ACTIVITÉ 6: CRÉER DES IMAGES AVEC LE MODULE TURTLE

Problématique: Comment utiliser les notions de programmation étudiées pour créer des dessins avec Python?

Objectifs: *À la fin de l'activité, vous serez capable de :*

- *Utiliser le module turtle pour réaliser des dessins.*
- *Réinvestir les variables, les boucles et les instructions conditionnelles.*
- *Créer et utiliser des procédures pour simplifier un programme.*
- *Réaliser des dessins de plus en plus complexes en combinant les différentes notions de programmation étudiées.*

Dans cette activité, nous allons utiliser le module turtle de Python pour revoir les différentes notions de programmation étudiées dans les activités précédentes.

À travers la réalisation de dessins géométriques de plus en plus complexes, vous utiliserez les variables, les boucles, les conditions et les procédures.

L'objectif final sera de réaliser des dessins de plus en plus complexes en combinant ces différentes notions.

Qu'est-ce que "turtle" ?

En Python, turtle est un module qui permet de réaliser des dessins à l'aide d'instructions de programmation.

Son fonctionnement est simple :

- On imagine une « tortue » qui se déplace sur l'écran en suivant les instructions du programme.
- La tortue possède un « crayon » : lorsqu'elle avance, elle trace une ligne. Si on relève le crayon, elle se déplace sans dessiner.
- Grâce à des instructions comme `forward()`, `right()` ou `color()`, on peut dessiner des figures géométriques, créer des motifs et même réaliser de petites animations.

Le principe ressemble à celui de Scratch, utilisé au collège, mais les instructions sont désormais écrites directement en Python.

Interpréteur Python:

Pour réaliser cette activité, nous utiliserons Trinket (https://trinket.strivemath.org), qui permet d'exécuter des programmes Python avec le module turtle.

Pour utiliser ce module, il faut commencer chaque programme par l'instruction :

```
from turtle import *
```

Lorsque vous écrivez un programme, la coloration syntaxique permet de distinguer les différents éléments du code.

En cas d'erreur, lisez attentivement le message affiché, repérez la ligne concernée et essayez de corriger votre programme avant de le relancer.

Conseil : Exécutez régulièrement votre programme pour vérifier son fonctionnement, plutôt que d'attendre d'avoir écrit toutes les instructions.

1. MEMENTO DE TURTLE

Voici les principales instructions qui peuvent être données :

`forward(distance)`	Avancer d'une distance donnée	`color(couleur)`	Couleur du tracé. <couleur> peut être une chaîne prédéfinie ('red', 'blue', 'green', etc.)
`backward(distance)`	Reculer d'une distance donnée	`width(épaisseur)`	Choisir l'épaisseur du tracé
`left(angle)`	Tourner à gauche d'un angle donné (en degrés).	`write(texte)`	Écrire un texte à la position de la tortue.
`right(angle)`	Tourner à droite d'un angle donné (en degrés).	`circle(rayon)`	Tracer un cercle de rayon donné.
`goto(x, y)`	Se déplacer jusqu'au point de coordonnées (x ; y).	`reset()`	Effacer le dessin et réinitialiser la tortue.

`up()`	Relever le crayon pour se déplacer sans tracer.	`begin_fill()`
		`# instructions`
		`end_fill()`
			Remplir un contour fermé. La première instruction commence le remplissage. La seconde la termine.

`down()`	Abaisser le crayon pour recommencer à tracer.		

Remarque: La fenêtre graphique possède un repère : le point (0 ; 0) est situé au centre, l'axe des abscisses est horizontal et celui des ordonnées est vertical.

Structure d'un programme Turtle
Pour utiliser le module *turtle*, on commence par importer ses instructions.

Chaque programme suivra la structure suivante :

```
from turtle import *

reset()
# Votre programme ici
done()
```

L'instruction `done()` se place à la fin du programme pour maintenir la fenêtre graphique ouverte dans les environnements qui le nécessitent.

Enregistrement de vos programmes
Pour conserver les programmes réalisés au cours de cette activité, vous utiliserez un document Google Docs.

1. Ouvrir un nouveau document Google Docs et le nommer « **SNT Activité 6** ».
2. Pour chaque exercice, selectionner tout votre code Python depuis Trinket (**Ctrl + A**), copier le (**Ctrl + C**) et le coller dans votre document (**Ctrl + V**).
3. Donner un titre à chaque programme, correspondant au nom indiqué dans l'exercice : **turtle01.py**, **turtle02.py, turtle03a.py**, etc.
4. Placer ce titre juste au-dessus du code correspondant.

Attention : Tous les programmes doivent être conservés dans un seul document Google Docs, en respectant l'ordre des exercices.

Lors de mon passage à votre poste, vous devrez être capables de me présenter le code de vos programmes et de me montrer leur fonctionnement.

2. SUCCESSION D'INSTRUCTIONS ET VARIABLES

Pour donner plusieurs instructions successives à la tortue, il suffit de les écrire les unes à la suite des autres, en allant à la ligne après chaque instruction.

Exercice 1: Tracer un carré

Voici un programme Python :

```
from turtle import *

reset()

forward(100)
right(90)
forward(100)

done()
```

a. Recopier ce programme dans un programme nommé **turtle01.py,** puis l'exécuter.

b. Compléter le programme pour que la tortue dessine un carré.

c. Modifier le programme pour que le carré soit tracé en rouge (aidez-vous du mémento).

À cocher une fois l'exercice terminé: ☐

On souhaite maintenant pouvoir tracer un carré plus grand ou plus petit.

Dans le programme précédent, il faudrait modifier la valeur 100 à chaque endroit où elle apparaît. Pour éviter ces répétitions, il est plus pratique d'introduire une variable appelée `taille`, dont il suffira de modifier la valeur une seule fois.

Par exemple :

```
taille = 100

forward(taille)
right(90)
```

Exercice 2: Utiliser une variable

a. Copier le programme **turtle01.py** en introduisant une variable taille, puis vérifier qu'il fonctionne.

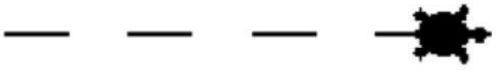
b. Modifier le programme pour qu'il demande à l'utilisateur de saisir un nombre entier qui déterminera la longueur du côté du carré.

c. Enregistrer ce programme au nom **turtle02.py**

À cocher une fois l'exercice terminé: ☐

3. BOUCLES "POUR"

Exercice 3a: Tracer une ligne en pointillés



À l'aide des instructions `up()` et `down()` du mémento, écrire un programme **turtle03a.py** qui reproduit la figure ci-contre.

Vous utiliserez une variable taille pour définir la longueur des pointillés (`taille = 7`).

À cocher une fois l'exercice terminé: ☐

Pour tracer une ligne en pointillés, on répète plusieurs fois les mêmes instructions. Si l'on souhaite tracer une ligne plus longue, il devient fastidieux de les recopier.

Pour simplifier le programme, on peut utiliser une boucle `for` :

```
from turtle import *

reset()

taille = 7

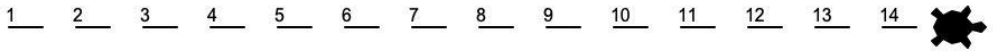
for i in range(1, 15):
    write(i)
    forward(taille)
    up()
    forward(taille)
    down()

done()
```

Les instructions indentées sont répétées 14 fois. La variable i prend successivement les valeurs entières de 1 à 14.

La commande `write(i)` permet d'afficher la valeur de i à la position de la tortue.

Exercice 3b: Recopier le programme ci-dessus dans un fichier nommé **turtle03b.py**, puis l'exécuter. Vérifier que la figure obtenue correspond à celle ci-dessous:



À cocher une fois l'exercice terminé: ☐

Exercice 4: Comprendre l'instruction `range()`

a. Copier le programme **turtle03b.py** dans un nouveau fichier nommé **turtle04.py.**

b. Remplacer `range(1, 15)` par `range(1, 15, 2)`, puis exécuter le programme.

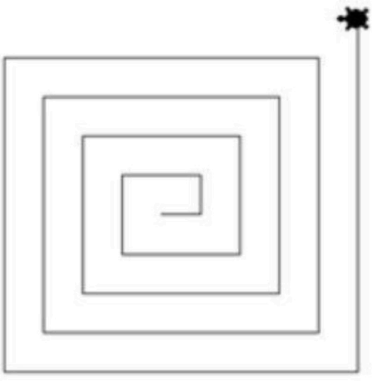
c. Comparer les deux résultats. Quel est le rôle du troisième nombre dans `range(1, 15, 2)` ?

.....

À cocher une fois l'exercice terminé: ☐

Exercice 5: Tracer une spirale carrée

Écrire un programme **turtle05.py** qui reproduit la figure ci-contre.



Pour cela :

- Utiliser une variable `taille = 10`.
- Les deux premiers segments doivent mesurer `taille`, les deux suivants 2 × `taille`, puis les deux suivants 3 × `taille`, et ainsi de suite.
- Utiliser une boucle for et la variable de boucle i pour faire varier la longueur des segments.

Indication : Vous pourrez utiliser l'instruction `forward(i * taille)`.

À cocher une fois l'exercice terminé: ☐

Exercice 6: Tracer un carré avec une boucle

Reprendre le programme de l'exercice 2 qui permet de tracer un carré.

Vous avez remarqué que certaines instructions sont répétées quatre fois.

Écrire un nouveau programme **turtle06.py** qui trace le même carré en utilisant une boucle `for`.

À retenir : Une boucle for permet de répéter plusieurs fois un ensemble d'instructions. Il n'est pas nécessaire d'utiliser la variable de boucle à l'intérieur des instructions répétées.

À cocher une fois l'exercice terminé: ☐

4. PROCÉDURES ET FONCTIONS

Chaque fois que l'on souhaite tracer un carré, il serait fastidieux de réécrire toutes les instructions nécessaires.

Pour éviter ces répétitions, on peut créer une procédure qui regroupe plusieurs instructions sous un même nom. Il suffit ensuite d'appeler cette procédure pour exécuter les instructions qu'elle contient.

Par exemple, voici la structure d'une procédure `carre(longueur)` que vous devrez compléter dans l'exercice 7.

```
def carre(longueur):

    # Instructions permettant de tracer un carré
    # à créer dans l'exercice 7

carre(100)  # Trace un carré de côté 100
```

Rappels :

- Une procédure est définie à l'aide du mot-clé `def`.
- Les instructions appartenant à la procédure doivent être indentées.
- Une procédure peut posséder un ou plusieurs paramètres.
- Pour exécuter une procédure, il faut l'appeler en indiquant les valeurs des paramètres.
- Le symbole # permet d'ajouter un commentaire qui n'est pas exécuté par Python.

À retenir : Contrairement à une procédure, une fonction renvoie une valeur à l'aide de l'instruction `return`.

Exercice 7: Tracer un carré avec une procédure

Écrire un programme **turtle07.py** qui:

a. Définit une procédure `carre(longueur)` permettant de tracer un carré dont le côté mesure longueur.

b. Appelle cette procédure pour tracer un carré de côté 30.

c. Modifie le programme pour tracer successivement trois carrés de côtés 20, 50 et 100.

À cocher une fois l'exercice terminé: ☐

Exercice 8: Tracer des triangles
Copier le programme **turtle07.py** dans un nouveau fichier nommé **turtle08.py**.

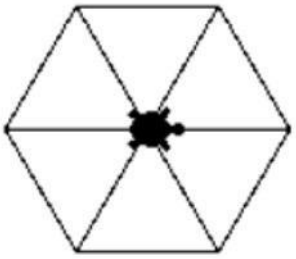


a. Ajouter une procédure triangle(taille) permettant de tracer un triangle équilatéral dont le côté mesure taille.

b. À l'aide des procédures `carre(longueur)` et `triangle(taille)`, d'une boucle `for` et des instructions `up()` et `down()`, reproduire la figure ci-dessus.

À cocher une fois l'exercice terminé: ☐

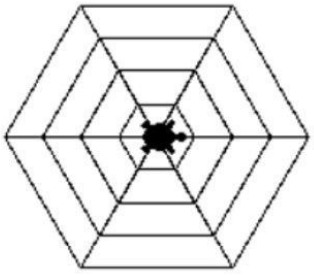
Exercice 9: Construire un hexagone
Écrire un programme nommé **turtle09.py** qui reproduit la figure ci-contre en utilisant :



- La procédure `triangle(taille)` créée précédemment.
- Une boucle for permettant de répéter le tracé des triangles.

Indication : Un tour complet correspond à 360°.

À cocher une fois l'exercice terminé: ☐



BONUS Défi: Une toile d'araignée
Écrire un programme nommé **turtle10.py** qui reproduit la toile d'araignée ci-contre.

Pour cela, vous utiliserez :

- La procédure `triangle(taille)` créée précédemment.
- Deux boucles for imbriquées, c'est-à-dire une boucle placée à l'intérieur d'une autre.

Indication : Utilisez deux variables différentes pour les boucles (par exemple i et j).

À cocher une fois l'exercice terminé: ☐

5. INSTRUCTIONS D'ENTRÉE-SORTIE

L'instruction `print()` permet d'afficher du texte ou le résultat d'un calcul dans la console Python.
Exemple :

```
print("La somme de 2 et 3 vaut : ", 2 + 3)
```

Remarque : Avec le module turtle, l'instruction `write()` permet d'afficher du texte directement dans la fenêtre graphique.

La fonction `input()` permet de demander à l'utilisateur de saisir une information.
Exemples :

```
age = int(input("Entrez votre âge : "))
nombre = float(input("donner un réel : "))
nom = input("Quel est votre nom ? ")
```

À retenir : La fonction `input()` renvoie toujours une chaîne de caractères (`str`). Pour effectuer des calculs avec la valeur saisie, il faut la convertir en nombre à l'aide de `int()` ou `float()`.

6. TESTS DE CONDITIONS

En Python, les instructions conditionnelles permettent d'exécuter différentes instructions selon qu'une condition est vraie ou fausse.

La structure générale est la suivante :

```
if <condition> :
    # Instructions si la condition est vraie

else:

    # Instructions si la condition est fausse
```

Rappel des opérateurs de comparaison :

< : strictement inférieur	> : strictement supérieur	<= : inférieur ou égal	>= : supérieur ou égal	== : égal	!= : différent de

On peut également combiner plusieurs conditions avec les opérateurs logiques `and` (et) et `or` (ou).

Exemple:

```
a = float(input("Donner la valeur de a : "))
if a > 0 :
    print("a est supérieur à 0.")
else :
    print("a est inférieur ou égal à 0.")
```

Exercice 11: Dessiner un polygone régulier

Écrire un programme nommé **turtle11.py** qui permet de dessiner un polygone régulier en fonction du nombre de côtés choisi par l'utilisateur.

Pour cela :

a. Demander à l'utilisateur de saisir un nombre entier n correspondant au nombre de côtés du polygone.

b. Si n est inférieur à 3, afficher le message *« Impossible de tracer un polygone »*.

c. Sinon, tracer un polygone régulier de n côtés, chaque côté mesurant 50 pixels. Vous utiliserez une boucle `for`.

d. Modifier le programme pour que le polygone soit tracé :
   - en rouge si le nombre de côtés est pair ;
   - en bleu si le nombre de côtés est impair.

Indications :

- Pour tracer un polygone régulier à n côtés, la tortue doit tourner de 360/n degrés après chaque côté.
- L'opérateur % permet de déterminer le reste d'une division entière.

À cocher une fois l'exercice terminé: ☐

7. BOUCLES "TANT QUE"

Une boucle `while` permet de répéter un ensemble d'instructions tant qu'une condition est vraie.
Sa syntaxe générale est la suivante :

```
while condition:
    # Instructions à répéter
```

Exemple:
Le programme suivant demande à l'utilisateur de saisir un nombre supérieur ou égal à 10. Tant que cette condition n'est pas respectée, il lui demande de saisir un nouveau nombre.

```
a = 0
while a < 10:
    a = float(input("Donner un nombre supérieur ou égal à 10 : "))
```

Attention : Il faut veiller à ce que la condition puisse devenir fausse, sinon la boucle risque de se répéter indéfiniment.

Exercice 12: Une tortue qui se promène au hasard
Écrire un programme nommé **turtle12.py** qui simule les déplacements aléatoires d'une tortue à l'intérieur d'un cercle.

Pour cela :

a. Définir deux variables :
   - `rayon = 100` : le rayon du cercle ;
   - `lm = 5` : la longueur d'un déplacement élémentaire.

b. Tracer un cercle de rayon `rayon`, centré à l'origine du repère (0 ; 0).

c. Placer la tortue au centre du cercle.

d. À chaque déplacement, la tortue avance d'une distance lm, puis choisit au hasard l'une des quatre possibilités suivantes :

- continuer tout droit ;
- tourner à droite de 90° ;
- tourner à gauche de 90° ;
- faire demi-tour.

Pour effectuer ce choix, utiliser l'instruction :
`i = randint(1, 4)`

Cette instruction génère un nombre entier aléatoire compris entre 1 et 4. Pour l'utiliser, ajouter au début du programme :

```
from random import *
```

e. Utiliser une boucle while pour que la tortue continue à se déplacer tant qu'elle se trouve à l'intérieur du cercle.

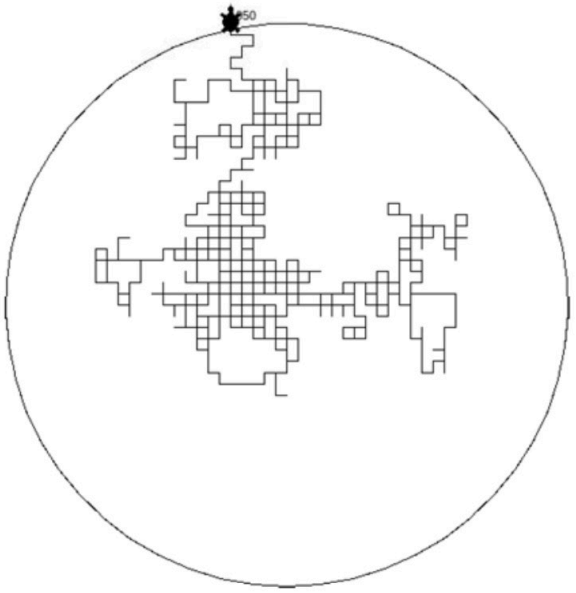
Indication : L'instruction `distance(0, 0)` renvoie la distance entre la tortue et l'origine du repère.

Vous pourrez donc utiliser une boucle de la forme :

```
while distance(0, 0) < rayon:

    # Instructions de déplacement
```

Vous devriez obtenir ce genre de figure.



À <u>cocher</u> une fois l'exercice terminé: ☐