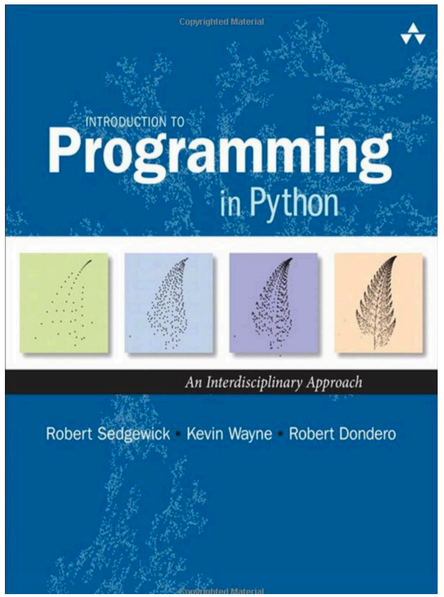
Parte II: Computación científica

Clase 16: Scientific computing with Python

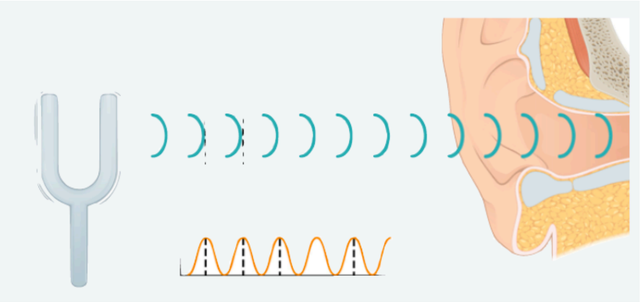
Diego Caro
dcaro@udd.cl



Basada en presentaciones oficiales de libro Introduction to Programming in Python (Sedgewick, Wayne, Dondero).

Disponible en https://introcs.cs.princeton.edu/python

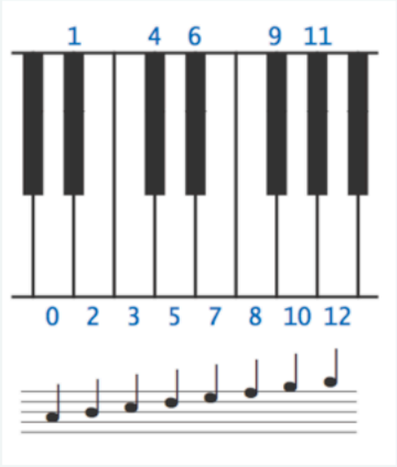
Sonido



- El **sonido** es la percepción de la vibración de moléculas.
- Un **tono musical** es un sonido periódico.
- Un **tono puro** es una onda sinusoidal.

Western musical scale

- Concert A is 440 Hz.
- 12 notes, logarithmic scale.



pitch	*i*	*frequency* $(440*2^{i/12})$	*sinusodial waveform*
A	0	440	
A♯ / B♭	1	466.16	
B	2	493.88	
C	3	523.25	
C♯ / D♭	4	554.37	
D	5	587.33	
D♯ / E♭	6	622.25	
E	7	659.26	
F	8	698.46	
F♯ / G♭	9	739.99	
G	10	783.99	
G♯ / A♭	11	830.61	
A	12	880	

http://www.cs.princeton.edu/courses/archive/fall18/cos126/lectures/CS.5.Functions.pdf

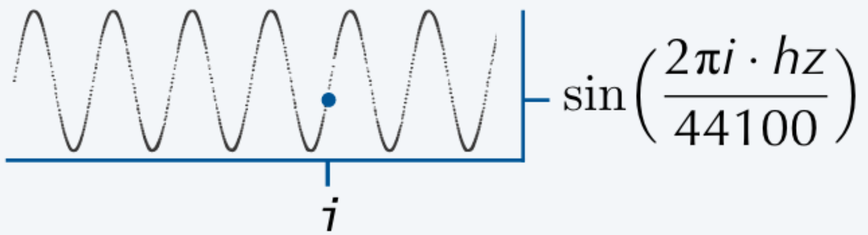
Audio digital

- Para representar una onda en el computador se debe "sample" en intervalos regulares.
- El computador solo puede representar números, "sampling" permite transformar la onda a una serie de números.

	samples/sec	*samples*	*sampled waveform*
1/40 second of concert A	5,512	137	
	11,025	275	
	22,050	551	
CD standard →	44,100	1102	

Hola mundo módulo stdaudio

```
 1 import math
 2 import stdaudio
 3 import sys
 4
 5 def tone(hz, duration):
 6     n = int(44100 * duration)
 7     note = [0.0]*(n+1)
 8     for i in range(n+1):
 9         note[i] = math.sin(2.0 * math.pi * i * hz / 44100)
10     stdaudio.playSamples(note)
11
12 hz = float(sys.argv[1])
13 duration = float(sys.argv[2])
14 tone(hz, duration)
```



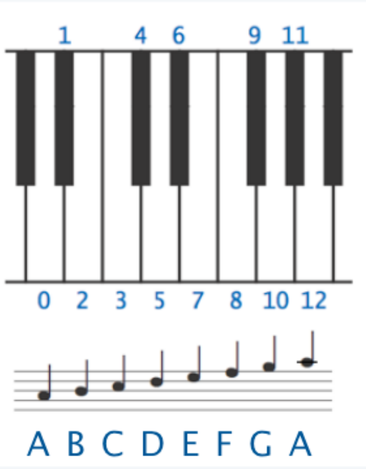

```
python3 playthatnote.py 440.0 3.0
python3 playthatnote.py 880.0 3.0
python3 playthatnote.py 220.0 3.0
python3 playthatnote.py 494.0 3.0
```

Reproducir canción

```
 1 import math
 2 import stdio # this is new!
 3 import stdaudio
 4
 5 SPS = 44100
 6 CONCERT_A = 440.0
 7 NOTES_ON_SCALE = 12.0
 8
 9 while not stdio.isEmpty():
10     pitch = stdio.readInt()
11     duration = stdio.readFloat()
12     hz = CONCERT_A * (2.0 ** (pitch / NOTES_ON_SCALE))
13     n = int(SPS * duration)
14     note = [0.0]*(n+1)
15     for i in range(n+1):
16         note[i] = math.sin(2.0 * math.pi * i * hz / SPS)
17     stdaudio.playSamples(note)
18
19 stdaudio.wait()
```

Lee desde teclado y convierte automáticamente a entero/float.

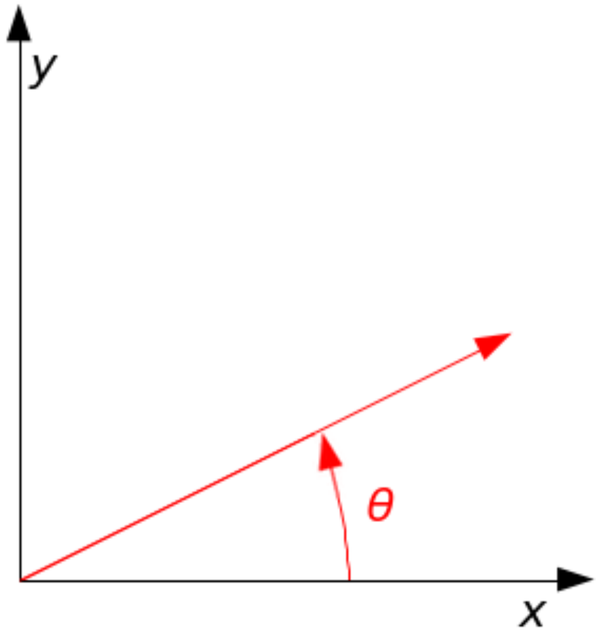
```
$ head elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
```



bouncingball-deluxe.py
balldeluxe.py
playthatnote.py

Rotaciones

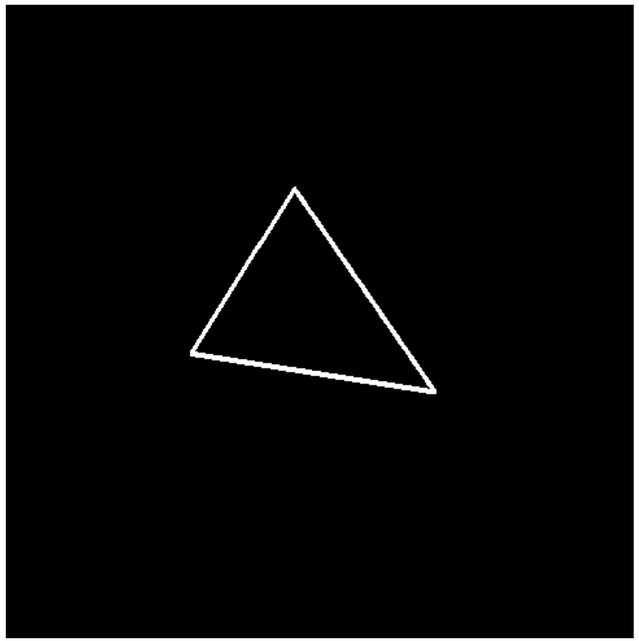
$$x' = x\cos\theta - y\sin\theta\,,$$
$$y' = x\sin\theta + y\cos\theta\,.$$

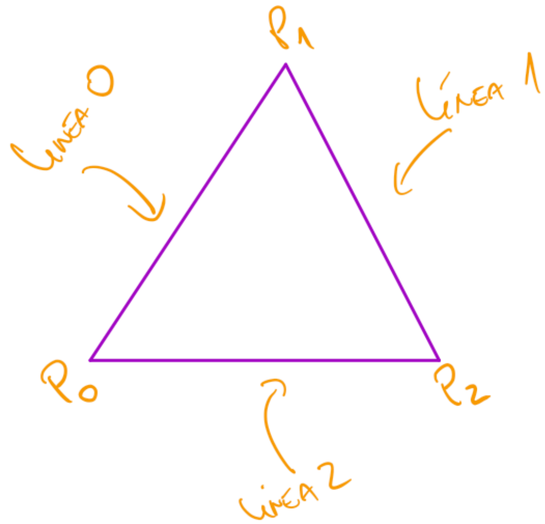


https://en.wikipedia.org/wiki/Rotation_matrix

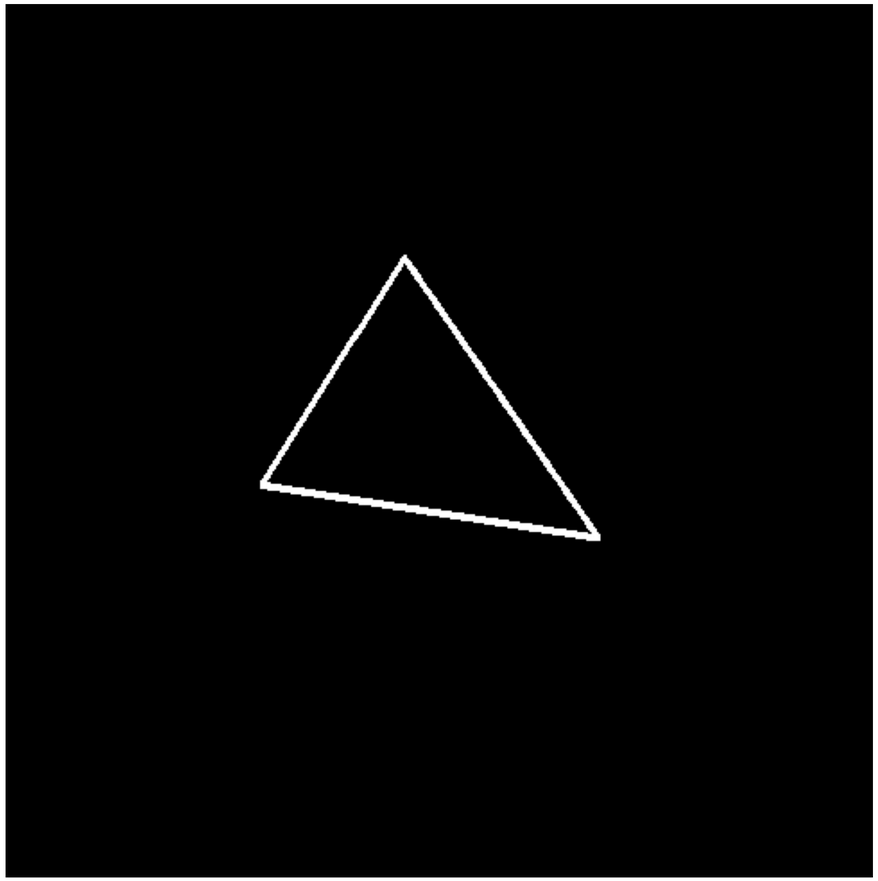
Ejemplo: rotando una nave espacial

- La nave espacial es un triángulo.
  - ... pero stddraw no dibuja triángulos! 😱
  - Podemos dibujarla usando tres líneas
- Luego rotamos los 3 puntos del triángulo, y boom!

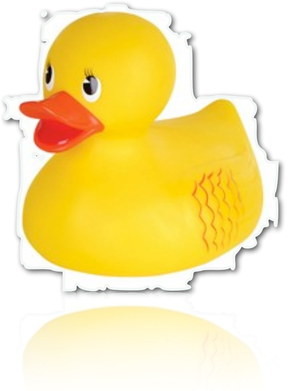




```python
import stddraw
from math import cos, sin

stddraw.setCanvasSize(500, 500)

stddraw.setXscale(-1.0, 1.0)
stddraw.setYscale(-1.0, 1.0)

points_x = [-0.3, 0  ,  0.3]
points_y = [-0.3, 0.4, -0.3]

# post-condicion: número de puntos en x es la misma que en y
assert len(points_x) == len(points_y)

n = len(points_x)

# velocidad angular
speed_rot = 0.1 # en radianes

while True:
    stddraw.clear(stddraw.BLACK)
    stddraw.setPenColor(stddraw.WHITE)

    # calculate rotations
    for i in range(n):
        newx = points_x[i]*cos(speed_rot) - points_y[i]*sin(speed_rot)
        newy = points_x[i]*sin(speed_rot) + points_y[i]*cos(speed_rot)
        points_x[i] = newx
        points_y[i] = newy

    # display triangle
    stddraw.polygon(points_x, points_y)

    # copy buffer to screen
    stddraw.show(0)
    stddraw.pause(20)
```



Dibuja un polígono con la lista de coordenadas en x y la lista de coordenadas en y



Rotaciones con matrices

```python
import stddraw
import numpy as np

stddraw.setCanvasSize(500, 500)

stddraw.setXscale(-1.0, 1.0)
stddraw.setYscale(-1.0, 1.0)

points = np.array([[-0.3, -0.3], [0, 0.3], [0.3, -0.3]])
n = points.shape[0] #numero de filas en matriz point

# velocidad angular
speed_rot = 0.1 # en radianes
matrix_rot = np.array([ [np.cos(speed_rot),-np.sin(speed_rot)],
                        [np.sin(speed_rot), np.cos(speed_rot)] ])

while True:
    stddraw.clear(stddraw.BLACK)
    stddraw.setPenColor(stddraw.WHITE)

    # calculate rotations
    for i in range(n):
        points[i] = matrix_rot.dot(points[i])
    # display triangle
    stddraw.polygon(points[:,0], points[:,1])

    # copy buffer to screen
    stddraw.show(0)
    stddraw.pause(20)
```

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

Cada fila representa un punto del polígono que queremos dibujar.