INFO-H-100 - Programmation
TP 13 — Turtle

Pour les exercices suivants, veillez à découpez vos programmes en fonctions pertinentes.

Documentation de Turtle accessible à : http ://docs.python.org/3.2/library/turtle.html

Ex. 1. A l'aide des fonctions `fd` et `left` de `turtle`, écrire une fonction qui dessine un polygone régulier à n côtés de longueur l.

Ex. 2. A l'aide des fonctions `penup`, `pendown` et `goto` de `turtle`, écrire une fonction qui dessine le segment de droite entre les points (x_1, y_1) et (x_2, y_2).

Ex. 3. En utilisant la fonction `segment` de l'exercice 2, tapez le code suivant dans un script Python :

```
from math import pi, sin, cos

def dt(x1, y1, angle, level):
    if level > 0:
        x2 = x1 + int(cos(angle) * level * 10.0)
        y2 = y1 + int(sin(angle) * level * 10.0)
        segment((x1, y1), (x2, y2))
        dt(x2, y2, angle - pi/9, level - 1)
        dt(x2, y2, angle + pi/9, level - 1)

def draw_tree(level):
    dt(0, -200, pi/2, level)

draw_tree(9)
```

Comprenez-vous le comportement de la fonction `dt` ?

Ex. 4. A l'aide de l'exercice 2, écrire une fonction qui relie, dans l'ordre, tous les points d'une liste d'au moins 2 points.

Ex. 5. Ecrire une fonction qui produit les points de $100\sin(x/20)$ entre deux bornes min_x et max_x avec un pas donné.

```
sample_sin(-200, 200, 5) # -> [(-200, 54.40211108893698), (-195, 31.951919362227365), ...
```

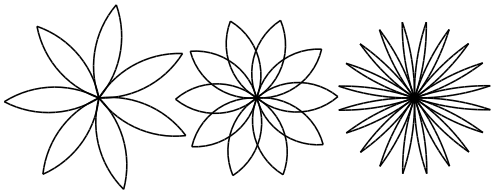
Ensuite, à l'aide des exercices précédents, dessiner cette liste.

Ex. 6. Ecrire une fonction qui produit les points d'une fonction $y = f(x)$ donnée en paramètres entre deux bornes min_x et max_x avec un pas donné.

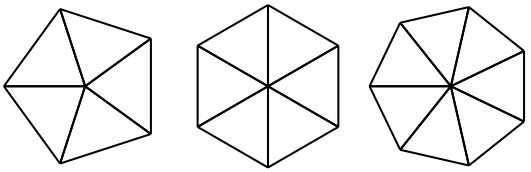
```
def f(x):
  return 100 * math.sin(x/20.0)

l = sample_fn(f, -200, 200, 5)

# -> [(-200, 54.40211108893698),
#     (-195, 31.951919362227365), ...
```

Ex. 7. A l'aide des exercices précédents, écrire une fonction qui dessine une fonction $y = f(x)$ agrandie de façon à ce que le dessin occupe tout l'espace entre $(-250, 250)$ et $(250, -250)$.

Ex. 8. Entrainez vous à dessiner des arcs de cercles à l'aide de la fonction `circle` de `turtle`. Ensuite, écrire une fonction qui dessine des fleurs de la manière suivante (conseil : dessiner des pétales) :



Ex. 9. Ecrire une fonction qui dessine des tartes de la manière suivante (conseil : dessiner des triangles isocèles) :



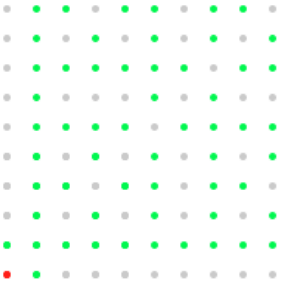
Ex. 10. A l'aide uniquement des fonctions `fd`, `left` et `setheading` (`turtle.setheading(0)` met à 0 l'angle de la tortue) de `turtle`, écrire une fonction qui dessine le segment de droite entre les points (x_1, y_1) et (x_2, y_2).

L'angle entre l'axe des ordonnées et la droite (x_1, y_1) et (x_2, y_2) est défini par

$$\arcsin\left(\frac{(y_2 - y_1)}{d((x_1, y_1), (x_2, y_2))}\right) * \frac{180}{\pi}$$

On considère que le point $(0, 0)$ est le point où la tortue commence son tracé (centre de l'écran).

Ex. 11. Dans le verger ci-dessous, seuls les arbres visibles de l'entrée du verger (le point rouge en bas à gauche), sont indiqués en vert. Les autres (les points gris) sont masqués par d'autres arbres. A l'aide de la fonction `dot`, `pencolor` et d'autres déjà vues, écrivez une fonction qui dessine un tel verger de coté `n` donné.



INFO-H-100 - Programmation
TP 13 — Turtle
Corrections

Solution de l'exercice 1:

```
import turtle

def draw_polygon(n,length):
    angle = 360.0/n
    for i in range(n):
        turtle.fd(length)
        turtle.left(angle)

draw_polygon(5,100)
draw_polygon(12,50)
```

Solution de l'exercice 2:

```
import turtle

def draw_segment(p1, p2):
    turtle.penup()
    turtle.goto(p1)
    turtle.pendown()
    turtle.goto(p2)

draw_segment((0,0), (100,100))
draw_segment((100,100),(200,0))
draw_segment((200,0),(100,-100))
draw_segment((100,-100),(0,0))
```

Solution de l'exercice 4:

```
def draw_polysegment(ls):
    if(len(ls)>=2):
        for i in range(1,len(ls)):
            draw_segment(ls[i-1],ls[i])

ls = [(-100,-100),(-100,100),(0,200),(100,100),
     (-100,100),(100,-100),(100,100),(-100,-100),
     (100,-100)]

draw_polysegment(ls)
```

Solution de l'exercice 5:

```
def sample_sin(min_x,max_x,step):
    ls = []
    x = min_x
    while x <= max_x:
        ls.append((x,100*math.sin(x/20.0)))
        x += step
    return ls

turtle.speed(0)
ls = sample_sin(-200,200,5)
draw_polysegment(ls)
```

Solution de l'exercice 6:

```
import math

def f(x):
    return 100 * math.sin(x/20.0)

def sample_fn(f,min_x,max_x,step):
    ls = []
```

```
    x = min_x
    while x <= max_x:
        ls.append((x,f(x)))
        x += step
    return ls
```

Solution de l'exercice 7:

```
import math
import turtle

def find_min_y(ls):
    minimum = ls[0][1]
    for point in ls:
        if(point[1] < minimum):
            minimum = point[1]
    return minimum

def find_max_y(ls):
    maximum = ls[0][1]
    for point in ls:
        if(point[1] > maximum):
            maximum = point[1]
    return maximum

def draw_sized(f,min_x,max_x):
    SIZE = 250
    l = sample_fn(f,min_x,max_x,1)
    x_ratio = float(2*SIZE)/abs(max_x-min_x)
    min_y = find_min_y(l)
    max_y = find_max_y(l)
    y_ratio = float(2*SIZE)/abs(max_y-min_y)
    for i in range(len(l)):
        x = -SIZE + (l[i][0]-min_x)*x_ratio
        y = -SIZE + (l[i][1]-min_y)*y_ratio
        l[i] = (x, y)
    draw_polysegment(l)

def f2(x):
    return math.cos(x/10.0)

draw_sized(f2,-100,100)
```

Solution de l'exercice 8:

```
import turtle

turtle.reset()

def draw_petal(radius, angle):
    for i in range(2):
        turtle.circle(radius, angle)
        turtle.left(180-angle)

def draw_flower(nb_petals, radius, angle):
    for i in range(nb_petals):
        draw_petal(radius,angle)
        turtle.left(360.0/nb_petals)

draw_flower(20,140,20)
```

Solution de l'exercice 9:

```
import turtle
import math

turtle.reset()

def draw_pie(n, r):
    theta = 360.0 / n
    for i in range(n):
        draw_isoceles_triangle(r, theta/2)
        turtle.left(theta)
```

```
def draw_isoceles_triangle(r, theta):
    y = r * math.sin(theta*math.pi/180)
    turtle.right(theta)
    turtle.fd(r)
    turtle.left(90+theta)
    turtle.fd(2*y)
    turtle.left(90+theta)
    turtle.fd(r)
    turtle.left(180-theta)

draw_pie(10,30)
```

Solution de l'exercice 10:

```
import turtle
import math

def distance(p1, p2):
    (x1,y1),(x2,y2) = p1, p2
    return math.sqrt((x1-x2)**2+(y1-y2)**2)

def angle(p1, p2):
    """pre: points != """
    (x1,y1),(x2,y2) = p1, p2
    sinus = (y2-y1)/distance((x1,y1),(x2,y2))
    angle = math.asin(sinus)*180/math.pi
    if(x1>=x2):                  #vers la gauche
        return - (angle + 180)
    else:                        #vers la droite
        return angle

def draw_segment(p1, p2):
    (x1,y1),(x2,y2) = p1, p2
    turtle.setheading(0)
    d = distance((x1,y1),(x2,y2))
    a = angle((x1,y1),(x2,y2))
    turtle.left(a)
    turtle.fd(d)

draw_segment((0,0), (100,100))
draw_segment((100,100),(200,0))
draw_segment((200,0),(100,-100))
draw_segment((100,-100),(0,0))
```

Solution de l'exercice 11:

```
import turtle

def draw_point(p):
    turtle.penup()
    turtle.goto(p)
    turtle.dot()

def gcd(a, b):
    """ Greatest common divisor. """
    # GCD using Euclid's algorithm.
    while b != 0:
        c = a % b
        a = b
        b = c
    return a

def orchard(n):
    turtle.hideturtle()
    for l in range(n):
        for c in range(n):
            if c == 0 and l == 0:
                turtle.pencolor("red")
            if gcd(l, c) > 1:
                turtle.pencolor("grey")
            draw_point((20 * c , 20 * l))
            turtle.pencolor("green")

orchard(10)
```