Exercice 1

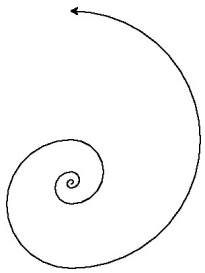
Ecrire un programme qui dessine une spirale en utilisant une succession de demi-cercles.

Corrigé

Il suffit de raccorder chaque demi-cercle qu'on dessine (instruction `circle(`r`, 180)`) au précédent, en augmentant le rayon r.
Si on l'augmente en ajoutant toujours la même valeur, on obtient une spirale régulière ; si on l'augmente de façon croissante (par exemple en multipliant par deux à chaque fois, on obtient une spirale « géométrique »).

```
circle(5,180)
circle(10,180)
circle(15,180)
circle(20,180)
circle(25,180)
circle(30,180)
circle(35,180)
circle(40,180)
```

```
circle(10,180)
circle(20,180)
circle(40,180)
circle(80,180)
circle(160,180)
circle(320,180)
```



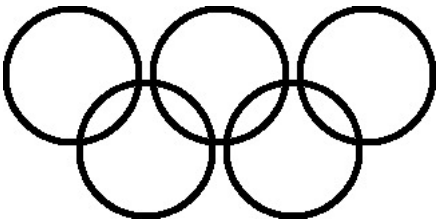
Exercice 2

Ecrire un programme qui dessine les cinq cercles de l'olympisme, d'abord en noir, puis en couleurs.

Corrigé

Première proposition : la séquence d'instruction est naturellement découpée en trois phases : les trois premiers cercles, puis un déplacement de la tortue, et les deux cercles suivants. Les placements ont été obtenus par tâtonnement.

```
# Jeux olympiques
ht()
# première ligne
pensize(4)
circle(50,360,500)
penup()
forward(2.2*50)
pendown()
circle(50,360,500)
penup()
forward(2.2*50)
pendown()
circle(50,360,500)
# déplacement pour la 2e ligne
penup()
backward(4.4*50)
right(90)
forward(1.1*50)
left(90)
forward(1.1*50)
pendown()
# deuxième ligne
circle(50,360,500)
penup()
forward(2.2*50)
pendown()
circle(50,360,500)
```

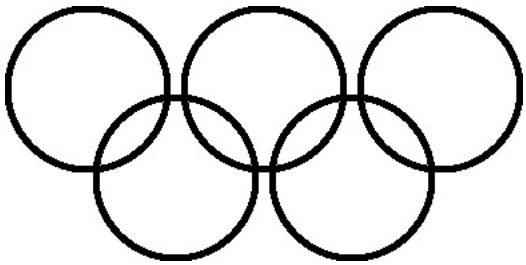


Le rayon est le paramètre important de ce dessin, toutes les dimensions en dépendent, et voici par conséquent une deuxième version où cette dépendance est plus visible : on introduit la variable `mon_rayon` qui fixe une fois pour toute la valeur de ce paramètre.

```
# Jeux olympiques
ht()
mon_rayon = 25
# première ligne
pensize(4)
circle(mon_rayon,360,500)
penup()
forward(2.2*mon_rayon)
pendown()
circle(mon_rayon,360,500)
penup()
forward(2.2*mon_rayon)
pendown()
circle(mon_rayon,360,500)
# déplacement pour la 2e ligne
penup()
backward(4.4*mon_rayon)
right(90)
forward(1.1*mon_rayon)
left(90)
forward(1.1*mon_rayon)
pendown()
# deuxième ligne
circle(mon_rayon,360,500)
penup()
forward(2.2*mon_rayon)
pendown()
circle(mon_rayon,360,500)
```

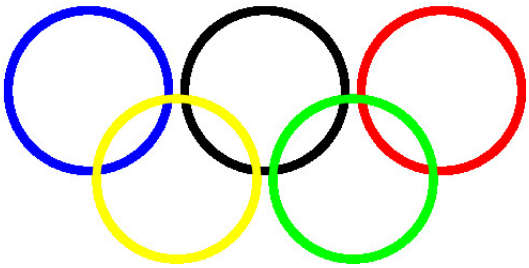
Découpage en fonctions : on observe que certaines parties du code sont répétées, et à partir de cette observation on peut définir des fonctions pour simplifier l'écriture. Le nom de la fonction doit être bien choisi.

```
# Jeux olympiques
def cercle_et_avance():
    circle(mon_rayon,360,500)
    penup()
    forward(2.2*mon_rayon)
    pendown()

def replacement_2e_ligne():
    penup()
    backward(6.6*mon_rayon)
    right(90)
    forward(1.1*mon_rayon)
    left(90)
    forward(1.1*mon_rayon)
    pendown()

ht()
mon_rayon = 60
pensize(4)

cercle_et_avance()
cercle_et_avance()
cercle_et_avance()
replacement_2e_ligne()
cercle_et_avance()
cercle_et_avance()
```



Dernière proposition : plutôt que d'avoir une variable globale `mon_rayon` on utilise les paramètres des fonctions. On ajoute aussi les couleurs olympiques, et on peut encapsuler tout ça dans une fonction.

```
def cercle_et_avance(r):
    circle(r,360,500)
    penup()
    forward(2.2*r)
    pendown()

def replacement_2e_ligne(r):
    penup()
    backward(6.6*r)
    right(90)
    forward(1.1*r)
    left(90)
    forward(1.1*r)
    pendown()

def anneaux_olympiques(rayon):
    ht()
    pensize(.1*rayon)

    color("blue")
    cercle_et_avance(rayon)
    color("black")
    cercle_et_avance(rayon)
    color("red")
    cercle_et_avance(rayon)
    replacement_2e_ligne(rayon)
    color("yellow")
    cercle_et_avance(rayon)
    color("green")
    cercle_et_avance(rayon)


penup() ; goto(-200,200) ; pendown()
anneaux_olympiques(20)

penup() ; goto(-100,-100) ; pendown()
anneaux_olympiques(75)
```



Exercice 3

[Programme à réaliser dans l'environnement `turtle`]
Ecrire une fonction qui affiche un triangle équilatéral dont la longueur du côté est passée en paramètre.

Corrigé

```
from turtle import *

# Version sans boucle:
def triangle(l):
   forward(l)
   left(360/3)
   forward(l)
   left(360/3)
   forward(l)
#   left(360/3)

# Version avec boucle:
def triangle(l):
   "Affiche un triangle equilateral de longueur l"
   for i in range(3):
      forward(l)
      left(360/3)

triangle(100)
```

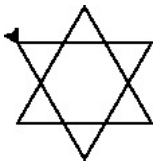
Exercice 4

[Programme à réaliser dans l'environnement `turtle`]
Ecrire un programme qui dessine une étoile à 6 branches en plaçant judicieusement deux triangles équilatéraux (réalisés par une fonction).

Corrigé

Une fois qu'on a dessiné le premier triangle, il faut non seulement « lever le crayon » pour se positionner plus haut, mais aussi choisir l'angle dans lequel on va se placer pour dessiner le second triangle. Il n'était pas nécessaire de définir une autre fonction triangle qui dessine dans l'autre sens.

```
from turtle import *

def triangle(l):
   "Affiche un triangle equilateral de longueur l"
   for i in range(3):
      forward(l)
      left(360/3)

# Dessin du premier triangle
triangle(100)
# positionnement pour dessiner le 2e triangle
left(90)    # maintenant on est oriente vers le haut
penup()     # on leve le crayon
forward(60) # on avance de 60
right(150)  # on tourne pour etre oriente vers le bas
pendown()   # on repose le crayon
# Dessin du 2e triangle
triangle(100)
```



Exercice 5

Ecrire une fonction qui affiche un triangle équilatéral coloré, dont la couleur et la longueur sont des paramètres.

Corrigé

```
def triangle(l,col):
   pen_color(col)
   forward(l)
   left(360/3)
   forward(l)
   left(360/3)
   forward(l)
#   left(360/3)

triangle(100, "red")
```

Exercice 6

[Programme à réaliser dans l'environnement `turtle`]
Ecrire un programme qui affiche un carré de 100 de côté. Proposer une version sans boucle et une version avec boucle.

Corrigé

```
from turtle import *

forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
```

```
from turtle import *

for i in range(4):
    forward(100)
    left(90)
```