Guide de démarrage

Remarque : les sections suivantes supposent que vous avez mis à jour votre TI-83 Premium CE Edition Python vers la version 5.7 de l'OS et des Apps ou une version ultérieure. Si vous n'avez pas effectué la mise à jour vers la version 5.7 ou une version ultérieure, veuillez consulter le site : https://education.ti.com/fr/produits-ressources/mise-a-jour-ti-83-premium-ce.
Veillez à sauvegarder ou archiver les fichiers et les données de votre TI-83 Premium CE Edition Python avant de procéder à la mise à jour.

Transférez le module Turtle à votre calculatrice et (si vous l'utilisez) au logiciel TI-SmartView

1) Transférez le module Turtle et le fichier Grid à votre TI-83 Premium CE Edition

Après avoir téléchargé le Zip et extrait les fichiers :

- Ouvrez votre logiciel de bureau TI Connect CE.
- Connectez votre TI-83 Premium CE Edition Python à votre ordinateur en utilisant le câble USB ordinateur-calculatrice fourni avec la calculatrice.
- Vérifiez que votre calculatrice connectée apparaît dans le panneau Calculatrices connectées de l'espace de travail Explorateur de calculatrices.
- Transférez les fichiers **TURTLE.8xv** et **GRID.8xv** vers la ou les calculatrices connectées en faisant glisser les fichiers dans la fenêtre des calculatrices connectées.

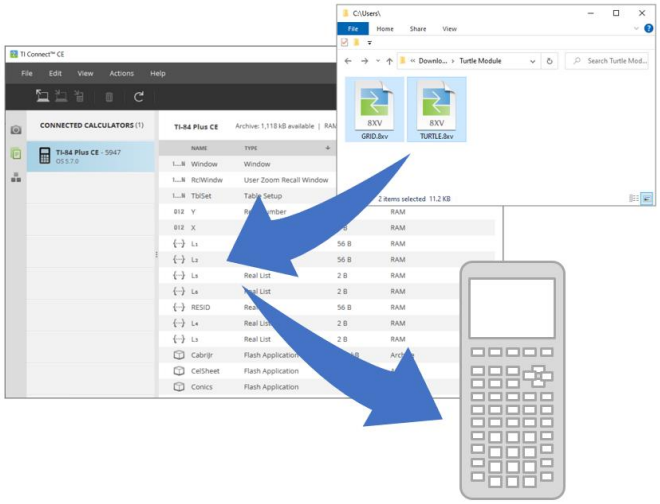
Remarque : le fichier GRID.8xv est une image utilisée comme grille de fond dans les programmes Turtle.

Remarque : les fichiers Turtle et Grid peuvent être aussi transférés d'une calculatrice à une autre à l'aide d'une liaison par câble USB d'unité à unité. Consultez le guide de votre calculatrice pour connaître le processus de transfert de fichiers de calculatrice à calculatrice.

- Ensuite, passez à l'étape 3) : **Importation du module Turtle**

Cette étape nécessite l'utilisation du logiciel (gratuit) TI Connect CE pour ordinateur PC ou Mac.

https://education.ti.com/fr/produits/logiciel-ordinateur/ti-connect-ce-sw



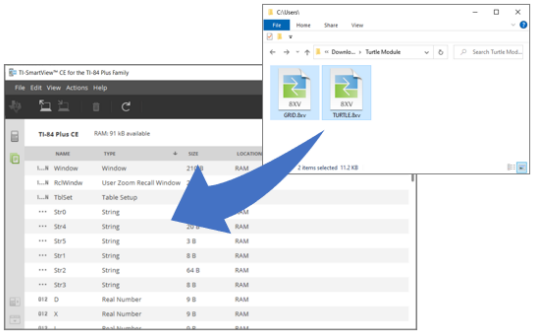
2) Transférez le module Turtle et le fichier Grid vers votre logiciel pour ordinateur TI-SmartView (PC ou Mac).

Cette étape est destinée aux personnes qui utilisent le logiciel pour ordinateur TI-SmartView CE pour les produits de la famille TI-83.

Après avoir téléchargé le Zip et extrait les fichiers :

- Faites glisser et déposez les fichiers TURTLE.8xv et GRID.8xv dans l'espace de travail Explorateur de l'émulateur TI-SmartView CE.

https://education.ti.com/fr/produits-ressources/smartview-ce



- Retournez à l'espace de travail de l'émulateur.

- Ensuite, passez à l'étape 3) :
  Importation du module Turtle

Remarque : au moment de la publication du module Turtle, la version **5.7 de l'OS/App** de **TI-SmartView CE pour la famille TI-83** n'était pas disponible. Voir la note de la section 3 concernant l'onglet [Compl.] et une méthode alternative pour importer le module Turtle en tapant à la main l'instruction d'importation.

3) Importation du module Turtle

Une fois le module Turtle et le fichier Grid transférés, créez un script Python et importez le module Turtle afin d'accéder aux sélections de son menu.

- Sur votre TI-83 Premium CE Edition Python (ou sur l'émulateur), ouvrez l'application Python. Vous pouvez la trouver à l'aide de la touche [**prgm**] ou de la touche [**apps**]. L'application Python s'ouvre sur un écran de gestion des scripts.
- Créez un nouveau script Python en sélectionnant l'onglet intitulé **Nouv** (appuyez sur la touche [**zoom**]).
- Donnez un nom au script Python (par exemple : **SQUARE**), puis sélectionnez **Ok**.

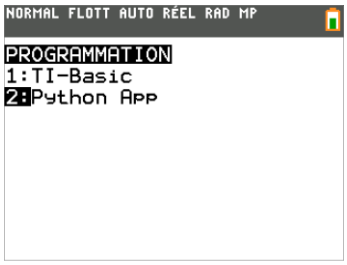
  Vous êtes maintenant dans l'écran de l'éditeur de scripts Python. Saisissez, ensuite, l'instruction d'importation Turtle et un objet Turtle.

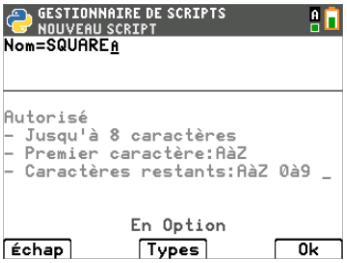
- Sélectionnez l'onglet [**Fns…**] (en bas de l'écran), puis flèche gauche vers l'onglet Modul (en haut de l'écran).
- En bas de l'écran, sélectionnez l'onglet [**Compl.**].
- Sélectionnez l'option **4:from turtle import *** pour coller l'instruction d'importation du module Turtle. Remarquez qu'il colle aussi un constructeur **t=Turtle()** pour assigner à **t** l'objet Turtle.
- Retournez à l'onglet Modul et remarquez l'apparition de **8:turtle…** dans le menu.

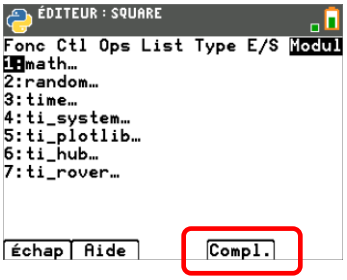
Remarque : l'onglet [**Compl.**] n'est visible qu'avec la version 5.7 ou ultérieure du système d'exploitation. Si vous utilisez une version antérieure, mettez à jour vers la version 5.7 ou ultérieure ; sinon, vous devez saisir manuellement l'instruction d'importation et le constructeur t=Turtle().

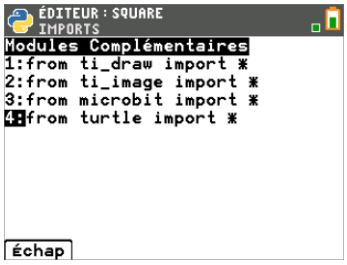
À ce stade, vous êtes prêt à accéder aux sélections de menu du module Turtle et à écrire un script Turtle !

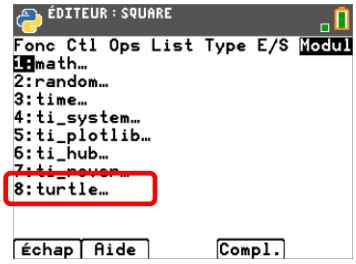
- Passez à la section suivante : **Créer votre premier programme Turtle : Dessinons un carré**











Créer votre premier programme Turtle : Dessinons un carré

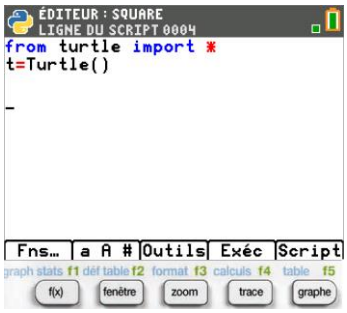
1) Entrez une instruction de boucle

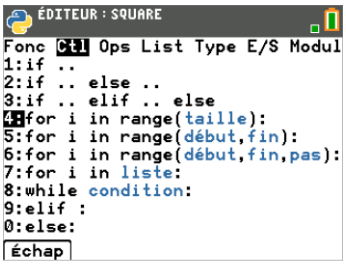
Suite de la section précédente (dans l'écran de l'Éditeur) :

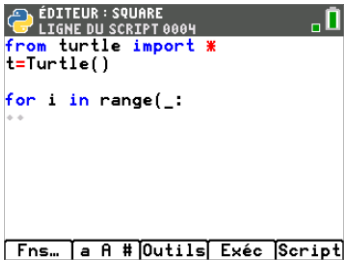
- Appuyez sur [**entrer**] pour aller à la ligne suivante.
- Sélectionnez l'onglet [**Fns…**] (appuyez sur la touche [**f(x)**]) puis flèche droite pour mettre en surbrillance l'onglet Ctl.
- Flèche vers le bas pour sélectionner **4:for i in range(taille) :**
- Appuyez sur [**entrer**] pour le coller dans l'Éditeur.

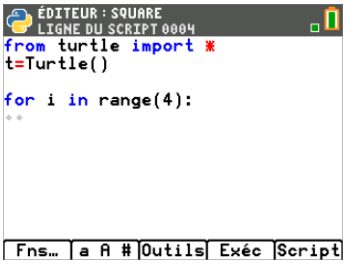
Notez que le curseur clignote à l'intérieur des parenthèses et qu'un retrait automatique est inséré sous l'instruction de la boucle "for".

- Entrez la valeur **4** entre les parenthèses. Cela détermine le nombre de cycles dans la boucle "for" qui est défini par l'indice i.







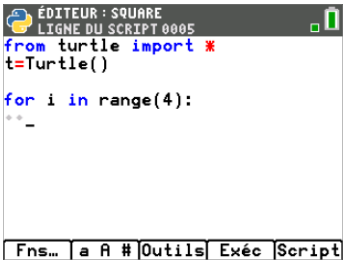


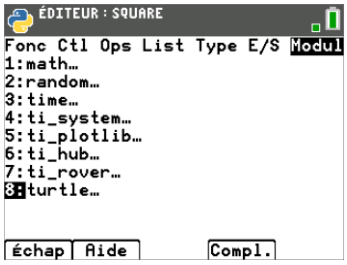
2) Saisissez une instruction pour faire avancer la tortue

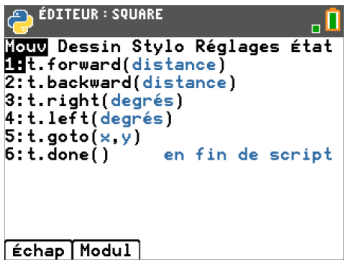
- Flèche vers le bas jusqu'à la ligne en retrait.
- Sélectionnez l'onglet [**Fns…**] (en bas de l'écran), puis flèche gauche vers l'onglet Modul (en haut de l'écran).
- Flèche vers le haut (ou le bas) pour sélectionner **8:turtle…**, puis appuyez sur [**entrer**] pour voir les sélections du menu du module Turtle.

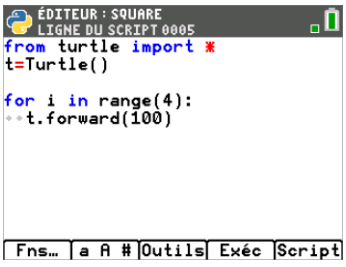
  Remarquez les onglets de menu et les sélections de menu dans le module Turtle. Nous allons commencer par sélectionner une méthode pour faire avancer la tortue.

- Sous l'onglet Mouv (Mouvement) avec le curseur sur **1:t.forward(distance)**, appuyez sur [**entrer**].
- Entrez la valeur **100** entre les parenthèses. C'est le nombre de pixels dont avancera la tortue.



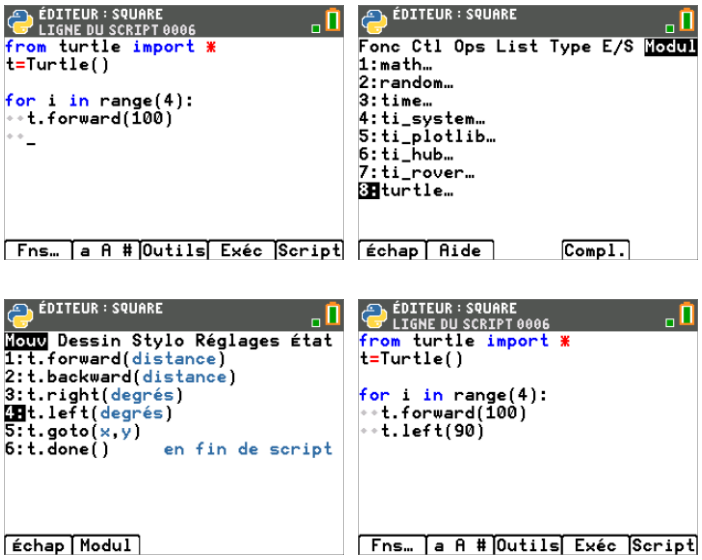






3) Entrez la direction et l'angle pour tourner

- À la fin de la ligne **t.forward(100)**, appuyez sur [**entrer**] pour créer une nouvelle ligne en retrait. (Ou bien [2nde] [entrer] pour passer à une nouvelle ligne).
- Sélectionnez l'onglet [**Fns…**] (en bas de l'écran), puis flèche gauche vers l'onglet Modul (en haut de l'écran).
- Flèche vers le haut (ou le bas) pour sélectionner **8:turtle…**, puis appuyez sur [**entrer**] pour voir les sélections du menu du module Turtle.
- Sous l'onglet Mouv avec le curseur sur **4:t.left(degrés)**, appuyez sur [**entrer**].
- Entrez, en degrés, l'angle dont vous voulez faire tourner la tortue vers la gauche. Pour un carré, on entrera **90** degrés.



4) Entrez une instruction "done" à la fin du script

Ainsi, le dessin Turtle reste sur votre écran lorsqu'il est terminé.

- À la fin de la ligne **t.left((90)** line, appuyez sur [**entrer**] pour créer une nouvelle ligne.
- Supprimez le retrait, en appuyant deux fois sur la touche [**suppr**]. (Ou bien, sélectionnez le menu [**Outils**], puis **2:Indent** ◂.)
- Revenir au menu du module Turtle comme précédemment.
- Sous l'onglet Mouv avec le curseur sur **6:t.done()**, appuyez sur [**entrer**].

Vous êtes prêt à exécuter votre premier programme Turtle !

5) Exécutez votre programme Turtle

Sélectionnez, tout simplement, l'onglet [**Exéc**] en bas de l'écran (en appuyant sur la touche [**trace**]).

Par défaut, la tortue est visible et vous la voyez dessiner. Notez également que, par défaut, la grille est visible, (l'échelle de la grille est de 25 pixels par carré).

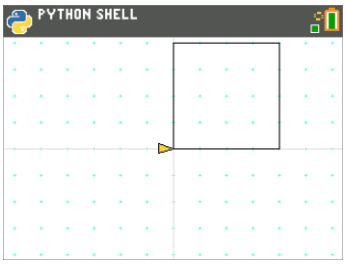
Lorsque vous avez fini d'admirer votre travail, appuyez sur la touche [**annul**] pour effacer l'écran. Remarquez que vous vous retrouvez sur un écran du Shell de Python. C'est là que votre script Python a été traité.

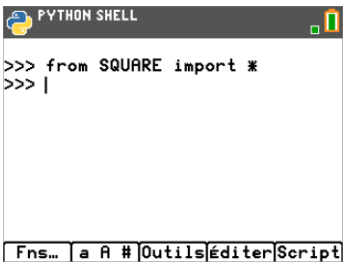
Retournez à l'éditeur en sélectionnant [**Éditer**] en bas de l'écran (appuyez sur la touche [**trace**]).

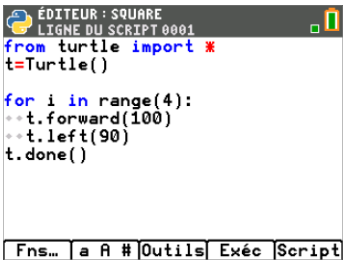
À partir de là, vous pouvez modifier votre script et l'exécuter de nouveau !

Challenges :

- Changer l'épaisseur du trait
- Changer la couleur du trait
- Cacher l'icône de la tortue
- Modifier la vitesse à laquelle la tortue se déplace
- Cacher la grille et l'indicateur d'échelle
- Remplir le carré





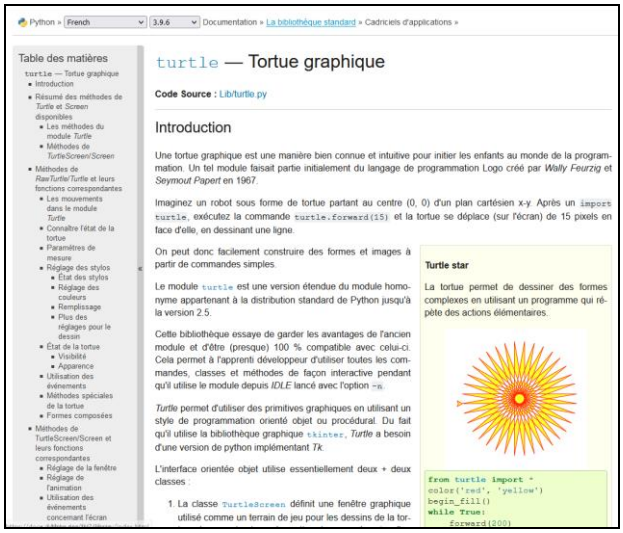


Méthodes du module Turtle

Tous les efforts ont été faits pour s'aligner sur la syntaxe et les comportements associés de l'API (Application Programming Interface) Turtle Graphics de Python dont on trouve la description sur le site Web de la documentation Python. Bien qu'il puisse y avoir de légères différences de comportement et de syntaxe dans la mise en œuvre, veuillez consulter ce site pour vous familiariser avec les définitions syntaxiques et les comportements de Turtle.

https://docs.python.org/fr/3/library/turtle.html?

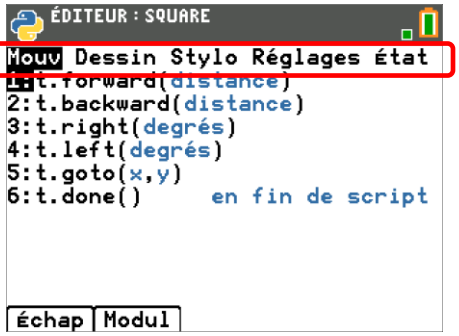
Veuillez consulter la section **Paramètres par défaut** et **exceptions connues** pour un aperçu des notes et des exceptions connues.



Sélections des menus du module

Voici les onglets des menus situés en haut de l'écran du module Turtle

- **Mouv** (Mouvement)
- **Dessin**
- **Stylo**
- **Réglages**
- **État**



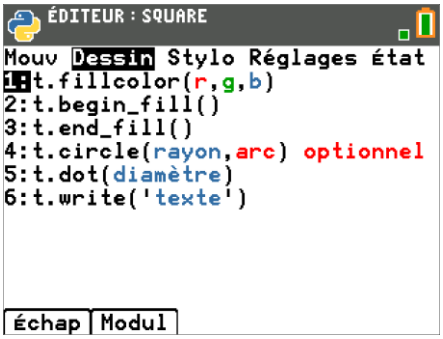
Mouv

- **t.forward(distance)** – indique la direction vers l'avant et la distance en pixels
- **t.backward(distance)** - indique la direction vers l'arrière et la distance en pixels
- **t.right(degrés)** – indique un virage à droite et l'angle en degrés.
- **t.left(degrés)** – indique un virage à gauche et l'angle en degrés.
- **t.goto(x,y)** – indique les coordonnées (x,y) du point vers lequel la tortue doit se déplacer.
- **t.done()** – utilisé à la fin du script pour afficher le dessin obtenu à l'aide de la tortue.

Le système de coordonnées Turtle est orienté avec (0,0) au centre de l'écran.

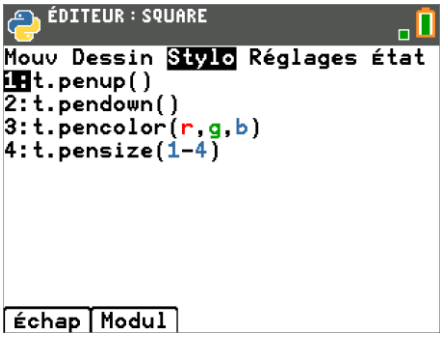
Dessin

- **t.fillcolor(r,g,b)** – indique une couleur de remplissage. RGB : **r**,**g**,**b** sont les arguments red, green et blue pour la couleur de remplissage, chacun des paramètres ayant une valeur comprise entre 0 et 255.
- **t.begin_fill()** – établit le moment où la méthode de remplissage commence.
- **t.end_fill()** – établit le moment où la méthode de remplissage se termine.
- **t.circle(rayon)** – dessine un cercle de rayon spécifié. Le centre du cercle est situé à un rayon à gauche de la tortue.
- **t.dot(diamètre)** – dessine un point de diamètre spécifié. Applique la couleur spécifiée à l'aide de t.pencolor(). Noir par défaut.
- **t.write('texte')** – écrit le texte spécifié dans la chaîne '**texte**'.



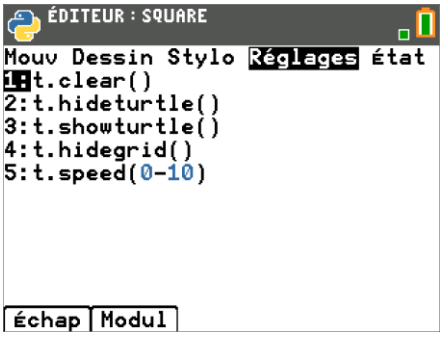
Stylo

- **t.penup()** – permet à la tortue de se déplacer sans tracer de ligne.
- **t.pendown()** – utilisé après t.penup(), permet à la tortue de dessiner une ligne.
- **t.pencolor(r,g,b)** – spécifie la couleur du stylo. RGB : **r**,**g**,**b** sont les arguments red, green et blue pour la couleur, chacun des paramètres ayant une valeur comprise entre 0 et 255.
- **t.pensize(1-4)** – change la taille du stylo. Quatre tailles de stylo : de 1 à 4. Valeur par défaut 1 lorsque t.pensize() n'est pas spécifié.



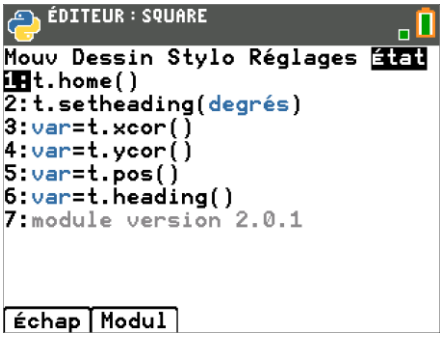
Réglages

- **t.clear()** – efface les lignes qui ont été tracées.
- **t.hideturtle()** – rend la tortue invisible.
- **t.showturtle()** – rend la tortue visible.
- **t.hidegrid()** – masque la grille.
- **t.speed(0-10)** – spécifie onze niveaux de vitesse de la tortue allant de 1 (lent) à 10 (rapide). La vitesse zéro (0) est le réglage le plus rapide. Valeur par défaut 5 lorsque t.speed() n'est pas spécifié.



État

- **t.home()** – déplace la tortue à l'origine (0,0) et fixe son cap à son orientation de départ à zéro (pointant vers la droite).
- **t.setheading(degrés)** – définit le cap de la tortue en degrés. Les valeurs positives sont dans le sens inverse des aiguilles d'une montre à partir du cap zéro. Les valeurs négatives sont dans le sens des aiguilles d'une montre.
- **var=t.xcor()** – Renvoie l'abscisse de la tortue.
- **var=t.ycor()** – Renvoie l'ordonnée de la tortue.
- **var=t.pos()** – Renvoie les coordonnées de la position actuelle de la tortue (x,y).
- **var=t.heading()** – Renvoie le cap actuel de la tortue.
- **module version 2.0.1** – Numéro de la version du module Turtle.



Paramètres par défaut et Exceptions	
Lorsque ces méthodes ne sont pas spécifiées dans un script, les valeurs par défaut suivantes sont appliquées :	
Turtle	**Par défaut : on**. Utilisez **t.hideturtle()** pour cacher l'icône de la tortue.
Grid	**Par défaut : on**. Utilisez **t.hidegrid()** pour cacher la grille. L'échelle de la grille est de 25 pixels par carré.
Speed	**Le réglage par défaut est 5 lorsqu'aucune vitesse n'est spécifiée**. Utilisez **t.speed()** pour changer la vitesse. Arguments valides de 0 à 10. Bien que les valeurs de 1 à 10 varient de lent à rapide, **speed(0)** donne le déplacement le plus rapide (conformément à l'API graphique Turtle).
Pen	**Par défaut : pensize = 1**. Pour ajuster la taille du stylo, utilisez **t.pensize()** avec comme argument 1, 2, 3 ou 4. **Par défaut : pencolor = black**. Utilisez **t.pencolor(**r,g,b**)** pour changer la couleur du trait. Les valeurs RGB (Red, Green, Blue) sont des arguments valides avec des valeurs comprises entre 0 et 255. **Par défaut : pen down**. Pour déplacer la tortue vers un emplacement sans tracer de ligne, utilisez la méthode **t.penup()**. Il faudra ensuite appliquer la méthode **t.pendow()** pour continuer à dessiner.
Fill	**Par défaut : color = black.** Utilisez **t.fillcolor(**r,g,b**)** pour spécifier une autre couleur. Les valeurs RGB (Red, Green, Blue) sont des arguments valides avec des valeurs comprises entre 0 et 255.

Exceptions

En raison des limitations de la taille de la mémoire du processeur Python, il se peut que vous rencontriez des erreurs de mémoire avec des programmes volumineux ou des programmes gourmands en ressources processeur. Si cela se produit, essayez ce qui suit :

a) n'importer que les fonctions nécessaires d'un module ; par exemple, **from random import randint** (qui importe uniquement la fonction **randint**) au lieu de **from random import *** (qui importe toutes les fonctions du module).

b) Ajuster l'étendue des valeurs utilisées dans un argument ; par exemple, **t.fillcolor(randint(150,255),0,0)** au lieu de **t.fillcolor(randint(0,255), randint(0,255), randint(0,255))**

c) Éliminer les parties du programme qui peuvent consommer une quantité excessive de mémoire. Dans l'exemple 4 (ci-dessous), nous supprimons les méthodes Fill du programme.

Exemples de Programmes:

Exemple 1: My First Star

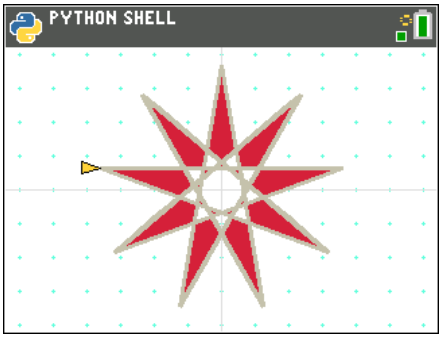
Dessine une seule étoile. Elle est tracée avec des lignes de couleur aléatoire et remplie avec une couleur aléatoire.

Notez que **t.pencolor()** peut être défini avec des valeurs R, G, B (Red, Green, Blue), valeurs comprises entre 0 et 255.

```
from random import *
from turtle import *
t=Turtle()
t.pensize(2)
t.penup()
t.goto(-90,16)
t.pendown()
t.pencolor(randint(0,255),randint(0,255),randint(0,255))
t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
t.begin_fill()
while True:
    t.forward(180)
    t.right(160)
    if t.heading() < 1:
        break
t.end_fill()
t.done()
```

Remarque : L'instruction **break** se trouve dans le catalogue. Appuyez sur [**2nde**] [**catalog**] et faites défiler jusqu'à la sélection **break**. **True** se trouve dans le menu de l'onglet Ops.

Nom du fichier : **STAR1**



Challenges :

- Changer l'angle de t.right() pour créer des étoiles de différentes formes. Que se passe-t-il quand l'angle est petit ?... grand ?
- Changer la distance de t.forward() pour obtenir une étoile plus grande, plus petite.

Exemple 2 : PhiX 177 Super Nova

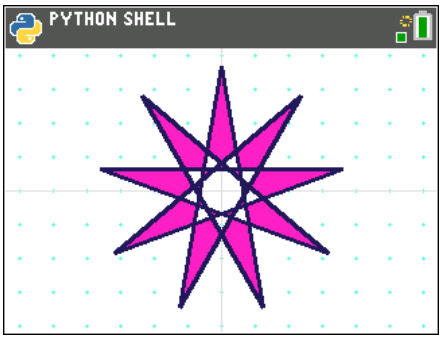
Développez le programme précédent en générant automatiquement une nouvelle étoile toutes les demi-secondes jusqu'à ce que vous appuyiez sur [annul].

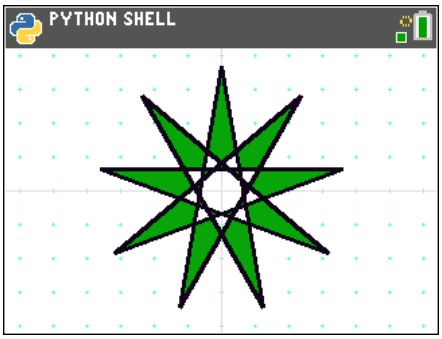
```
from time import *
from random import *
from turtle import *
t=Turtle()
t.hideturtle()
t.pensize(2)
t.speed(10)
while not escape():
  t.penup()
  t.goto(-90,16)
  t.pendown()
  t.pencolor(randint(0,255),randint(0,255),randint(0,255))
  t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
  t.begin_fill()
  while True:
    t.forward(180)
    t.right(160)
    h=t.heading()
    if h<1:
      break
  t.end_fill()
  sleep(1.5)
t.done()
```

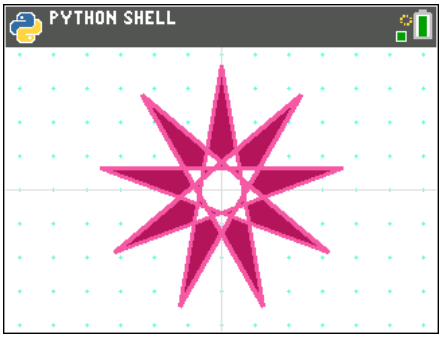
Remarque : Le module **time** est importé afin que la fonction **sleep()** puisse être utilisée pour mettre en pause chaque étoile pendant un moment avant que la suivante ne soit dessinée.

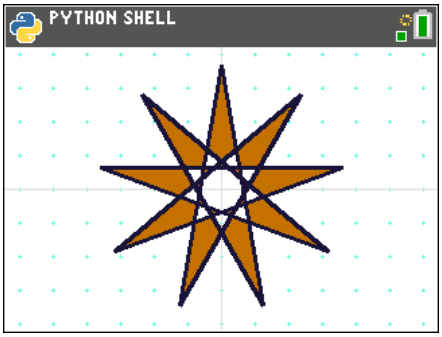
Remarque : L'instruction **while not escape():** se trouve dans le menu du module **ti_system…**. Lorsque vous utilisez le module **Turtle**, il n'est pas nécessaire d'importer le module **ti_system** pour utiliser uniquement **while not escape():**.
Par contre, l'utilisation d'autres fonctions du module **ti_system** nécessitera l'importation du module complet.

Nom du fichier : **STAR2**







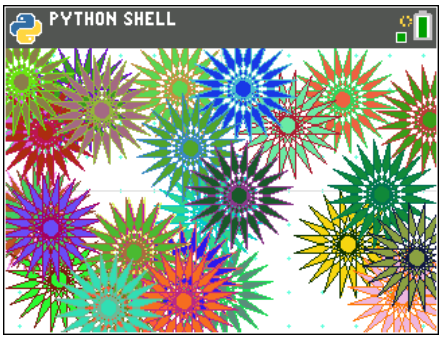


Exemple 3 : Zinnias in the Summer

S'appuyer sur les programmes précédents et rendre aléatoire la position des étoiles sur l'écran.

```
from random import randint
from turtle import *
t=Turtle()
t.pensize(1)
t.hideturtle()
t.speed(0)
while not escape():
    t.penup()
    t.goto(randint(-200,120), randint(-100,100))
    t.pendown()
    t.pencolor(randint(0,255),randint(0,255),randint(0,255))
    t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
    t.begin_fill()
    while True:
        t.forward(80)
        t.left(162)
        h=t.heading()
        if h<1:
            break
    t.end_fill()
    t.done()
```

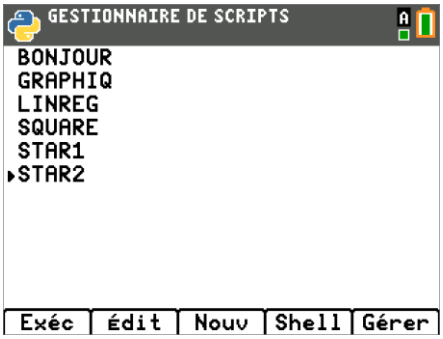
Nom du fichier : **STAR3**

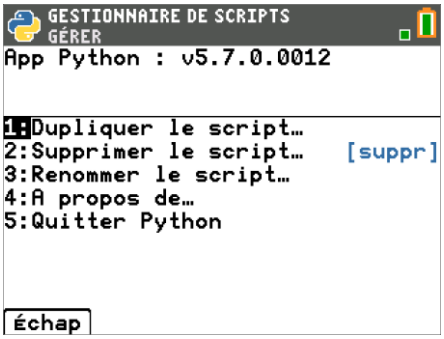


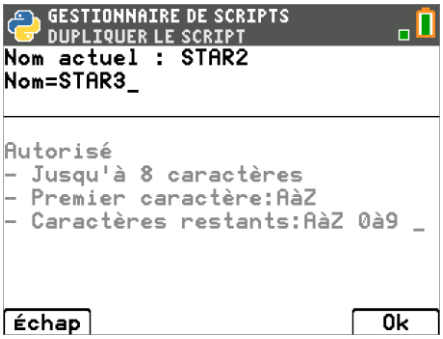
Challenges :

- À la place d'étoiles, tracer des carrés ou des rectangles
  - Randomiser la taille des carrés.
  - Randomiser la longueur et la largeur des rectangles.
- Adjuster la couleur du stylo et du remplissage afin d'obtenir des nuances de la même couleur.

Astuce : Lorsque vous voulez travailler sur un script existant, il peut être pratique d'en faire une copie, puis de modifier le code de la copie. Ceci peut être réalisé à partir du gestionnaire de scripts 1) en sélectionnant le script à copier (dans cet exemple, STAR2), puis 2) en sélectionnant l'onglet [**Gérer**], puis 3) en sélectionnant l'option **1:Dupliquer le script…** et enfin 4) en entrant le nom du nouveau script (dans cet exemple, STAR3).



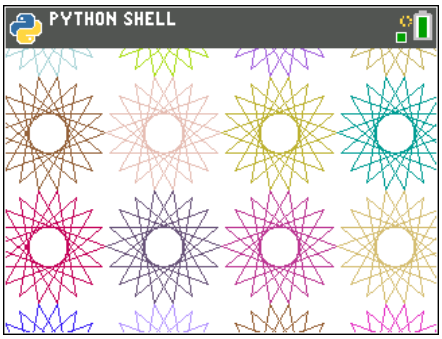




Exemple 4 : Grandma's Quilt
S'appuie sur les programmes précédents, mais dispose les étoiles dans un modèle ordonné de "couette".

```
from random import randint
from turtle import *
t=Turtle()
t.hideturtle()
t.hidegrid()
t.speed(0)
for j in range(112,-150,-84):
  for i in range(-169,170,86):
    t.penup()
    t.goto(i, j)
    t.pendown()
    t.pencolor(randint(0,255),randint(0,255),randint(0,255))
    while True:
      t.forward(80)
      t.left(140)
      h=t.heading()
      if h<1:
        break
t.done()
```

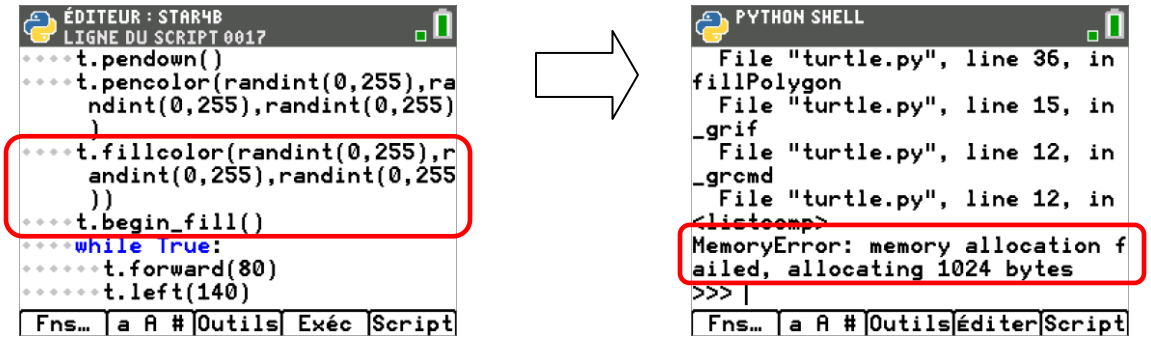
Nom du fichier : **STAR4**



Challenges :

- Ajuster le modèle de "couette" afin que les étoiles se chevauchent verticalement et horizontalement.
- Changer le nombre de points sur l'étoile.
- Tracer des cercles ou des carrés à la place d'étoiles.

Remarque : notez que dans cet exemple, nous n'utilisons pas de méthode Fill. Cela est dû au fait que le processeur Python de la TI-83 Premium CE Edition Python dispose d'une mémoire limitée. Si un programme dépasse la mémoire disponible dans le processeur Python, une erreur de mémoire peut se produire, comme illustré ci-dessous lorsqu'une méthode Fill est utilisée dans l'exemple 4. Si une erreur de mémoire se produit, modifiez votre programme pour optimiser (minimiser) la taille du programme. Ici, nous avons supprimé les méthodes Fill. Dans d'autres cas, il peut s'agir d'importer uniquement les fonctions d'un module dont on a besoin, par exemple **from random import randint** (qui importe uniquement la fonction **randint**) au lieu de **from random import *** (qui importe toutes les fonctions du module).

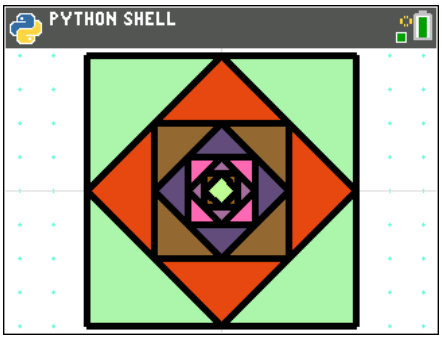


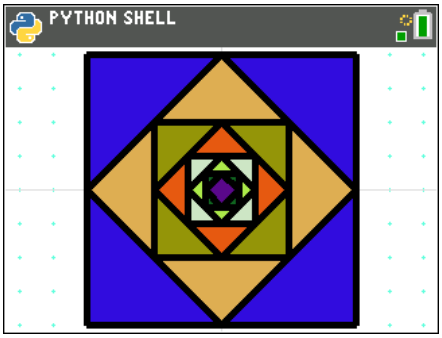
Exemple 5 : Stained Glass

Construit une série de carrés inclus les uns dans les autres.

```
from random import randint
from ti_system import escape
from math import sqrt
from time import sleep
from turtle import *
t=Turtle()
t.speed(0)
t.pensize(3)
t.hideturtle()
while not escape():
  d=200
  t.penup()
  t.goto(-d/2,-d/2)
  t.setheading(0)
  t.pendown()
  for i in range(8):
    t.fillcolor(randint(0,255),randint(0,255),randint(0,255))
    t.begin_fill()
    for j in range(4):
      t.forward(d)
      t.left(90)
    t.end_fill()
    t.penup()
    t.forward(d/2)
    t.left(45)
    t.pendown()
    d=sqrt((d**2)+(d**2))/2
  sleep(1.5)
t.done()
```

Nom du fichier : **SQRLOOP**





Challenges :

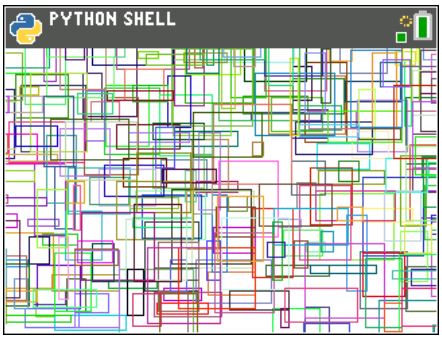
- Augmenter ou diminuer le nombre de carrés dessinés.
- Changer l'angle d'orientation des carrés dessinés.
- Créer un pavage avec des carreaux plus petits qui ressemble au dessin original.

Exemple 6 : Threadbare

Toujours à l'aide d'une stratégie de boucle classique, dessiner un tableau aléatoire de rectangles. Le tracé continue jusqu'à ce que vous appuyiez sur [**annul**].

```
from random import randint
from ti_system import escape
from turtle import *
t=Turtle()
t.hidegrid()
t.hideturtle()
t.speed(0)
while not escape():
  t.penup()
  t.goto(randint(-200,150), randint(-110,100))
  t.pendown()
  t.pencolor(randint(0,255), randint(0,255), randint(0,255))
  b=randint(5,60)
  h=randint(5,60)
  for i in range(2):
    t.forward(b)
    t.left(90)
    t.forward(h)
    t.left(90)
t.done()
```

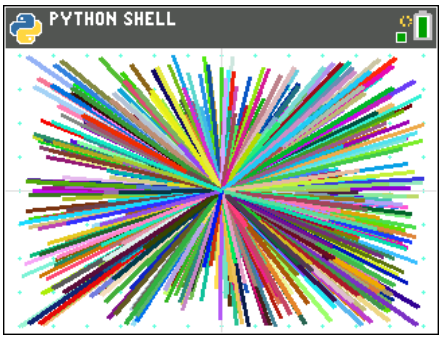
Nom du fichier : **RANDREC**



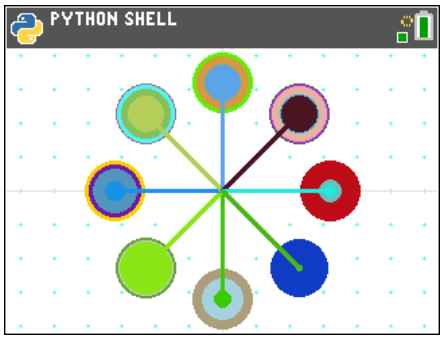
Challenges :

- Remplir les rectangles.
- Tracer des carrés à la place des rectangles.
- Ajustez la taille des rectangles ou des carrés.
- Ajustez la couleur pour qu'elle soit plus uniforme ; composée de nuances de la même couleur.

Défi : écrivez votre propre code pour créer les deux images ci-dessous.



Nom du fichier : **SPLAT1**

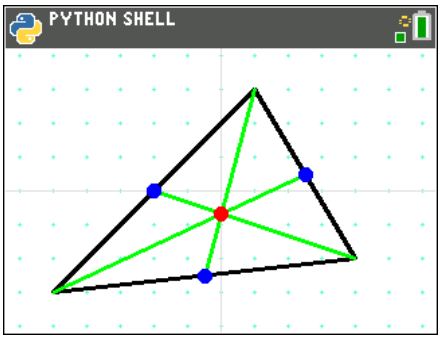


Nom du fichier : **SPLAT2**

Exemple 7 : Centroid
Construit un triangle, ses médianes et son centre de gravité.

```
from turtle import *
t=Turtle()
t.speed(1)
t.hideturtle()
t.pensize(2)

# calcule le milieu d'un segment
def midpoint(pt1,pt2):
return ((pt1[0] + pt2[0])/2, (pt1[1] + pt2[1])/2)

# tracé d'un point
def plot_point(pt):
t.penup()
t.goto(pt)
t.pendown()
t.dot(10)

# les sommets du triangle
v1 = (25,75)
v2 = (-125,-75)
v3 = (100,-50)

# calcule le centre de gravité
centroid =((v1[0]+v2[0]+v3[0])/3,(v1[1]+v2[1]+v3[1])/3)

# calcule les milieux des côtés
mid_1_2 = midpoint(v1,v2)
mid_2_3 = midpoint(v2,v3)
mid_1_3 = midpoint(v1,v3)

# trace le triangle en noir
t.penup()
t.goto(v1)
t.pendown()
t.goto(v2)
t.goto(v3)
t.goto(v1)

# trace les trois médianes en vert
t.pencolor(0,255,0)

t.penup()
t.goto(mid_1_2)
t.pendown()
t.goto(v3)

t.penup()
t.goto(mid_2_3)
t.pendown()
t.goto(v1)

t.penup()
t.goto(mid_1_3)
t.pendown()
t.goto(v2)

# trace le centre de gravité en rouge
t.pencolor(255,0,0)
plot_point(centroid)

# trace les milieux des côtés en bleu
t.pencolor(0,0,255)
plot_point(mid_1_2)
plot_point(mid_2_3)
plot_point(mid_1_3)

t.done()
```

Nom du fichier : **CENTROID**



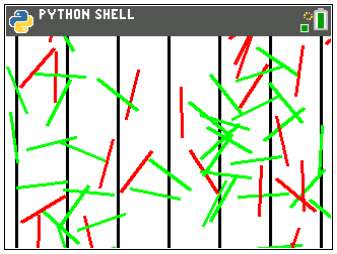
Challenge :

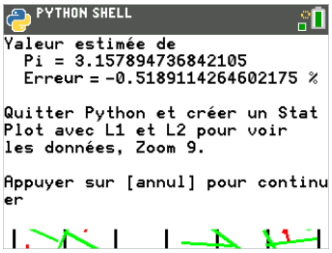
Changer les coordonnées d'un ou plusieurs sommet(s) et observer le résultat.

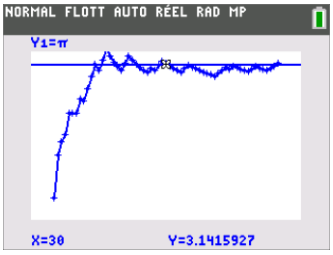
Exemple 8 : Buffon's Needle
Simule l'expérience de l'aiguille de Buffon pour estimer la valeur de pi.

```
from ti_system import *
from random import *

# utilisez le Shell avant d'entrer dans l'environnement Turtle
disp_clr()
length=int(input("Longueur d'aiguille ou [entrer] pour 50 ?") or '50')
spacing=int(input("Espace entre les lignes ou [entrer] pour 50 ?") or '50')
needles=int(input("Nombre d'aiguilles ou [entrer] pour 60 ?") or '60')
the_pies=[]
the_count=[]
# configuration de l'environnement Turtle
from turtle import *; t=Turtle()
t.hidegrid()
t.hideturtle()
t.pensize(2)
t.speed(0)
w,h=320,210
the_lines=[] # liste des abscisses de chaque ligne
n_lines=int((w/spacing)/2)+2 # nombre de lignes
crossing = 0 # aiguilles rencontrant une ligne
estimate = 0 # valeur estimée calculée de pi
# tracé des lignes et ajout à the_lines
for x in range(-n_lines*spacing,(n_lines+1)*spacing,spacing):
  the_lines.append(x)
  t.penup()
  t.goto(x,-h/2)
  t.pendown()
  t.goto(x,h/2)
# dessine les aiguilles
for i in range(needles+1):
  x1=randint(0,w)-w//2
  y1=randint(0,h)-h//2
  angle=random()*360
  t.penup()
  t.goto(x1,y1)
  t.setheading(angle)
  t.forward(length)
  x2= t.xcor()
  y2= t.ycor()
  t.pencolor(255,0,0)
# teste si une aiguille touche une ligne
  for n in range(len(the_lines)):
    if((x1<=the_lines[n] and x2>=the_lines[n]) or (x2<=the_lines[n] and x1>=the_lines[n])):
      t.pencolor(0,255,0)
      crossing+=1
  t.pendown()
  t.goto(x1,y1)
# formule de Buffon
  try:
estimate=(2*length*i)/(crossing*spacing)
  except:
# ça marche jusqu'à ce que l'on tombe sur une division par zéro…
    pass
  the_pies.append(estimate)
  the_count.append(i)
store_list("1",the_count)
store_list("2",the_pies)
error=(pi-estimate)*100/pi
sleep(2)
disp_at(7,"    Appuyez sur [annul] pour continuer","left")
disp_wait()
disp_clr()
print("Valeur estimée de Pi = ", estimate,"\n  Erreur =",error,"%\n")
print("Quitter Python et créer un Stat Plot avec L1 et L2 pour voir \nles données, Zoom 9.")
print("\nAppuyer sur [annul] pour continuer")
t.done()
```

Nom du fichier : **BUFFON**







Quittez l'application Python. Faites un tracé statistique avec L1 et L2. ZoomStat et Trace.

Challenge :

Exécutez la simulation en spécifiant différentes longueurs d'aiguilles et différents espacements entre les lignes.

Note : en raison des limitations de la mémoire, il est possible que des erreurs de mémoire apparaissent, même avec les entrées par défaut.