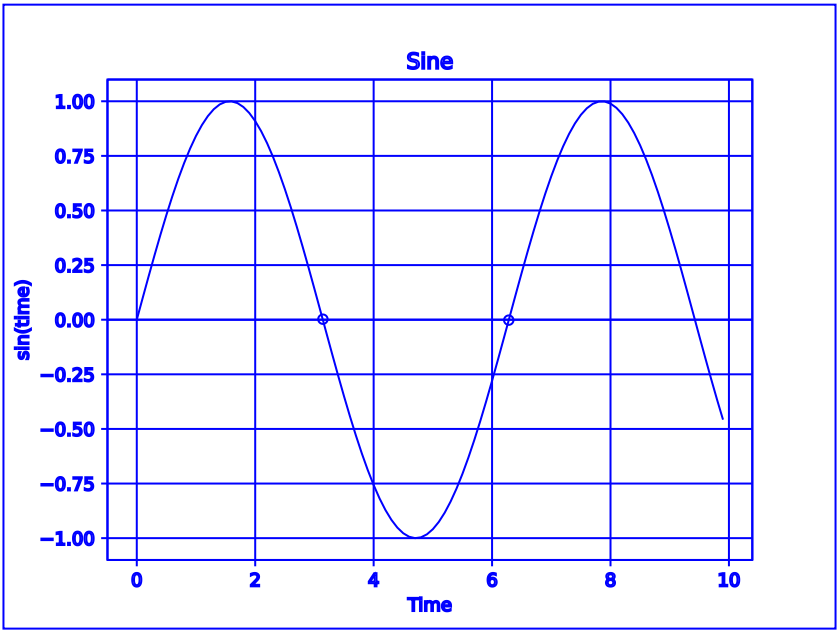
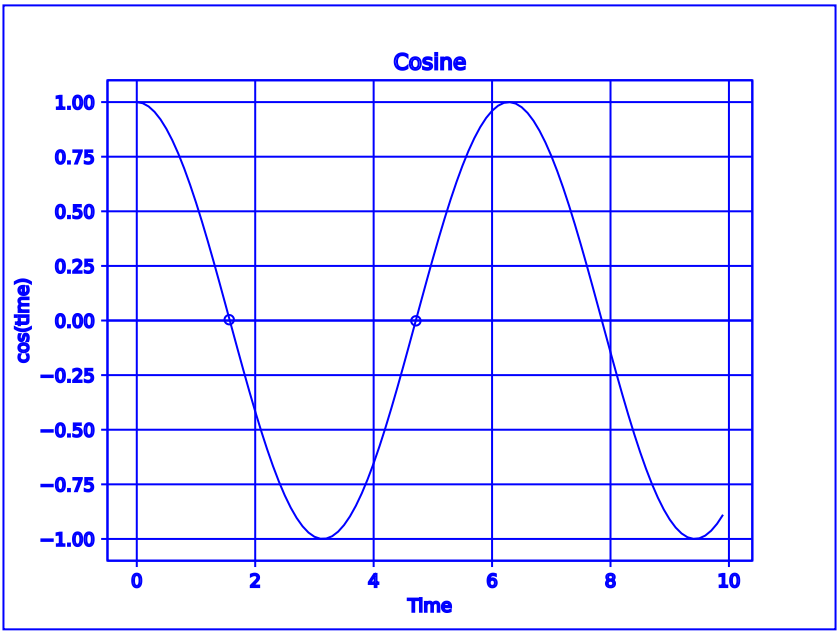
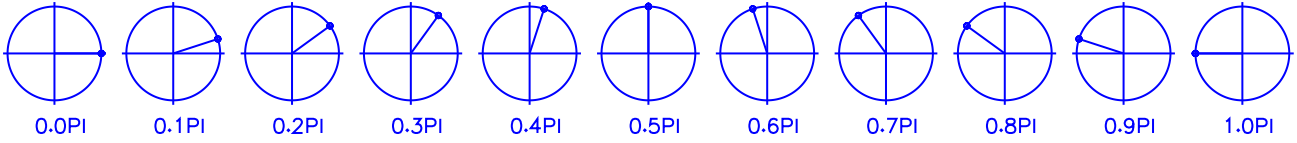
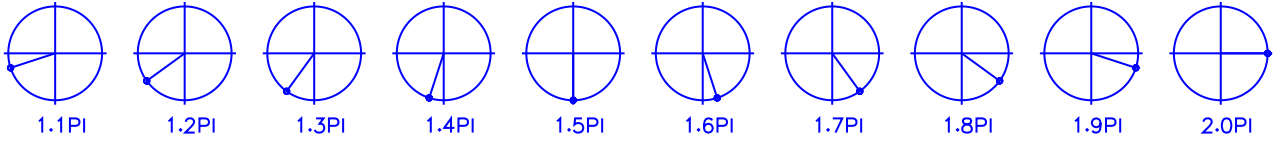
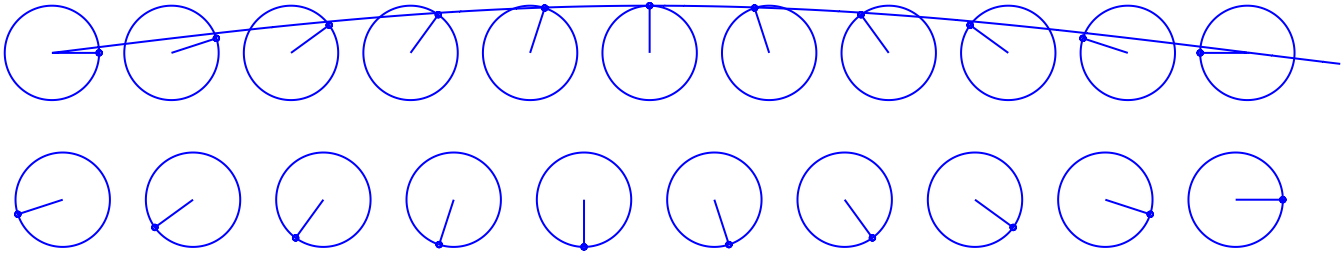

Sine
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
sin(time)
0
2
4
6
8
10
Time


Cosine
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
cos(time)
0
2
4
6
8
10
Time





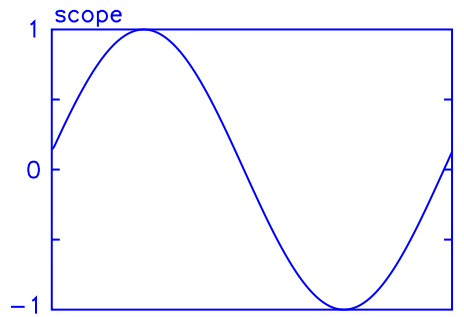


sine and cosine – are functions revealing the shape of a right triangle. Looking out from a vertex with angle theta, sin(theta) is the ratio of the opposite side to the hypotenuse, while cos(theta) is the ratio of the adjacent side to the hypotenuse.

$$\sin(\theta) = \frac{\text{Opp}}{\text{Hyp}}$$

$$\cos(\theta) = \frac{\text{Adj}}{\text{Hyp}}$$

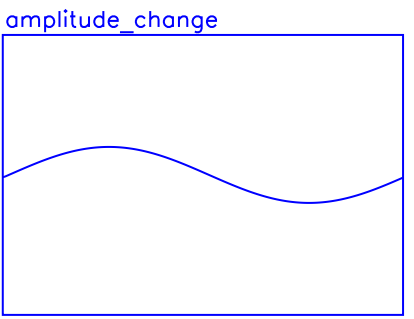
Generating a sine tone in Pure Data




```
; scope yticks 0 0.5 1
; scope ylabel −1 −1 0 1
```

```
osc~ 1
tabwrite~ scope
```

// this array has 44,100 collected samples as the X axis.




```
osc~ 1
*~ 0.2
tabwrite~ amplitude_change
```

// multiplying the signal with a value that is less than 1 will lower the amplitude.

Sine as a function of time (t):

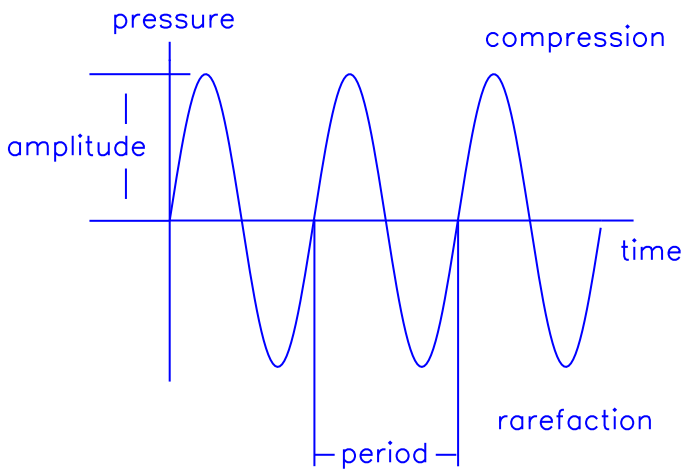
$$y(t) = A\sin(2\pi ft + \varphi) = A\sin(\omega t + \varphi)$$

A, amplitude, the peak deviation of the function from zero.

f, ordinary frequency, the number of oscillations (cycles) that occur each second of time.

ω, $2\pi f$, angular frequency, the rate of change of the function argument in units of radians per second.

φ, phase, specifies (in radians) where in its cycle the oscillation is at t = 0.

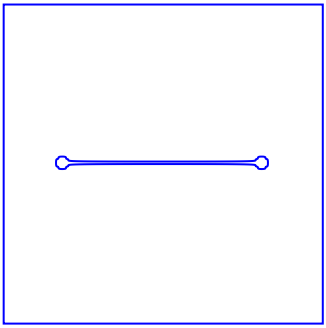


In matplotlib:

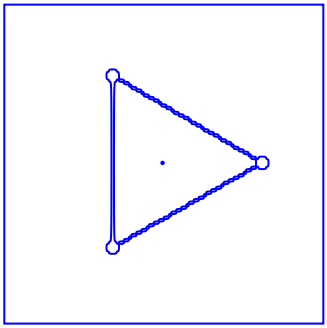
```
# parameters
time = np.arange(0, math.pi*6, 0.1) # x axis
frequency = 1 # 1 Hz
phase = 0 # phase (in radians)

# A 1 Herz sine wave. Completes a cycle every second.
y = np.sin((math.pi*2)*frequency*time + phase)
```

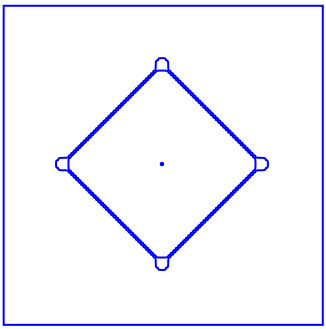
vertices = 2



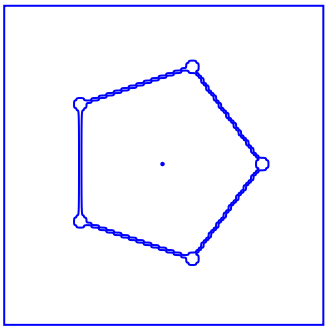
vertices = 3



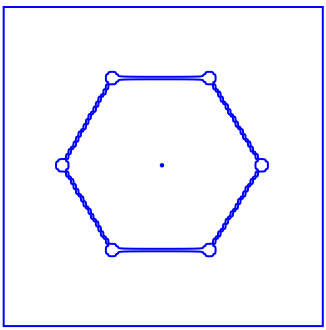
vertices = 4



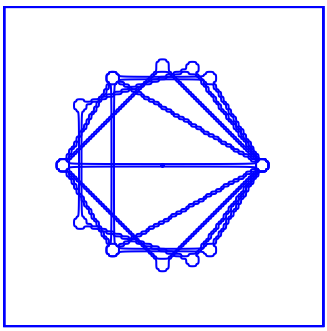
vertices = 5



vertices = 6

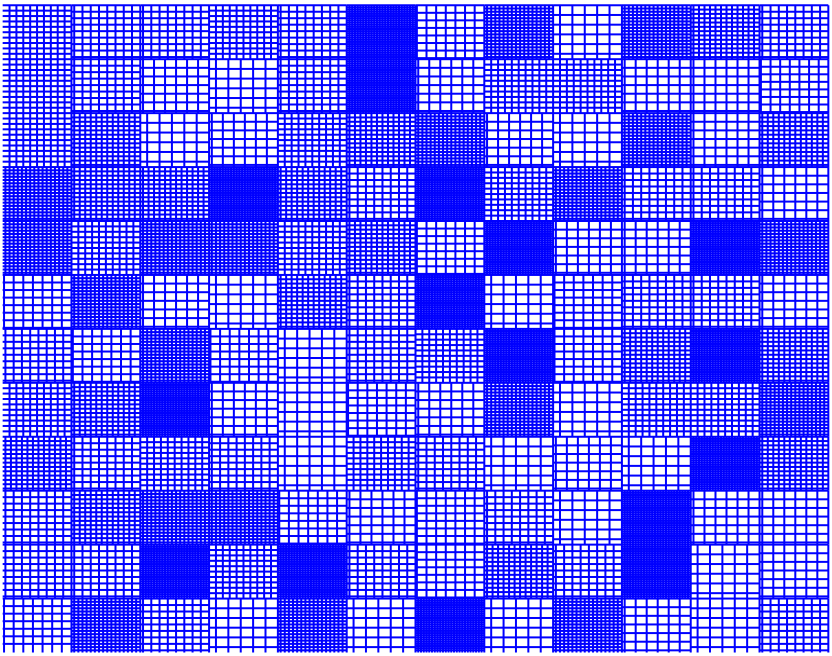


combined



```
radius=40
vertices=0
max_vertices=6
t=1

function _draw()
  cls()
  -- connect vertices with lines
  for angle=0, vertices do
    if (angle>0 and angle < vertices) then
      line(cos(angle/vertices)*radius+63,
           sin(angle/vertices)*radius+63,
           cos((angle-1)/vertices)*radius+63,
           sin((angle-1)/vertices)*radius+63, 6)
    end
    -- when line is <= 1 away from last vertex
    if (angle>=vertices-1 and angle<=vertices) then
      line(cos(angle/vertices)*radius+63,
           sin(angle/vertices)*radius+63,
           cos((0)/vertices)*radius+63,
           sin((0)/vertices)*radius+63, 6)
    end
  end
  -- draw vertices
  for angle=0, vertices do
    circfill(cos(angle/vertices)*radius+63,
             sin(angle/vertices)*radius+63, 2, 8)
  end
  -- draw midpoint
  pset(63, 63)
  -- fluctuate number of vertices using sin()
  vertices = (sin(t) * (max_vertices-1)/2) +
               (max_vertices-1)/2 + 1
  t+=0.004
end
```



Processing

```
void draw() {

  for(int x=0; x<width; x+=pixelsize) {
    for(int y=0; y<height; y+=pixelsize) {
      int randomValue= floor(random(2, pixelsize/5));
      for (int x2=x; x2<x+pixelsize; x2+=randomValue) {
        line(x2,y,x2,y+pixelsize);
      }
      for (int y2=y; y2<y+pixelsize; y2+=randomValue) {
        line(x,y2,x+pixelsize,y2);
      }
    }
  }

}
```

generating white noise