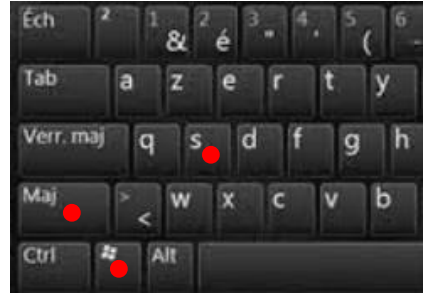


L'objectif principal de ce second TP est d'utiliser la structure « `for .. in range(...)` » vue en cours dans des situations différentes.

On demande de rédiger un compte-rendu au format `.doc` ou `.odt` à transférer en fin d'activité par l'intermédiaire de l'onglet **Mon Compte** du site <https://nsibrantly.fr> en utilisant le code : **tp2** . Ce compte-rendu contiendra :

- les réponses aux différentes questions posées,
- les captures d'écran **des morceaux de codes** écrits et celles **des résultats des exécutions**. Pour faire ces captures, utiliser *l'Outil Capture d'écran* de Windows (touches clavier `windows+Shift+s`)

Attention à repérer correctement les titres de paragraphe.

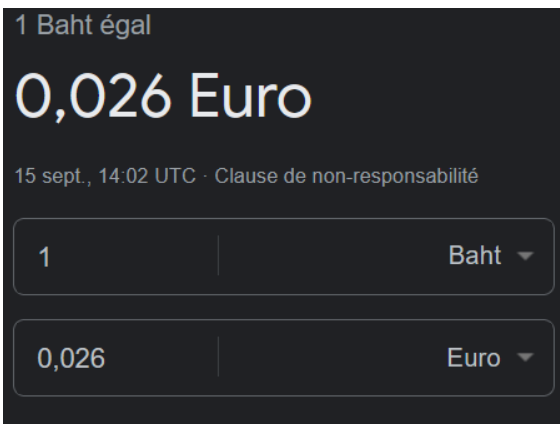


1- CODE A TROUVER :

⇒ Ecrire un code qui demande de saisir un entier pour afficher ensuite dans la console, un texte identique à celui donné sur l'exemple ci-contre :

```
Combien ? : 5
1 fois
2 fois
3 fois
4 fois
5 fois
```

2- CODE DONNANT DES EXEMPLES DE CONVERSION MONETAIRE ENTRE BATHS THAÏLANDAIS ET EUROS :



La monnaie nationale en Thaïlande est le bath. Au 15 septembre 2023, le taux de change était celui donné sur la figure ci-contre.

⇒ Ecrire un code comprenant une boucle `for` et qui permet d'afficher dans la console, les 10 lignes ci-contre :

```
>>> (executing file "ex1.py")
1 Bath(s) = 0.026 €
10 Bath(s) = 0.26 €
100 Bath(s) = 2.6 €
1000 Bath(s) = 26.0 €
10000 Bath(s) = 260.0 €
100000 Bath(s) = 2600.0 €
1000000 Bath(s) = 26000.0 €
```

Info : $10^1 = 10^{**}1 = 10$
 $10^2 = 10^{**}2 = 100$
 $10^3 = 10^{**}3 = 1000$

3- QUE FAIT CE CODE ? :

⇒ Exécuter le code donné ci-contre :

```
for i in range(10) :
    print(2*i , end=" ")
```

Remarque : l'instruction `end=" "`, inséré en dernier argument dans les parenthèses de la fonction `print()`, permet de remplacer *le saut à la ligne* qui termine par défaut le `print()` par ici un caractère espace " ". Ainsi les différents `print()` exécutés l'un après l'autre, permettent d'écrire sur la même ligne et ici les différentes valeurs écrites sont séparées par un espace. En écrivant `end = "***"`, le saut à la ligne est remplacé par 2 étoiles .

Application :

⇒ Ecrire un code qui comporte 2 lignes seulement et qui affiche dans la console :

```
0+ 2+ 4+ 6+ 8+ 10+ 12+ 14+ 16+ 18+
```

4- CODE QUI AFFICHE DES NOMBRES AU CARRE :

⇒ Ecrire un code qui demande à l'utilisateur combien de nombres au carré il faut afficher. Après saisi de ce nombre, le code les affiche à l'écran. Par exemple :

```
>>> (executing file "ex4.py")
Combien de nombres au carré on affiche ? :10
1 4 9 16 25 36 49 64 81 100
```

5- CODE QUI AFFICHE LA SOMME DE NOMBRES AU CARRE :

⇒ Modifier le code précédent, afin qu'il puisse afficher en plus la somme des nombres affichés. Par exemple :

```
>>> (executing file "ex4.py")
Combien de nombres au carré on affiche ? : 10
1 + 4 + 9 + 16 + 25 + 36 + 49 + 64 + 81 + 100 = 385
```

6- CODE INCOMPLET :

Le code ci-contre est incomplet. Il manque l'argument du 2^{ème} appel à la fonction `range()`.

Son exécution avec une valeur saisie de 5, affiche dans la console :

```
n = int(input("Combien ? : "))
for i in range(1,n+1) :
    for j in range( ) :
        print("*",end="")
    print()
```

```
Combien ? : 5
```

```
*
**
***
****
*****
```

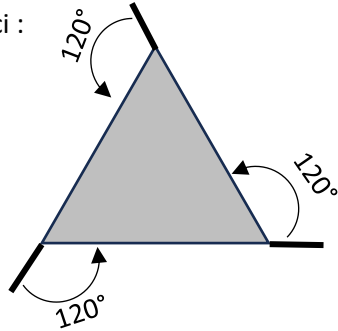
⇒ Compléter le code et essayer.



7- QUE DESSINE CE CODE ? :

Le code ci-contre permet de tracer une forme géométrique sur une fenêtre graphique « turtle ».

Rappel maths : sur un triangle *équilatéral*, les angles internes sont à 60° . Ainsi, on retrouve 120° ici :

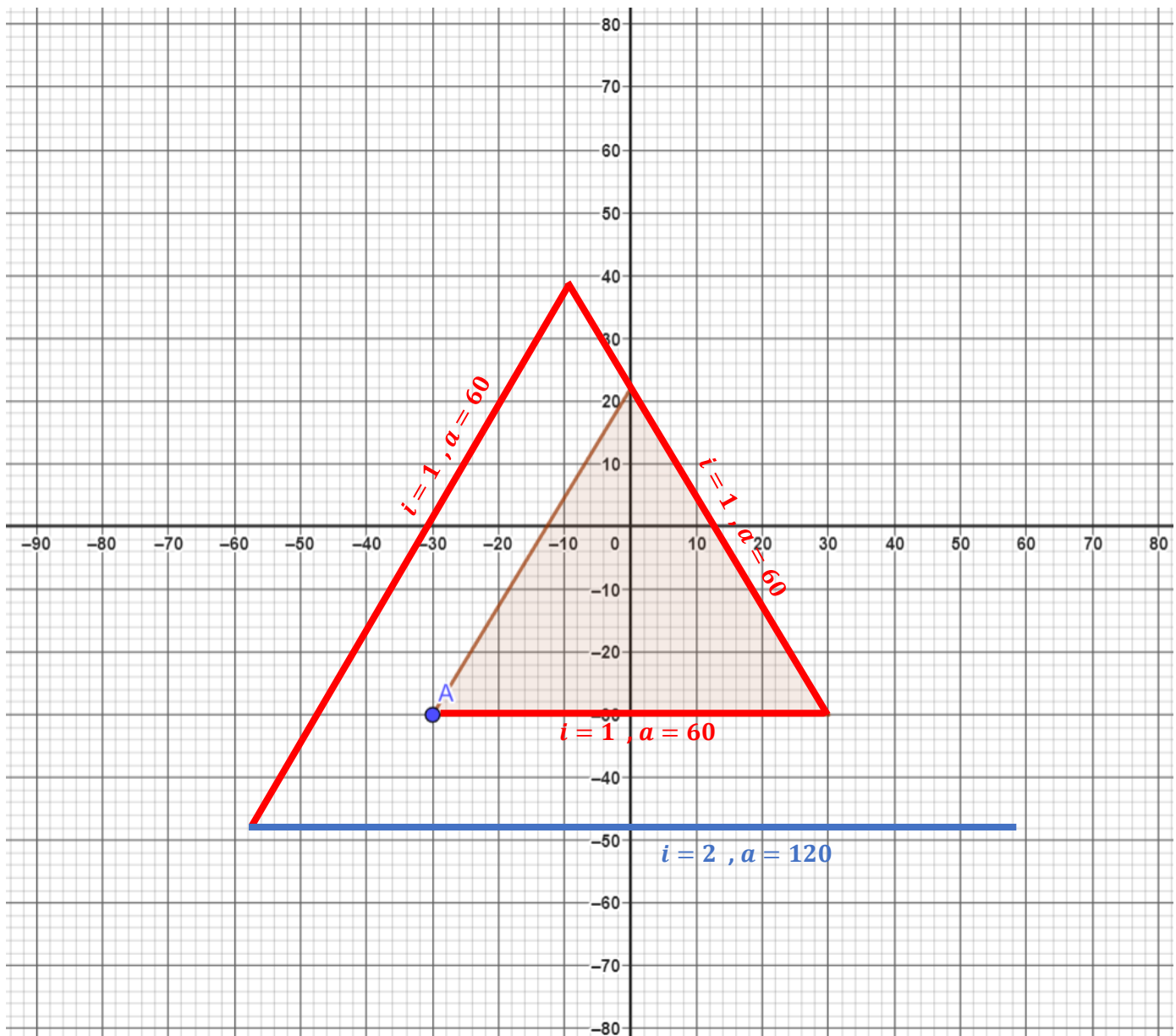


```

1 from turtle import *
2 n = int(input("Combien ? : "))
3 up()
4 a = 60
5 goto(-a/2, -a/2)
6 down()
7 for i in range(1,n) :
8     width(0.8*i)
9     forward(a)
10    left(120)
11    forward(a+20)
12    left(120)
13    forward(a+40)
14    left(120)
15    a = a + 60
16
17 mainloop()

```

⇒ Exécuter ce code et le tester pour une valeur n saisie de 20.



⇒ Afin de bien comprendre dans le détail ce qui se passe, on demande de compléter au crayon, le croquis précédent et distribué à l'échelle 1 : 1 (1mm = 1 px) , **pour l'itération $i = 2$** .

8- ON AMELIORE LE CODE PRECEDENT ? :

Le code ci-contre est incomplet. Il permet de faire exactement la même chose qu'avant, mais en évitant la répétition des fonctions `left()` et `forward()` .

⇒ Compléter correctement ce code et le tester.

```
1 from turtle import *
2 n = int(input("Combien ? : "))
3 up()
4 a = 60
5 goto(-a/2, -a/2)
6 down()
7 for i in range(1,n) :
8     width(0.8*i)
9     for j in range(3):
10         .....
11         .....
12         a = a + .....
13
14 mainloop()
```

9- TRACE D'UN CARRE DE COULEUR QUELCONQUE :

La bibliothèque `turtle` comprend une multitude de fonctions dont la documentation est donnée par exemple, sur la page : <https://docs.python.org/fr/3/library/turtle.html> . Dans cet exercice, on utilisera la fonction `numinput()` de `turtle` qui permet de réaliser des saisies au clavier, à partir d'une fenêtre de type pop-up. La valeur retournée sera un `float`. La documentation de cette fonction est donnée sur ce lien : <https://docs.python.org/fr/3/library/turtle.html#turtle.numinput> . On la commente ci-dessous :

The screenshot shows the documentation for `turtle.numinput()`. At the top, the function signature is: `turtle.numinput(title, prompt, default=None, minval=None, maxval=None)`. Below it, a 'Paramètres:' section lists five parameters: `title` (chaîne de caractères), `prompt` (chaîne de caractères), `default` (un nombre facultatif), `minval` (un nombre facultatif), and `maxval` (un nombre facultatif). A red arrow points from the `minval` and `maxval` parameters in the code below to this list. The main text describes the function's behavior: it pops up a dialog window for number input, with `title` as the window title and `prompt` as the text. It mentions that if `minval` and `maxval` are provided, the input must be within that range. At the bottom, a code block shows: `>>> screen.numinput("Poker", "Your stakes:", 1000, minval=10, maxval=10000)`. A red arrow points from a callout box to the `minval=10, maxval=10000` part of this code.

on utilisera directement la fonction `numinput()` ⇒ pas besoin d'écrire `screen`.

Il faut définir les arguments décrits ici :

Le code ci-dessous est incomplet. En l'exécutant, un carré de 600 px de coté et rempli d'une couleur définie par l'utilisateur est tracé.

⇒ Compléter ce code :

```
from turtle import *
title("Ma fenêtre de ouf !")
up()
goto(-300,-300)
down()
r = numinput("et 1", "Proportion de rouge (entre 0 et 1) :", 0.5, minval=0, maxval=1)
= numinput("et encore", "Proportion de vert (entre 0 et 1) :", 0.5, minval=0, maxval=1)
= numinput("dernier", "Proportion de bleu (entre 0 et 1) :", 0.5, minval=0, maxval=1)

fillcolor(r,g,b)

begin_fill()
for j in range(4):
    [redacted]
end_fill()

mainloop()
```

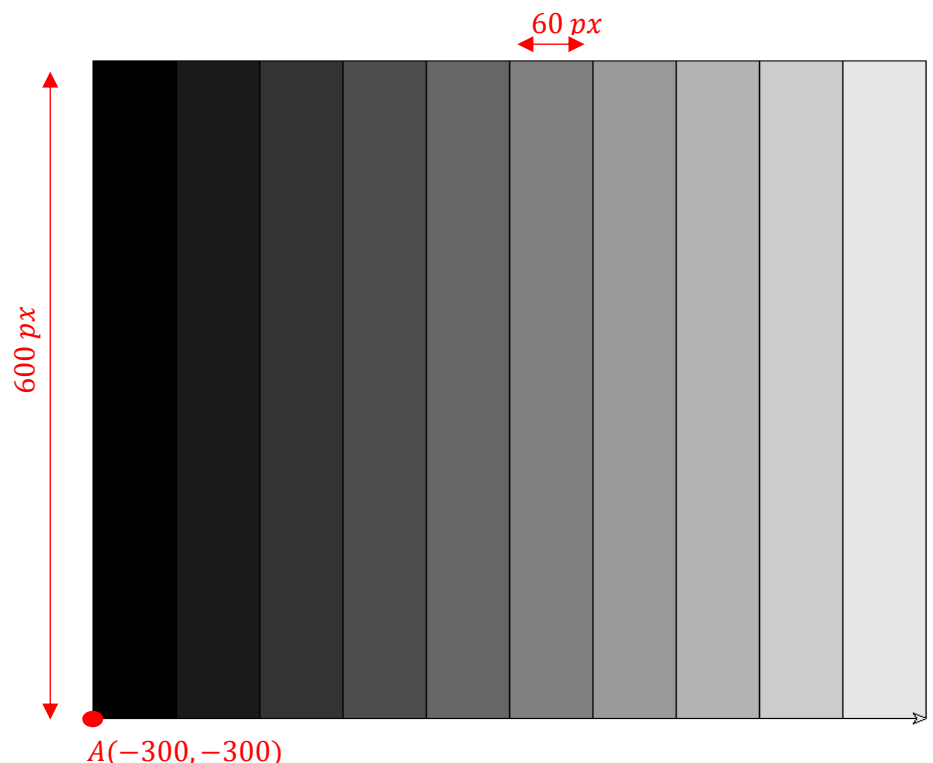
Info utile : Pour définir une couleur avec 20% de rouge, 20% de vert et 40% de bleu par exemple, on utilise la fonction `fillcolor()` avec comme argument : `fillcolor(0.2,0.2,0.4)` .

10- DEGRADE DE GRIS :

⇒ Ecrire un code qui permette d'afficher la figure ci-dessous :

Le premier rectangle possède une largeur de 60 pixels et une hauteur de 600 pixels et est rempli de la couleur noire.

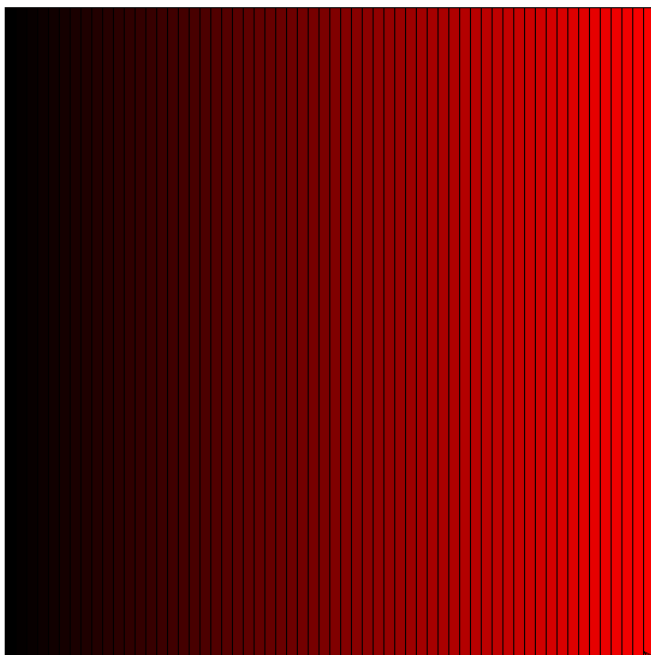
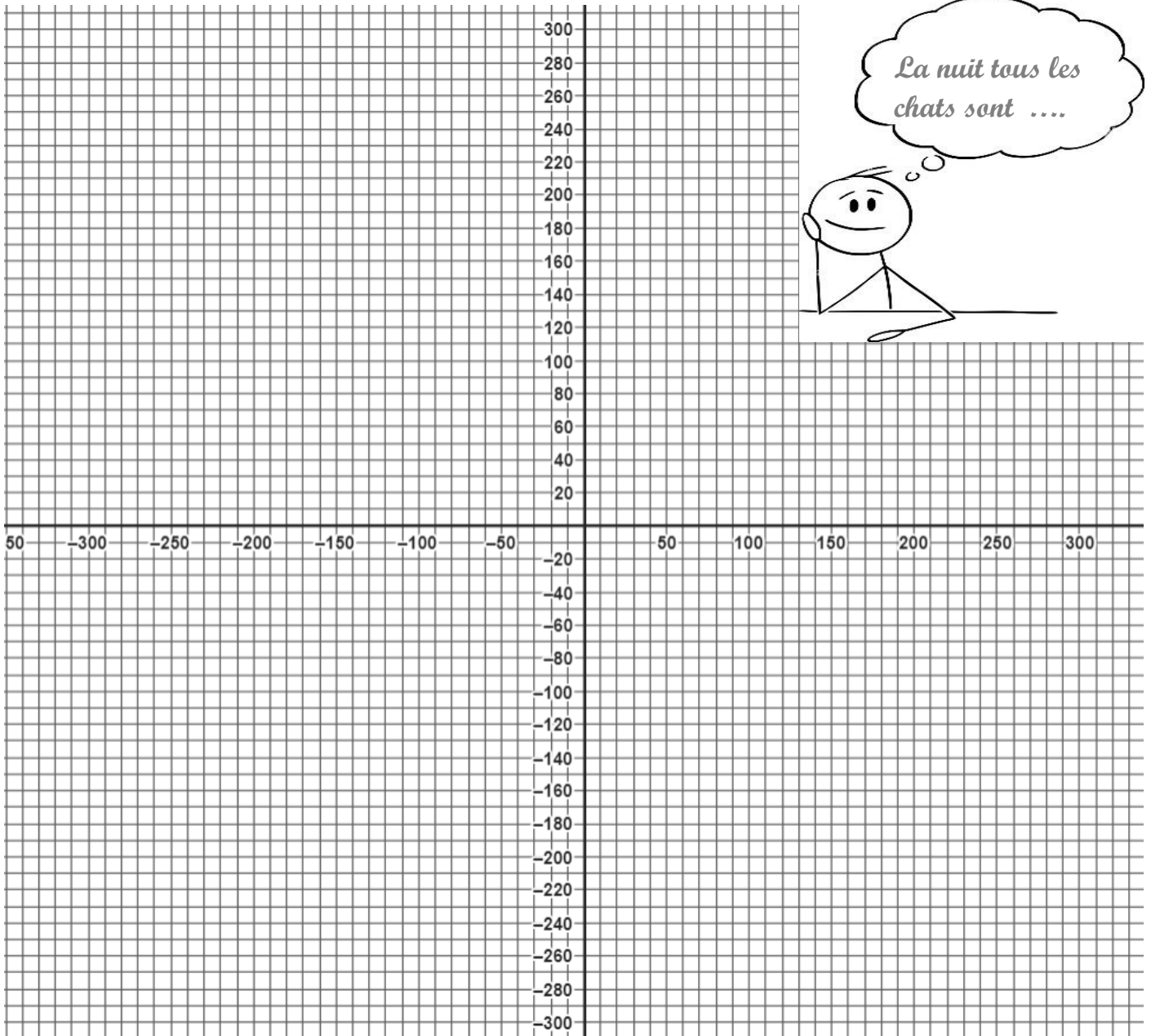
Chaque rectangle est rempli uniformément d'une couleur d'abord noire puis par un gris de plus en plus clair, proportionnellement à sa position. C'est en fait le même rectangle qui est décalé en abscisse de la largeur du premier. Utiliser une séquence de 2 boucles pour tracer cette figure.



Info utile pour obtenir un gris :

- avec `fillcolor(0,0,0)` → on a du noir
- avec `fillcolor(0.1,0.1,0.1)` → on a du gris foncé
- avec `fillcolor(0.8,0.8,0.8)` → on a du gris clair
- avec `fillcolor(1,1,1)` vous aurez du blanc

⇒ Compléter au crayon, le croquis ci-dessous et distribué, pour l'itération $i = 3$ seulement. Noter la valeur des variables utilisées à côté de la figure.



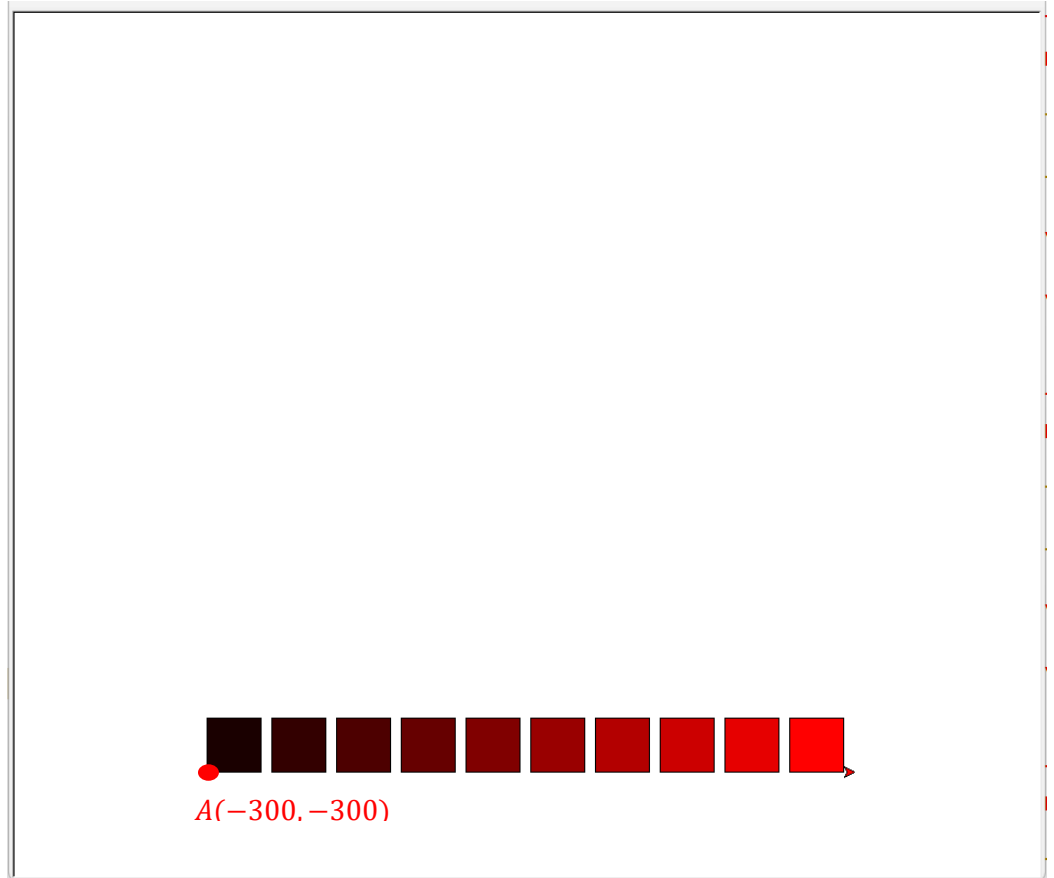
⇒ Modifier le code afin d'obtenir la figure ci-contre.

BONUS

11- DES CARRÉS ET ENCORE DES CARRÉS :

⇒ Ecrire un code qui permette de dessiner 10 carrés de 50 pixels de côté, séparés par une bande de 10 px de large.

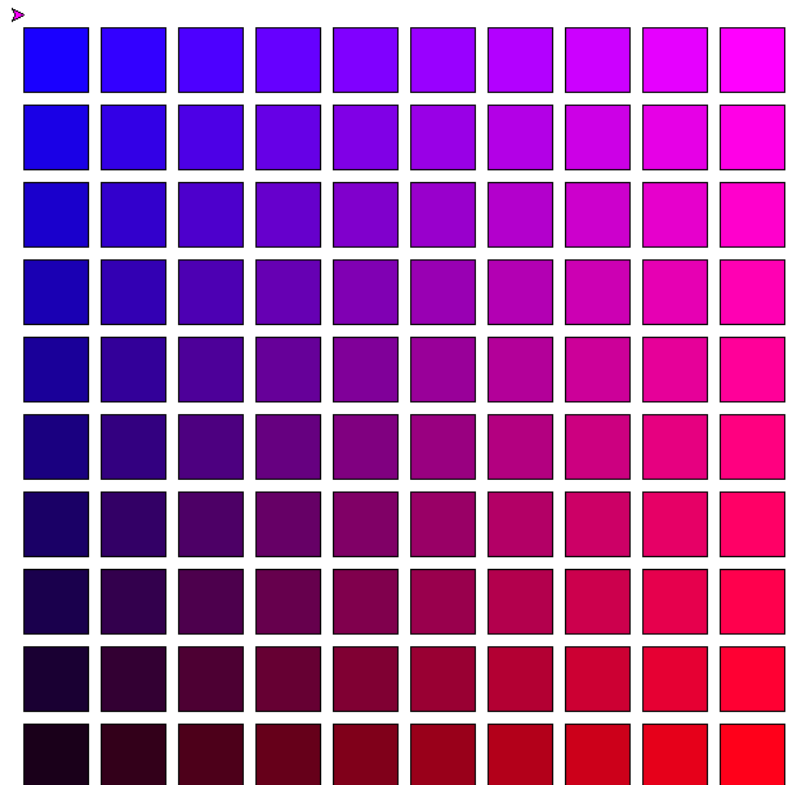
Le premier carré démarre au point A de coordonnées $A(-300, -300)$. Vous utiliserez une architecture à 2 boucles. Les couleurs seront celles de la figure ci-contre.



12- ON RAJOUTE UNE TROISIEME BOUCLE :

⇒ Ecrire un code qui permette de dessiner 100 carrés de 50 pixels de côté, séparés par une bande de 10 px de large.

Vous utiliserez une architecture à 3 boucles. Les couleurs seront celles de la figure ci-contre.



13- UN TEXTE A DECODER :

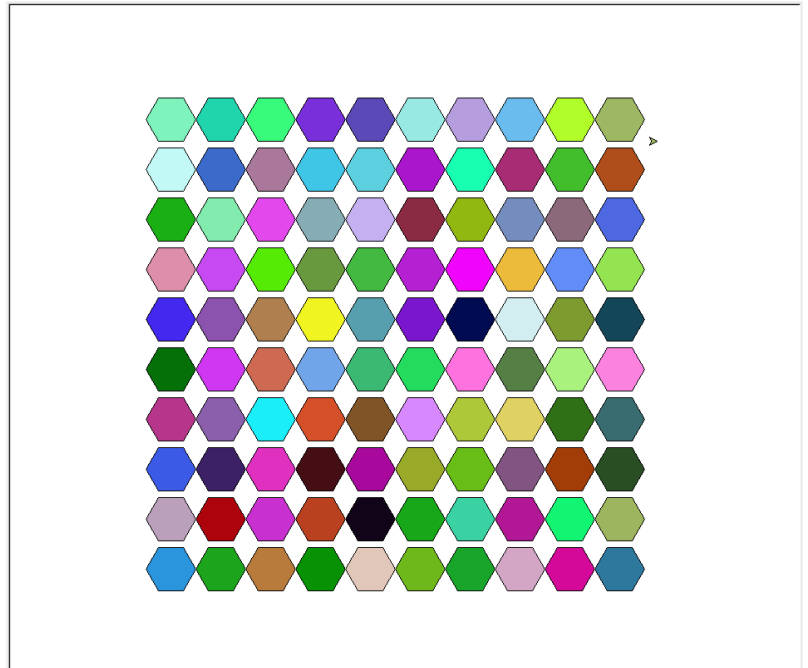
Le texte ci-dessous est incompréhensible car les caractères qui le composent ont été écrits à l'envers. Ainsi le premier caractère est en fait le dernier caractère du texte décodé :

« ednom ua euqinu iot ruop iares eJ .ednom ua euqinu iom ruop sares uT .ertua'l ed nu'l nioseb snorua suon ,sesiovirppa'm ut is ,siaM .sdraner ellim tneç à elbalbmes draner nu'uq iot ruop sius en eJ .sulp non iom ed nioseb sap sa'n ut tE .iot ed nioseb sap ia'n ej tE .snoçrag stitep ellim tneç à elbalbmes tuot noçrag titep nu'uq iom ruop erocne se'n uT .draner el tid ,rûs neiB »

⇒ Ecrire un code qui permette de décoder ce texte. (Commencer par copier-coller ce texte pour le placer dans une variable de type chaîne de caractère). De quel livre est tiré ce texte ?

14- DES ALVEOLES DE RUCHE :

⇒ Ecrire un code qui permette de dessiner 100 hexagones. Les couleurs de chacun d'eux est définie aléatoirement en utilisant la fonction random().



15- UN DERNIER CODE ENTIEREMENT DANS L'ALEATOIRE :

⇒ Ecrire un code qui permette de dessiner 10 figures dont la forme polygonale, la position et les couleurs sont définies aléatoirement :

- la couleur en utilisant la fonction random()
- la forme en utilisant la fonction randint(3,8) qui définira le nombre de côté du polygone. La longueur des cotés sera de 100 px.
- Les coordonnées du 1^{er} point sont définies avec la fonction randint(-200,200)

Info intéressante : Pour un polygone de n coté, la tortue devra tourner d'un angle de

$\frac{360}{n}$ degrés .

