Buzzer

A buzzer creates vibrations that produce sound. When an electrical current flows through the buzzer's coil, it creates a magnetic field that moves a small magnet back and forth rapidly, causing a contact to vibrate and create sound waves in the air that we can hear as a buzzing sound.

PWM Frequency and Duty Cycle

Frequency of the sound waves changes the tone of the buzzer.
Our ears can detect changes in the tone (frequency) between 10 and 10000 cycles per second.
Duty Cycle sets the volume of the buzzer. The range is technically 0 to 65535 (using 16 bit binary) to represent. However, anything below 5000 may be very difficult to detect. Typically, a duty cycle of **10000** appears to be the optimal level for buzzers.
A 0 value represents silence.

```
#Library
from machine import Pin, PWM
import time
#setup
buzzer=PWM(Pin(0))

#Algorithm
buzzer.freq(1000)
buzzer.duty_u16(10000)
time.sleep(2)
buzzer.duty_u16(0)

```

Musical Notes are created varying the **frequency** of the sound wave.

	0	1	2	3	4	5	6	7	8
C	16.35	32.7	65.41	130.81	261.63	523.25	1046.5	2093	4186
C#	17.32	34.65	69.3	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.63
D#	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489	4978
E	20.6	41.2	82.41	164.81	329.63	659.25	1318.51	2637	5274
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#	23.12	46.25	92.5	185	369.99	739.99	1479.98	2959.96	5919.91
G	24.5	49	98	196	392	783.99	1567.98	3135.96	6271.93
G#	25.96	51.91	103.83	207.65	415.3	830.61	1661.22	3322.44	6644.88
A	27.5	55	110	220	440	880	1760	3520	7040
A#	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951	7902.13

https://muted.io/note-frequencies #see column 5

Each musical note consists on a duty cycle (how loud to play), the key , a play time (whole note, half note or quarter note) and a pause in between the notes in a song.

musicalnote.py

```
import time
#setup
buzzer=PWM(Pin(0))

#Do  C key
buzzer.duty_u16(10000)  #duty cycle
buzzer.freq(523)        #frequency
time.sleep(.2)          #play time
buzzer.duty_u16(0)      #stop
time.sleep(.1)          #stop time

#Re  D key
buzzer.duty_u16(10000)
buzzer.freq(587)
time.sleep(.2)
buzzer.duty_u16(0)
time.sleep(.1)

#Mi  E Key
buzzer.duty_u16(10000)
buzzer.freq(659)
time.sleep(.2)
buzzer.duty_u16(0)
time.sleep(.1)
```

We can create signature tunes. For example, a tune for Door Opening / and reverse for Door Closing

See. **signaturetune.py**

twinkle1.py

```
#Library
from machine import PWM, Pin
import time
#setup
buzzer=PWM(Pin(0))
#notes variables
A = 440
As = 466
B = 494
C = 523
Cs = 554
D = 587
Ds = 622
E = 659
F = 698
Fs = 740
G = 784
Gs = 830

volume=10000
def playtone(note,vol,playtime,stoptime):
    buzzer.duty_u16(vol)
    buzzer.freq(note)
    time.sleep(playtime)
    buzzer.duty_u16(0)
    time.sleep(stoptime)


#twinkle twinkle litte star the hard way
playtone(C,volume,.2,.1)
playtone(C,volume,.2,.1)
playtone(G,volume,.2,.1)
playtone(G,volume,.2,.1)
playtone(A,volume,.2,.1)
playtone(A,volume,.2,.1)
playtone(G,volume,.2,.2)

playtone(F,volume,.2,.1)
playtone(F,volume,.2,.1)
playtone(E,volume,.2,.1)
playtone(E,volume,.2,.1)
playtone(D,volume,.2,.1)
playtone(D,volume,.2,.1)
playtone(C,volume,.2,.1)
```

twinkletwinkle.py

```
#Library
from machine import PWM, Pin
import time
#setup
buzzer=PWM(Pin(0))

notes = dict(
A = 440,
As = 466,
B = 494,
C = 523,
Cs = 554,
D = 587,
Ds = 622,
E = 659,
F = 698,
Fs = 740,
G = 784,
Gs = 830)

volume=10000

def playtone(note,vol,playtime,stoptime):
   buzzer.duty_u16(vol)
   buzzer.freq(note)
   time.sleep(playtime)
   buzzer.duty_u16(0)
   time.sleep(stoptime)

def playline(song):
   for x in range(len(song)):
      print(int(notes[song[x]]),song[x])
      playtone(int(notes[song[x]]),volume,.3,.2)
   time.sleep(.3)

#duty cycle is the volume
duty_cycle=volume #0 to 65535
buzzer.duty_u16(duty_cycle)

#frequency is the tone
#https://noobnotes.net/twinkle-twinkle-little-star-traditional/
#twinkle twinkle little star the easier way
playline('CCGGAAG')
playline('FFEEDDC')
playline('GGFFEED')
playline('GGFFEED')
playline('CCGGAAG')
playline('FFEEDDC')
```

#This is a ChatGPT generated program for twinkle twinkle little stars **AiTwinkle.py**

```
import time
from machine import Pin, PWM
# Define the GPIO pin number for the speaker
speaker_pin = 0
# Define the frequency of each note
#https://muted.io/note-frequencies/
#see column 5
C = 262
D = 294
E = 330
F = 349
G = 392
A = 440
B = 494
C2 = 523
# Define the duration of each note
quarter_note = 0.25
half_note = 0.5
whole_note = 1
# Define the notes of Twinkle Twinkle Little Star
twinkle_twinkle = [
   (C2, quarter_note), (C2, quarter_note), (G, quarter_note),
   (G, quarter_note), (A, quarter_note), (A, quarter_note),
   (G, half_note),
   (F, quarter_note), (F, quarter_note), (E, quarter_note),
   (E, quarter_note), (D, quarter_note), (D, quarter_note),
   (C2, half_note),
   (G, quarter_note), (G, quarter_note), (F, quarter_note),
   (F, quarter_note), (E, quarter_note), (E, quarter_note),
   (D, half_note),
   (G, quarter_note), (G, quarter_note), (F, quarter_note),
   (F, quarter_note), (E, quarter_note), (E, quarter_note),
   (D, half_note),
   (C2, quarter_note), (C2, quarter_note), (G, quarter_note),
   (G, quarter_note), (A, quarter_note), (A, quarter_note),
   (G, half_note),
   (F, quarter_note), (F, quarter_note), (E, quarter_note),
   (E, quarter_note), (D, quarter_note), (D, quarter_note),
   (C2, half_note),
]
# Initialize the PWM object for the speaker
speaker_pwm = PWM(Pin(speaker_pin))
# Set the PWM frequency to the highest possible value
speaker_pwm.freq(10000)
# Play each note of Twinkle Twinkle Little Star
for note in twinkle_twinkle:
   frequency, duration = note
   speaker_pwm.freq(frequency)
```

```
speaker_pwm.duty_u16(10000)
time.sleep(duration)
speaker_pwm.duty_u16(0)
time.sleep(0.05)
```