The Joy of Creating Art with Code.

Presented by Neeraj Pandey
NEERAJ PANDEY

@NEERAJP99

ASHOKA UNIVERSITY

Sophomore student at Ashoka University.

Software Development, Generative Art, Distributed Computing and Quantitative Finance.
POINTS FOR DISCUSSION

Generative Art - Principles and Elements
History behind Generative Art
Intro to Processing.py
Geometry, Algorithms and Randomness
Examples using Processing.py

@NEERAJP99
Art created through the use of an autonomous system.

ALGORITHMS, MATHEMATICS, GENETIC SEQUENCES
**PRINCIPLES AND ELEMENTS**

**Elements:** color, form, line, shape, space, the randomness and the texture.

**Principles:** rhythm, contrast, harmony, balance, movement, proportion

Substrate, Jared Tarbell
Created using Python mode for Processing
HISTORY OF GENERATIVE ART

Hommage à Paul Klee - Frieder Nake, 1965
One of the earliest and best-known pieces of generative art.

By Georg Nees, 1968
PROCESSING FOUNDATION

Processing
P5.js
Processing.py
Processing for Pi
Processing for Android
OVERVIEW

```python
x, y = 250, 250

def setup():
    size(500, 500)
    background(0)

def draw():
    global x, y
    fill(255)
    ellipse(x, y, 25, 25)
```
USING MATHEMATICS AND ALGORITHMS
RANDOM

Generates random floating point numbers.
RANDOM

Generates random floating point numbers.

```python
1 random()
2 # floating value between [0, 1)
3 random([min], [max])
4 # Random value between [min, max)
5 random(x)
6 # Random value between [0, x)
```

Without processing, use random module in Python.
(https://docs.python.org/3/library/random.html)
DRAWING A POINT

We consider a 2-D cartesian plane, and each point as a vector.
CREATING A POINT AND LINE

```python
1 point(x, y)
2 # Creates a point at (x, y) from the origin.
3 point(x, y, z)
4 # Creates a point at (x, y, z).
5 dist(x1, y1, x2, y2)
6 # Calculates the distance between 2 points.
7 line(x1, y1, x2, y2)
8 # Create a line by joining 2 points.
9
10 stroke(color)
11 # Sets the color used to draw lines
12 # and borders around shapes.
13 strokeWeight(x)
14 # Sets the thickness of the stroke.
```

```python
1 context.set_line_width(0.02)
2 context.move_to(x, y)
3 context.line_to(x1, y1)
4 context.stroke()
```
Using straight lines
Using vector operations
CURVE VERTEX & BEZIER CURVES

Curve Vertex: It specifies the vertex coordinates for curves.

Bezier Curves: It is a versatile mathematical curve in vector graphics.
Creating Bezier Curves

```python
1  # Draws the bezier curves on the screen.
2  # Provide the control point of the anchor points
3  bezier(x1, y1, x2, y2, x3, y3, x4, y4)
4  bezier(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4)
```

```python
1  # Sets the current point to a specific point A.
2  ctx.move_to(ax, ay)
3  # It draws a curve from the current point A to the point D,
4  # using B and C as handles.
5  ctx.curve_to(bx, by, cx, cy, dx, dy)
```
USING BEZIER CURVES TO CREATE WAVES
def setup():
    global offset
    offset = random(100)
    size(700, 700)
    background(0)
    stroke(255)
    strokeWeight(0.5)
    noFill()

def draw():
    global offset
    x, y = [], []
    m, n = 20, 70
    if frameCount <= 600:
        for i in range(4):
            x.append(width * noise(offset + m))
            y.append(height * noise(offset + n))
            m += 10
            n += 10
    bezier(x[0], y[0], x[1], y[1], x[2], y[2], x[2], y[2]);
    offset += 0.005;
USING BEZIER CURVES TO CREATE WAVES
USING CURVEVERTEX()

1 # An implementation of Catmull-Rom splines.
2 # Provide x, y, z coordinates
3 curveVertex(x, y)
4 curveVertex(x, y, z)
def setup():
    size(700, 700)
    background("#013840");

def draw():
    for i in range(40, width - 30, 40):
        for j in range(40, height - 30, 40):
            strokeWeight(1.3);
            stroke(255);
            noFill();
            beginShape();
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            endShape();
    noLoop();
def setup():
    size(700, 700)
    background("#013840");

def draw():
    for i in range(40, width - 30, 40):
        for j in range(40, height - 30, 40):
            strokeWeight(1.3);
            stroke(255);
            noFill();
            beginShape();
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            curveVertex(i + random(-10, 10), j + random(-10, 10));
            endShape();
    noLoop();
CREATING BASIC SHAPES

```python
1  ellipse(a, b, c, d)
2  # Creates an ellipse at point a,b
3  # with "c" width and "d" height
4  rect(a, b, c, d, tl, tr, br, bl)
5  # Creates a rectangle at point a,b
6  # width width "c" and height "d"
7  square(x, y, c)
8  # Creates a square
```

```python
1  context.rectangle(a, b, c, d)
2  # Creates a rectangle
3  context.arc(x, y, radius, start_angle, stop_angle)
4  # Creates an arc, with angles in radians
5  # with stop angle as math.pi*2 for ellipse
```
USING RECTANGLE SHAPE
def setup():
    size(500, 700)
    background(0)

def draw():
    for y in range(0, 700, 10):
        x1 = random(0, 500)
        x2 = 500 - x1
        noStroke()
        fill(random(255))
        rect(0, y, int(x1), 15)
        fill(random(255))
        rect(int(x1), y, 500, 15)
    noLoop()
Using Shapes
LINEAR INTERPOLATION

This function interpolates within the range [start..end] based on the amount parameter, where amount parameter is typically within a [0..1] range.

```python
1 lerp(start, stop, amt)
2  # Calculates a number between two numbers
3  # at a specific increment.
```
EXAMPLE USING LERP COLOR()

```python
def setup():
    size(600, 600)
    background(255)
    colorMode(HSB, 360, 100, 100);

def draw():
    initialColor = color(0, 50, 100);
    finalColor = color(45, 80, 100);
    noStroke()
    for i in range(width):
        linearIntValue = map(i, 0, width, 0, 1.0)
        # Calculates a color between two colors at a specific increment.
        linearIntColor = lerpColor(initialColor, finalColor, linearIntValue)
        fill(linearIntColor)
        rect(i, 0, 25, height)
    noLoop()
```
PERLIN NOISE / SIMPLEX NOISE
PERLIN NOISE / SIMPLEX NOISE

```python
1 noise(x, [y], [z])
2 # Returns the Perlin noise value
3 noiseSeed(seed)
4 # Sets the seed value for noise()
5 noiseDetail(lod, falloff)
6 # Adjusts the level of detail of produced
7 # by perlin noise
```

Without processing, use noise module in Python.
(https://pypi.org/project/noise/)
ADD NOISE TO THE PIXEL COLORS

```python
1 def setup():
2     size(500, 500)
3     pixelDensity(1)
4     noiseDetail(40)
5
6 def draw():
7     xoff = 0
8     loadPixels()
9     for x in range(0, width, 1):
10        yoff = 0
11        for y in range(0, height, 1):
12           index = (x + y * width)
13           r = noise(xoff, yoff) * 255
14           if random(1) > 0:
15               pixels[index] = color(r)
16           yoff += 0.02
17           xoff += 0.02
18     updatePixels()
19     noLoop()
```
ADD RANDOM TO THE PIXEL COLORS
USING NOISE AND TRIGONOMETRY
def draw():
    global rad, r, g, b, translateX, translateY, opacity, zoff
    noFill()
    stroke(r, g, b, opacity)
    beginShape()
    a = 0
    while a < TWO_PI:
        xoff = map(cos(a), -1, 1, 0, 10)
        yoff = map(sin(a), -1, 1, 0, 10)
        noiseFactor = map(noise(xoff, yoff, zoff), 0, 1, 100, 150)
        x = width / 2 + rad * noiseFactor * cos(a)
        y = height / 2 + rad * noiseFactor * sin(a)
        curveVertex(x, y)
        a += 0.1
    endShape(CLOSE)
    zoff += 0.1
    rad -= 0.02
USING NOISE AND TRIGONOMETRY

```python
if r > 255:
    r = 0
if g > 255:
    g = 0
if b > 255:
    b = 0
if opacity == 0:
    opacity = 255
opacity -= 1
r += 1
g += 1
b += 1
```
USING NOISE AND TRIGONOMETRY
PERLIN NOISE

Using Perlin Noise Field and Perlin Noise generated random noise points (grain like texture).
PERLIN NOISE FIELD
def setup():
    global points
    size(700, 700)
    background(255)
    points = []
    # Loop over and create random x, y vectors
    for i in range(2000):
        newVector = PVector(random(width + 100), random(height))
        # Push the newly created vector points to the points array
        points.append(newVector)
```python
def draw():
    global points
    for i in range(len(points)):
        noFill()
        noiseSeed(2)
        vectorObject = points[i]
        stroke("#333");
        strokeWeight(0.3);
        beginShape()
        for i in range(20):
            # Generate noise values and map it between 0 and 2*PI radians
            noiseValue = map(
                noise(vectorObject.x / 500, vectorObject.y / 500),
                0,
                1,
                0,
                2*PI
            )
            xl = vectorObject.x
            yl = vectorObject.y
            x2 = xl + cos(noiseValue)
            y2 = yl + sin(noiseValue)
            vertex(xl, yl)
            vectorObject.x = x2
            vectorObject.y = y2
        endShape(OPEN)
```
PERLIN NOISE FIELD
Using Geometrical patterns, fractals and chaos theory to generate aesthetic art pieces
SIERPINSKI TRIANGLE

An equilateral triangle, subdivided recursively into smaller equilateral triangles with one recursive call each time.
Modified Sierpinski Triangle
Mandelbrot Set

A geometrical figure where each part has the same statistical characters.

\[ z = x + yi \]
\[ z_{n+1} = z_n^2 + C \]
Julia Set
The Logistic Map

Source: Wikimedia
CHAOS THEORY

Deterministic, unpredictable
A small change in the initial state can result in very large difference in the final outcome
Attractors

Lorenz system

De Jong Attractor

\[ x_{t+1} = \sin(a \cdot y_t) - \cos(b \cdot x_t) \]
\[ y_{t+1} = \sin(c \cdot x_t) - \cos(d \cdot y_t) \]

https://examples.pyviz.org/attractors/attractors.html
SIMULATING PAINT

Creating oil, water color paint effects on our 2D/3D canvas.
CREATE POLYGON, DIVIDE SIDES

```python
1  def setup():
2      global polygon
3      size(800, 800)
4      background(255)
5      colorMode(HSB, 360, 200, 150, 1)
6      # Creating a polygon
7      sideAngle = 0.5
8      while sideAngle < 20:
9          sideX = sin(sideAngle) * width / 4
10         sideY = cos(sideAngle) * width / 4
11         polygon.append({"sideX": sideX, "sideY": sideY})
12         sideAngle += 0.5
13      # Midpoint using Gaussian Distribution
14      mid = 50
15      while mid > 5:
16          for i in range(3):
17              polygon = getMid(polygon, mid)
18          mid /= 2
```
FUNCTIONS TO DIVIDE THE SIDE USING RANDOM GAUSSIAN

```python
1 def getCenterPoint(previous, current, standardDeviation):
2     sideX = random_gauss(previous["sideX"] +
3     (current["sideX"] -
4     previous["sideX"]) / 2,
5     standardDeviation)
6     sideY = random_gauss(previous["sideY"] +
7     (current["sideY"] -
8     previous["sideY"]) / 2,
9     standardDeviation)
10    return { "sideX": sideX, "sideY": sideY }
11
12 def getMid(mean, standardDeviation):
13     newVector = [mean[0]]
14     for i in range(1, len(mean), 1):
15         previousValue = mean[i - 1]
16         currentValue = mean[i]
17         midPoint = getCenterPoint(previousValue, currentValue, standardDeviation)
18         newVector.append(previousValue)
19         newVector.append(midPoint)
20     newVector.append(mean[len(mean) - 1])
21     return newVector
```
ADD VERTEX TO THE NEW SIDES

```python
def draw:
    global polygon, randomGauss
    for i in range(20):
        push()
        noStroke()
        fill(200, 100, 80, 0.02)
        beginShape()
        translate(width * 0.5, height * 0.5);
        for j in range(len(polygon)):
            currentVector = polygon[j]
            x = random_gauss(currentVector['sideX'], random(25))
            y = random_gauss(currentVector['sideY'], random(25))
            vertex(x, y)
        endShape(CLOSE)
        pop()
    noLoop()
```
ADD A CUSTOM RANDOM GAUSSIAN METHOD

```python
# Custom function to return a random number fitting a Gaussian distribution.
def random_gauss(m, sd):
    value = 1
    while value >= 1:
        x1 = random(2) - 1
        x2 = random(2) - 1
        value = x1 * x1 + x2 * x2
    value = math.sqrt(-2 * math.log(value) / value)
    y1 = x1 * value
    y2 = x2 * value
    mean = m or 0
    standardDeviation = sd or 1
    return y1 * standardDeviation + mean
```
Inspiration: https://tylerxhobbs.com/essays/2017/a-generative-approach-to-simulating-watercolor-paints
SIMPLE IMPLEMENTATION OF SORTING PIXELS

```python
def draw():
    global img, imgNew
    image(img, 0, 0)
    pixelArray = img.pixels
    # Looping through rows and columns
    for x in range(img.height):
        newPixelArray = list()
        for y in range(img.width):
            # Get pixel positions
            pixel = img.get(y, x)
            # Get the pixel hex value and convert it into rgb
            rgbPixel = convert_to_rgb(str(pixel))
            rgbPixel.append(255)
            newPixelArray.append(rgbPixel)
            # Sort the new pixel array
            for j in range(len(newPixelArray) - 2):
                # for i in range (len(newPixelArray) - 2):
                if (((newPixelArray[j][0] + newPixelArray[j][1] + newPixelArray[j][2]) >
                     (newPixelArray[j+1][0] + newPixelArray[j+1][1] + newPixelArray[j+1][2])))
                    newPixelArray[j], newPixelArray[j+1] = newPixelArray[j+1], newPixelArray[j]
        # Set the image pixels to the second image
        for i in range(img.width):
            a = newPixelArray[i][0]
            b = newPixelArray[i][1]
            c = newPixelArray[i][2]
            imgNew.set(i, x, cl)
        imgNew.updatePixels()
    image(imgNew, 0, 0);
    noLoop()
```
SIMPLE IMPLEMENTATION OF SORTING PIXELS

```python
1 def setup():
2    global img, imgNew
3    size(700, 400)
4    background(0)
5    img = loadImage("sketch.jpg")
6    imgNew = loadImage("sketch.jpg")
7    imgNew.resize(700, 500)
8    img.resize(700, 400)
9    img.loadPixels()
10   colorMode(RGB)
11
12 def convert_to_rgb(hexcode):
13    hexcode = hexcode.lstrip('–')
14    return list(int(hexcode[i:i+2], 16) for i in (0, 2, 4))
```
Pixel Sorting Algorithms
v2.0
THANK YOU