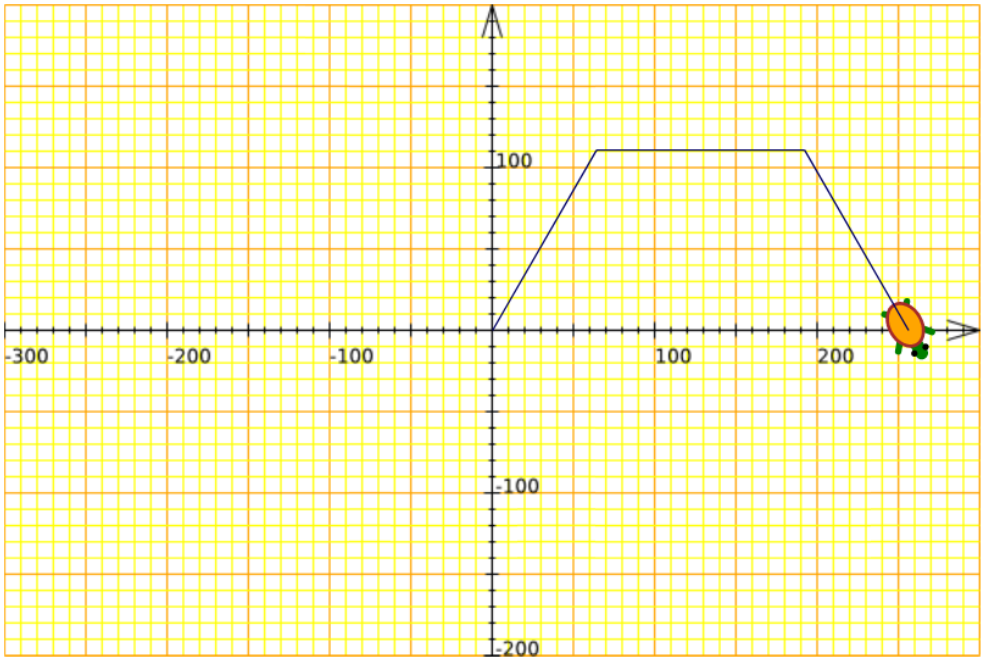


En choisissant la constante 0, on peut remplacer, au clavier, le nombre 0 par tout autre nombre choisi, par exemple 128 :

À partir de ce moment (et tant qu'on ne modifie pas la variable L), le mot « L » est un synonyme de 128. On récupère la valeur de la variable L, dans le menu des variables :



Si on remplace dans le programme précédent, tous les 64 par des L, on obtient bien un dessin deux fois plus grand :



Exercice : Reproduire à l'aide de la tortue, la figure 4 de Sierpiński (on prendra L=64).

Langage Python

Une fois que le programme semble correct, il est possible d'en obtenir la traduction Python, en cliquant sur le bouton « éditeur » :

Cela ouvre un éditeur (logiciel permettant d'écrire du Python), dans lequel se trouve le script Python donnant les commandes au robot.

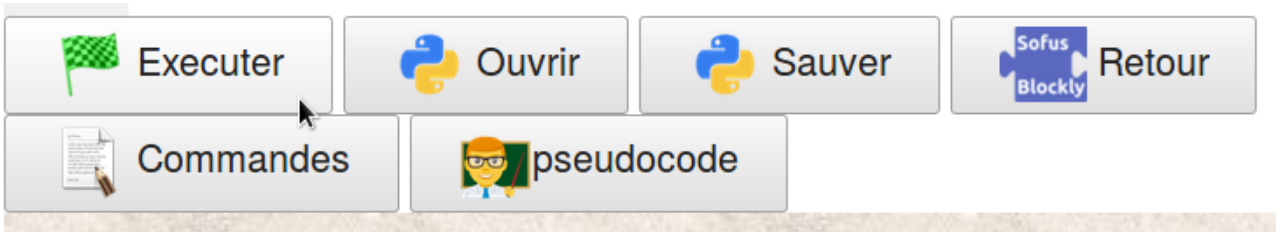
```
 1  from turtle import *
 2
 3  L = 128
 4  reset()
 5  left(60)
 6  forward(L)
 7
 8  right(60)
 9  forward(L)
10  right(60)
11  forward(L)
```

En Python, le groupe nominal représenté par un astérisque, désigne tout ce qui est dans le module concerné (ici, « turtle »). La première ligne a donc pour effet d'importer, depuis le module **`turtle`**, tout ce qu'il contient. En particulier les fonctions

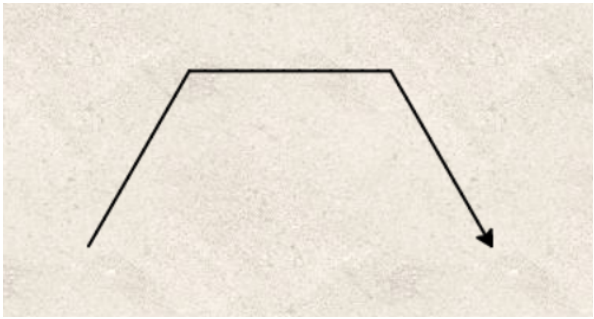
- **`reset()`** qui réinitialise le graphisme tortue
- **`forward(distance)`** qui fait avancer la tortue de **`distance`** pixels
- **`left(angle)`** qui fait tourner la tortue vers la gauche d'un **`angle`** en degrés
- **`right(angle)`** qui fait tourner la tortue vers la droite d'un **`angle`** en degrés

Avec l'habitude, il devient plus rapide d'écrire directement en Python, que de passer par Sofus.

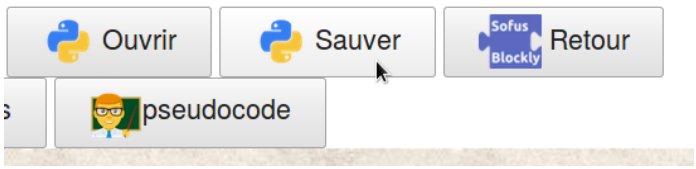
Pour que la tortue de Python fasse le parcours, il faut là aussi cliquer sur le bouton « exécuter » :



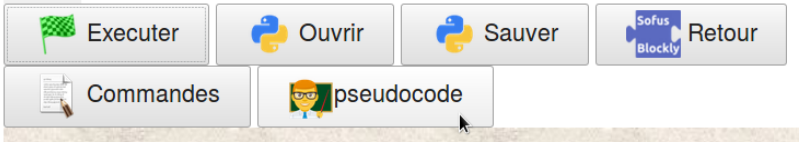
Mais la tortue de Python n'a pas le même aspect que celle de Sofus, et elle évolue dans un bac à sable qui la ralentit un peu :



Il est possible d'enregistrer le script Python dans un fichier dont le nom finit par « .py » (par exemple **`NomPrénom1.py`**) et ensuite de l'exécuter depuis un autre éditeur Python :



Mais attention : SofusPy974 utilise la version 2 de Python. Les calculatrices par exemple utilisent la version 3. Cela ne devrait pas poser de problème si on évite d'utiliser les lettres accentuées pour les noms de variables par exemple. Le bouton que voici :



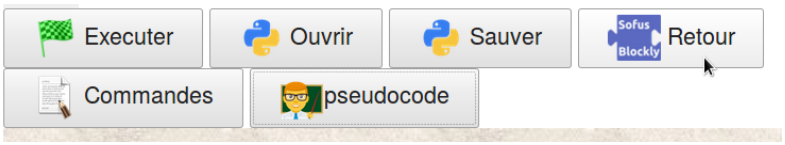
fait inscrire par SofusPy974 la description du programme Python par un « pseudo-code » qui décrit l'algorithme en langage naturel proche du français) :

```
L ← 128
initialiser le graphisme tortue
tourner vers la gauche de(60)
avancer de(L)
```

```
1  from turtle import *
2
3  L = 128
4  reset()
5  left(60)
6  forward(L)
7
```

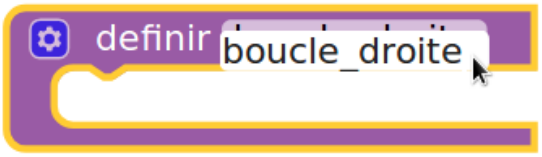
Notion de fonction

Depuis l'éditeur Python, on peut revenir à celui de Sofus (mais ça effacera les modifications effectuées dans l'éditeur Python), en cliquant sur le bouton qui a remplacé « éditeur » :



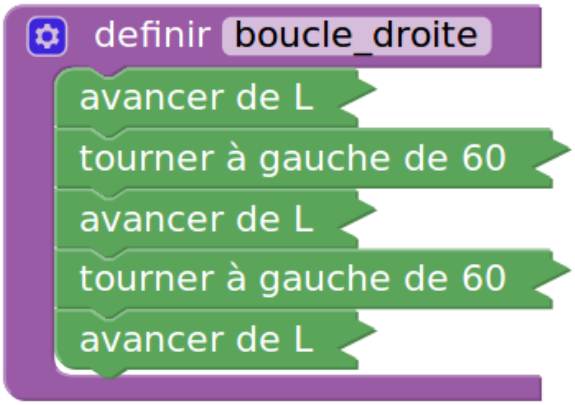
Un autre raccourci possible est de remplacer des choses répétitives (comme les trois traits du côté droit de la tortue) par un bloc unique regroupant ces commandes. Ce bloc est dans la catégorie « sous-programmes » :

En attrapant le premier bloc de cette catégorie, on peut cliquer sur le nom « faire quelque chose » pour le remplacer par un autre nom, au clavier :

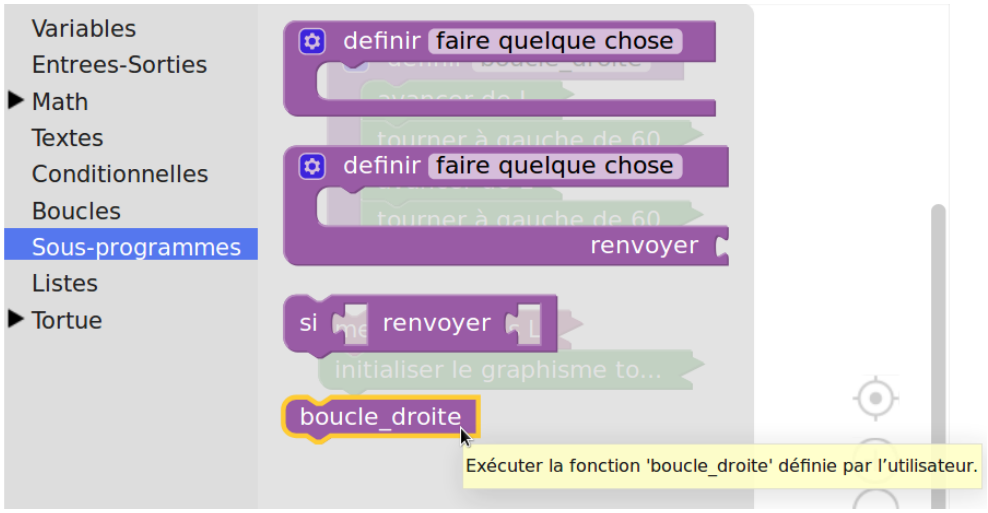


Pour éviter les problèmes de nom, on choisit de mettre un espace souligné (« underscore » en anglais, ou « tiret du 8 ») et donc, au lieu de « boucle droite », on préférera écrire « boucle_droite ».

Pour définir ce qu'on entend par « boucle_droite », on déplace des blocs comme précédemment.Par exemple pour une boucle à droite, on voit sur les figures de Sierpiński que le robot doit avancer puis tourner à gauche puis avancer encore et tourner à gauche et enfin avancer une dernière fois :

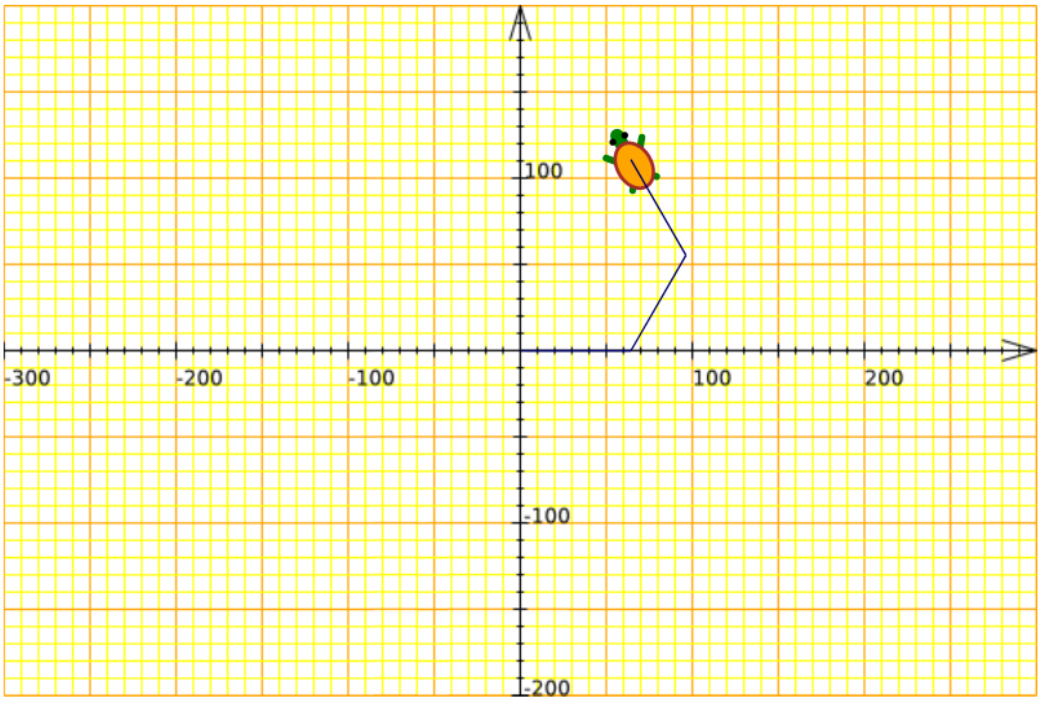


Une fois que la fonction est définie, elle apparaît dans la catégorie des « sous-programmes » :



Il suffit alors d'attraper ce bloc pour le glisser dans l'espace de travail, chaque fois qu'on veut que le robot effectue une boucle à droite.

Côté Sofus on dit qu'on a inventé un nouveau bloc (ou une nouvelle instruction) pour faire dessiner une boucle à droite :



Côté Python on dit qu'on a ***défini une fonction***. La syntaxe Python le rappelle, les trois lettres **`def`** étant précisément le début du mot « définir » :

```
1  from turtle import *
2  def boucle_droite():
3      global L
4      forward(L)
5      left(60)
6      forward(L)
7      left(60)
8      forward(L)
9
10
11 L = 64
12 reset()
13 boucle_droite()
14
```

La boucle mauve de Sofus reste à l'état de trace en Python, les lignes 3 à 8 étant décalées vers la droite (ou « indentées »). Et la ligne 13 montre que pour utiliser une fonction en Python, on donne juste le nom de cette fonction, suivi de parenthèses (ici, vides). Ainsi, **`reset`**, **`forward`** et **`left`** sont aussi des fonctions.

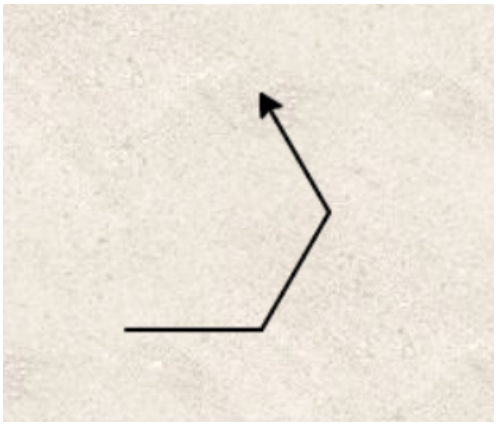
L'éditeur Python permet de cacher ce qui est à l'intérieur de le définition de la fonction, pour améliorer la lisibilité du script. Pour cela, il suffit de cliquer sur le petit triangle vers le bas qui est à côté du numéro de ligne :

```
1  from turtle import *
2  def boucle_droite():
3      global L
4      forward(L)
5      left(60)
6      forward(L)
7      left(60)
8      forward(L)
9
10
11 L = 64
12 reset()
13 boucle_droite()
```

Après le clic, le script a l'air raccourci :

```
1  from turtle import *
2  def boucle_droite():
9
10
11 L = 64
12 reset()
13 boucle_droite()
```

En cliquant sur « exécuter » on voit que le robot fait bien une boucle sur sa droite :



En cliquant sur « pseudocode » on constate que la définition est décrite par le mot « algorithme » :

```
algorithme boucle_droite()
    avancer de(L)
    tourner vers la gauche de(60)
    avancer de(L)
```

Mais le nom de l'algorithme reste celui d'une fonction :

```
L ← 64
initialiser le graphisme tortue
boucle_droite()
```

Ceci dit, on peut copier-coller le « pseudo-code » vers un logiciel de traitement de texte :

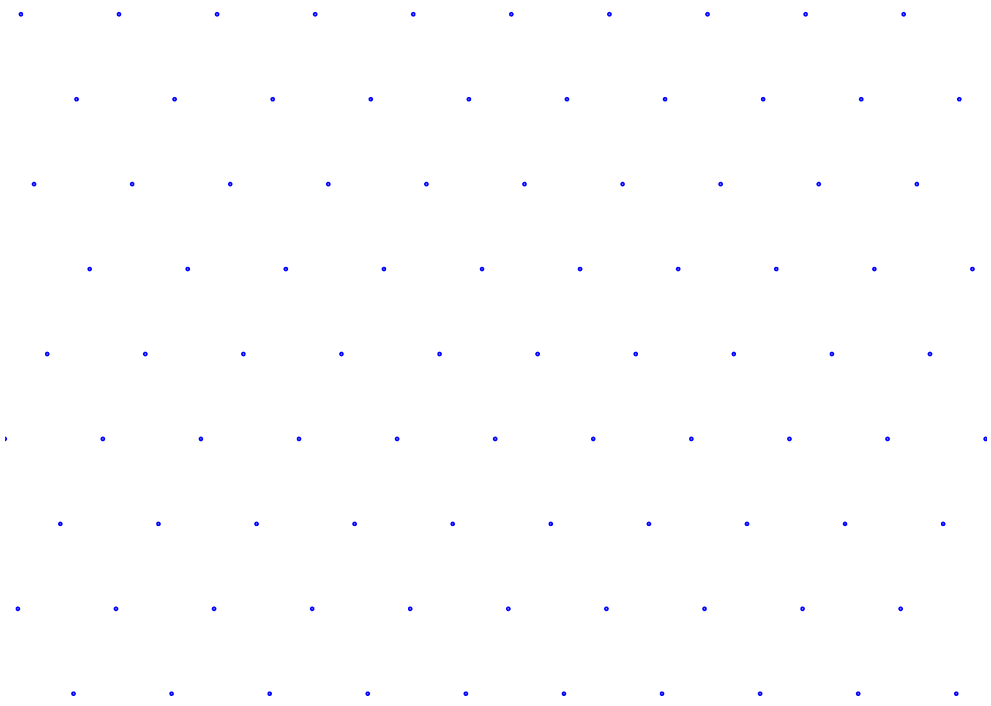
```
algorithme boucle_droite()
    avancer de(L)
    tourner vers la gauche de(60)
    avancer de(L)
    tourner vers la gauche de(60)
    avancer de(L)


L ← 64
initialiser le graphisme tortue
boucle_droite()
```

Puis le modifier pour le rendre plus intelligible :

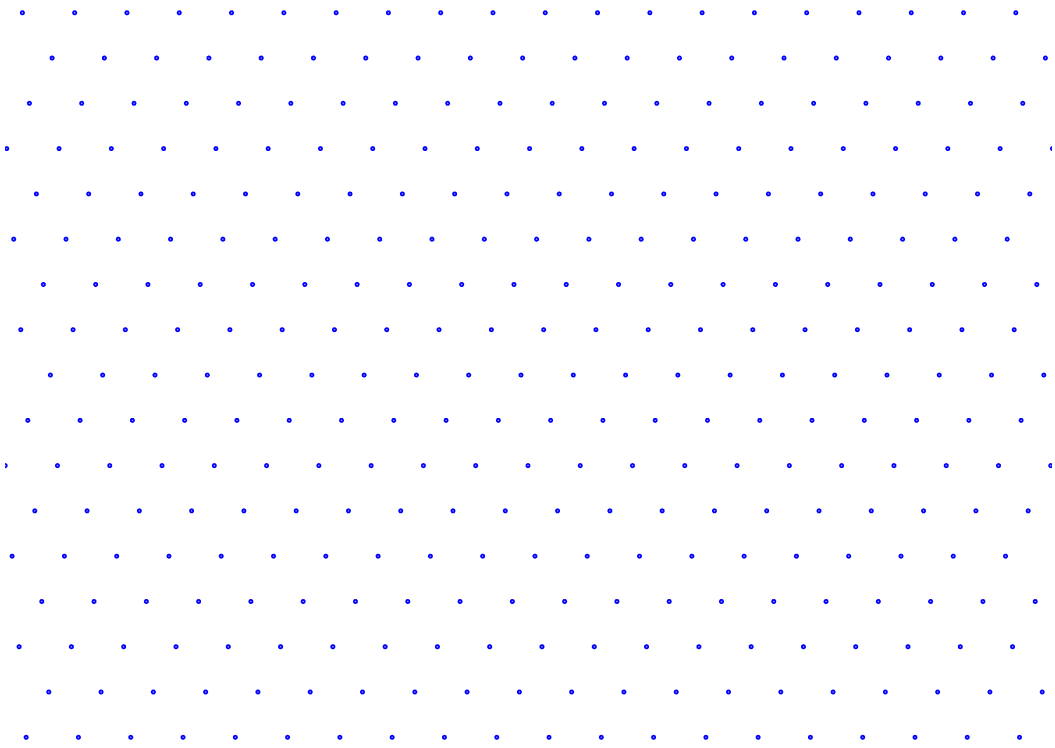
```
Pour effectuer une boucle à droite :
    avancer de L pixels
    tourner vers la gauche de 60°
    avancer de L pixels
    tourner vers la gauche de 60°
    avancer de L pixels

Mettre 64 dans L
initialiser le graphisme tortue
effectuer une boucle à droite
```

Exercices

Reproduire la figure 5 de Sierpiński ci-dessous (on pourra s'aider des points qui sont équidistants) :



Reproduire la figure 6 de Sierpiński ci-dessous :



Écrire un script Python qui fait dessiner par la tortue la figure 5 de Sierpiński (avec une longueur de 32 pixels par segment) :

```
from turtle import *
```

Écrire un script Python qui fait dessiner par la tortue la figure 6 de Sierpiński (avec une longueur de 16 pixels par segment) :

```
from turtle import *
```