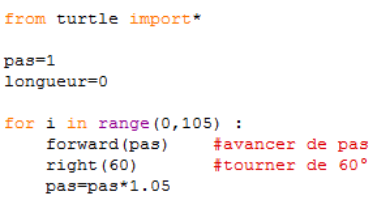
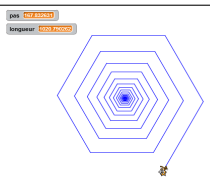
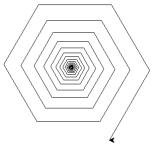
Algorithmes sur Python

1 Partie 1 : mise en place et transition avec le collège

1.1 La spirale - activité enseignant

Vous pourrez adapter cette activité pour la classe

1. Réaliser les spirales sur Scratch et Python :

Scratch	Python
	
	


```
from turtle import*

pas=1
longueur=0

for i in range(0,105) :
    forward(pas)      #avancer de pas
    right(60)         #tourner de 60°
    pas=pas*1.05
```

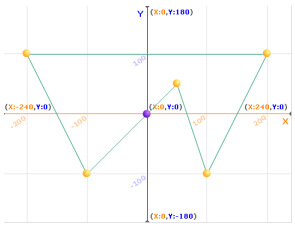
2. Quelle est la longueur de la spirale ?

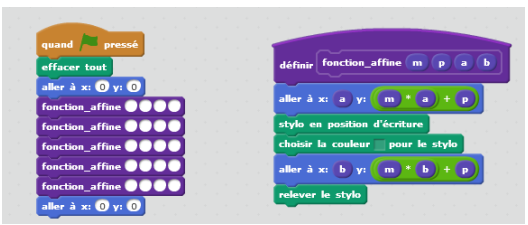
3. Sur chacun des programmes, compléter-le pour obtenir la longueur de la spirale et modifier les pour obtenir une longueur de spirale supérieur ou égale à 1000.

4. Sur tableur organiser un tableau pour retrouver le résultat.
Intérêt du tableur : comprendre la notion de boucle à travers le copier-coller, l'initialisation des variables (une variable par colonne)

1.2 Tracer d'un polygone par des fonctions affines - activité enseignant

1. Sur **Scratch** on a placé 5 points (lutins en jaune). Le lutin violet se déplace sur chaque point suivant le motif suivant.Compléter l'algorithme du lutin violet :





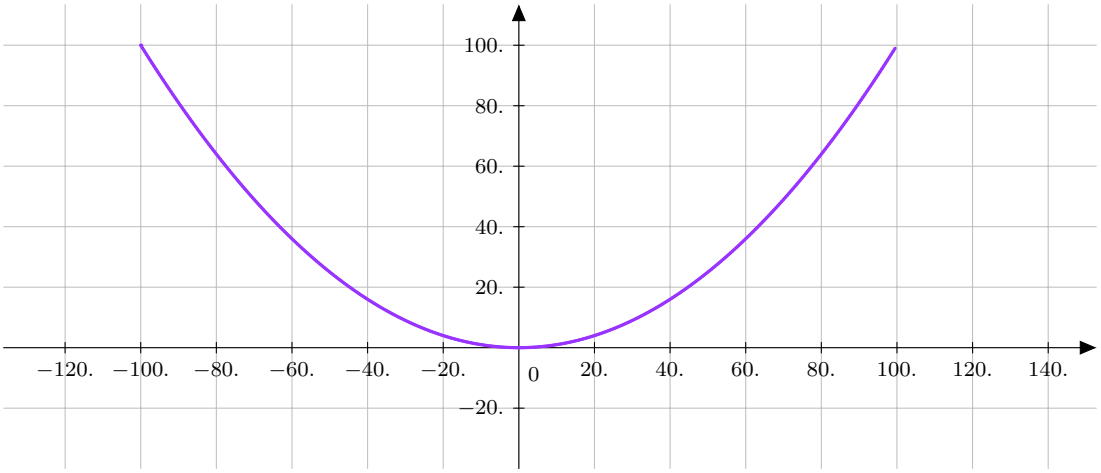
2. Créer le même motif sur **Python** (correction à la suite).

```
from turtle import*
#####
## Les fonctions
#####
def fonction_affine(m,p,x_A,x_B):
    up()
    goto(x_A,m*x_A+p)
    down()
    goto(x_B,m*x_B+p)

def coefficient_directeur(x_A,y_A,x_B,y_B):
    return (y_B-y_A)/(x_B-x_A)

def ordonnee_origine(x_A,y_A,x_B,y_B):
    return y_A-coefficient_directeur(x_A,y_A,x_B,y_B)*x_A


fonction_affine(coefficient_directeur(-200,100,200,100),ordonnee_origine(-200,100,200,100),-200,200)
fonction_affine(coefficient_directeur(200,100,100,-100),ordonnee_origine(200,100,100,-100),200,100)
fonction_affine(coefficient_directeur(100,-100,50,50),ordonnee_origine(100,-100,50,50),100,50)
fonction_affine(coefficient_directeur(-100,-100,50,50),ordonnee_origine(-100,-100,50,50),50,-100)
fonction_affine(coefficient_directeur(-100,-100,-200,100),ordonnee_origine(-100,-100,-200,100),-100,-200)
```

1.3 Tracer d'une courbe et longueur d'une courbe

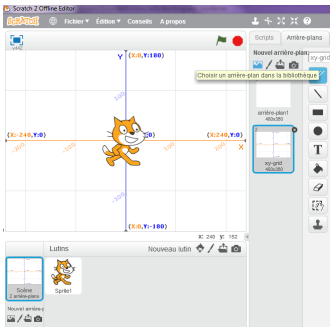
1.3.1 Activité de l'élève - TD

Sur Scratch, le chat doit parcourir une courbe $\mathcal{C}$ définie par une fonction f sur l'intervalle $[-100\ ;\ 100]$ par l'expression $f(x) = 0,01x^2$:



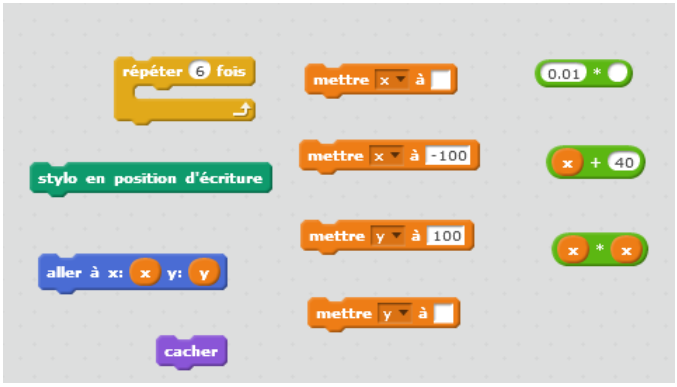
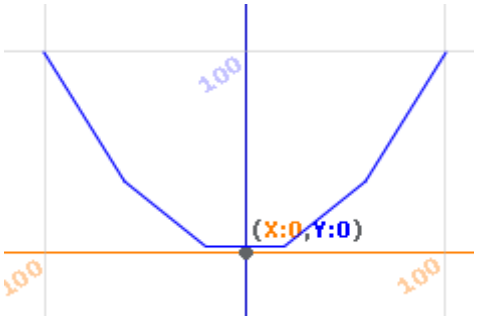
1. Mise en place du déplacement du chat sur Scratch :

   (a) choisir l'arrière plan XY-Grid :



   (b) Commencer par éditer les commandes suivantes qui permettront au chat de se replacer comme il faut à chaque tentative déclenchée par le drapeau vert :

(c) Les commandes suivantes proposent un chemin qui approche grossièrement la courbe $\mathcal{C}$, replacer les afin d'obtenir le résultat proposé :

commandes	résultat
	

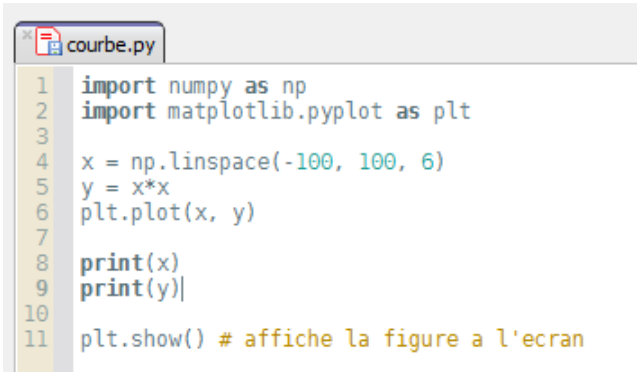
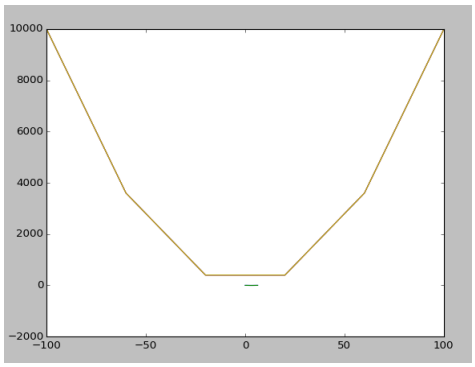
(d) En répétant 21 fois les commandes de la **boucle**, l'algorithme donne un tracer plus précis, plus proche de la courbe $\mathcal{C}$. Modifier l'algorithme en remplaçant 6 par 21 et en modifiant une commande de la boucle.

2. Modifier l'algorithme pour tracer la courbe $\mathcal{C}'$ décrite par la fonction g définie sur l'intervalle $[-100\ ;\ 100]$ par $g(x) = 0.0001x^3 - x$.

3. Algorithme sur Python : On reprend la fonction f définie sur $[-100\ ;\ 100]$ par $f(x) = x^2$.

(a) Ouvrir le logiciel Python Pyzo : 

(b) Saisir les instructions sans erreurs suivantes :

instructions	résultats (graphique et affichage)
	

courbe.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  x = np.linspace(-100, 100, 6)
5  y = x*x
6  plt.plot(x, y)
7
8  print(x)
9  print(y)
10
11 plt.show() # affiche la figure a l'ecran
```



```
[-100.  -60.  -20.   20.   60.  100.]
[ 10000.   3600.    400.    400.   3600.  10000.]
```

(c) Modifier les instructions pour avoir 21 segments qui approchent la courbe $\mathcal{C}$.

(d) Modifier l'algorithme pour tracer la courbe $\mathcal{C}'$.

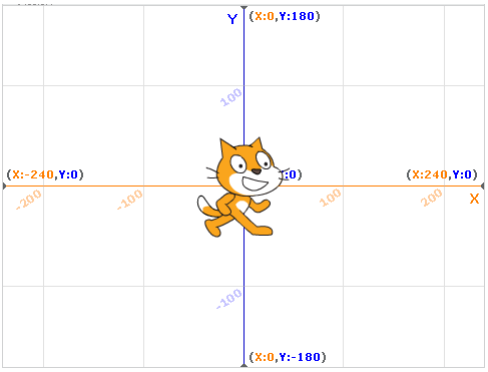
1.3.2 Activité de l'enseignant

1. Faire le TD élève,
2. Compléter le TD élève pour obtenir la longueur approchée des courbes $\mathcal{C}$ et $\mathcal{C}'$.

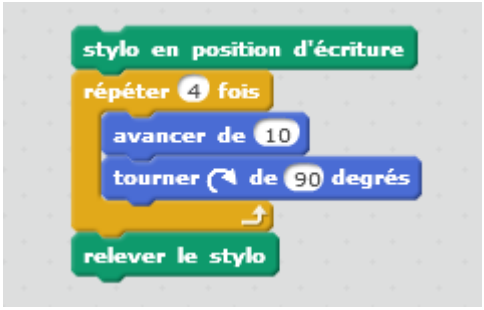
1.4 Frises et pavages

1.4.1 Activité de l'élève - TD1

Le but de l'exercice est de déplacer un motif suivant une translation de vecteur $\vec{u}$ définit par ses coordonnées. Choisir la grille-XY pour scène :



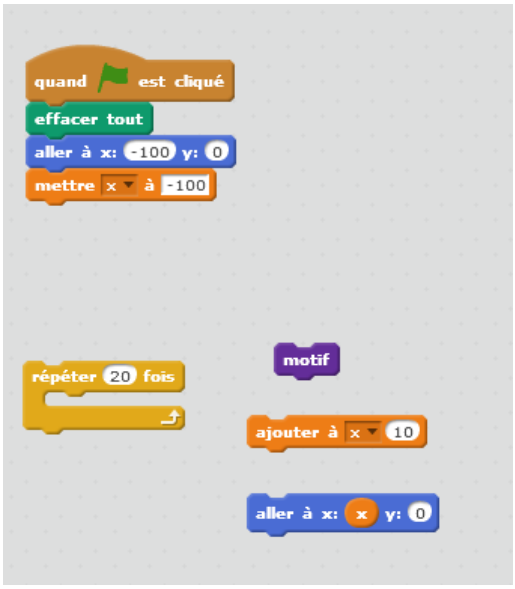
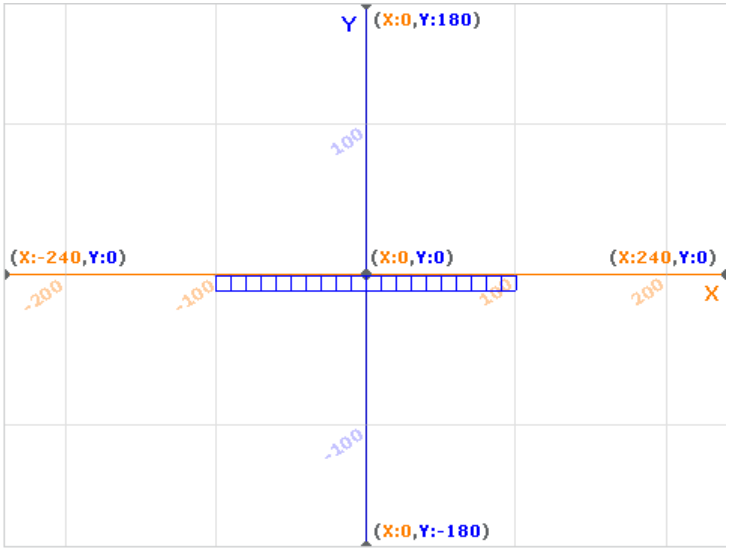
1. Création du motif à répéter :
   Quelle figure décrit le code suivant ?



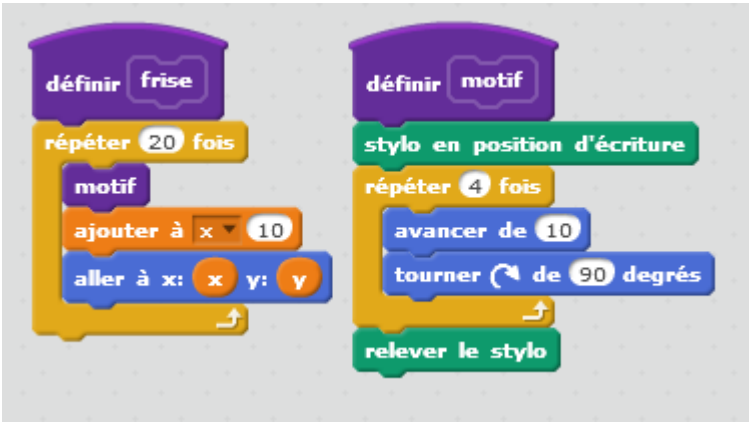
   Créer le motif sur Scratch.

2. On souhaite déplacer ce motif 20 fois suivant le vecteur $\vec{u}$ de coordonnées $\begin{pmatrix} 10 \\ 0 \end{pmatrix}$; l'abscisse est 10 et l'ordonnée est 0. On commence la frise au point de coordonnées $(-100\ ;\ 0)$.
   La fonction motif permettra de produire la frise :

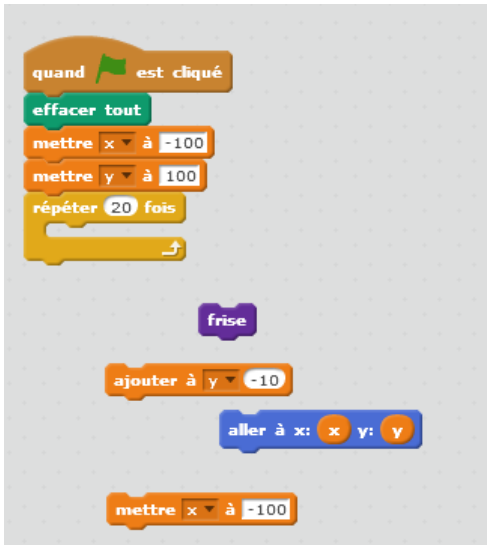
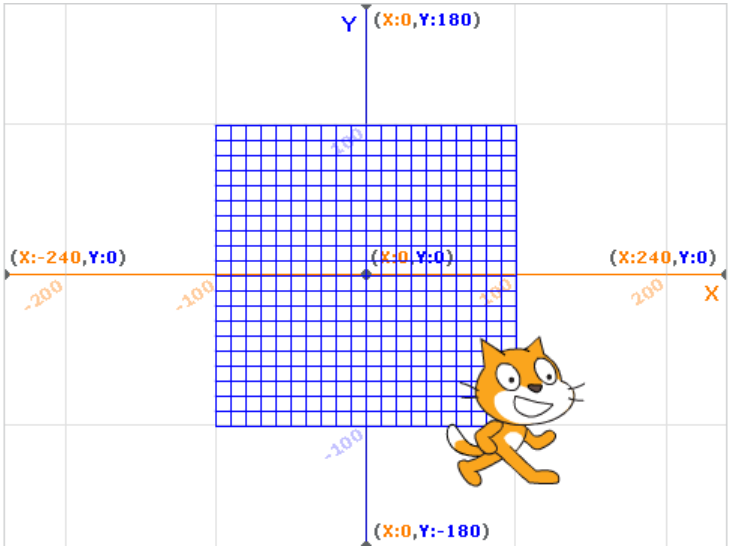
   Remettre dans l'ordre les instructions suivantes pour obtenir le résultat suivant :

Codes, instructions	Résultat
	

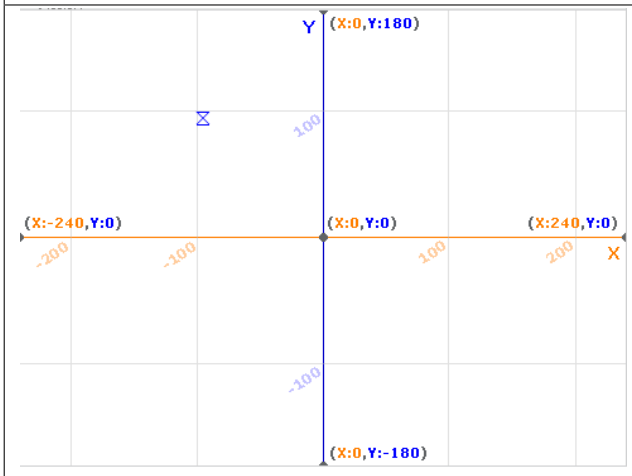
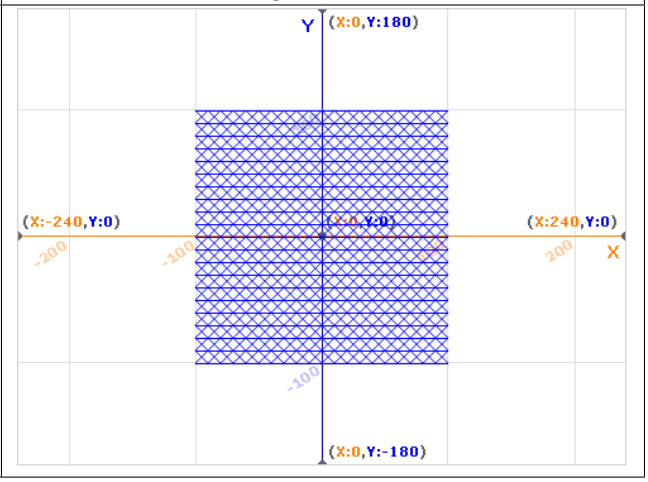
3. Appelons le résultat précédent frise et codons cette frise dans l'algorithme par la fonction suivante à compléter :



On souhaite déplacer cette frise 20 fois suivant le vecteur $\vec{v}$ de coordonnées $\begin{pmatrix} 0 \\ 10 \end{pmatrix}$; l'abscisse est 0 et l'ordonnée est 10. On commence la figure (pavage) au point de coordonnées $(-100\ ;\ 100)$.
La fonction frise permettra de produire la figure (pavage) :

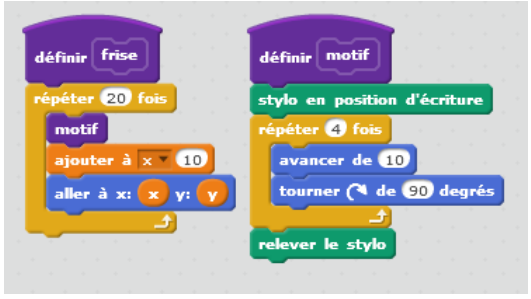
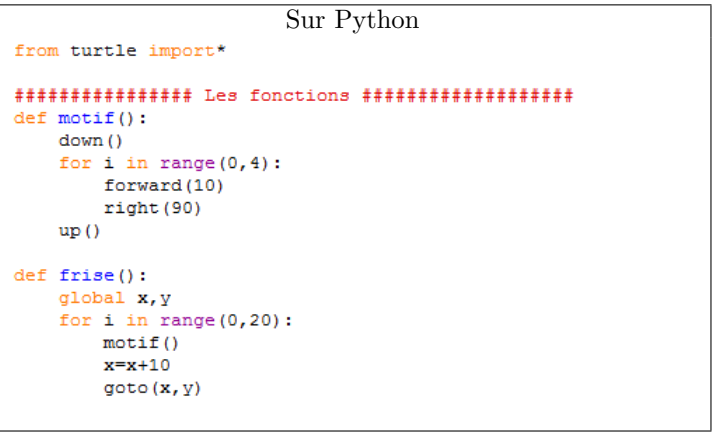
Codes, instructions	Résultat
	

4. Modifier pour obtenir le motif suivant puis la figure suivantes :

motif	figure finale
	

1.4.2 Activité de l'élève - TD2

1. À partir du TD sur Scratch, retrouver les correspondances entre les codes pour définir le même tracer motif et frise sur Python :

Sur Scratch	Sur Python
	

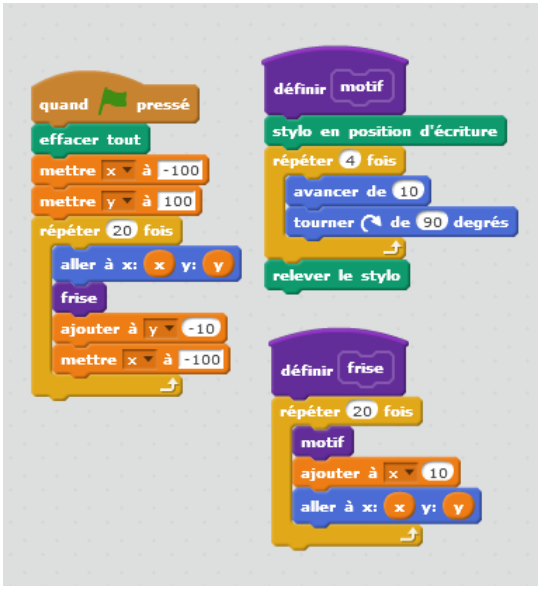
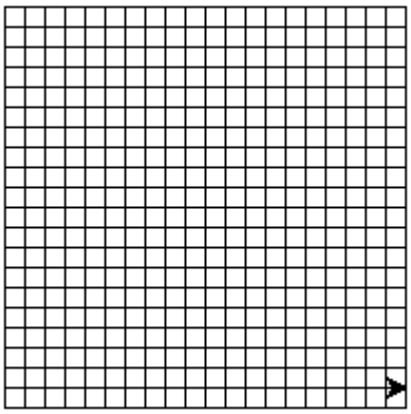

```
from turtle import*

################ Les fonctions ###################
def motif():
    down()
    for i in range(0,4):
        forward(10)
        right(90)
    up()

def frise():
    global x,y
    for i in range(0,20):
        motif()
        x=x+10
        goto(x,y)
```

- Déclaration d'une fonction :
- Appelle d'une fonction :
- Boucle répéter :
- Avancer la tortue :
- Tourner à droite de 90 degré :
- Lever le crayon :
- Baisser le crayon :

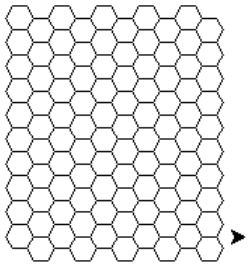
2. Recopier le début du code Python sur Pyzo (ou IDLE) :

```
from turtle import*

################ Les fonctions ###################
def motif():
    down()
    for i in range(0,4):
        forward(10)
        right(90)
    up()

def frise():
    global x,y
    for i in range(0,20):
        motif()
        x=x+10
        goto(x,y)


############## Le programme principal ###############
clear()
speed(0) #permet d'accélérer la réalisation entre 0 et 10
up()     #par défaut le stylo est en position d'écriture, il faut donc le lever
x=-100
y=100
```

3. En vous inspirant de la réalisation sur Scratch, compléter l'algorithme sur Python pour obtenir le pavage suivant :

Codes sur Scratch	Représentation du résultat
	

Exercice : un pavage hexagonal Le but de l'exercice est d'obtenir le pavage suivant :



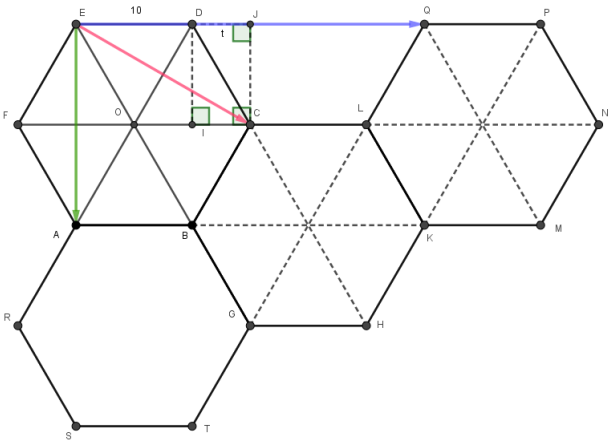
à partir de la frise suivante :

avec le motif suivant :

Recopier et compléter le programme suivant en remplaçant les points d'interrogation par ce qu'il convient (on pourra ne pas recopier les commentaires de speed, int et up) :

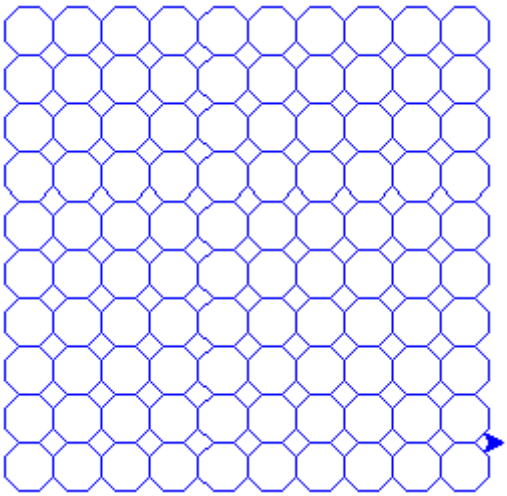
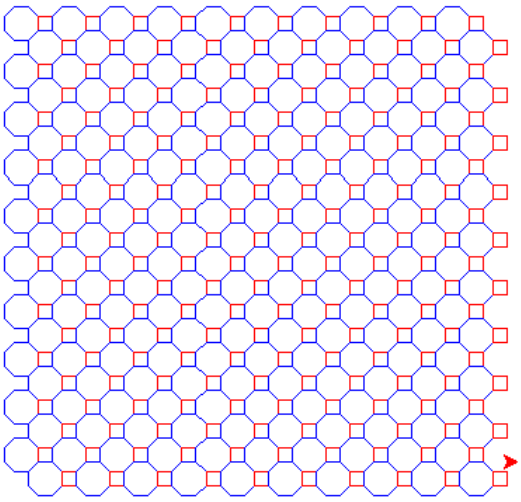
```
from turtle import*
from math import*

################ Les fonctions ###################
def motif():
    down()
    for i in range(0,?):
        forward(10)
        right(?)
    up()

def frise():
    global x,y
    for i in range(0,5):
        motif()
        x=x+?
        goto(x,y)


############## Le programme principal ###############
clear()
speed(0) #permet d'accélérer la réalisation entre 0 et 10
up()     #par défaut le stylo est en position d'écriture, il faut donc le lever
x=-100
y=100

for i in range(0,20):
    goto(x,y)
    frise()
    if i/2==int(i/2):       #int(nombre) donne la partie entière d'un nombre.
        x=?
    else :
        x=?
    y=?
```

Aide pour comprendre la représentation :



1.4.3 Prolongement des activités - activité de l'enseignant

1. Faire les deux TD précédents
2. Compléter ces TD pour obtenir les pavages suivants : Sur Python Pyzo (ou IDLE) réaliser l'**un** des pavages suivants :

Facile	Difficile
	

2 Partie 2 : Python pour les Mathématiques

2.1 Les fonctions statistiques et représentations graphiques

2.1.1 Activité de l'élève - TD1

Soit la série statistique discrète suivante (cette série peut représenter les températures observées chaque jour du mois de septembre à 14h dans une ville donnée) :

valeurs (x_i)	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
effectifs (n_i)	1	2	1	2	2	1	1	2	3	2	3	2	2	3	2	1

Sur Tableur, on souhaite mettre en place la recherche du calcul d'une moyenne, des fréquences et des fréquences cumulées croissantes :

1. Ouvrir le fichier statistiques1.calc.
2. Compléter le fichier par à l'aide des formules suivantes et de glisser-coller si nécessaire :
   - =B2/B$18
   - =somme(B2:B17)
   - =B2*A2
   - =D2
   - =somme(B2:B17)/B18
   - =E2+D3

   résultat :

	A	B	C	D	E
1	valeurs	effectifs	produit (valeurs*effectifs)	fréquences	fréquences cumulées croissantes
2	12	1	12	3,33%	3,33%
3	13	2	26	6,67%	10,00%
4	14	1	14	3,33%	13,33%
5	15	2	30	6,67%	20,00%
6	16	2	32	6,67%	26,67%
7	17	1	17	3,33%	30,00%
8	18	1	18	3,33%	33,33%
9	19	2	38	6,67%	40,00%
10	20	3	60	10,00%	50,00%
11	21	2	42	6,67%	56,67%
12	22	3	66	10,00%	66,67%
13	23	2	46	6,67%	73,33%
14	24	2	48	6,67%	80,00%
15	25	3	75	10,00%	90,00%
16	26	2	52	6,67%	96,67%
17	27	1	27	3,33%	100,00%
18	total	30	20,1	moyenne	
19					

Sur Python, on souhaite mettre en place la recherche de moyenne et fréquences sur Python.

1. Deux listes Lvaleur et Leffectif seront les listes qui définissent la série statistique :

   - Lvaleur=[12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
   - Leffectif=[1,2,1,2,2,1,1,2,3,2,3,2,2,3,2,1]

   La fonction *len*(*liste*) de Python permet de calculer la taille d'une *liste*, soit son nombre d'éléments. L[i] permet de donner la valeur du i^{eme} l'élément de la liste L. Une liste sur Python commence au rang 0 : Lvaleur[0] est 12.

   (a) Déterminer *len*(*Lvaleur*) : *len*(*Lvaleur*) =

   (b) Pour i=3 donner Lvaleur[i].

2. Le programme suivant permet de calculer l'effectif total (avec la fonction total), la moyenne (avec la fonction) moyenne, les fréquences (avec la fonction fréquence), et les effectifs ou les fréquences cumulées croissantes (avec la fonction listecumulee) :

```
from math import*


Lvaleur=[12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
Leffectif=[1,2,1,2,2,1,1,2,3,2,3,2,2,3,2,1]

def total(L): #la fonction "total" dépend" d'une liste L (paramètre)
    s=0 #initialisation de la variable s à 0
    for i in range(0,len(L)):
        s=__________
    return s

def moyenne(L1,L2): #la fonction "moyenne" dépend de deux listes L1 et L2
    s=0  #initialisation de la variable somme à 0
    for i in range(0,len(L1)):
        s=__________
    return s/total(L2)

def frequence(L): #la fonction "fréquence" dépend d'une liste L
    Lf=[] #initialisation d'une liste à une liste vide
    s=total(L) #initialsation de la variable s au total de la liste L
    for i in range(__________,__________):
        Lf.append(__________) #.append(élément) permet d'ajouter "l'élément" à la fin de la liste
                    #round(nombre,2) permet d'arrondir le nombre à deux décimales
    return Lf

def listecumulee(L): #la fonction "listecumlee" dépend d'une liste L
    Lc=[L[0]] # initialisation de la variable Lc à la valeur L[0] (première valeur de la liste L)
    for i in range(__________,__________):
        Lc.append(__________)
    return Lc

#affichages
print('valeurs : ',Lvaleur)
print('effectifs :',Leffectif)
print('effectif total : ',total(Leffectif))
print('frequences : ',frequence(__________))
print('effectifs cumulés croissantes : ',listecumulee(__________))
print('fréquences cumumées croissantes : ',listecumulee(frequence(__________)))
print('moyenne : ',moyenne(__________,__________))
```

   (a) Ouvrir le fichier statistiques1.py avec IDLE

   (b) Dans chacune des fonctions, remplacer les traits soulignés par l'un des codes suivants qui convient :

round(L[i]+Lc[i-1],2)	round(L[i]/s,2)	1	L1[i]*L2[i]+s	s+L[i]	0	len(L)

   (c) Dans les affichages, compléter les traits soulignés par la bonne liste.

2.1.2 Activité de l'élève - TD2

Soit la série statistique discrète suivante (cette série peut représenter les températures observées chaque jour du mois de septembre à 14h dans une ville donnée) :

valeurs (x_i)	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
effectifs (n_i)	1	2	1	2	2	1	1	2	3	2	3	2	2	3	2	1

Sur Tableur, on souhaite mettre en place la recherche de la médiane (deuxième quartile) et des quartiles 1 et 3 à l'aide des fréquences cumulées croissantes :

1. Ouvrir le fichier statistiques1.calc utilisé lors du dernier TD.
2. En cellule F2 saisir la formule =SI(ET(E2>=0,25;E1<0,25);A2;0) puis glisser-coller vers le bas.
3. De la même manière, construire une formule pour déterminer le deuxième quartile (médiane) en cellule G2, puis une formule pour déterminer le troisième quartile en cellule H2.
   résultat :

	A	B	C	D	E	F	G	H
1	valeurs	effectifs	produit (valeurs*effectifs)	fréquences	fréquences cumulées croissantes	Q1	Q2	Q3
2	12	1	12	3,33%	3,33%	0	0	0
3	13	2	26	6,67%	10,00%	0	0	0
4	14	1	14	3,33%	13,33%	0	0	0
5	15	2	30	6,67%	20,00%	0	0	0
6	16	2	32	6,67%	26,67%	16	0	0
7	17	1	17	3,33%	30,00%	0	0	0
8	18	1	18	3,33%	33,33%	0	0	0
9	19	2	38	6,67%	40,00%	0	0	0
10	20	3	60	10,00%	50,00%	0	20	0
11	21	2	42	6,67%	56,67%	0	0	0
12	22	3	66	10,00%	66,67%	0	0	0
13	23	2	46	6,67%	73,33%	0	0	0
14	24	2	48	6,67%	80,00%	0	0	24
15	25	3	75	10,00%	90,00%	0	0	0
16	26	2	52	6,67%	96,67%	0	0	0
17	27	1	27	3,33%	100,00%	0	0	0
18	total	30	20,1	moyenne				

Sur Python, on souhaite mettre en place la recherche des quartiles à partir de la liste des fréquences cumulées sur Python.

1. Reprendre le fichier statistique1.py
2. Le fonction quartile (pour n'importe quel quartile Q1, Q2 ou Q3) est donnée dans le désordre (les parenthèses sont à compléter) :

quartile():	listecumulee()	frequence()	L1[i]
L1,L2,numero	while	i+1	L2
i=0	def	i=	<
Lc[i]	Lc=	0.25*numero:	return

   Recopier le code de la fonction quartile dans l'ordre en faisant attention aux indentations.

3. Compléter l'affichage pour tester votre programme :

```
print('quartile 1 : ',quartile(__________,__________,__________))
print('quartile 2 : ',quartile(__________,__________,__________))
print('quartile 3 : ',quartile(__________,__________,__________))
```

2.1.3 Activité enseignant

1. Refaire les deux TD
2. Insérer un diagramme en bâtons et un diagramme en boites de la série :
   bibliothèque Mathplotlib
   aide à la réalisation :

```
plt.figure(1)
plt.subplot(2,1,1) #placement du premier graphique
plt.title(" diagramme en bâtons")
plt.xlabel("valeurs de la série")
plt.ylabel("effectifs")
plt.axis(abscisse min,abscisse max,ordonnée min,ordonnée max)
plt.bar(Liste des valeurs,Liste des effectifs,largeur du bâton, color='red')

plt.subplot(2,1,2) #placement du deuxième graphique
plt.subplots_adjust(hspace=0.5)
plt.title(" diagramme en boîtes")
plt.xlabel("valeurs de la série")
plt.axis(abscisse min,abscisse max,ordonnée min,ordonnée max)
plt.boxplot(Liste,vert=0,positions=[1])

plt.show()
```

3 Application aux simulations statistiques

3.1 activité des élèves

Arthur lance plusieurs fois un dé et il compte le nombre de lancés nécessaires pour obtenir 6. Au bout de 10 expériences, il s'aperçoit qu'il obtient 6 avant le sixième lancé.
Le but de l'exercice est de simuler n parties du lancé d'un dé jusqu'à obtenir 6 et d'exploiter les résultats statistiques afin de se faire une idée de l'expérience aléatoire.
simulation sur Python

1. Ouvrir Pyzo-Python
2. En Python on donne la fonction *de()* qui permet de simuler le lancer d'un dé équilibré à six face :

```
#temps d'attente pour obtenir 6
import random

#fonctions
def de():
    return random.randint(1,6)
#fonction python de la bibliothèque random pour choisir un entier entre 1 et 6
```

   Recopier le code et tester le avec la commande d'affichage : *print(de())*

3. On souhaite simuler une partie : On lance le dé jusqu'à obtenir 6 et on compte le nombre de lancés nécessaires.
   On donne l'algorithme à la main :

   Algorithme :
   numerode ← de()
   nombrelance ← 1
   tant que numerode..... :
   nombrelance ← ....
   numerode ← de()

(a) Compléter l'algorithme à la main qui dépend de deux variables *numerode* et *nombrelance*

(b) Après la fonction *de()* coder cet algorithme sur Python et tester-le à l'aide de la commande d'affichage *print()*

4. (a) Deux listes *listevaleur* et *listeeffectif* permettent de retenir les résultats de la simulation, elles sont initialisées chacune à une liste vide : saisir le code suivant entre la fonction *de()* et l'algorithme précédent :
   - listevaleur=[]
   - listeeffectif=[]

   (b) Recopier le code suivant à la suite de l'algorithme précédent :

```
if len(listevaleur)<nombrelance:
    for i in range(len(listevaleur),nombrelance):
        listevaleur.append(i+1) #.append permet d'ajouter l'élément i+1 à la liste
        listeeffectif.append(0) #.append permet d'ajouter l'élément 0 à la liste
listeeffectif[nombrelance-1]=listeeffectif[nombrelance-1]+1
```

   Tester le code avec l'affichage des deux listes : vous venez de simuler une seule partie du lancé de dé jusqu'à obtenir 6.

   (c) Compléter le code avec une simulation de 10 parties et la boucle *pour* et la commande associée *for i in range(0,10):*. Vous ferez attention à l'indentation pour toutes les instructions qui se trouvent dans la boucle pour.

   (d) enregistrer le fichier sous le nom attentenumerode.py.

5. Étude statistique :

   (a) Ouvrir le fichier statistique1.py qui contient les fonctions statistiques (*moyenne, quartiles, médiane,etc...).*

   (b) À l'aide de copier-coller organiser-le code de la manière suivante dans le fichier attentede.py :

```
#titre du td : temps d'attente pour obtenir 6

#Les fonctions : lancé dé et statistiques

#programme principal : simulation des 10 lancers, affichage des listes

#affichage des résultats statistiques : appels des fonctions statitistiques et affichage des résultats

#représentation graphique : un diagramme en bâtons
```

6. finir de compléter le code avec celui de la représentation graphique avec la commande de bibliothèque *import matplotlib.pyplot as plt*

```
plt.bar(listevaleur, listeeffectif, 0.05, color=blue)
plt.xlabel('nombre lancés')
plt.ylabel('effectif des parties')
plt.title('temps attente numéro dé')

plt.savefig('resultatattentenumerode.png')
plt.show()
```

7. A partir de quel nombre n de parties le graphique devient-il *stable* ?

3.2 activité enseignant

1. faire l'activité des élèves
2. Dans le même esprit créer une autre activité de simulation dont l'élève ne connait pas les probabilités correspondantes.

3.3 Algorithme de dichotomie : introduction et statistiques

3.3.1 L'activité des élèves - TD1

Problème ouvert : *Ton voisin pense à un nombre entier compris en 0 et 100. Le jeu consiste à deviner ce nombre le plus rapidement possible (c'est à dire avec le plus petit nombre de tentatives), ton voisin ne peut donner comme indication que "plus" si le nombre que tu proposes est plus petit que celui à deviner ou "moins" si le nombre que tu proposes est plus grand que le nombre à deviner.*

1. Écrire (à la main) ton algorithme pour trouver ce nombre.
2. Mettre en place cet algorithme sur Python (une variable nbdevine permet de stocker le nombre à deviner, une variable nbpropose sera proposée par l'utilisateur (qui sera la machine en traduisant l'algorithme de la question précédente) et comparer avec le nombre à deviner, l'algorithme donnera le nombre de coups nécessaire pour trouver ce nombre).
   Bibliothèques utiles :
   - import math
   - import random
   - import matplotlib.pyplot as plt

   Instructions utiles :
   - random.randint(a,b) (permet de trouver choisir au hasard un entier compris entre a et b).
   - int(a) (partie entière du nombre décimal a).

Note : On peut s'attendre à différents algorithmes construits par les élèves et parmi ces algorithmes celui de dichotomie

3.3.2 L'activité des élèves - TD2

Comparaison de d'algorithmes : reprise de l'algorithme du TD1.

1. Faire une simulation de 1000 parties en stockant les nombres de coups réalisés par partie dans une liste.
2. Reprendre les fonctions statistiques et faire une représentation graphique des résultats obtenus.
3. Comparer la rapidité de votre algorithme avec un de vos camarades.

3.3.3 Activité de l'enseignant

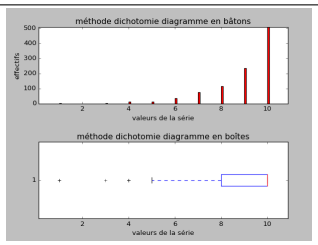
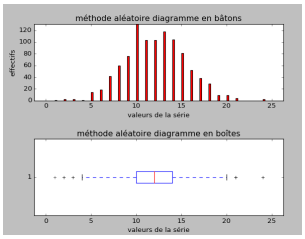
1. Faire les deux activités précédentes.
2. Comparer deux algorithmes : dichotomie et recherche au hasard. indication de correction :

dichotomie

```
## intialisation des parties

n=1000
Listdicho=[]

for i in range(0,n):
    a=1
    b=1000
    nombre_coups=0
    nombre_propose=0
    nombre_deviner=random.randint(1,1000)
    #print(nombre_deviner)
    while nombre_deviner != nombre_propose :
        nombre_coups=nombre_coups+1
        nombre_propose=int((a+b)/2)
        if nombre_deviner<nombre_propose :
            b=nombre_propose
        else :
            a=nombre_propose
        #print(a," ; ",b," ; ",nombre_coups)
    Listdicho.append(nombre_coups)
```

aléatoire

```
## intialisation des parties

Listalea=[]

for i in range(0,n):
    a=1
    b=1000
    nombre_coups=0
    nombre_propose=0
    nombre_deviner=random.randint(1,1000)
    #print(nombre_deviner)
    while nombre_deviner != nombre_propose :
        nombre_coups=nombre_coups+1
        nombre_propose=random.randint(a,b)
        if nombre_deviner<nombre_propose :
            b=nombre_propose-1
        else :
            a=nombre_propose+1
        #print(a," ; ",b," ; ",nombre_coups)
    Listalea.append(nombre_coups)
```

dichotomie	aléatoire
effectif total : 1000 moyenne : 8.979 écart-type : 1.440332947620098 médiane : 10.0 premier quartile : 8 deuxième quartile : 10 troisième quartile : 10 premier décile : 7 neuvième décile : 10 mode : 10 étendue : 9	effectif total : 1000 moyenne : 12.024 écart-type : 3.309897883621185 médiane : 12.0 premier quartile : 10 deuxième quartile : 12 troisième quartile : 14 premier décile : 8 neuvième décile : 16 mode : 10 étendue : 23
	

3.4 Algorithme de dichotomie et résolution d'une équation

Problème :
Utiliser l'algorithme de dichotomie pour trouver une valeur approchée d'une solution d'équation $f(x) = k$:
ABCD est un carré de côté x, le point B est sur le segment [AE] tel que AE=10. Pour quelle valeur de x l'aire du trapèze AECD est-elle égale à 6 unité d'aire ?
Utiliser l'algorithme de dichotomie sur Python pour résoudre le problème.

3.5 Statistiques - ICN (d'après Laurent Renaud)

En lien avec l'ICN et les enseignements de langues, il pourrait être intéressant de comparer les romans suivants les langues (par exemple "le tour du monde en 80 jours").

3.5.1 Partie 1 : ICN

L'activité suivante permet de définir une série statistique qualitative : les occurrences des lettres de l'alphabet dans un texte quelconque.

1. Enregistrer au format .txt sous le nom texte.txt le poème suivant (le bloc note permet cet enregistrement) - Charles Beaudelaire (1821-1867) :

L'albatros

Souvent, pour s'amuser, les hommes d'équipage
Prennent des albatros, vastes oiseaux des mers,
Qui suivent, indolents compagnons de voyage,
Le navire glissant sur les gouffres amers.

A peine les ont-ils déposés sur les planches,
Que ces rois de l'azur, maladroits et honteux,
Laissent piteusement leurs grandes ailes blanches
Comme des avirons traîner à côté d'eux.

Ce voyageur ailé, comme il est gauche et veule !
Lui, naguère si beau, qu'il est comique et laid !
L'un agace son bec avec un brûle-gueule,
L'autre mime, en boitant, l'infirme qui volait !

Le Poète est semblable au prince des nuées
Qui hante la tempête et se rit de l'archer ;
Exilé sur le sol au milieu des huées,
Ses ailes de géant l'empêchent de marcher.

2. Le code suivant permet de mettre un dans un tableau le texte précédent :

```
texte=open(fichier,'r') #ouverture du fichier texte en mode read ("w" mode wrtie, "a" mode ajout)
tableau=texte.readlines() #chaque ligne du texte est une liste de caractère, le tableau est une liste de listes ligne.
texte.close()
print(tableau)
```

3. La fonction suivante compte tous les caractères du texte (espaces et apostrophes compris) :

```
def compte_caractere(fichier) :
    texte=open(fichier,'r')
    tableau=texte.readlines()
    texte.close()
    nb_caracteres=0
    for ligne in tableau:
        nb_caracters=nb_caracteres+len(ligne)-1
    nb_caracteres=nb_caracteres+1
    return nb_caracteres

print(compte_caractere("texte.txt"))
```

4. La fonction suivante permet de scanner un caractère du texte :

```
def scan_ligne(ligne,type_caractere):
    nb_apparitions=0
    for caractere in ligne:
        if(caractere==type_caractere):
            nb_apparitions=nb_apparitions+1
    return nb_apparitions
```

5. La fonction comptecaractere complétée permet de compter les caractères du texte et renvoie l'effectif associé et le nombre total de caractères y compris les espaces et les apostrophes :

```
def compte_caractere(fichier) :
    texte=open(fichier,'r') #ouverture du fichier texte en mode read ("w" mode wrtie, "a" mode ajout)
    tableau=texte.readlines()
    texte.close()
    nb_caracteres=0
    alphabet_minuscule=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
    alphabet_majuscule=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
    occurrences=26*[0]
    for ligne in tableau:
        nb_caracters=nb_caracteres+len(ligne)-1
        for i in range(0,26):
            occurrences[i]=occurrences[i]+scan_ligne(ligne,alphabet_minuscule[i])+scan_ligne(ligne,alphabet_majuscule[i])
    return nb_caracteres+1,occurrences
```

3.5.2 Partie 2 : mathématiques

reprendre les fonctions effectif total et fréquences

(a) Pour la liste des occurrences, donner les fréquences des occurrences et en donner un graphique.

(b) *Application :*

i. Donner le diagramme de la fréquence des occurrences du texte du tour du monde en 80 jours de Jules Verne (disponible sur Internet).

ii. Comparer le graphique obtenu avec la liste des points des lettres du Scrabble : $[1, 3, 3, 2, 1, 4, 2, 4, 1, 8, 10, 1, 2, 1, 1, 3, 8, 1, 1, 1, 1, 4, 10, 10, 10, 10]$

iii. Comparer les occurrences du texte en français et en anglais.

3.6 Simulations statistiques et échantillonnage (document d'accompagnement)

On considère un échantillon de taille n constitué des résultats de n répétitions indépendantes de la même expérience, qui a une probabilité p de succès, et on relève la fréquence f du succès de l'expérience.

```
import random

def echantillon(p,n):
#simule n répétitions indépendantes d'une expérience
#dont le succès est de probabilité p,
#et renvoie la fréquence des expériences réussies
    succes = 0
    for i in range(n):
        if random.random() <= p:
            succes = succes+1
    f = succes /n
    return f
```

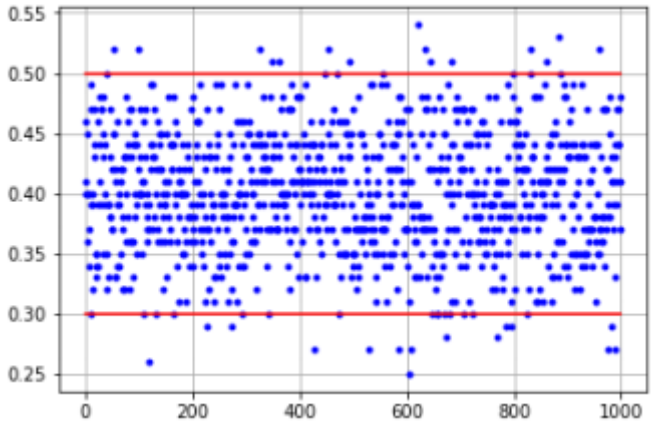
Une évaluation echantillon(0.2,10000) peut renvoyer 0.1981, par exemple (le résultat est évidemment aléatoire). Le programme de mathématiques invite à observer avec quelle probabilité la fréquence f se retrouve dans l'intervalle de fluctuation $[p - \frac{1}{\sqrt{n}} \ ; \ p + \frac{1}{\sqrt{n}}]$.

```
def fluctuation(p,n):
'''renvoie (a,b) bornes de l'intervalle de fluctuation
au seuil de 95 % pour des échantillons de taille n
et une proportion p du caractère'''
    return(p-1/sqrt(n),p+1/sqrt(n))
```

On peut écrire la fonction suivante pour en avoir un aperçu graphique.

```
def grapheFluctuation(p,n,nbEchantillons):
    a,b = fluctuation(p,n)
    L = []
    for i in range(nbEchantillons):
        L.append(echantillon(p,n))
    plt.plot(list(range(0,nbEchantillons)),L,'b.')
    plt.grid()
    plt.plot([0,nbEchantillons-1],[a,a],'r-')
    plt.plot([0,nbEchantillons-1],[b,b],'r-')
    plt.show()
```

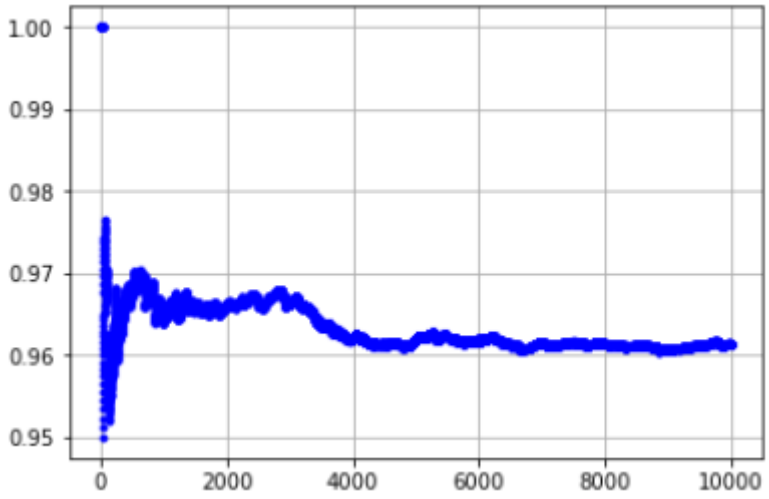
Ici p est la probabilité de succès d'une expérience, n est la taille de chaque échantillon et nbEchantillons le nombre d'échantillons sur lesquels on mène l'observation.
On trace en rouge les horizontales correspondant aux bornes de l'intervalle de fluctuation considéré, et chaque point bleu représente la fréquence obtenue pour un échantillon.
Voici ce que produit l'appel grapheFluctuation(0.4,100,1000).



On peut aussi choisir d'évaluer la proportion des échantillons tels que la fréquence f observée est dans l'intervalle de fluctuation en fonction du nombre d'échantillons simulés.
On observe que cette proportion est rapidement supérieure à 0,95. C'est ce que fait la fonction suivante, qui utilise un compteur d'échantillons corrects, c'est-à-dire tels que la fréquence correspondante est dans l'intervalle souhaité.

```
def grapheProportionDansIntervalleFluctuation(p,n,nbEchantillons):
    a,b = fluctuation(p,n)
    L = []
    corrects = 0
    for i in range(1,nbEchantillons):
        f = echantillon(p,n)
        if f >= a and f <= b:
            corrects = corrects + 1
        proportion = corrects/i
        L.append(corrects/i)
    plt.plot(list(range(1,nbEchantillons)),L,'b.')
    plt.grid()
    plt.show()
```

Voici le résultat de l'appel grapheProportionDansIntervalleFluctuation(0.4,100,10000).



Cet exercice est une reprise d'une activité proposé dans l'ancien document d'accompagnement qui pouvait en partie se faire sur tableur :

	A	B	C	D	E	F	G	H	I	J	K
1	taille n des l'échantillons		2000	probabilité p du succès		0,5					
2	numéro de simulations	1	2	3	4	5	6	7	8	9	10
3	fréquences f :	0,501	0,505	0,509	0,513	0,499	0,517	0,5005	0,5075	0,515	0,4845
4	f-1/racine(n)	0,4786393202	0,48263932	0,48663932	0,49063932	0,47663932	0,49463932	0,47813932	0,48513932	0,49263932	0,46213932
5	f+1/racine(n)	0,5233606798	0,52736068	0,53136068	0,53536068	0,52136068	0,53936068	0,52286068	0,52986068	0,53736068	0,50686068
6	affiche 0 si p est dans l'intervalle 1 sinon	0	0	0	0	0	0	0	0	0	0
7		1	1	0	0	0	1	0	1	0	1
8		0	1	1	1	0	1	0	1	0	1
9		1	0	0	0	0	0	0	1	0	0
10		1	0	0	1	1	0	1	1	1	1
11		0	1	0	0	1	1	1	1	0	1