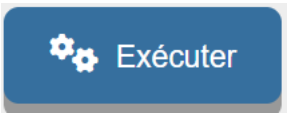
Institut national supérieur du professorat et de l'éducation
Académie de Paris

DU 2nd degré - Mathématiques
Année 2025-2026
Katia.Odiot@inspe-paris.fr

TP1bis : Géométrie avec Python

Console Python via Capytale	Exécuter	Console	Vue graphique
			

Partie B - Module turtle, boucles bornées et fonctions sans retour : 2267-10265772

1. Ouvrir un éditeur Python, recopier le programme ci-contre, l'exécuter puis appeler la fonction **dessin()** dans la console.
   Avec EduPython, remplacer la fonction done() par exitonclick() permettant de fermer la fenêtre graphique en cliquant sur la zone de dessin.

2. On souhaite identifier l'effet des principales instructions de ce programme en les désactivant l'une après l'autre. Pour cela, il suffit d'insérer le caractère « # » en début de ligne.
   En Python, ce caractère indique le début d'un commentaire. Le texte qui suit sur la même ligne n'est pas interprété.

```
from turtle import *

def carre():
    down()
    begin_fill()
    for i in range(4):
        forward(50)
        right(90)
    end_fill()
    up()

def ligne():
    for i in range(3):
        carre()
        forward(60)

def dessin():
    color("red")
    x = 0
    y = 0
    for i in range(3):
        goto(x, y)
        ligne()
        y = y - 60
    done()
```

Désactiver tour à tour chacune des lignes ci-dessous (une seule à la fois), puis exécuter le programme à nouveau pour vérifier l'effet obtenu.

- **a)** color("red")
- **b)** down()
- **c)** forward(50)
- **d)** right(90)
- **e)** goto(x,y)
- **f)** begin_fill() et end_fill()
- **g)** **from** turtle **import** *
- **h)** exitonclick()

3. Pour mieux comprendre chaque instruction, on va maintenant modifier certaines lignes du programme. Effectuer tour à tour chacune des modifications ci-dessous, exécuter le programme et appeler la fonction **dessin()** dans la console pour observer l'effet obtenu.
   - **a)** Remplacez color("red") par color("blue") ;
   - **b)** Remplacez forward(50) par forward(100) ;
   - **c)** Remplacez right(90) par right(60) ;
   - **d)** Remplacez 4 par une autre valeur dans **for** i **in** range(4) ;
   - **e)** Remplacez 3 par une autre valeur dans **for** i **in** range(3) ;
   - **f)** Modifiez les valeurs initiales des variables x et y.

4. Indiquez dans le tableau ci-dessous le sens de chaque instruction :

Instruction	Description
color("red")	
down()	
forward(50)	
right(90)	
up()	
goto(x,y)	
from turtle **import** *	
exitonclick()	
for i **in** range(4):	

5. On va maintenant ajouter des paramètres à la fonction **carre()**.

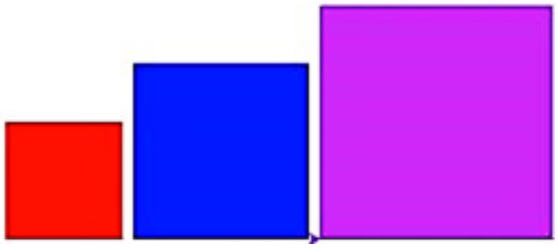
a) Modifier le programme précédent pour tracer un unique carré vert de côté 100.

b) Modifier la fonction **carre()** :

```
def carre(c, couleur) :
    fillcolor(couleur)
    down()
    begin_fill()
    for i in range(4) :
        forward(c)
        left(90)
    end_fill()
    up()
```

c) Modifier les autres fonctions pour fournir deux arguments à la nouvelle fonction **carre()**. On écrira donc : **carre(100,'green')**.

d) Enfin, modifier le programme pour obtenir la figure ci-dessous :



Partie C - Structures, fonctions à valeur de retour : 1497-10265923

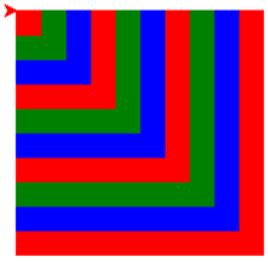
On considère le programme ci-contre.

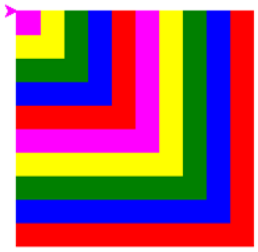
```
from turtle import *

def carre(c, teinte):
    color(teinte)
    down()
    begin_fill()
    for i in range(4):
        forward(c)
        right(90)
    end_fill()
    up()

def couleur(n):
    if n == 0:
        return "red"
    elif n == 1:
        return "blue"
    else:
        return "green"

def carres_emboites(n):
    up()
    c = 200
    pas = 20
    i_couleur = 0
    while c > pas:
        teinte = couleur(i_couleur%3)
        carre(c, teinte)
        i_couleur = i_couleur + 1
        c = c - pas
    done()
```

1. Recopier et exécuter le programme avec n = 10.

2. Que se passerait-il si on supprimait la ligne « c = c - pas » de ce programme ?
   Tentez de répondre à cette question sans exécuter le programme.

3. Modifier le programme pour obtenir successivement chacun des deux dessins suivants :

a) 

b) 

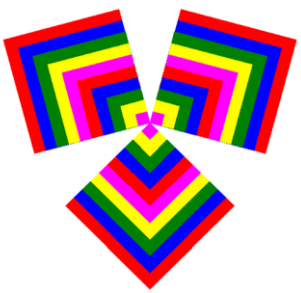
Indications :

- *L'opérateur « % » permet de calculer le reste d'une division euclidienne.*
- *La suite des couleurs utilisées dans le dessin **b)** est (de bas en haut) : red, blue, green, yellow, magenta (répétée deux fois).*

4. En programmation, on évite le plus possible de se répéter. C'est pourquoi, dès que l'on est amené à utiliser plusieurs fois la même séquence d'instructions, on l'extrait pour en faire une fonction.

a) Modifier la fonction **carres_emboites(n)** pour lui fournir deux paramètres : **c_max**, taille du grand carré, et **pas**, la différence de taille des côtés entre 2 carrés successifs :
def carres_emboites(c_max,pas): …

b) Écrire une nouvelle fonction permettant d'obtenir le motif ci-contre en faisant appel à la fonction **carres_emboites(c_max,pas)**.

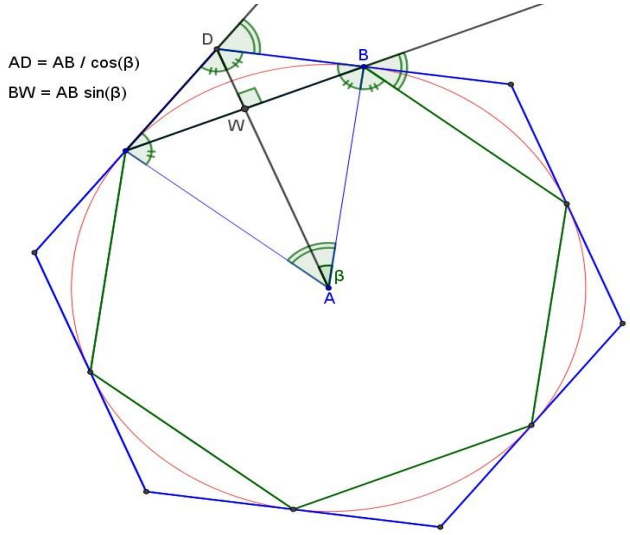


BONUS - Polygones réguliers, un peu de trigonométrie

Dans cette partie, on pourra utiliser les fonctions trigonométriques fournies par le module math de Python 3, ainsi que la constante pi qui fournit une valeur approchée de π, en faisant précéder le programme de l'instruction : **from math import cos, sin, tan, pi**.
Attention, contrairement à la tortue, ces fonctions comprennent les angles en radians…
Il est conseillé de tester les fonctions demandées une par une en écrivant à chaque fois un petit programme.

1. Écrire une fonction **polygone_regulier(n, c)** permettant de tracer un polygone à n côtés de longueur c. Le tracé du premier côté doit commencer depuis la position et la direction initiales de la tortue et dans le sens direct (anti-horaire).

2. Écrire une fonction **polygone_inscrit(n,r)** permettant de tracer un polygone régulier à n côtés (en vert sur la figure ci-dessous), inscrit dans un cercle imaginaire de rayon r (en rose) dont le centre est la position initiale de la tortue (point A).



On placera le premier sommet du polygone (point B) dans la direction initiale de la tortue (ici supposée orientée selon le vecteur).
Cette fonction devra réutiliser la fonction **polygone_regulier(n, c)**.

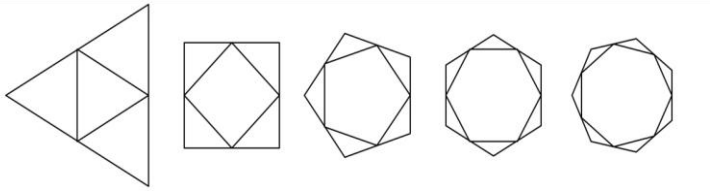
3. Écrire une fonction **polygone_exinscrit(n, r)** permettant de tracer un polygone régulier à n côtés (en bleu sur la figure), exinscrit au cercle décrit précédemment.

On tracera le polygone de manière à ce que le milieu de chacun de ses côtés soit un sommet du polygone inscrit tracé par la fonction précédente (voir la figure).
Cette fonction devra réutiliser la fonction **polygone_inscrit(n,r)**.

4. Tester l'ensemble des fonctions réalisées à l'aide du programme principal suivant :

```
up()
goto(-220, 0)
rayon = 40
for i in range(3, 8):
    polygone_inscrit(i, rayon)
    polygone_exinscrit(i, rayon)
    forward(2 * rayon + 30)
exitonclick()
```

dont le résultat attendu est montré ci-contre :



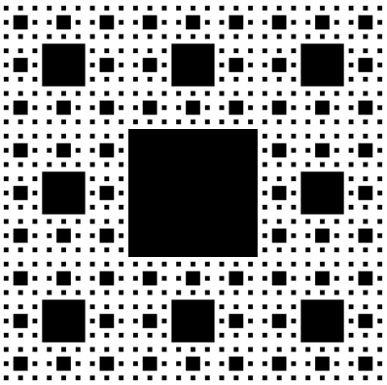
BONUS : Fonctions récursives – Tapis de Sierpinski

Dans cette partie, on va utiliser des fonctions récursives, c'est à dire des fonctions qui s'appellent elles-mêmes (avec des arguments différents), selon le même principe que la récurrence en mathématiques.
Par exemple, voici ci-contre fonction récursive qui calcule la factorielle d'un nombre entier n.

```
def factorielle(n) :
    if n <= 1 :  # cas de base
        return 1
    else :  # cas général
        return n * factorielle (n – 1)  # appel récursif
```

Pour ne pas s'appeler elle-même indéfiniment, une fonction récursive commence en général par un test permettant de distinguer les cas de base (sans appels récursifs) des cas généraux (qui peuvent engendrer un ou plusieurs appels récursifs).

Pour plus de détails sur la programmation de fonctions récursives, on pourra se référer par exemple au document http://www.irem.univ-paris-diderot.fr/up/recursif_terminal.pdf.



On souhaite reproduire le dessin ci-contre appelé *tapis de Sierpinski*.

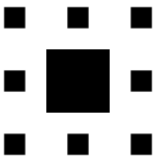
Si l'on observe bien ce dessin, on se rend compte que la structure est récursive : le dessin tout entier est répété plusieurs fois (à une échelle plus petite) à l'intérieur de lui-même.

Supposons qu'on travaille dans une zone carrée de côté c.

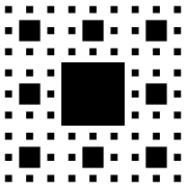
- On dessine d'abord un carré central (ici rempli en noir) de côté c/3.
- Puis dans chacun des huit petits carrés de côté c/3 restants, on reproduit le même dessin à l'échelle 1/3.

Pour que le processus ne soit pas infini, on s'arrête à une « profondeur » fixée, appelée **génération**.
Voici, par exemple, les dessins obtenus aux générations 1 à 4 :

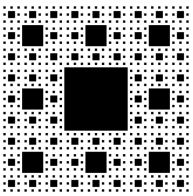
Génération 1



Génération 2



Génération 3



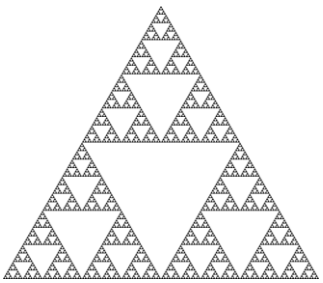
Génération 4

1. Écrire une fonction **carre_centre(c)** qui dessine un carré rempli (dans la couleur courante) et centré autour de la position initiale de la tortue.
2. Écrire une fonction *récursive* **tapis_sierpinski(c, n)** qui dessine un tapis de Sierpinski de génération n et de côté c centré sur la position initiale de la tortue. Ici, on suppose que n est un entier strictement positif.

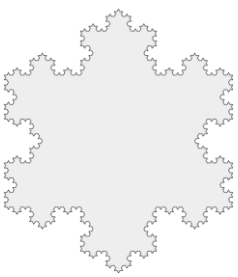
Indications :

- *Si n == 1, cette fonction doit seulement dessiner un carré centré de côté c/3.*
- *Et si n > 1, elle doit en plus dessiner huit tapis de Sierpinski de côté c/3 et de génération n-1...*

3. En vous inspirant de cette fonction, créer des fonctions vous permettant de reproduire les dessins suivants :



Triangle de Sierpinski



Flocon de Koch