Python Mode for Processing

Download Processing at

<https://processing.org/download/>

Excellent documentation at

<https://processing.org/reference/> and <http://py.processing.org/reference/>

Example Sketches

Random rectangles:

|  |  |
| --- | --- |
|  | Tools:  setup  draw  random  fill  rect  mouseX  mouseY |

def **setup**():

    size(600,600)

def **draw**():

    fill(random(255),

         random(255),

         random(255))

    rect(mouseX, mouseY,

              random(100),

              random(150))

Tree Fractal:

|  |  |
| --- | --- |
|  | Tools:  Recursion  translate  rotate  mouseX  mouseY |

def **setup**():

    size(600,600)

def **draw**():

    background(255)

    translate(width/2,height-50)

    rotate(PI)

    level = mouseX/40

    angle = mouseY/4.0

    tree(100,angle,level)

def tree(sz,angle,level):

    if level > 0:

        line(0,0,0,sz)

        translate(0,sz)

        rotate(radians(angle))

        tree(sz\*.8,angle,level -1)

        rotate(radians(-2\*angle))

        tree(sz\*.8,angle,level -1)

        rotate(radians(angle))

        translate(0,-sz)

def tree2(sz,level):#asymmetrical tree

    if level > 0:

        line(0,0,0,sz)

        translate(0,sz)

        rotate(radians(30))

        tree(sz\*.8,level -1)

        rotate(radians(-50))

        tree(sz\*.9,level -1)

        rotate(radians(20))

        translate(0,-sz)

The Color Grid

|  |  |
| --- | --- |
|  | Tools:  colorMode(HSB)  noStroke  nested loop to draw a grid  dist |

gridsz = 20

def **setup**():

    size(600,600)

    colorMode(HSB)

    noStroke()

def **draw**():

    background(0)

    sz = width/float(gridsz)

    for x in range(gridsz):

        for y in range(gridsz):

            d = dist(sz\*x,sz\*y,mouseX,mouseY)

            ratio = 0.95

            fill(0.5\*d,255,255)

            rect(sz\*x,sz\*y,sz\*ratio,sz\*ratio)

The Wavy Grid of Squares

|  |  |
| --- | --- |
|  | Tools:  Embedded Loop for creating a grid  map  sin  Time Variable |

sz = 10 #gridsize

t = 0

def **setup**():

    size(600,600)

    noStroke()

def **draw**():

    global t

    cellsz = width/sz

    background(255)

    for y in range(sz):

        for x in range(sz):

            col = map(sin(t+x+y),-1,1,0,255)

            fill(col)

            rect(x\*cellsz,y\*cellsz,cellsz,cellsz)

    t += 0.1

The Rotating Triangles:

|  |  |
| --- | --- |
|  | Tools:  Time Variable  pushMatrix  popMatrix  translate  rotate  beginShape  endShape  vertex  mouseX |

'''Rotating Triangles

June 3, 2018'''

t = 0

num\_tris = 120

def **setup**():

    size(600,600)

    noFill()

def **draw**():

    global t,num\_tris

    background(255)

    translate(width/2,height/2)

    for i in range(num\_tris):

        pushMatrix()

        rotate(i\*TWO\_PI/float(num\_tris))

        translate(200,0)

        shift = map(mouseX,0,width,0,5.0)

        rotate(shift\*(t+i))

        tri(100)

        popMatrix()

    t += 0.005

def tri(sz):

    '''Draws an Equilateral Triangle,

    located at its center of size sz'''

    beginShape()

    for i in range(3):

        vertex(sz\*cos(i\*TWO\_PI/3),

               sz\*sin(i\*TWO\_PI/3))

    endShape(CLOSE)

The Ball Trail:

|  |  |
| --- | --- |
|  | Tools:  sin  cos  saveFrame  Time Variable  translate |

NUM\_BALLS = 10

t = 0

def **setup**():

    size(600,600)

    noStroke()

def **draw**():

    global t, NUM\_BALLS

    background(255)

    translate(width/2,height/2)

    for i in range(1,NUM\_BALLS+1):

#color changes from 0 to 255 to “fade to white”

        fill(255,(255-255\*(i-1)/NUM\_BALLS),(255-255\*(i-1)/NUM\_BALLS))

#circle follows Lissajous-like path with phase shift delay

# between balls

        ellipse(200\*cos(t+i\*0.05),200\*sin(2\*(t+i\*0.05)),20,20)

#uncomment to save screenshots to create a .gif file

    #saveFrame("#####.png")

    t += .03 #increment time

The Bouncing Ball

|  |  |
| --- | --- |
|  | Tools:  position  velocity |

xcor,ycor = 0,0

xvel,yvel = 2,3

def **setup**():

    size(600,600)

def **draw**():

    global xcor, ycor, xvel, yvel

    #uncomment next line to see difference

    #background(0)

    xcor += xvel

    ycor += yvel

    #if the ball reaches a wall, switch direction

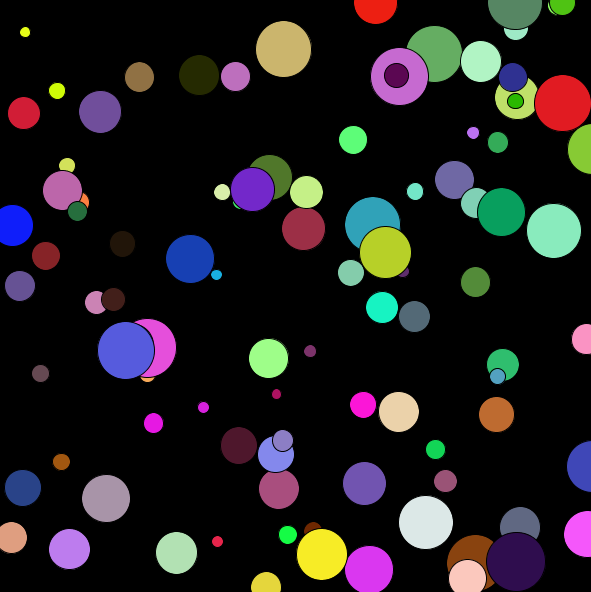
    if xcor > width or xcor < 0:

        xvel = -xvel

    if ycor > height or ycor < 0:

        yvel = -yvel

    ellipse(xcor,ycor,20,20)

The Bouncing Balls: Using Python classes

ballList=[] #empty list to put the balls in

class Ball:

    def \_\_init\_\_(self,x,y):

        '''How to initialize a Ball'''

        self.xcor = x

        self.ycor = y

        self.sz = random(10,60)

        self.xvel = random(-2,2)

        self.yvel = random(-2,2)

        self.col = color(random(255),

                         random(255),

                         random(255))

    def update(self):

'''How to update and draw a Ball'''

        self.xcor += self.xvel

        self.ycor += self.yvel

        #if the ball reaches a wall, switch direction

        if self.xcor > width - self.sz or self.xcor < self.sz:

            self.xvel = -self.xvel

        if self.ycor > height - self.sz or self.ycor < self.sz:

            self.yvel = -self.yvel

        fill(self.col)

        ellipse(self.xcor,self.ycor,self.sz,self.sz)

def **setup**():

    size(600,600)

    for i in range(100): #create 100 balls, put them in the ball list

        ballList.append(Ball(random(width),

                             random(height)))

def **draw**():

    background(0) #black

    for ball in ballList: #loop through the ball list

        ball.update() #and update each ball