# 8 Vistas & Sonidos

*No hagan música para una audiencia vasta y desconocida o para el mercado o el rating, ni siquiera para algo tan tangible como el dinero. Aunque sea crucial ganarse la vida, esa no debería ser la inspiración. Háganlo por ustedes mismos.*
-Billy Joel

Página anterior: Be My Valentine
Arte Fractal por Sven Geier (www.sgeier.net)

Mencionamos anteriormente que la noción de computación en estos días se extiende mucho más allá de los simples cálculos numéricos. Escribir programas de control de robots es computación, al igual que hacer proyecciones acerca de la población mundial. Utilizando dispositivos como los iPods, pueden disfrutar música, videos y shows de radio y de televisión. La manipulación de sonido e imagen también pertenece al reino de la computación y en este capítulo les introduciremos el tema. Ya han visto cómo, usando el robot, pueden tomar fotos de varias escenas. También pueden tomar fotos similares de su cámara digital. Usando las técnicas computacionales básicas que han aprendido hasta ahora, verán en este capítulo cómo pueden realizar computación sobre formas y sonidos. Aprenderán a crear imágenes usando computación. Las imágenes que crearán pueden ser utilizadas para visualizar datos e incluso para fines estéticos para explorar sus facetas creativas. También presentaremos algunos fundamentos del sonido y de la música y les mostraremos cómo también pueden llevar a cabo formas similares de computación utilizando la música.

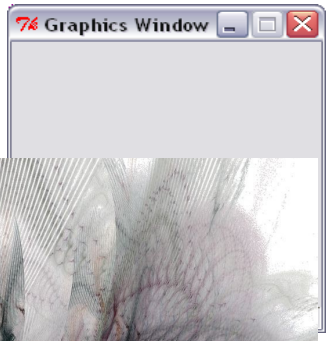
## Vistas: Dibujar

Si han usado una computadora durante cualquier cantidad de tiempo, deben haber usado algún tipo de aplicación para dibujar. Usando una típica aplicación de dibujo, pueden dibujar varias formas, colorearlas, etc. También pueden crear dibujos usando los comandos de dibujo provistos en el módulo de la librería Myro. Para

dibujar cualquier cosa, primero deben tener un lugar donde dibujarlo: un bastidor o una ventana. Pueden crear esa ventana usando el comando:

```
myCanvas = GraphWin()
```

**Una ventana de gráficos**



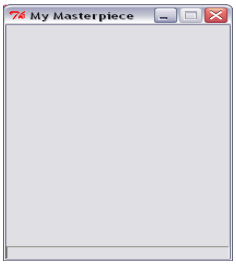

Si han ingresado el comando de arriba en el IDLE, inmediatamente verán emerger una pequeña ventana gris (ver imagen abajo a la derecha). Esta ventana les servirá como el bastidor para crear dibujos. Por defecto, la ventana creada por el comando `GraphWin` es de 200 píxeles de alto y 200 píxeles de ancho y su nombre es



`"Graphics Window"`. No es una forma muy inspiradora para comenzar, pero el comando `GraphWin` tiene algunas otras variaciones. Primero, para cerrar la pantalla, pueden usar el comando:

```
myCanvas.close()
```

**Mi obra maestra 200x300**




Para crear una ventana de gráficos de cualquier tamaño y con un nombre especificado por ustedes, pueden usar el comando de abajo:

```
myCanvas = GraphWin("Mi obra maestra", 200, 300)
```

El comando de arriba crea una ventana llamada `"Mi obra maestra"` que tendrá 200 píxeles de ancho y 300 de altura (ver imagen a la derecha). Pueden cambiar el color de fondo de un gráfico como se muestra abajo:

```
myCanvas.setBackground("white")
```

Pueden nombrar cualquiera de los colores en el comando de arriba, desde los colores comunes como “rojo”, “azul”, “gris”, “amarillo”, o colores más exóticos, desde “AntiqueWhite” (blanco antiguo) hasta “LavenderBlush” (toque de lavanda) o “WhiteSmoke” (humo blanco). Los colores se pueden crear de muchas maneras como veremos abajo. Varios miles de nombres de colores han sido pre-asignados (**Google**: lista de nombres de colores) y pueden ser usados en el comando de arriba.

Ahora que saben cómo crear una paleta (y hacerla desaparecer) e incluso cómo determinar un color de fondo, es hora de ver qué se puede dibujar dentro de ella.

Pueden crear y dibujar todo tipo de objetos geométricos: puntos, líneas, círculos, rectángulos, e incluso textos e imágenes. Dependiendo del tipo de objeto que deseen dibujar, primero deben *crearlo* y luego dibujarlo. Sin embargo, antes de hacer esto, también deben conocer el sistema de coordenadas de la ventana de gráficos.

En una ventana de gráficos con un ancho W y una altura H (por ejemplo. 200x300 píxeles), el píxel (0, 0) está en la esquina derecha superior y el píxel (199, 299) estará en la esquina derecha inferior. Es decir, las coordenadas-x aumentan a medida que van hacia la derecha y las coordenadas-y aumentan a medida que van hacia la izquierda.

El objeto más simple que pueden crear es un *punto*. Esto se hace de la siguiente manera:

```
p = Point(100, 50)
```

Es decir, p es un objeto que es un Punto cuya coordenada-x está en 100 y cuya coordenada-y está en 50. Esto sólo crea un objeto Punto. Para dibujarlo, deben enunciar el comando:

```
p.draw(myCanvas)
```

La sintaxis del comando de arriba puede parecer un poco extraña al principio. Lo vieron brevemente cuando presentamos las listas en el Capítulo 5 pero no lo desarrollamos. Definitivamente es diferente a lo que han visto hasta ahora. Pero si piensan acerca de los objetos que han visto hasta el momento: números, strings, etc., la mayoría tienen operaciones estándares definidas (como +, *, /, etc.). Pero al pensar en objetos geométricos, no hay notaciones estándares. Los lenguajes de programación como Python proveen facilidades para modelar cualquier tipo de objetos y la sintaxis que usamos acá es la sintaxis estándar que puede ser aplicada a todo tipo de objetos. La forma general del comando enunciado para un objeto es:

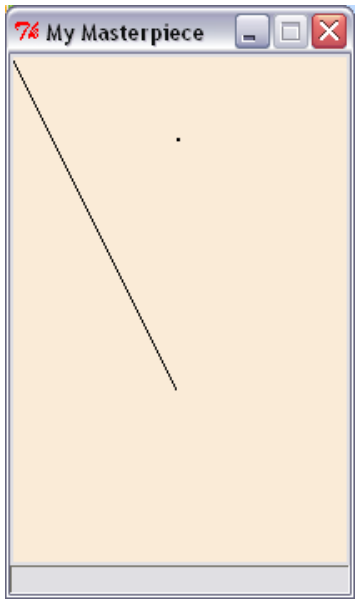
```
<objecto>.<funcion>(<parametros>)
```

Por lo tanto, en el ejemplo de arriba, <objecto> es el nombre p que anteriormente fue definido para ser un objeto Point, <funcion> es dibujar, y <parametros> es myCanvas. La función draw requiere a la ventana de gráficos como parámetro. Es decir que le están pidiendo al punto representado por p que sea dibujado en la

ventana especificada como su parámetro. Los objetos `Point` tienen otras funciones disponibles:

```
>>> p.getX()
100
>>> p.getY()
50
```

Es decir, dado un objeto `Point`, pueden obtener sus coordenadas x- e y-. Los objetos se crean usando sus *constructores* como el constructor `Point(x, y)` de arriba. Usaremos muchos constructores en esta sección para crear los objetos gráficos. Un objeto línea puede ser creado de manera similar a los objetos puntos. Una línea requiere que se especifiquen los dos puntos extremos. Por lo tanto una línea de (0, 0) a (100, 200) puede ser creada como:




```
L = Line(Point(0,0), Point(100,200))
```

Y pueden dibujar la línea usando el mismo comando de dibujo de arriba:

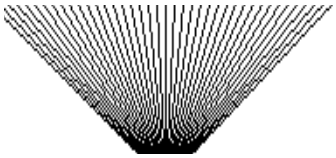
```
L.draw(myCanvas)
```

El dibujo de la derecha muestra los dos objetos que hemos creado y dibujado hasta el momento. Con respecto a `Point`, pueden obtener los valores de los puntos extremos de una línea:

```
>>> inicio = L.getP1()
>>> inicio.getX
0
>>> fin = L.getP2()
>>> fin.getY()
200
```

Aquí hay un pequeño loop de Python que puede ser usado para crear y dibujar varias líneas:

```
for n in range(0, 200, 5):
    L=Line(Point(n,25),Point(100,100))
    L.draw(myCanvas)
```



En el loop de arriba (los resultados se muestran a la derecha), el valor de `n` comienza en 0 y aumenta de a 5 después de cada iteración hasta llegar a 200 sin incluirlo 200 (por ej. 195). Para cada valor de `n` se crea un objeto `Line` (línea) nuevo con las coordenadas de inicio `(n, 25)` y el punto final en `(100, 100)`.

**Realizar la siguiente actividad:** Prueben todos los comandos introducidos hasta el momento. Luego observen los efectos producidos por el loop de arriba. Modifiquen el incremento de 5 en el loop de arriba a diferentes valores (1, 3, etc.) y observen el efecto. Luego, prueben el siguiente loop:

```
for n in range(0, 200, 5):
    L = Line(Point(n, 25), Point(100, 100))
```

```
    L.draw(myCanvas)
    wait(0.3)
    L.undraw()
```

La función de `undraw` (desdibujar) hace exactamente lo que implica el nombre. En el loop de arriba, para cada valor que toma `n`, se crea una línea (como se observa arriba), se dibuja, y luego, después de una espera de 0.3 segundos, se borra. Nuevamente, modifiquen el valor del incremento y observen el efecto. Prueben también modificar la cantidad de tiempo en el comando `wait`.

También pueden dibujar varias formas geométricas: círculos, rectángulos, óvalos y polígonos. Para dibujar un círculo (o cualquier forma geométrica), primero deben crearla y después dibujarla:

```
C = Circle(centerPoint, radius)
c.draw(myCanvas)
```

`centerPoint` es un objeto `Point` y su radio está especificado en píxeles. Por lo tanto, para dibujar un círculo centrado en (100, 150) con un radio de 30, pueden usar los siguientes comandos:

```
C = Circle(Point(100, 150), 30)
c.draw(myCanvas)
```

Los rectángulos y los óvalos se dibujan de modo similar (ver detalles al final del capítulo). Todos los objetos geométricos tienen muchas funciones en común. Por ejemplo, pueden obtener el punto central de un círculo, un rectángulo o un óvalo utilizando el comando:

```
centerPoint = C.getCenter()
```

De manera predeterminada, todos los objetos se dibujan en negro. Hay varias formas de modificar o especificar los colores de los objetos. Para cada objeto pueden especificar n color para su contorno así como un color de relleno. Por ejemplo, para dibujar un círculo centrado en (100, 150), radio 30, contorno color rojo, y relleno color amarillo:

```
C = Circle(Point(100, 150), 30)
C.draw(myCanvas)
C.setOutline(“red”)
C.setFill(“yellow”)
```

Por cierto, `setFill` (determinar relleno) y `setOutline` (determinar contorno) tienen el mismo efecto en los objetos `Point` y `Line` (dado que no hay lugar para rellenar con ningún color). Además, la línea o el contorno dibujado es siempre se un píxel de ancho. Pueden modificar este ancho usando el comando

```
setWidth(<pixels>):
C.setWidth(5)
```

El comando de arriba modifica el ancho del contorno del círculo a 5 píxeles.

**Realizar la siguiente actividad:** Prueben todos los comandos introducidos aquí. Además, miren al final del capítulo para ver detalles sobre cómo dibujar otras formas.

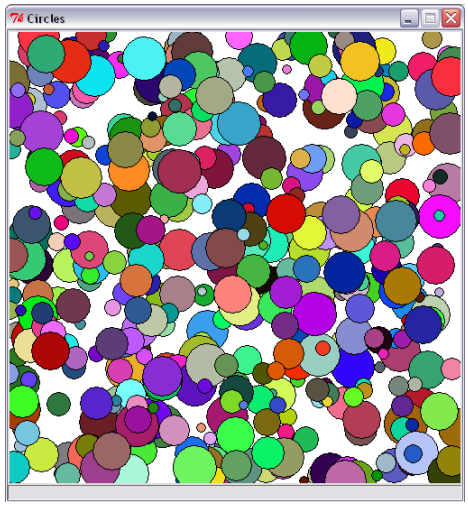
Anteriormente mencionamos que varios colores tienen nombres asignados que pueden ser usados para seleccionar colores. También pueden crear colores propios especificando sus valores de rojo, verde y azule. En el Capítulo 5 mencionamos que cada color está formado por tres valores: RGB o valores de color rojo, verde y azul. Cada uno de estos valores puede estar en rango de 0..255 y se llama un valor-color 24-bit. Con esta manera de especificar colores, el color con valores (255, 255, 255) (es decir rojo = 255, verde = 255, y azul = 255) es blanco; (255, 0, 0) es rojo puro, (0, 255, 0), es azul puro, (0, 0, 0) es negro, (255, 175, 175) es rosa, etc. Pueden tener tantos colores como 256x256x256 (¡más de 16 millones de colores!). Dados los valores RGB específicos, pueden crear un color nuevo usando el comando color_rgb:

```
myColor = color_rgb(255, 175, 175)
```

**Realizar la siguiente actividad:** El programa de abajo dibuja varios círculos de tamaños al azar con colores al azar. Pruébenlo y observen los resultados. Una pantalla de muestra se puede ver a la derecha (abajo). Modifiquen el programa para incorporar un número para la cantidad de círculos que se dibujen. `randrange(m,n)` devuelve un número al azar en el rango`[m..n-1]`.

```
# Program to draw a bunch of # random colored circles
from myro import *
from random import *
def makeCircle(x, y, r):
    # creates a Circle centered at point (x, y) of radius r
    return Circle(Point(x, y), r)

def makeColor():
    # creates a new color using random RGB values
    red = randrange(0, 256)
    green = randrange(0, 256)
    blue = randrange(0, 256)
    return color_rgb(red, green,blue)
def main():
    # Create and display a
    # graphics window
    width = 500
    height = 500
    myCanvas =
GraphWin('Circles',width,height)
    myCanvas.setBackground("white")
    # draw a bunch of random
    # circles with random
    # colors.
    N = 500
    for i in range(N):
        # pick random center
        # point and radius
        # in the window
        x = randrange(0,width)
        y = randrange(0,height)
        r = randrange(5, 25)
        c = makeCircle(x, y, r)
```



```
        # select a random color
        c.setFill(makeColor())
    c.draw(myCanvas)
main()
```

Observen nuestro uso de las funciones para organizar el programa. Desde la perspectiva del diseño, las dos funciones `makeCircle` (crear círculo) y `makeColor` (crear color) se escriben de manera diferente. Esto es sólo para fines ilutrativos. Podrían, por ejemplo, definir `makeCircle` del mismo modo que `makeColor` para que no tome ningún parámetro y genere los valores de x, y, y radio de la siguiente manera:

```
def makeCircle():
    # creates a Circle centered at point (x, y) of radius r
    x = randrange(0,width)
    y = randrange(0,height)
    r = randrange(5, 25)

    return Circle(Point(x, y), r)
```

Desafortunadamente, aunque este cambio se ve muy sencillo, la función no va a andar. Para generar los valores de x y de y necesita conocer el ancho y la altura de la ventana de gráficos. Pero el ancho y la altura no están disponibles ni accesibles en la función de arriba. Este es un tema de alcance de nombres en un programa Python: ¿Cuál es el alcance de accesibilidad de un nombre en un programa?

Python define el alcance de un nombre en forma *textual* y por *léxico*. Es decir, cualquier nombre está visible en el texto del programa/función *después* de que ha sido definido. Observen que la noción de después es una noción textual. Es más, Python restringe la accesibilidad de un nombre al texto de la función en la que está definido. Es decir, los nombres ancho y altura están definidos dentro de la función main y por lo tanto no están visibles en ningún lugar fuera de main. De modo similar, la variable rojo, verde y azul se consideran locales a la definición de `makeColor` (formar color) y no están accesibles fuera de la función `makeColor`.

Entonces, ¿cómo puede `makeCircle`, si han decidido que generará los valores `x` e `y` relativos al tamaño de la ventana, obtener el acceso al ancho y a la altura de la ventana? Hay dos soluciones para esto. Primero, pueden hacerlos pasar como parámetros. En ese caso, la definición de `makeCircle` sería:

```
def makeCircle(w, h):
    # creates a Circle centered at point (x, y) of radius r
    # such that (x, y) lies within width, w and height, h
    x = randrange(0,w)
    y = randrange(0,h)
    r = randrange(5, 25)

    return Circle(Point(x, y), r)
```

Entonces, la forma en que usarían la función de arriba en el programa main sería usando el comando:

```
C = makeCircle(width, height)
```

Es decir, pasan los valores de `width` (ancho) y `height` (altura) a `makeCircle` como parámetros. La otra forma de definir `makeCircle` sería exactamente como se mostró en primera instancia:

```
def makeCircle():
    # creates a Circle centered at point (x, y) of radius r
    x = randrange(0,width)
    y = randrange(0,height)
    r = randrange(5, 25)

    return Circle(Point(x, y), r)
```

Sin embargo, moverían las definiciones de `width` y `height` afuera y antes de las definiciones de todas las funciones:

```
from myro import *
from random import *
width = 500
height = 500
def makeCircle():
    …
def makeColor():
  …
def main():
  …
```

Dado que las variables se definen fuera de cualquier función y antes de las definiciones de las funciones que las utilizan, pueden acceder a su valor. A esta altura pueden estar preguntándose ¿cuál es la mejor versión? O incluso, ¿para qué molestarse? La primera versión era igualmente buena. La respuesta a esta pregunta es similar a la forma de escribir un párrafo en un ensayo. Se pueden escribir párrafos de muchas maneras. Algunas versiones serán preferibles a otras. En programación, la primera regla que se usa en el alcance de los nombres es: asegurarse de que solamente las partes del programa  que se supone que tienen acceso al nombre tengan permitido el acceso. Esto es similar a la razón por la cual no compartirían su contraseña con nadie, o su código de tarjeta bancaria, etc. En nuestra segunda solución, hicimos que los nombres `width` y `height` estén *globalmente* visibles para el programa entero que sigue a continuación. Esto implica que aún `makeColor` puede tener acceso a ellos ya sea si lo necesita o no.

Quizás a esta altura quieran argumentar: ¿cuál es la diferencia si se hacen esas variables visibles para `makeColor` mientras uno se cuide de no usarlas en esa función? Tienen razón, no hay diferencia. Pero requiere una responsabilidad extra de su parte para asegurarse de que así sea. ¿Pero qué les garantiza que alguien que está modificando el programa no lo haga?

Usamos un programa simple para ilustrar decisiones simples pero potencialmente peligrosas que pueblan el panorama de la programación como minas de tierra. Los programas pueden romperse si nombres como estos son *mal utilizados* en un programa. Peor aún, los programas no se rompen, sino que conducen a resultados incorrectos. Sin embargo, en el nivel de decidir qué variables hacer locales y cuáles hacer globales, las decisiones son simples, y uno sólo necesita ejercitar prácticas de programación seguras. Concluiremos esta sección de gráficos con algunos ejemplos de creación de animaciones.

Cualquier objeto dibujado en la ventana de gráficos puede ser movido usando el comando `move(dx, dy)`. Por ejemplo, para mover el círculo 10 píxeles a la derecha y 5 píxeles hacia abajo, pueden usar el comando:

```
C.move(10, 5)
```

**Realizar la siguiente actividad:** Escribamos un programa que mueva el círculo de manera aleatoria en la ventana de gráficos. Primero, ingresen el siguiente programa y pruébenlo.

```
# Moving circle; Animate a circle...
from myro import *
from random import *
def main():
    # create and draw the graphics window
    w = GraphWin(“Moving Circle”, 500, 500)
    w.setBackground(“white”)
    # Create a red circle
    c = Circle(Point(250, 250), 50)
    c.setFill(“red”)
    c.draw(w)
    # Do a simple animation for 200 steps
    for i in range(200):
        c.move(randrange(-4, 5), randrange(-4, 5))
        wait(0.2)
main()
```

Observen en el programa de arriba, que estamos moviendo el círculo alrededor de manera aleatoria en las direcciones x- e y-. Prueben cambiar el rango de los movimientos y observen el comportamiento. Prueben modificar los valores para que el círculo sólo se mueva en dirección horizontal o sólo en dirección vertical. También observen que hemos tenido que *desacelerar* la animación insertando el comando wait (espera) después de cada movimiento. Comenten el comando wait y vean qué sucede. Puede parecer que no sucedió nada pero en realidad ¡los 200 movimientos pasaron tan rápido que sus ojos no pudieron registrar un solo movimiento! Usando esto de base, podemos ahora escribir un programa más interesante. Observen el programa de abajo:

```
# Moving circle; Animate a circle...
from myro import *
from random import *
def main():
    # create and draw the graphics window
    winWidth = winHeight = 500
    w = GraphWin(“Bouncing Circle”, winWidth, winHeight)
    w.setBackground(“white”)
    # Create a red circle
    radius = 25
    c = Circle(Point(53, 250), radius)
    c.setFill(“red”)
    c.draw(w)

    # Animate it
    dx = dy = 3
    while timeRemaining(15):
        # move the circle
        c.move(dx, dy)
```

```
        # make sure it is within bounds
        center = c.getCenter()
        cx, cy = center.getX(), center.getY()
        if (cx+radius >= winWidth) or (cx-radius <= 0):
            dx = -dx
        if (cy+radius >= winHeight) or (cy-radius <= 0):
            dy = -dy
wait(0.01)
main()
```

Durante 15 segundos, verán un círculo rojo rebotando alrededor de la ventana. Estudien el programa de arriba para ver cómo mantenemos el círculo dentro de la ventana todo el tiempo y cómo la dirección del rebote de a pelota se modifica. Cada vez que cambien la dirección, hagan que la computadora emita un beep:

```
computer.beep(0.005, 440)
```

Si los entusiasma la posibilidad de animar un círculo, imaginen lo que pueden hacer si tienen muchos círculos y otras formas animadas. Además, conecten el controlador *game pad* y vean si pueden controlar el círculo (o cualquier otro objeto) usando los controles del mismo. Esto es muy similar a controlar el robot. Diseñen un juego interactivo de computadora que aproveche esta nueva modalidad de input. También pueden diseñar juegos multi-usuarios dado que pueden conectar múltiples controladores game pad a su computadora. Vean la documentación de Referencias para más detalles sobre cómo obtener input de varios controladores game pad.

## Dibujar Textos & Imágenes

Así como se pueden ubicar formas, también se pueden colocar textos en la ventana de gráficos. La idea es la misma. Primero crean el texto usando el comando:

```
myText = Text(<anchor point>, <string>)
```

y luego lo dibujan.

Pueden especificar la tipografía, tamaño y estilo del texto. No entraremos en esos detalles aquí. Se sintetizan en la referencia al final del texto. Cuando tengamos la oportunidad, usaremos estas funciones abajo en otros ejemplos.

Las imágenes en este sistema son tratadas igual que cualquier otro objeto. Pueden crear una imagen usando el comando `Image`:

```
myPhoto = Image(<centerPoint>, <file name>)
```

Deben tener preparada una imagen en uno de los formatos de imágenes comunes (como JPEG, GIF, etc.) y alojado en un archivo (`<file name>`). Una vez que el objeto imagen se ha creado, puede ser dibujado, borrado o movido, al igual que las otras formas.

Lo que ustedes hagan con esta funcionalidad recién descubierta depende de su creatividad. Pueden usar gráficos para visualizar algunos datos: ploteo del crecimiento de la población mundial, por ejemplo; o crear arte, un juego interactivo, o incluso una historia animada. Pueden combinar su robot, el controlador game pad, e incluso sonido para enriquecer la experiencia multi-media. Los ejercicios al final

del capítulo presentan algunas ideas para explorar esto aún más. Abajo, ahondaremos en los sonidos.

## Sonido

Como hemos visto, pueden hacer que el robot haga *beeps* recurriendo a la función `beep()`, así:

```
beep(1, 440)
```

Este comando lo instruye al robot a reproducir un tono de 440 Hz por 1 segundo. Tratemos primero de analizar qué hay en el tono de *440 Hz*. Primero, las letras *Hz* son una abreviación de *Hertz*. El nombre proviene de un físico alemán, Heinrich Rudolph Hertz, quien fue pionero en el trabajo de la producción de ondas electromagnéticas en el siglo 19. Hoy usamos el **Hertz** (o *Hz*) como unidad para especificar frecuencias.

$1 Hertz = 1 cycle / second$

El uso más común de las frecuencias en la actualidad es para especificar las velocidades del reloj de la CPU de la computadora. Por ejemplo, una típica PC hoy en día corre a la velocidad reloj de pocos GigaHertz (or GHz).

$1 GigaHertz = 10^9 cycles / second$

Las frecuencias se relacionan con movimientos o vibraciones periódicas (repetitivas). El tiempo que le lleva a un movimiento o vibración repetirse se llama *período de tiempo*. La frecuencia y el período de tiempo están inversamente relacionados. Es decir, se llama frecuencia al número de ciclos o repeticiones en un segundo. Por lo tanto, 1 Hertz hace referencia a cualquier movimiento o vibración que se repite cada 1 segundo. En el caso de la frecuencia del reloj de la computadora, una computadora que corre a 4 Gigahertz ¡está repitiendo 4 billones de veces por segundo! Otros ejemplos de movimientos periódicos incluyen: la rotación de la tierra sobre su eje (1 ciclo cada 24 * 60 * 60 = 86400 segundos o a la frecuencia de 0.00001157 ciclos/segundo), un típico CD de audio da vueltas 400 veces por segundo, una unidad de CD en su computadora valorado en 52x gira el CD 52 * 400 = 20800 veces por segundo, los colibríes pueden batir sus alas a frecuencias que oscilan entre 20-78 veces/segundo (¡algunas llegan hasta 200!).

El sonido es una compresión y refracción (o regreso a su estado original) periódica de aire (para simplificarlo, asumamos que el medio es el aire). Un ciclo de un sonido comprende una compresión y una refracción. Por lo tanto, un beep a 440 Hz representa 440 ciclos completos de compresión y refracción. Generalmente, un oído humano es capaz de escuchar frecuencias en un rango de 20 Hz a 20000 Hz (o 20 Kilo Hertz). Sin embargo, la capacidad varía de persona a persona. Además, muchos dispositivos electrónicos no son capaces de producir frecuencias en ese rango completo. 20-20KHz se considera alta-fidelidad para un estéreo o componentes de audio *home theater*. Examinemos primero el rango de sonidos audibles que puede producir el Scribbler. Para sacar un sonido del Scribbler, deben darle una frecuencia y duración (en segundos) en que debe ser reproducido el sonido. Por ejemplo, para emitir un sonido de 440 Hz por 0.75 segundos:

```
beep(0.75, 440)
```

El oído humano es capaz de distinguir sonidos que difieren solamente por pocos Hertz (tan poco como 1 Hz) sin embargo, esta habilidad varía de persona a persona. Prueben los comandos:

```
beep(1, 440)
beep(1, 450)
```

¿Pueden distinguir entre los dos tonos? A veces ayuda ubicarlos en un loop para que puedan escuchar los tonos alternados repetidamente como para poder distinguirlos entre sí. El siguiente ejercicio puede ayudarlos a determinar qué frecuencias son capaces de distinguir.

**Ejercicio:** Usando el ejemplo de arriba, intenten ver cuánto se acercan a distinguir frecuencias cercanas. Como se sugirió antes, quizás deseen emitir los tonos alternándolos durante 5-10 segundos. Comiencen con 440 Hz. ¿Pueden notar la diferencia entre 440 y 441? 442? etc. Una vez que han establecido su alcance, prueben otra frecuencia, digamos 800. ¿Es la misma la distancia que pueden distiguir?

**Realizar la siguiente actividad:** Pueden programar al Scribbler para crear una sirena, repitiendo dos tonos diferentes (bastante similar al ejemplo de arriba). Deberán experimentar con distintos pares de frecuencias (pueden estar cercar o muy distantes entre sí) para producir una sirena que suene realista. Escriban su programa para emitir una sirena durante 15 segundos. ¡Cuánto más fuerte el volumen mejor!

También pueden hacer que Myro emita directamente un beep desde la computadora, en lugar del robot, con el comando:

```
computer.beep(1, 440)
```

Desafortunadamente, no pueden hacer que el robot y la computadora toquen a dúo. ¿Por qué not? Prueben estos comandos:

```
beep(1, 440)
computer.beep(1, 440)
beep(1, 880)
computer.beep(1, 880)
beep(1, 440)
computer.beep(1, 440)
```

¿Qué ocurre? Prueben sus soluciones a los ejercicios de arriba emitiendo los sonidos desde la computadora en lugar de desde el Scribbler.

## Escalas Musicales

En la música occidental, una *escala* está dividida en 12 notas (de 7 notas mayores: ABCDEFG –la, si, do, re, mi, fa, sol). Además hay octavas Una octava en C se compone de las notas que se muestran abajo:

C C#/Db D D#/Eb E F F#/Gb G G#/Ab A A#/Bb B

C# (pronunciada “C sostenido”) es el mismo tono que Db (pronunciada “D bemol”).

Las frecuencias correspondientes a una nota específica, digamos C, se multiplican (o dividen) por 2 para obtener la misma nota en una octava más alta (o más baja). En un piano hay disponibles varias octavas en el despliegue de teclas. ¿Cuál es la relación entre estos dos tonos?

```
beep(1, 440)
beep(1, 880)
```

El segundo tono es exactamente una octava del primero. Para subir un tono una octava, simplemente multiplican la frecuencia por 2. Del mismo modo, para hacer que un tono sea una octava más bajo, lo dividen por 2. Las notas que indican una octava pueden anotarse de la siguiente manera:

```
C0 C1 C2 C3 C4 C5 C6 C7 C8
```

Es decir, C0es la nota para C en la octava más baja (or 0). La quinta octava (numerada 4) es comúnmente referida como la octava del medio. Por lo tanto, C es la nota C en la octava del medio La frecuencia que corresponde a C es 261.63 Hz. Intenten emitirla en el Scribbler. También prueben C5 (523.25) que es dos veces la frecuencia de C4 y C3 (130.815). En sintonías comunes (*iguales temperamentos*) las 12 notas son equidistantes. Por lo tanto, si la frecuencia duplica cada octava, cada nota sucesiva es de una distancia de $2^{1/12}$. Es decir, si C4 es 261.63 Hz, C# (o Db) será:

C#4/Db4 = $261.63 * 2^{\frac{1}{12}} = 277.18$

Por lo tanto, podemos computar todas las sucesivas frecuencias:

*D*4 = $277.18 * 2^{1/12} = 293.66$

D#4/Eb = $293.66 * 2^{\frac{1}{12}} = 311.13$

etc.

El tono más bajo que puede reproducir el Scribbler es A0 y el más alto es C8. A0 tiene una frecuencia de 27.5 Hz, y C8 tiene una frecuencia de 4186 Hz. ¡Eso es un rango bastante alto! ¿Pueden escuchar el espectro completo?

```
beep(1, 27.5)
beep(1, 4186)
```

**Ejercicio:** Escriban un programa Scribbler para reproducir las 12 notas en una octava usando la computación de arriba. Pueden asumir en su programa que C0 es 16.35 y luego usar eso para computar todas las frecuencias en una octava dada (C4 es $16.35 * 2^4$). Su programa debería utilizar una octava (un número de 0 hasta 8), produzcan todas las notas en esa octava y también impriman una tabla de frecuencia para cada nota en esa octava.

## Hacer Música

Reproducir canciones por frecuencias es bastante molesto. Myro tiene un conjunto de funciones para abstraer este proceso. Una canción Myro es un *string* de caracteres compuesto de la siguiente manera:

NOTE1 [NOTE2] WHOLEPART

donde `[]` significa opcional. Cada una de estas notas /acordes está compuesto sobre su propia línea, o separado por comas donde:

NOTE1 is either a frequency or a NOTENAME (NOTE1 es o una frecuencia o un NOTENAME- nombre de nota)

NOTE2 is the same, and optional. Use for Chords. (NOTE2 es lo mismo, y opcional. Usar para acordes).

WHOLEPART is a number representing how much of

a whole note to play. (WHOLEPART es un número que representa cuánto reproducir de una nota entera).

`NOTENAMES` son *strings* insensibles a las mayúsculas o minúsculas (*case insensitive*). Aquí hay una escala entera de `NOTENAMES`:

C C#/Db D D#/Eb E F F#/Gb G G#/Ab A A#/Bb B C

Esta es la octava predeterminada. También es la quinta octava, que también puede escribirse así:

C5 C#5/Db5 D5 D#5/Eb5 E5 F5 F#5/Gb5 G5 G#5/Ab5 A5 A#5/Bb5 B5 C6

El Formato Myro Song (Canción Myro) replica las teclas del piano, por lo cual va desde A0 hasta C8. La octava del medio en un teclado es la número 4, pero usamos la 5 como la octava predeterminada. Ver http://en.wikipedia.org/wiki/Piano_key_frequencies para más detalles. Aquí hay una escala:

“C 1; C# 1; D 1; D# 1; E 1; F 1; F# 1; G 1; G# 1; A 1; A# 1; B 1; C 1;”

La escala, una octava más baja, y tocada como una polka:

“C4 1; C#4 ½; D4 ½; D#4 1; E4 ½; F4 ½; F#4 1; G4 ½; G#4 ½; A4 1; A#4 ½; B4 ½; C4 1;”

También hay algunos nombres de notas especiales más, incluyendo `PAUSE(pausa)`, `REST (descanso)`, pueden dejar el número de octava apagado de las notas de octava predeterminadas si lo desean. Usen “#” para los sostenidos, y “b” para los bemoles.

`WHOLEPART` puede ser una notación decimal o una división. Por ejemplo:

Ab2 .125

o

Ab2 1/8

Representa el A bemol en la segunda octava (dos más abajo que del medio).

A modo de ejemplo, prueben reproducir lo siguiente:

c 1

c .5

c .5

c 1

c .5

c .5

e 1

c .5

c .5

c 2

e 1

e .5

e .5

e 1

e .5

e .5

g 1

e .5

e .5

e 2

¿Lo reconocen?

Pueden dejar líneas en blanco, y los comentarios deben comenzar con un signo #. Las líneas también pueden estar separadas por comas.

## Usando una canción

Para el siguiente ejercicio, necesitarán tener un objeto para reproducir la canción. Necesitarán iniciar el robot de una manera un poco distinta. En lugar de:

```
initialize()
```

hagan lo siguiente:

```
robot = Scribbler()
```

Ahora que tienen una canción, probablemente la querrán reproducir. Si su canción está en un archivo, pueden leerla:

```
s = readSong(filename)
```

y reproducirla en el robot:

```
robot.playSong(s)
```

o en la computadora:

```
computer.playSong(s)
```

También pueden usar `makeSong(text)` para hacer una canción. Por ejemplo:

```
s = makeSong(“c 1; d 1; e 1; f 1; g 1; a 1; b 1; c7 1;”)
```

y luego reproducirla como se muestra arriba.

Si quieren reproducir la canción más rápido o más despacio, podrían cambiar todos los números `WHOLENOTE`. Pero si sólo queremos modificar el tempo, hay una manera más sencilla:

```
robot.playSong(s, .75)
```

El segundo argumento para `playSong` es la duración de una nota entera en segundos. El tempo estándar toca una nota entera en aproximadamente .5 segundos. Los números más grandes se reproducirán más lentamente, y los números más chicos se reproducirán más rápido.

## Síntesis

Pueden usar la ventana de gráficos como modo de visualizar cualquier cosa. En la ventana de gráficos pueden dibujar pueden dibujar todo tipo de formas: puntos, líneas, círculos, rectángulos, óvalos, polígonos, textos e incluso imágenes. También pueden animar estas formas como lo deseen. Lo que hagan con estas capacidades básicas de dibujos está limitado solamente por su creatividad y sus habilidades como programadores. Pueden combinar las vistas creadas con sonidos y otras interfaces, como el controlador game pad, o incluso su robot. Las funcionalidades multimedia introducidas en este capítulo pueden ser usadas en todo tipo de situaciones para crear programas interesantes.

## Referencia Myro

Abajo sintetizamos todos los comandos de gráficos mencionados en este capítulo. Los instamos a revisar el manual de referencias para más funcionalidades de gráficos.

`GraphWin()`
`GraphWin(<title>, <width>, <height>)`
Devuelve un objeto ventana de gráficos. Crea una ventana de gráficos con titulo, `<title>` y dimensiones `<width>` x `<height>`. Si no se especifican parámetros, la ventana creada es de 200x200 píxeles.

`<window>.close()`
Closes the displayed graphics window <window>.

`<window>.setBackground(<color>)`
Establece el color de fondo de la ventana al color especificado. `<color>` puede ser un color nombrado (Google: lista de nombres de colores –*color name list*), o un nuevo color creado usando el comando `color_rgb` (ver abajo).

`color_rgb(<red>, <green>, <blue>)`
Crea un nuevo color usando los valores `<red>`, `<green>`, y `<blue>` especificados. Los valores pueden estar dentro del rango 0..255.

`Point(<x>, <y>)`
Crea un objeto punto en la ubicación (`<x>`, `<y>`) en la ventana.

`<point>.getX()`
`<point>.getY()`
Devuelve las coordenadas *x* e *y* del objeto punto `<point>`.

`Line(<start point>, <end point>)`
Crea un objeto línea en `<start point>` que termina en `<end point>`.

`Circle(<center point>, <radius>)`
Crea un objeto círculo centrado en `<center point>` con un radio de `<radius>` píxeles.

`Rectangle(<point1>, <point2>)`
Crea un objeto rectángulo con esquinas opuestas ubicadas en `<point1>` y `<point2>`.

`Oval(<point1>, <point2>)`
Crea un objeto óvalo en la caja definida por los puntos de la esquina `<point1>` y `<point2>`.

`Polygon(<point1>, <point2>, <point3>,…)`
`Polygon([<point1>, <point2>, …])`
Crea un polígono con los puntos dados como vértices.

`Text(<anchor point>, <string>)`
Crea un texto anclado (la esquina inferior izquierda del texto) en `<anchor point>`. El texto en sí mismo se define por `<string>`.

`Image(<centerPoint>, <file name>)`
Crea una imagen centrada en `<center point>` del archivo de imágenes `<file name>`. La imagen puede ser en formato GIF, JPEG, o PNG.

Todos los objetos gráficos responden a los siguientes comandos:

`<object>.draw(<window>)`
Dibuja el `<object>` en la ventana de gráficos especificada `<window>`.

`<object>.undraw()`
Deshace el dibujo `<object>`.

`<object>.getCenter()`
Devuelve el punto central del `<object>`.

`<object>.setOutline(<color>)`
`<object>.setFill(<color>)`
Establece el color de delineado y de relleno del `<object>` en el `<color>` especificado.

`<object>.setWidth(<pixels>)`
Establece el ancho del delineado del `<object>` en `<pixels>`.

`<object>.move(<dx>, <dy>)`
Mueve el objeto `<dx>`, `<dy>` de su posición actual.

Las siguientes funciones relacionadas con el sonido se presentaron en este capítulo.

`beep(<seconds>, <frequency>)`
`beep(<seconds>, <f1>, <f2>)`
Hace que el robot genere un beep durante la frecuencia de tiempo `<seconds>` especificada. Pueden especificar una sola frecuencia `<frequency>` o una mezcla de dos: `<f1>` y `<f2>`.

`<robot/computer object>.beep(<seconds>, <frequency>)`
`<robot/computer object>.beep(<seconds>, <f1>, <f2>)`
Hace que el robot o la computadora realice un beep durante el tiempo `<seconds>` en la frecuencia especificada. Pueden especificar una sola frecuencia `<frequency>` o una mezcla de los dos: `<f1>` y `<f2>`.

`robot.playSong(<song>)`
Reproduce la <song> (canción) en el robot.

readSong(<filename>)
Lee un archivo de canción de <filename>.

`song2text(song)`
Convierte una <song> al formato texto.

`makeSong(<text>)`
`text2song(<text>)`
Convierte el `<text>` (texto) en un formato de canción.

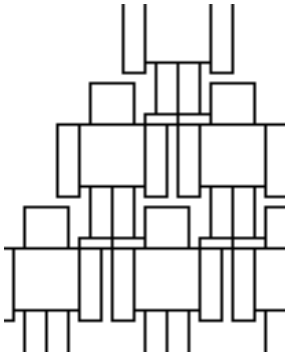
## Referencias Python

En este capítulo presentamos *alcances de las normas* informales para nombres en programas Python. Mientras que estas se pueden complicar bastante, para nuestros fines deben conocer la distinción entre un *nombre local* que es local dentro de la esfera de una función, versus un *nombre global* definido por fuera de la función. El ordenamiento del texto define lo que es accesible.

## Ejercicios

**1.** Usando los comandos de gráficos aprendidos en este capítulo, escriban un programa para generar un dibujo estacional: invierno, verano, playa o una escena festiva.

**2.** Escriban un programa que tenga la función `drawRobot(x, y, w)` de modo tal que dibuje la figura de un robot como el que se muestra al lado. El robot está ubicado en (`x`, `y`) y está dibujado en la ventana, `w`. También noten que el robot está hecho enteramente de rectángulos (8 rectángulos).



**3.** Usando la función `drawRobot` del ejercicio previo, dibujen una pirámide de robots como se muestra en el dibujo a la derecha.

**4.** Supongan que cada robot del ejercicio previo está coloreado usando uno de los colores: rojo, azul, verde y amarillo. Cada vez que se dibuja un robot, utiliza el siguiente color en la secuencia. Una vez que se terminó la secuencia, la recicla. Escriban un programa para dibujar la pirámide de arriba de manera tal que los robots aparezcan en estos colores a medida que son dibujados. Pueden decidir modificar `drawRobot` para tener un parámetro adicional: `drawRobot(x, y, c, w)` o pueden escribirlo como para que `drawRobot` decida qué color elegir. Completen ambas versiones y comparen los programas resultantes. Discutan lo méritos y/o debilidades de estas versiones con sus amigos [Una ayuda: Usen una lista de nombres de colores.]

**5.** El comando beep puede reproducir dos tonos simultáneamente. En este ejercicio demostraremos un fenómeno interesante. Primero, prueben los siguientes comandos:

```
beep(2, 660)
beep(2, 665)
```

Quizás les resulte posible diferenciar los dos tonos. A continuación, prueben reproducir los dos tonos juntos usando el comando:

```
beep(2, 660, 665)
```

Escucharán latidos pulsantes. Este fenómeno se llama *pulsación* y es muy común cuando se reproducen tonos puros juntos. Expliquen por qué sucede esto.

**6.** Realicen una búsqueda en la Web acerca de *fractales* y escriban programas para dibujar algunos fractales simples. Por ejemplo, *Copos de Nieve Koch*, *Triángulos Sierpinski*, *Conjuntos de Mandelbrot*, *Conjuntos de Julia*, etc.