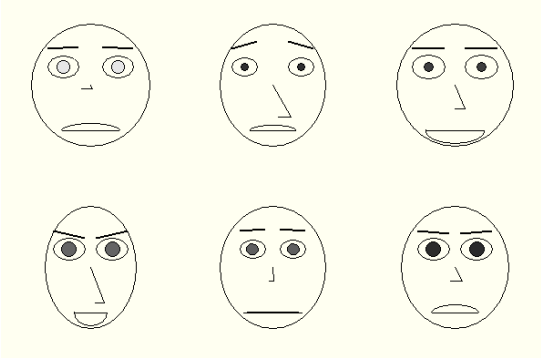


Chapitre 11

Figures de Chernoff

11.1. Nouveaux thèmes abordés dans ce chapitre

- tkinter : segments, rectangles, polygones, cercles, ovales, secteurs
- couleurs

11.2. Code de l'exemple

Pour expliquer comment dessiner des formes, le plus simple est de regarder un exemple :

exemple.py

```
# Exemples de dessins

from tkinter import *
from random import choice

def change_couleur():
    global coul
    palette=['snow', 'ghost white', 'white smoke', 'gainsboro', 'floral white', 'old
lace', 'linen', 'antique white', 'papaya whip', 'blanched almond', 'bisque', 'peach
puff', 'navajo white', 'lemon chiffon', 'mint cream', 'azure', 'alice blue',
'lavender', 'lavender blush', 'misty rose', 'dark slate gray', 'dim gray', 'slate
gray', 'light slate gray', 'gray', 'light grey', 'midnight blue', 'navy',
'cornflower blue', 'dark slate blue', 'slate blue', 'medium slate blue', 'light
slate blue', 'medium blue', 'royal blue',  'blue', 'dodger blue', 'deep sky blue',
'sky blue', 'light sky blue', 'steel blue', 'light steel blue', 'light blue',
'powder blue', 'pale turquoise', 'dark turquoise', 'medium turquoise', 'turquoise',
'cyan', 'light cyan', 'cadet blue', 'medium aquamarine', 'aquamarine', 'dark green',
'dark olive green', 'dark sea green', 'sea green', 'medium sea green', 'light sea
green', 'pale green', 'spring green', 'lawn green', 'medium spring green', 'green
yellow', 'lime green', 'yellow green', 'forest green', 'olive drab', 'dark khaki',
'khaki', 'pale goldenrod', 'light goldenrod yellow', 'light yellow', 'yellow',
'gold', 'light goldenrod', 'goldenrod', 'dark goldenrod', 'rosy brown', 'indian
red', 'saddle brown', 'sandy brown', 'dark salmon', 'salmon', 'light salmon',
'orange', 'dark orange', 'coral', 'light coral', 'tomato', 'orange red', 'red', 'hot
pink', 'deep pink', 'pink', 'light pink', 'pale violet red', 'maroon', 'medium
violet red', 'violet red', 'medium orchid', 'dark orchid', 'dark violet', 'blue
violet', 'purple', 'medium purple', 'thistle', 'snow2', 'snow3', 'snow4',
'seashell2', 'seashell3', 'seashell4', 'AntiqueWhite1', 'AntiqueWhite2',
'AntiqueWhite3', 'AntiqueWhite4', 'bisque2', 'bisque3', 'bisque4', 'PeachPuff2',
'PeachPuff3', 'PeachPuff4', 'NavajoWhite2', 'NavajoWhite3', 'NavajoWhite4',
'LemonChiffon2', 'LemonChiffon3', 'LemonChiffon4', 'cornsilk2', 'cornsilk3',
```

```
'cornsilk4', 'ivory2', 'ivory3', 'ivory4', 'honeydew2', 'honeydew3', 'honeydew4',
'LavenderBlush2', 'LavenderBlush3', 'LavenderBlush4', 'MistyRose2', 'MistyRose3',
'MistyRose4', 'azure2', 'azure3', 'azure4', 'SlateBlue1', 'SlateBlue2',
'SlateBlue3', 'SlateBlue4', 'RoyalBlue1', 'RoyalBlue2', 'RoyalBlue3', 'RoyalBlue4',
'blue2', 'blue4', 'DodgerBlue2', 'DodgerBlue3', 'DodgerBlue4', 'SteelBlue1',
'SteelBlue2', 'SteelBlue3', 'SteelBlue4', 'DeepSkyBlue2', 'DeepSkyBlue3',
'DeepSkyBlue4', 'SkyBlue1', 'SkyBlue2', 'SkyBlue3', 'SkyBlue4', 'LightSkyBlue1',
'LightSkyBlue2', 'LightSkyBlue3', 'LightSkyBlue4', 'SlateGray1', 'SlateGray2',
'SlateGray3', 'SlateGray4', 'LightSteelBlue1', 'LightSteelBlue2', 'LightSteelBlue3',
'LightSteelBlue4', 'LightBlue1', 'LightBlue2', 'LightBlue3', 'LightBlue4',
'LightCyan2', 'LightCyan3', 'LightCyan4', 'PaleTurquoise1', 'PaleTurquoise2',
'PaleTurquoise3', 'PaleTurquoise4', 'CadetBlue1', 'CadetBlue2', 'CadetBlue3',
'CadetBlue4', 'turquoise1', 'turquoise2', 'turquoise3', 'turquoise4', 'cyan2',
'cyan3', 'cyan4', 'DarkSlateGray1', 'DarkSlateGray2', 'DarkSlateGray3',
'DarkSlateGray4', 'aquamarine2', 'aquamarine4', 'DarkSeaGreen1', 'DarkSeaGreen2',
'DarkSeaGreen3', 'DarkSeaGreen4', 'SeaGreen1', 'SeaGreen2', 'SeaGreen3',
'PaleGreen1', 'PaleGreen2', 'PaleGreen3', 'PaleGreen4', 'SpringGreen2',
'SpringGreen3', 'SpringGreen4', 'green2', 'green3', 'green4', 'chartreuse2',
'chartreuse3', 'chartreuse4', 'OliveDrab1', 'OliveDrab2', 'OliveDrab4',
'DarkOliveGreen1', 'DarkOliveGreen2', 'DarkOliveGreen3', 'DarkOliveGreen4',
'khaki1', 'khaki2', 'khaki3', 'khaki4', 'LightGoldenrod1', 'LightGoldenrod2',
'LightGoldenrod3', 'LightGoldenrod4', 'LightYellow2', 'LightYellow3',
'LightYellow4', 'yellow2', 'yellow3', 'yellow4', 'gold2', 'gold3', 'gold4',
'goldenrod1', 'goldenrod2', 'goldenrod3', 'goldenrod4', 'DarkGoldenrod1',
'DarkGoldenrod2', 'DarkGoldenrod3', 'DarkGoldenrod4', 'RosyBrown1', 'RosyBrown2',
'RosyBrown3', 'RosyBrown4', 'IndianRed1', 'IndianRed2', 'IndianRed3', 'IndianRed4',
'sienna1', 'sienna2', 'sienna3', 'sienna4', 'burlywood1', 'burlywood2',
'burlywood3', 'burlywood4', 'wheat1', 'wheat2', 'wheat3', 'wheat4', 'tan1', 'tan2',
'tan4', 'chocolate1', 'chocolate2', 'chocolate3', 'firebrick1', 'firebrick2',
'firebrick3', 'firebrick4', 'brown1', 'brown2', 'brown3', 'brown4', 'salmon1',
'salmon2', 'salmon3', 'salmon4', 'LightSalmon2', 'LightSalmon3', 'LightSalmon4',
'orange2', 'orange3', 'orange4', 'DarkOrange1', 'DarkOrange2', 'DarkOrange3',
'DarkOrange4', 'coral1', 'coral2', 'coral3', 'coral4', 'tomato2', 'tomato3',
'tomato4', 'OrangeRed2', 'OrangeRed3', 'OrangeRed4', 'red2', 'red3', 'red4',
'DeepPink2', 'DeepPink3', 'DeepPink4', 'HotPink1', 'HotPink2', 'HotPink3',
'HotPink4', 'pink1', 'pink2', 'pink3', 'pink4', 'LightPink1', 'LightPink2',
'LightPink3', 'LightPink4', 'PaleVioletRed1', 'PaleVioletRed2', 'PaleVioletRed3',
'PaleVioletRed4', 'maroon1', 'maroon2', 'maroon3', 'maroon4', 'VioletRed1',
'VioletRed2', 'VioletRed3', 'VioletRed4', 'magenta2', 'magenta3', 'magenta4',
'orchid1', 'orchid2', 'orchid3', 'orchid4', 'plum1', 'plum2', 'plum3', 'plum4',
'MediumOrchid1', 'MediumOrchid2', 'MediumOrchid3', 'MediumOrchid4', 'DarkOrchid1',
'DarkOrchid2', 'DarkOrchid3', 'DarkOrchid4', 'purple1', 'purple2', 'purple3',
'purple4', 'MediumPurple1', 'MediumPurple2', 'MediumPurple3', 'MediumPurple4',
'thistle1', 'thistle2', 'thistle3', 'thistle4', 'gray1', 'gray2', 'gray3', 'gray4',
'gray5', 'gray6', 'gray7', 'gray8', 'gray9', 'gray10', 'gray11', 'gray12', 'gray13',
'gray14', 'gray15', 'gray16', 'gray17', 'gray18', 'gray19', 'gray20', 'gray21',
'gray22', 'gray23', 'gray24', 'gray25', 'gray26', 'gray27', 'gray28', 'gray29',
'gray30', 'gray31', 'gray32', 'gray33', 'gray34', 'gray35', 'gray36', 'gray37',
'gray38', 'gray39', 'gray40', 'gray42', 'gray43', 'gray44', 'gray45', 'gray46',
'gray47', 'gray48', 'gray49', 'gray50', 'gray51', 'gray52', 'gray53', 'gray54',
'gray55', 'gray56', 'gray57', 'gray58', 'gray59', 'gray60', 'gray61', 'gray62',
'gray63', 'gray64', 'gray65', 'gray66', 'gray67', 'gray68', 'gray69', 'gray70',
'gray71', 'gray72', 'gray73', 'gray74', 'gray75', 'gray76', 'gray77', 'gray78',
'gray79', 'gray80', 'gray81', 'gray82', 'gray83', 'gray84', 'gray85', 'gray86',
'gray87', 'gray88', 'gray89', 'gray90', 'gray91', 'gray92', 'gray93', 'gray94',
'gray95', 'gray97', 'gray98', 'gray99']
    coul = choice(palette)

def dessiner_ligne():
    global coul
    can.create_line(10, 10, 100, 200, width=2, fill=coul)

def dessiner_rectangle():
    global coul
    can.create_rectangle(10, 10, 100, 200, fill=coul)

def dessiner_polygone():
    global coul
    sommets = [150, 100, 200, 120, 250, 30, 360, 20, 300, 210, 150, 150]
    can.create_polygon(sommets, outline='red', fill=coul)

def dessiner_disque():
    global coul
    can.create_oval(200, 300, 300, 400, fill=coul)
```

```
def dessiner_ovale():
    global coul
    can.create_oval(100, 250, 150, 350, fill=coul)

def dessiner_secteur():
    global coul
    can.create_oval(100, 250, 150, 350, start=45, extent=90, fill=coul)

#------ Programme principal -------
coul = 'cyan'   # couleur initiale
fen = Tk()
can = Canvas(fen,bg='white',height=400,width=400)
can.pack(side=LEFT)
# Boutons
bou1 = Button(fen,text='Quitter',width=10,command=fen.quit)
bou1.pack(side=BOTTOM)
bou2 = Button(fen,text='Ligne',width=10,command=dessiner_ligne)
bou2.pack()
bou3 = Button(fen,text='Rectangle',width=10,command=dessiner_rectangle)
bou3.pack()
bou4 = Button(fen,text='Polygone',width=10,command=dessiner_polygone)
bou4.pack()
bou5 = Button(fen,text='Disque',width=10,command=dessiner_disque)
bou5.pack()
bou6 = Button(fen,text='Ovale',width=10,command=dessiner_ovale)
bou6.pack()
bou7 = Button(fen,text='Secteur',width=10,command=dessiner_secteur)
bou7.pack()
bou8 = Button(fen,text='Autre couleur',width="0,command=change_couleur)
bou8.pack()
fen.mainloop()
fen.destroy()
```

11.3. Analyse du programme

On a utilisé dans ce programme des objets classiques en dessin : segments, rectangles, polygones et ovales.

11.3.1. Les couleurs

Les couleurs peuvent se définir par leur nom. Nous avons profité de ce petit script pour les énumérer toutes. C'est un peu idiot, mais comme ces noms ne sont pas faciles à trouver dans la documentation Python, autant le faire une fois pour toutes.

```
def change_couleur():
    global coul
    palette=['snow', 'ghost white', 'white smoke', 'gainsboro', ...]

                (voir le code du §11.2 pour avoir la liste complète)
```

Une autre manière, plus pratique, de décrire une couleur est de donner le codage RVB (rouge, vert, bleu) ou, en anglais, RGB (red, green, blue). La combinaison de ces trois couleurs donnera 16'777'216 (256^3) couleurs.

Attention ! On parle ici de combinaisons de lumières, et non pas de peintures. Ainsi, du rouge avec du bleu donnera du magenta, du bleu avec du vert donnera du cyan, et du rouge avec du vert donnera du jaune. La combinaison des trois couleurs donnera du blanc.

En Python, la couleur aura le format suivant : #rrggbb, où rr sera un nombre, exprimé en base 16, compris entre 0 (`00`) et 256 (`ff`), qui donnera la valeur du rouge. Idem pour le vert (gg) et le bleu (bb). Par exemple, `#ff0000` est du rouge, `#ffff00` du jaune, `#000000` du noir, `#ffffff` du blanc, etc. On aurait ainsi pu écrire :

```
coul = '#00ffff'   # cyan
```

11.3.2. Segment

Un segment de droite est défini par les coordonnées de ses deux extrémités.

```
def dessiner_ligne():
    global coul
    can.create_line(10, 10, 100, 200, width=2, fill=coul)
```

Dans l'exemple, le segment va du point (10, 10) au point (100, 200). Il a une largeur de 2 pixels et une couleur définie par `coul`.

Attention ! L'origine (0, 0) est dans le coin supérieur gauche de la fenêtre. Le coin inférieur droit aura comme coordonnées (larg, haut), dans notre exemple (400, 400) :

```
can = Canvas(fen, bg='white', height=400, width=400)
```

Notons enfin que l'on peut prolonger le segment en une ligne polygonale : il suffira d'ajouter des sommets à la liste. Par exemple :

```
    can.create_line(10, 10, 100, 200, 150, 50, width=2, fill=coul)
```

11.3.3. Rectangle

Un rectangle est défini par les coordonnées des extrémités d'une de ses diagonales. Ses côtés seront parallèles aux axes.

```
def dessiner_rectangle():
    global coul
    can.create_rectangle(10, 10, 100, 200, fill=coul)
```

Dans l'exemple, on a donné comme extrémités de la diagonale le coin supérieur gauche (10, 10) et le coin inférieur droit (100, 200). Le rectangle a un contour (`outline`) noir et une couleur de remplissage (`fill`) définie par `coul`.

11.3.4. Polygone

Un polygone est défini par la suite des coordonnées de ses sommets. Le dernier sommet sera relié au premier.

```
def dessiner_polygone():
    global coul
    sommets = [150, 100, 200, 120, 250, 30, 360, 20, 300, 210, 150, 150]
    can.create_polygon(sommets, outline='red', fill=coul)
```

Dans l'exemple, la suite des sommets est (150, 100), (200, 120), (250, 30), (360, 20), (300, 210) et (150, 150).

11.3.5. Disque et ovale

Un ovale est défini par le rectangle (voir § 11.3.3) dans lequel il est inscrit. Si le rectangle est un carré, on obtient bien évidemment un disque.

```
def dessiner_disque():
    global coul
    can.create_oval(200, 300, 300, 400, fill=coul)

def dessiner_ovale():
    global coul
    can.create_oval(250, 250, 400, 350, fill=coul)
```

11.3.6. Secteur

Un secteur est une « part de gâteau » d'un disque ou d'un ovale. La syntaxe est la presque même que pour un ovale, mais on ajoute deux paramètres : l'inclinaison de départ (`start`) et l'angle du secteur (`extent`). Ces angles sont donnés en degrés et on tourne dans le sens inverse des aiguilles d'une montre.

On tourne donc dans le sens trigonométrique positif.

L'exemple ci-dessous dessinera un quart d'ovale, incliné de 45°, inscrit dans le rectangle défini par les sommets (100, 250) et (150, 350).

```
def dessiner_secteur():
    global coul
    can.create_arc(100, 250, 150, 350, start=45, extent=90, fill=coul)
```

Exercice 11.1

Modifiez les valeurs des paramètres dans le code du § 11.2 et observez quelles sont les répercussions sur les formes dessinées.

Exercice 11.2 : la marche de l'ivrogne

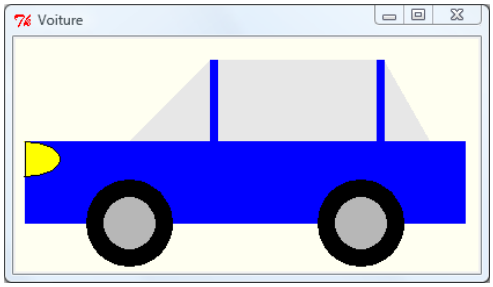
Un ivrogne se déplace en titubant. À chaque pas, il a le choix entre quatre directions : en haut, en bas, à gauche, à droite. Il prend une de ces quatre directions au hasard.

En partant du centre d'un carré, dessinez la trajectoire de l'ivrogne après 1000 pas.

Exercice 11.3

Reproduisez ce dessin en utilisant un polygone, trois rectangles, deux disques et un secteur.

Vous pouvez aussi dessiner un engin plus à votre goût...

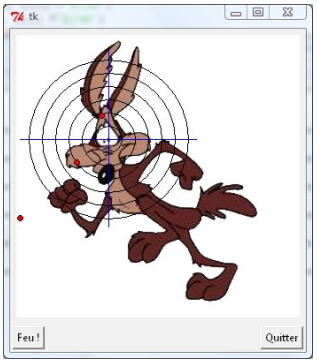


Exercice 11.4

Il existe aussi une méthode appelée `create_image` qui place une image dans un canevas.

```
photo = PhotoImage(file ='coyote.gif')
can.create_image(156, 156, image=photo)
```

Les coordonnées (156, 156) permettent de placer l'image dans le canevas. Le centre de l'image sera positionné sur ces coordonnées.

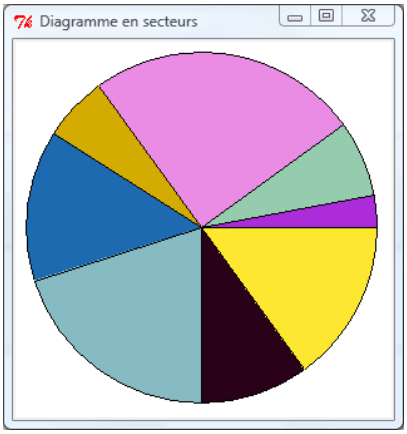


1. Insérez une image dans une fenêtre. Vous trouverez Wile Coyote à l'adresse www.apprendre-en-ligne.net/pj/chernoff/
2. Ses dimensions sont de 312 pixels sur 312 pixels.
3. Dessinez une cible centrée sur l'œil gauche du coyote (voir image ci-dessus)
4. Insérez un bouton « Quitter » et un bouton « Feu! »
5. Quand on pressera le bouton « Feu! » trois points rouges apparaîtront au hasard dans le plus petit carré englobant la cible, donnant l'illusion que l'on tire sur le coyote.

Exercice 11.5

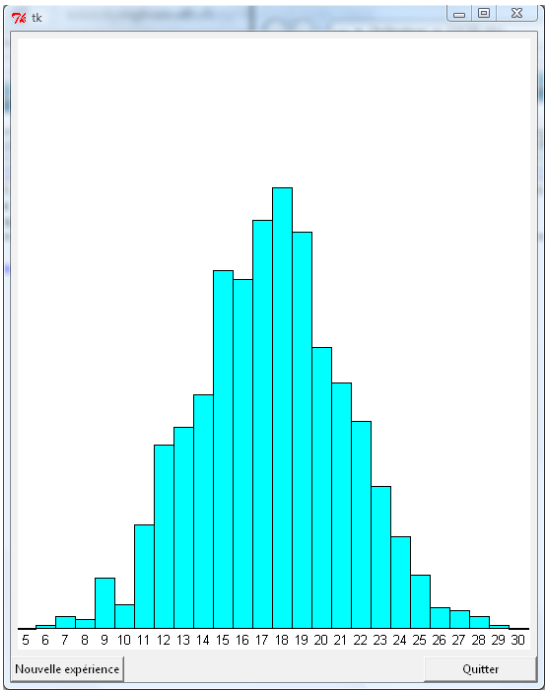
Écrivez un programme Python qui dessine un diagramme à secteurs (un camembert) d'après un tuple de pourcentages donnés. Chaque part de gâteau aura une couleur aléatoire. Le camembert ci-dessous a été obtenu à partir du tuple (3,7,25,6,14,20,10,15).

La somme des huit nombres fait bien 100...



Exercice 11.6

Écrivez un programme Python qui dessine un histogramme (voir exemple ci-dessous). Les nombres en bas de l'image sont les sommes possibles quand on lance 5 dés : ces sommes vont de 5=1+1+1+1+1 à 30=6+6+6+6+6. La hauteur est proportionnelle au nombre de fois que la somme correspondante est survenue. On appelle cela une distribution.



Les nombres seront tirés au sort dans le programme, si bien que l'on pourra faire plusieurs séries de lancers, en cliquant sur le bouton « Nouvelle expérience ». Chaque série comportera plusieurs centaines de lancers.

Avant de démarrer la partie graphique, le programme demandera le nombre de dés à utiliser (entre 1 et 10). Il faudra donc adapter les sommes...

Exercice 11.7

Modifiez le programme de l'exercice 11.6 afin de représenter la distribution obtenue en suivant le protocole de l'exercice 1.9.

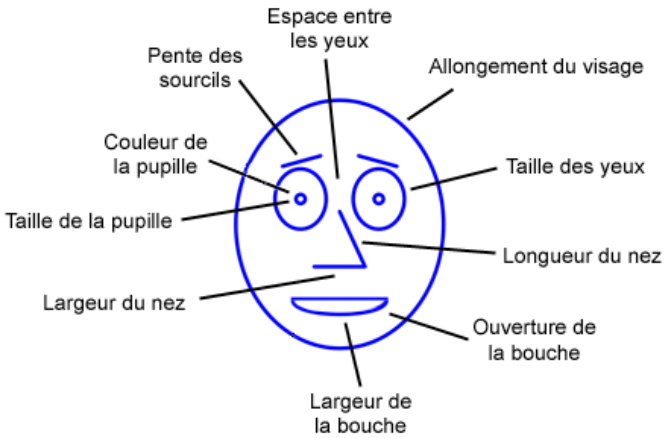
11.4. Les figures de Chernoff

Cette méthode a été inventée par Herman **Chernoff**, mathématicien et physicien américain.

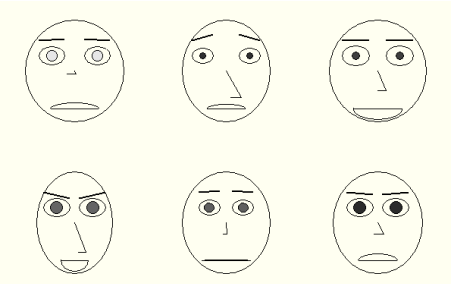
Chernoff, H. (1973). *The use of faces to represent points in k-dimensional space graphically*, Journal of the American Statistical Association 68 (342) : 361-368

Trouver une représentation graphique intuitive et facilement compréhensible de données à n dimensions lorsque n est supérieur à 3 n'est pas simple. Les représentations les plus courantes utilisent habituellement une, deux ou les trois dimensions spatiales et parfois quelques dimensions supplémentaires via un coloriage adéquat. Les figures de Chernoff permettent d'augmenter le nombre de dimensions à représenter. Elles tirent profit de notre capacité à déceler de très légers changements dans les expressions faciales.

Une figure de Chernoff s'obtient en associant à chaque composante du vecteur représentant les données un trait d'une expression faciale. La première composante permettra par exemple de préciser la forme de la tête, la seconde fixera la taille des yeux, la troisième leur aspect, la quatrième la distance entre ceux-ci, etc. En utilisant ainsi les traits les plus frappants d'un visage, il est possible de représenter un nombre de dimensions largement supérieur à trois.



Voici un exemple utilisant les caractéristiques ci-dessus. On peut ainsi comparer des entités comportant 10 paramètres, par exemple des élèves d'une classe (les paramètres seraient les notes dans différentes disciplines), des pays (PIB, population, chômage, etc.), des aliments en fonction de la composition, etc.



11.5. Code du programme

chernoff.py

```
# figures de Chernoff

from random import random
from math import *
from tkinter import *

def gris(c):
    # choisit un niveau de gris (gray1 à gray99)
    c = int(c*100)
    if c==100:
```

```
        return "white"
    elif c==0:
        return "black"
    else:
        return "gray"+str(c)

def visage(a):
    # a : allongement du visage. 0=rond
    a = 1-0.25*a
    # 0.75 <= a <= 1
    can.create_oval(centre-a*cote/2.6, centre-cote/2.6/a,
                    centre+a*cote/2.6, centre+cote/2.6/a)

def nez(l, g):
    # l : longueur du nez
    # g : profondeur du nez
    # 0 <= l <= 1
    l = 38*(l+0.1)
    k = 2+2*g
    # 2 <= k <= 4 pour calculer l'angle du nez
    can.create_line(centre, centre, centre+l*cos(pi/k), centre+l*sin(pi/k),
                    centre+l*cos(pi/k)-20*g-5, centre+l*sin(pi/k))

def yeux(e, p, o, f, c):
    # e : espacement entre les yeux
    # p : rayon de la pupille
    # o : rayon de l'oeil
    # f : froncement des sourcils
    # c : couleur de la pupille
    e = cote/10+cote/18*e
    p = 5*p+cote/60
    o = cote/40*o+cote/15
    # pupilles
    can.create_oval(centre-e-p, centre-0.1*cote-p,
                    centre-e+p, centre-0.1*cote+p, fill=gris(c))
    can.create_oval(centre+e-p, centre-0.1*cote-p,
                    centre+e+p, centre-0.1*cote+p, fill=gris(c))
    # oeil
    can.create_oval(centre-e-o, centre-0.1*cote-0.7*o,
                    centre-e+o, centre-0.1*cote+0.7*o)
    can.create_oval(centre+e-o, centre-0.1*cote-0.7*o,
                    centre+e+o, centre-0.1*cote+0.7*o)
    # sourcils
    f = f - 0.5
    can.create_line(centre-e-o, centre-0.2*cote,
                    centre-e+o, centre-0.2*cote+30*f, width=2)
    can.create_line(centre+e+o, centre-0.2*cote,
                    centre+e-o, centre-0.2*cote+30*f, width=2)

def bouche(m,j):
    # m : largeur
    # j : hauteur
    j = j-0.5
    m = m+0.02
    if j<0:
        can.create_arc(centre-cote/6*m, centre+0.25*cote+cote/6*j,
                       centre+cote/6*m, centre+0.25*cote-cote/6*j,
                       start=0, extent=180)
    else:
        can.create_arc(centre-cote/6*m, centre+0.25*cote+cote/6*j,
                       centre+cote/6*m, centre+0.25*cote-cote/6*j,
                       start=180, extent=180)

def chernoff(a, l, g, e, p, o, f, c, m, j):
    # coefficients entre 0 et 1
    visage(a)
    nez(l,g)
    yeux(e,p,o,f,c)
    bouche(m,j)

def reinit():
    can.delete(ALL)
    a = random()
    l = random()
```

```
    g = random()
    e = random()
    p = random()
    o = random()
    f = random()
    c = random()
    j = random()
    m = random()
    chernoff(a,l,g,e,p,o,f,c,m,j)

cote = 300
centre = 1.2*cote/2

root = Tk()
root.title("Figures de Chernoff")
can = Canvas(root, height=1.2*cote, width=1.2*cote, background="ivory")
can.pack()
b = Button(root,text='Une autre!',command=reinit)
b.pack()
chernoff(1,1,1,1,1,1,1,1,1,1)
root.mainloop()
```

Comme ce programme ne contient rien de nouveau, nous laissons le lecteur l'analyser seul. Nous noterons simplement que la variable `cote` définit à la fois la taille de la fenêtre (`1.2*cote`) et celle du visage, qui est placé au centre de la fenêtre (`centre = 1.2*cote/2`).

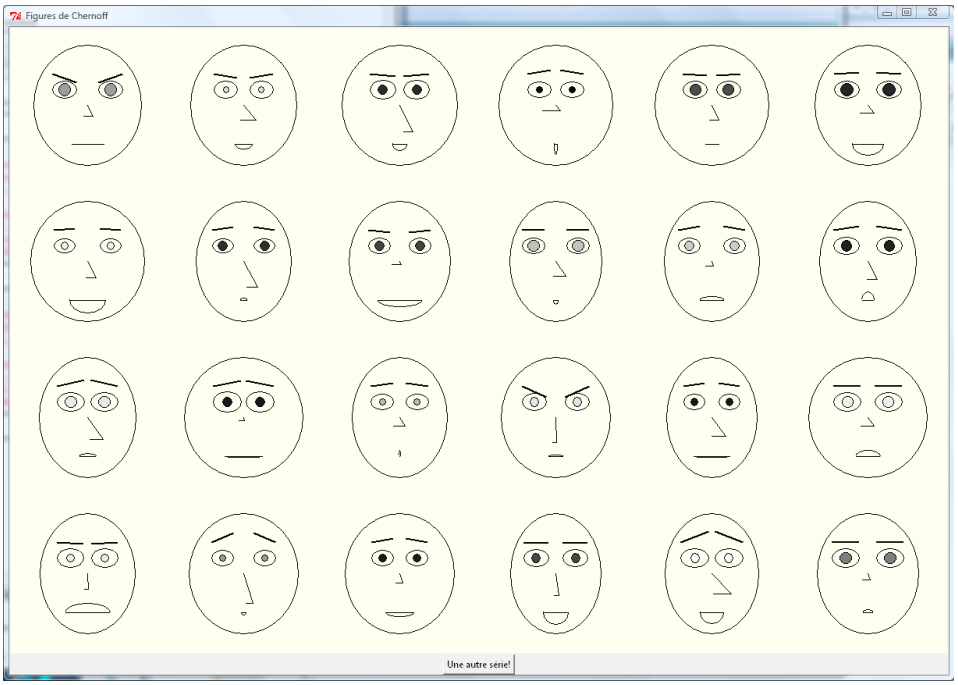
Exercice 11.8

Modifiez le programme du § 11.5. Ajoutez un 11ème paramètre *b* influant sur la forme des yeux. Inspirez-vous de la fonction `visage()`. Pour $b = 0.5$, l'œil sera parfaitement rond. Pour $0 \leq b < 0.5$, l'œil sera allongé horizontalement et pour $0.5 < b \leq 1$, l'œil sera allongé verticalement.

Exercice 11.9

Reprenez le programme chernoff.py du chapitre 9 et créez une classe `chernoff` qui permettra de définir et dessiner une figure de Chernoff.

Disposez ensuite 24 figures aléatoires sur une feuille, comme ci-après :



chernoff.py

Exercice 11.10

Modifiez le programme de l'exercice 11.9 pour lire et visualiser le fichier suivant, qui contient les notes de 12 élèves :

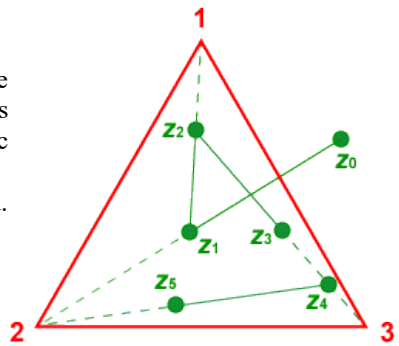
```
Lucien 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0 6.0
Paul 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
Malo 4.0 4.0 4.0 4.0 4.0 4.0 4.0 4.0 4.0 4.0
Julie 3.5 5.5 5.0 4.5 4.5 4.5 3.5 4.0 5.5 6.0
Marie-France 4.0 2.5 3.5 5.5 5.5 6.0 5.0 4.5 4.0 5.0
Géraldine 6.0 5.5 4.5 3.5 2.5 4.0 2.0 3.5 4.0 5.0
Octavia 6.0 2.5 5.5 5.5 3.5 6.0 5.0 4.5 4.0 5.0
Jules 3.5 2.5 3.5 5.5 3.5 6.0 5.0 4.5 3.0 5.0
René 6.0 2.5 3.5 3.5 3.5 1.5 5.0 4.5 4.5 6.0
Yvan 5.0 2.5 3.5 5.5 4.5 6.0 5.0 4.5 5.0 2.0
Agapanthe 2.0 2.5 3.0 5.5 5.5 5.0 5.0 5.5 4.0 3.0
Véronique 4.0 2.5 4.5 5.5 5.0 6.0 5.0 4.5 4.0 5.0
```

Écrivez le nom de l'élève sous chaque tête, puis comparez visuellement ces élèves.

Exercice 11.11

Dessinez les trois sommets d'un triangle équilatéral sur une feuille. Placez ensuite un point z_0 au hasard sur cette feuille. Vous allez maintenant construire et dessiner une suite de points avec les règles suivantes :

1. prendre l'un des trois sommets du triangle au hasard. Appelons-le s.
2. dessiner le point z_{i+1} qui est au milieu du segment $z_i s$
3. incrémenter i et retourner en 1.



a) Avant de programmer cela, essayez d'imaginer quel sera le résultat.
b) Programmez cet algorithme. Générez une suite de 50'000 points dans une fenêtre de 800 pixels de large. Qu'obtenez-vous ?

Exercice 11.12

Essayez les règles de l'exercice 11.11 avec une autre forme initiale que le triangle et/ou en modifiant l'endroit du segment où se trouvera le nouveau point (par exemple en prenant le tiers du segment plutôt que le milieu).

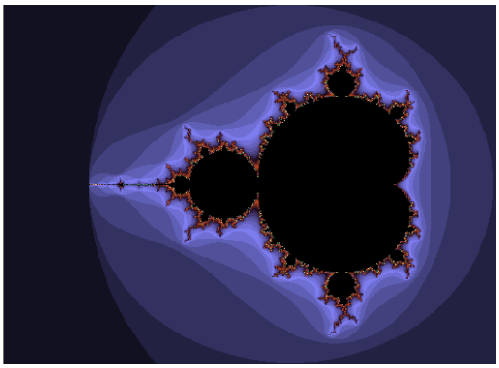
Exercice 11.13 : ensemble de Mandelbrot

Vous voyez ci-contre une célèbre figure fractale appelée « ensemble de Mandelbrot ». Elle est extrêmement complexe, et pourtant pas si difficile que cela à produire.

Mickaël Launay vous explique comment la dessiner dans sa vidéo intitulée « À la découverte de l'ensemble de Mandelbrot - Myriogon #6 » (https://www.youtube.com/watch?v=dQeIUrLKM9s).

Il utilise le langage Javascript, mais ce n'est pas une difficulté.

Suivez ses indications pour écrire votre programme en Python et amusez-vous !

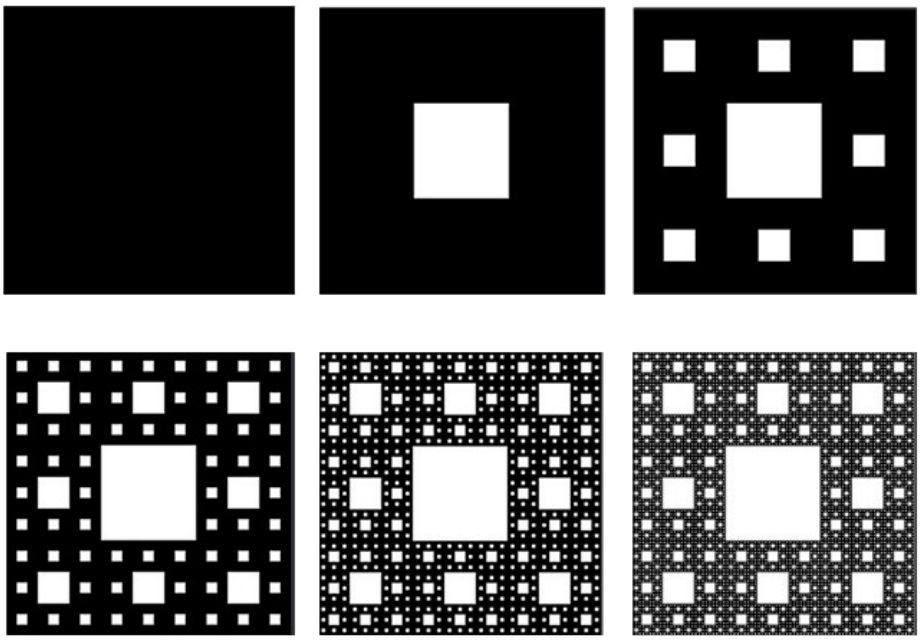


Exercice 11.14 : le tapis de Sierpinksi

Le tapis de Sierpinski est une figure fractale. L'objet de départ est un carré plein. On divis ce carré en 9 carrés égaux. On évide le carré du centre (3 fois plus petit).

À chaque itération on agit de même avec tous les carrés.

Wacław **Sierpiński**
(1882 - 1969)



Écrivez un programme Python qui reproduit cette figure.

Remarque : la procédure décrite ci-dessus n'est pas la plus facile à programmer. Il est plus simple de dessiner le tapis ligne par ligne.

Exercice 11.15 : la tortue de Logo

Logo est un langage de programmation essentiellement connu pour sa fameuse tortue graphique (qui l'a cantonné à une image faussement puérile). Logo est en fait un langage complet de haut niveau, proche du Lisp, apparu en 1967 au MIT. Pour en savoir plus sur ce langage, voir http://fr.wikipedia.org/wiki/Logo_(langage).

Logo permet de faire des dessins en donnant des ordres à un instrument appelé *tortue*. La tortue laisse une trace (ou pas) de ses déplacements. On peut imaginer qu'elle tient un crayon dans sa bouche.

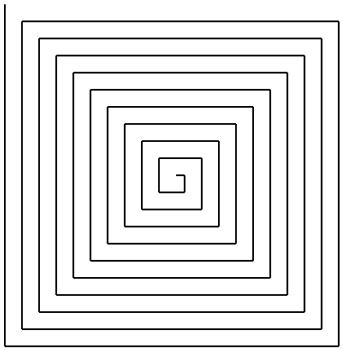
Dans cet exercice, vous allez programmer en Python certaines commandes de la tortue. Le « pré » de la tortue sera un carré de 800 pixels sur 800 pixels. Vous les utiliserez ensuite pour réaliser des figures géométriques.

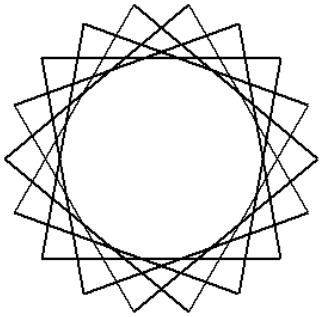
Commandes à programmer

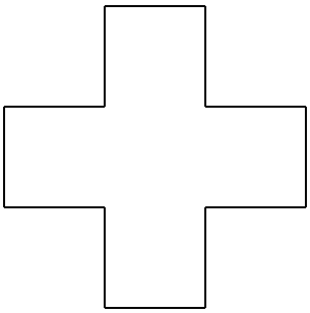
`origine`	la tortue est placée au centre du pré, aux coordonnées (400; 400)
`fpos(x0,y0)`	la tortue est positionnée en (x0; y0)
`fcap(angle0)`	la tortue s'oriente dans une direction, en degrés. 0° représente l'Est, 90° le Nord, 180° l'Ouest et 270° le Sud.
`av(d)`	la tortue avance de *d* pixels
`re(d)`	la tortue recule de *d* pixels
`td(a)`	la tortue tourne sur elle-même de *a* degrés vers la droite
`tg(a)`	la tortue tourne sur elle-même de *a* degrés vers la gauche
`lc`	lever le crayon (la tortue ne dessinera rien en se déplaçant)
`bc`	abaisser le crayon (la tortue dessinera en se déplaçant)
`cc(nom)`	choisir la couleur du crayon, donnée par son nom

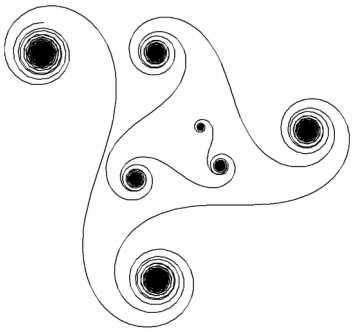
Figures à reproduire avec la tortue

Utilisez les commandes ci-dessus pour reproduire les figures suivantes. Vous pouvez les combiner à des instructions Python, par exemple pour faire des boucles.









La dernière figure semble très difficile à reproduire, et pourtant le code n'a que 11 lignes !

11.6. Ce que vous avez appris dans ce chapitre

- Les couleurs peuvent être désignées par un nom (la liste est longue...), ou par le système RVB.
- Il existe dans le module externe `tkinter` des procédures pour dessiner des segments, des rectangles, des polygones, des ovales et des secteurs. On peut modifier leurs caractéristiques (position, couleur, ...) en cours d'exécution du programme.
- On peut importer des images.
- Vous avez vu des figures fractales.