TURTLE sur Python

Nouveau fichier →Tortue→Créer

```
from turtle import *
import turtle as tortue
```

→cette commande n'est pas obligatoire

Vocabulaire :

forward(n) et **back(n)** : faire avancer ou reculer la tortue de n

left(a) et **right(a)** : faire tourner à gauche ou à droite la tortue de a

Exemple : Dessiner un carré de côté 90

```
from turtle import *
import turtle as tortue
for i in range(4) :              # pour i variant de 0 à 3
    tortue.forward (60)          # avancer de 90
    tortue.left(90)              # tourner à gauche de 90°
tortue.mainloop()
```

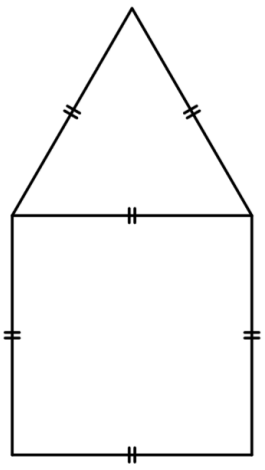
```
from turtle import *
for i in range(4) :
    forward (60)
    left(90)
mainloop()
```

La boucle d'événements

Les fonctions mainloop et done permettent de démarrer la boucle d'événements.
Dans le monde des événements, nous utilisons une boucle infinie. De cette manière, le programme continue d'écouter les interactions jusqu'à ce que l'utilisateur informe qu'il a terminé en fermant la fenêtre par exemple. Dès lors, nous sortons de la boucle et le programme peut s'achever.
Ces fonctions mettent en place cette boucle infinie pour nous. D'après la documentation, nous devons utiliser une de ces fonctions tout à la fin de notre programme turtle.

Exercice : Dessiner la figure suivante :

```
from turtle import *

for i in range(4) :
    forward (100)
    right(90)
left(60)
forward(100)
right(120)
forward(100)

mainloop()
```

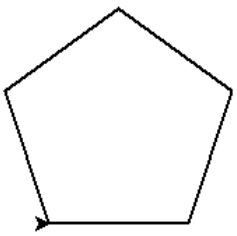


Exercice : Tracer un polygone régulier ayant n côtés :

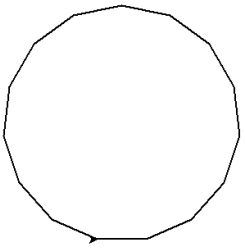
```
from turtle import *

n = int(input("Nombre  de côtés (au moins 3)"))
pensize(2)
for i in range(n):
    forward(100/(n/5))
    left(360/n)

mainloop()
```

NB : il faut adapter la longueur de côtés pour que la figure soit complète dans l'espace graphique

→si n = 5 :



→si n = 15 :



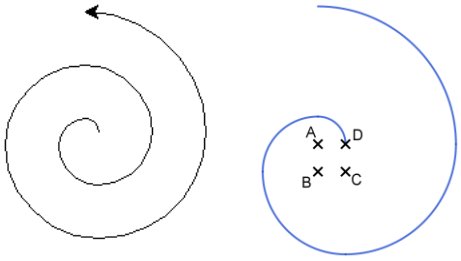
Vocabulaire pour tracer des cercles :

circle(rayon) : réaliser un cercle complet de rayon donné
circle(rayon,angle) : réaliser un arc de cercle d'angle donné d'un cercle de rayon donné

→si rayon > 0 : Trace un cercle à partir de la position de la tortue et en tournant dans le sens trigonométrique.
→si rayon < 0 : Trace un cercle dans le sens anti-horaire.
→si l'angle est précisé : trace un arc de cercle avec une ouverture de angle (en degré).
→si l'angle n'est pas précisé : le cercle est tracé dans son intégralité.

Exercice : Tracer une spirale

```
from turtle import *
n = 10
for i in range(n):
    circle(10*i,90)
mainloop()
```



Attention, on ne connaît a priori pas le centre de ce cercle... c'est d'ailleurs ce qui peut faire l'intérêt de l'algorithme!
La spirale ci-contre est obtenue en traçant bout à bout des quarts de cercle de centres successifs A, B, C et D.

Exercice : Tracer un œuf :

On veut tracer l'œuf ci-contre, avec les données suivantes :

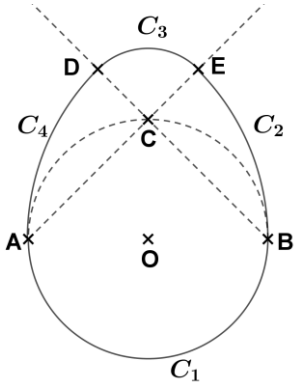
$OA = OB = OC = r$,

C_1 est un demi-cercle de diamètre $[AB]$,

C_2 est un arc de cercle de centre A et passant par B et E,

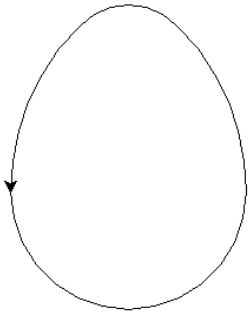
C_3 est un arc de cercle de centre C et passant par E et D,

C_4 est un arc de cercle de centre B et passant par D et A.



Règle essentielle :
le rayon est perpendiculaire à toutes les tangentes du cercle.

```
from turtle import *
from math import *
r=100
right(90)          # on part du point A
circle(r, 180)     # 1er  demi-cercle
circle(2 * r, 45)
      # 2ème demi-cercle, 45° car ABC est isocèle et droit
circle(2*r - r*sqrt(2), 90)    # le 3è rayon vaut AB - AC
circle(2 * r, 45)

mainloop()
```



Vocabulaire :

reset() : effacer ou fermer fenêtre
showturtle() ou **hideturtle()** : pour montrer ou ne pas afficher la tortue
up() ou **down()** : pour lever ou baisser le crayon

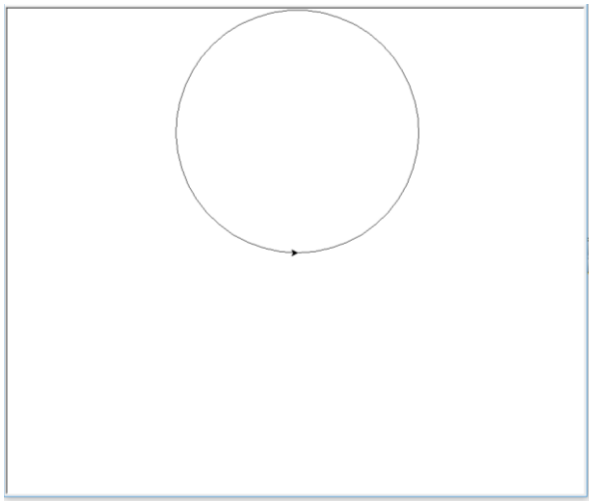
Pour centrer les figures dans la fenêtre

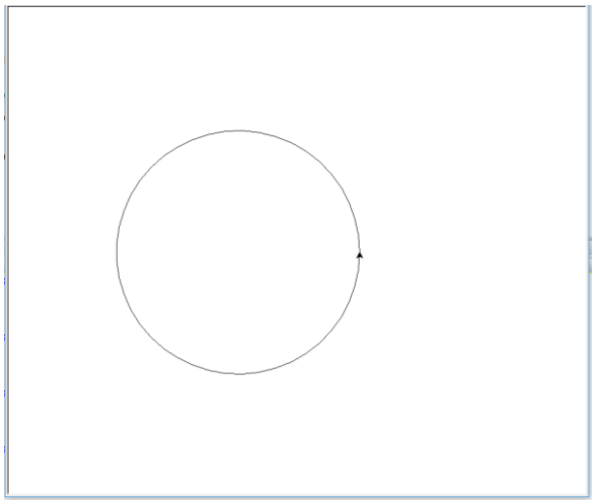
Sur les deux figures ci-dessous, nous avons tracé deux cercles :

```
circle(200)
```

Dans la 2ème figure, on a ajouté les instructions suivantes :

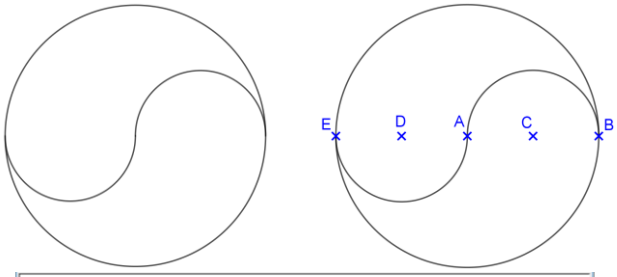
```
up()                # désactive le tracé de la tortue (« lever le traceur »)
forward(100)        # avance le curseur de départ de 100 (vers la droite, par défaut)
down()              # fin de la désactivation (« abaisser le traceur »)
left(90)            # oriente le traceur vers le haut
circle(200)
```

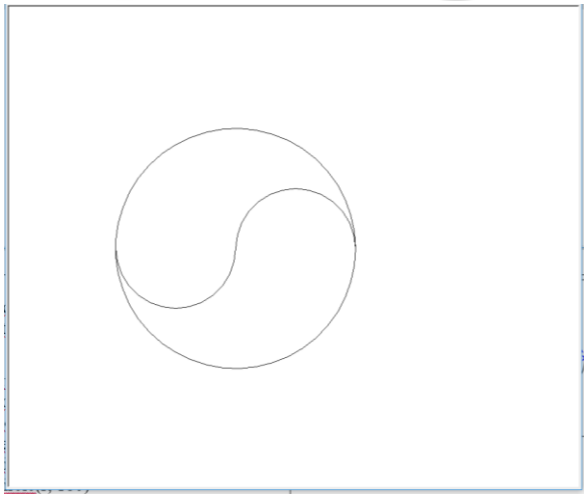




Exercice : Le yin et le yang

```
from turtle import *

r = 100
up()
forward(r)
down()
left(90)
circle(2 * r)
circle(r, 180)      # ½ cercle diam [AB]
circle(-r, 180)     # ½ cercle diam [AE]

hideturtle()      # cacher la tortue
mainloop()
```





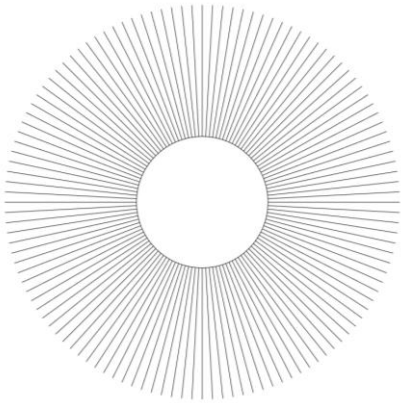
Vocabulaire sur la vitesse de traçage :

speed(valeur)

→speed(0) # met la vitesse de traçage la plus rapide, **par défaut : speed(1)**

Exercice : Un soleil avec 120 rayons

Voici une figure pénible à réaliser avec GEOGEBRA
Turtle va être d'une grande aide :



```
from turtle import *
speed(10)
n = int(input("Nombre  de rayons (au moins 3)"))
for i in range(n):
    right(90)              # on va tracer vers le bas
    forward(100)           # on trace un trait de longueur 100
    back(100)              # on trace à reculons
    left(90)
    circle(50, 360/n)      # 360 / 120 = 3°
hideturtle()
mainloop()
```

Vocabulaire pour définir la couleur du crayon :

pencolor(texte) ou **pencolor(rouge,vert,bleu)**

On peut entrer un texte entre guillemets parmi (entre autres) :
'aqua', 'beige', 'black', 'blue', 'brown', 'chocolate', 'fuchsia', 'gold', 'gray', 'green', 'indigo', 'khaki', 'maroon', 'orange', 'red', 'white', ...

On peut aussi définir sa propre couleur en paramétrant les composantes rouge, vert et bleu de la couleur (chaque composante étant un nombre entre 0 et 1).

Vocabulaire pour modifier l'épaisseur du tracé :

pensize(*nombre***)** →par défaut, **pensize() est un trait de taille 1**

Vocabulaire pour modifier la couleur de fond :

bgcolor(*paramètres***)** :

→elle prend en paramètres la couleur à appliquer :

- soit sous la forme d'une chaîne de caractères (le nom ou le code hexadecimal),
- soit sous la forme d'un tuple (code RGB : (*Red*, *Green*, *Blue*) avec des valeurs entre 0 et 1 ici). Il existe de nombreux sites sur internet pour vous renseigner sur le code hexadécimal ou le code RGB d'une couleur.

Exemples :

```
bgcolor("black")          # met le fond en noir
print(bgcolor())          # affiche 'black'

bgcolor("#00FF00")        # met le fond en vert
bgcolor((0.5, 0, 1))      # met le fond en violet
```

Vocabulaire pour colorier des figures :

Le crayon de Turtle possède deux couleurs : une couleur pour tracer ainsi qu'une couleur pour remplir :

pencolor(paramètres) pour modifier la couleur des tracés
fillcolor(paramètres) pour remplir les figures dessinées

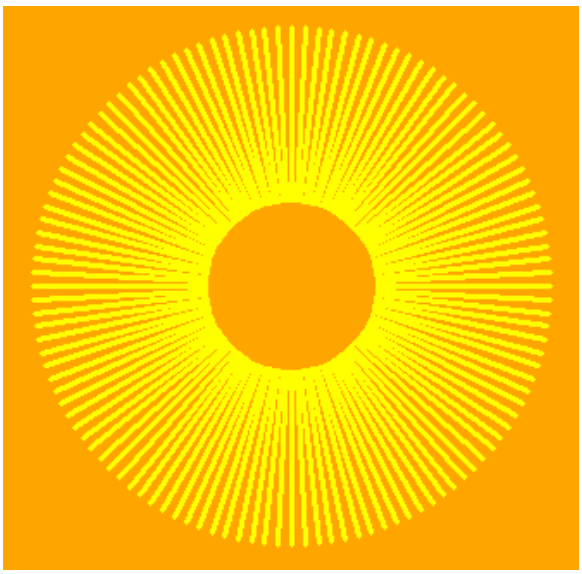
→avec fillcolor(), il faut utiliser les commandes :

begin_fill() et **end_fill()** pour préciser le début et la fin du remplissage

NB : si les bords et le contenu sont de même couleur : **color**(paramètre)
sinon : **color(c1,c2)** : modifie la couleur du trait c1 et la couleur du remplissage c2. On peut aussi les modifier séparement avec pencolor(c) et fillcolor(c).

Exercice :

```
from turtle import *
speed(10)
n = int(input("Nombre  de côtés (au moins 3)"))
pencolor("yellow")
bgcolor("orange")
for i in range(n):
    right(90)           # on va tracer vers le bas
    forward(100)        # on trace un trait de longueur 100
    back(100)           # on trace à reculons
    left(90)
    circle(50, 360/n)     # 360 / 120 = 3°
hideturtle()
mainloop()
```



Vocabulaire pour ouvrir et positionner le curseur sur la fenêtre turtle :

setup(largeur, hauteur, position en largeur, position en hauteur)
→largeur (*width*) de notre fenêtre,
→sa hauteur (*height*),
→la position en largeur (*startx*)
→position en hauteur (*starty*) du coin en haut à gauche de notre fenêtre par rapport au coin en haut à gauche de l'écran

Exemples :

```
setup(640, 480, 100, 100)      # Largeur : 640px, Hauteur : 480px, pos x : 100px, pos y : 100px
setup(200, 200)                # Largeur : 200px, Hauteur : 200px, position centrée
setup(startx = 0, starty = 0)  # Largeur : 50%, Hauteur : 75%, position : coin haut gauche écran
setup()                        # Largeur : 50%, Hauteur : 75%, position centrée
```