PES University

Cloud Computing and Big Data

Polygon Generator

Team Details

- Ishita Chaudhary- PES1UG19CS190
- Abdul Mannan- PES1UG19CS007
- Rithika Shankar- PES1UG19CS387

[RR campus]

Problem Statement

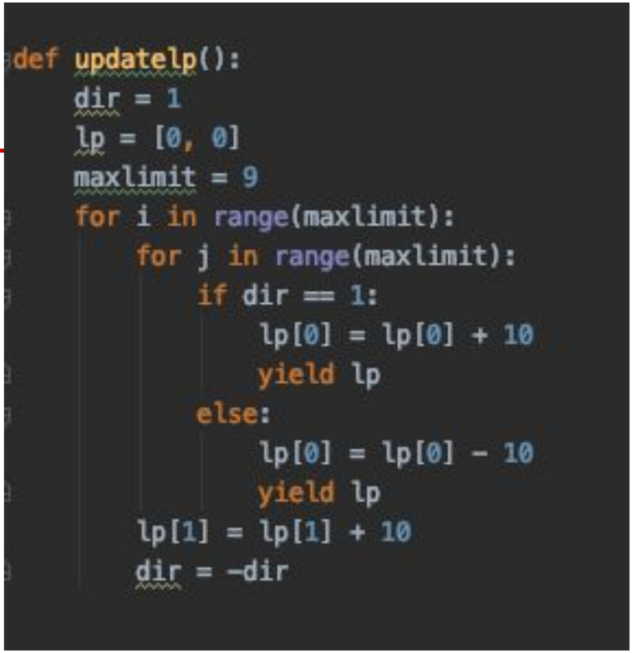
The aim of this assignment is to generate random polygons having vertices from 10 upto 500, and save these shapes as WKT in a text file. Additionally, we find-

- The file sizes and time taken to generate a large number of polygons
- The data distribution on how our data is spread out in a given landscape

Generating non-overlapping points of the polygon

- We have created a generator function to randomly generate the vertices(10-500) of the polygons, while ensuring that none of the polygons overlap when plotted on a single canvas.
- The variable lp[0,0] initially is used to increment each of the coordinates[x,y] generated for one polygon by the same amount. Using the generator function updatelp(), we increment ‘lp’ and yield the value in every iteration(for each polygon).
- This value of lp is passed to the function random_points_gen(vertices, lp), to generate the coordinates of the next polygon. Since the variable lp is added to each set of random points generated, we ensure that none of the polygons created using these points overlap (as adding lp shifts the polygon)
- The variable max_limit defines the number of figures in a row. Once the limit is met, it generates the next rows of polygons above it and so on.

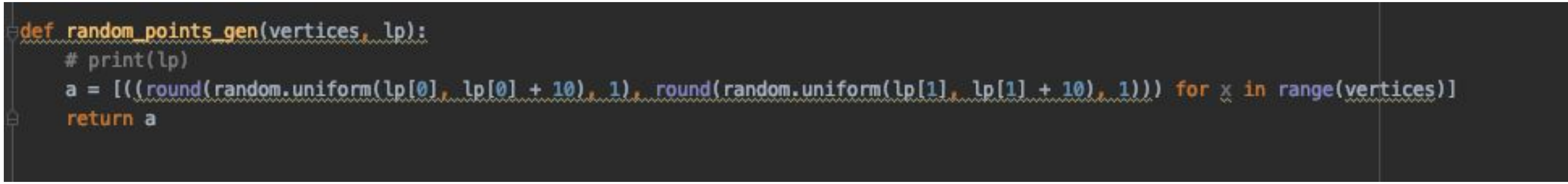
Functions used to create non-overlapping points of a polygon, and arranging it into a grid on the canvas



```
def updatelp():
    dir = 1
    lp = [0, 0]
    maxlimit = 9
    for i in range(maxlimit):
        for j in range(maxlimit):
            if dir == 1:
                lp[0] = lp[0] + 10
                yield lp
            else:
                lp[0] = lp[0] - 10
                yield lp
        lp[1] = lp[1] + 10
        dir = -dir
```

Ensures no polygons overlap

Generates the random coordinates



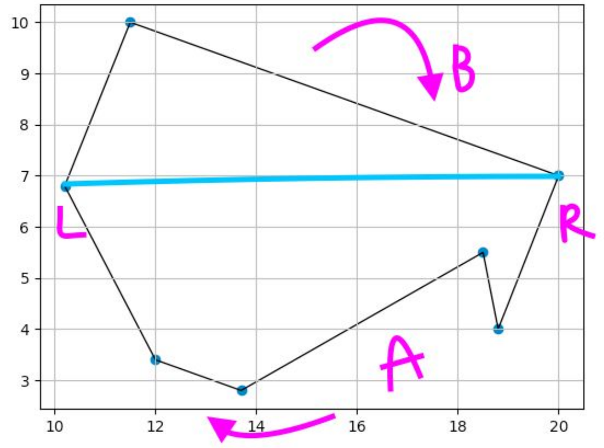
```
def random_points_gen(vertices, lp):
    # print(lp)
    a = [((round(random.uniform(lp[0], lp[0] + 10), 1), round(random.uniform(lp[1], lp[1] + 10), 1))) for x in range(vertices)]
    return a
```

Arranging the points into a polygon

For each set of vertices generated, we sort the points in that list in a clockwise order, ensuring that the vertices form a non-intersecting polygon.

Method-

- First, find the most extreme (rightmost (R) and leftmost(L) points) and remove them from the list
- Find the equation of the line passing through L and R, iterate through the list, separating the points that lie below this line intc an array A and the points that lie above into a different array B
- Arrange the points in array A in decreasing(reverse) order of x-coordinate
- Arrange the points in array B in increasing order of x-coordinate



Arranging the points into a polygon

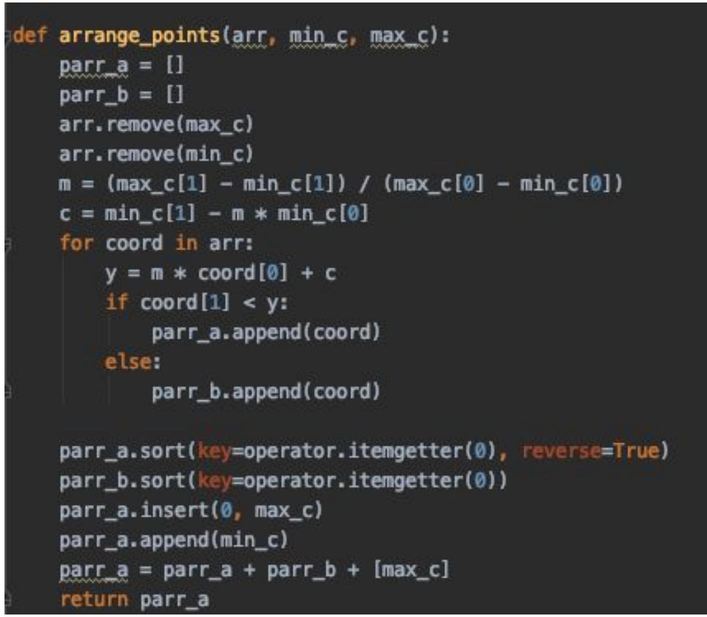
- Insert L to the starting of the array A and append R to the end
- Concatenate the array B to the end of the above array
- Additional: Add the first point(L) to the end as well,

  to match the WKT format of a polygon

Final sorted array: R + A + L + B + R

(in clockwise order)

Time Efficiency: O(n log n)

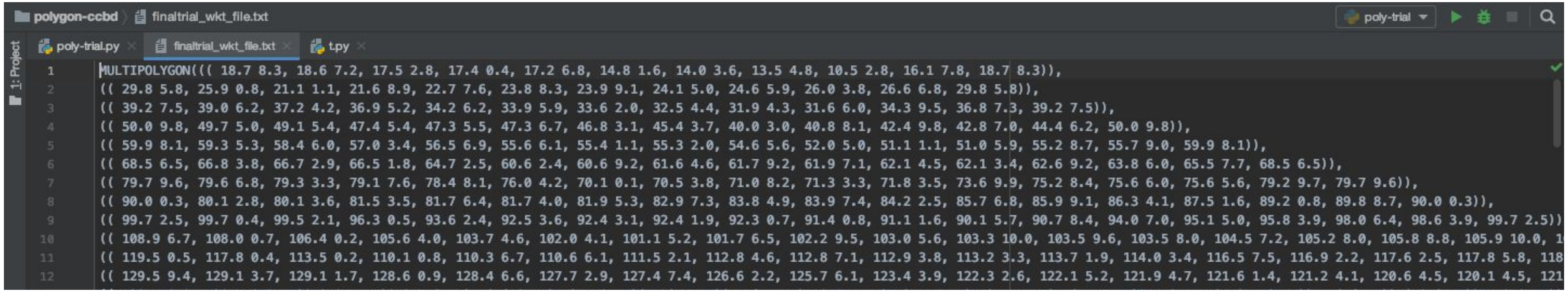


```
def arrange_points(arr, min_c, max_c):
    parr_a = []
    parr_b = []
    arr.remove(max_c)
    arr.remove(min_c)
    m = (max_c[1] - min_c[1]) / (max_c[0] - min_c[0])
    c = min_c[1] - m * min_c[0]
    for coord in arr:
        y = m * coord[0] + c
        if coord[1] < y:
            parr_a.append(coord)
        else:
            parr_b.append(coord)

    parr_a.sort(key=operator.itemgetter(0), reverse=True)
    parr_b.sort(key=operator.itemgetter(0))
    parr_a.insert(0, max_c)
    parr_a.append(min_c)
    parr_a = parr_a + parr_b + [max_c]
    return parr_a
```

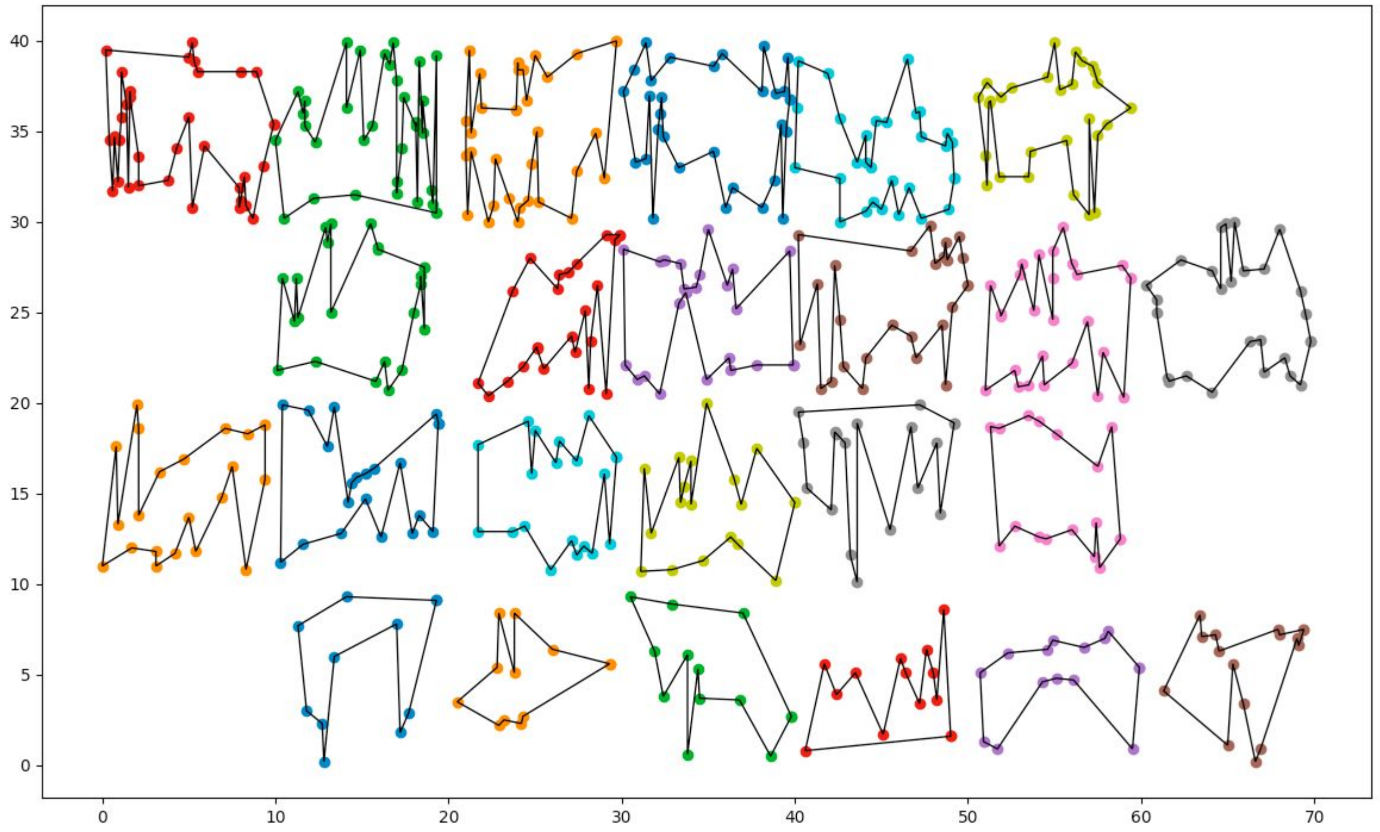
Storing and displaying

- Using string operations, we converted the polygons to a WKT format and store these in a list
- Using that list and filename.writelines(list) method, we store these polygons in a text file in a single step
- We can then display these polygons using pylab and matplotlib on a single canvas, with no overlapping polygons!



```
MULTIPOLYGON((( 18.7 8.3, 18.6 7.2, 17.5 2.8, 17.4 0.4, 17.2 6.8, 14.8 1.6, 14.0 3.6, 13.5 4.8, 10.5 2.8, 16.1 7.8, 18.7 8.3)),
(( 29.8 5.8, 25.9 0.8, 21.1 1.1, 21.6 8.9, 22.7 7.6, 23.8 8.3, 23.9 9.1, 24.1 5.0, 24.6 5.9, 26.0 3.8, 26.6 6.8, 29.8 5.8)),
(( 39.2 7.5, 39.0 6.2, 37.2 4.2, 36.9 5.2, 34.2 6.2, 33.9 5.9, 33.6 2.0, 32.5 4.4, 31.9 4.3, 31.6 6.0, 34.3 9.5, 36.8 7.3, 39.2 7.5)),
(( 50.0 9.8, 49.7 5.0, 49.1 5.4, 47.4 5.4, 47.3 5.5, 47.3 6.7, 46.8 3.1, 45.4 3.7, 40.0 3.0, 40.8 8.1, 42.4 9.8, 42.8 7.0, 44.4 6.2, 50.0 9.8)),
(( 59.9 8.1, 59.3 5.3, 58.4 6.0, 57.0 3.4, 56.5 6.9, 55.6 6.1, 55.4 1.1, 55.3 2.0, 54.6 5.6, 52.0 5.0, 51.1 1.1, 51.0 5.9, 55.2 8.7, 55.7 9.0, 59.9 8.1)),
(( 68.5 6.5, 66.8 3.8, 66.7 2.9, 66.5 1.8, 64.7 2.5, 60.6 2.4, 60.6 9.2, 61.6 4.6, 61.7 9.2, 61.9 7.1, 62.1 4.5, 62.1 3.4, 62.6 9.2, 63.8 6.0, 65.5 7.7, 68.5 6.5)),
(( 79.7 9.6, 79.6 6.8, 79.3 3.3, 79.1 7.6, 78.4 8.1, 76.0 4.2, 70.1 0.1, 70.5 3.8, 71.0 8.2, 71.3 3.3, 71.8 3.5, 73.6 9.9, 75.2 8.4, 75.6 6.0, 75.6 5.6, 79.2 9.7, 79.7 9.6)),
(( 90.0 0.3, 80.1 2.8, 80.1 3.6, 81.5 3.5, 81.7 6.4, 81.7 4.0, 81.9 5.3, 82.9 7.3, 83.8 4.9, 83.9 7.4, 84.2 2.5, 85.7 6.8, 85.9 9.1, 86.3 4.1, 87.5 1.6, 89.2 0.8, 89.8 8.7, 90.0 0.3)),
(( 99.7 2.5, 99.7 0.4, 99.5 2.1, 96.3 0.5, 93.6 2.4, 92.5 3.6, 92.4 3.1, 92.4 1.9, 92.3 0.7, 91.4 0.8, 91.1 1.6, 90.1 5.7, 90.7 8.4, 94.0 7.0, 95.1 5.0, 95.8 3.9, 98.0 6.4, 98.6 3.9, 99.7 2.5))
(( 108.9 6.7, 108.0 0.7, 106.4 0.2, 105.6 4.0, 103.7 4.6, 102.0 4.1, 101.1 5.2, 101.7 6.5, 102.2 9.5, 103.0 5.6, 103.3 10.0, 103.5 9.6, 103.5 8.0, 104.5 7.2, 105.2 8.0, 105.8 8.8, 105.9 10.0, 1
(( 119.5 0.5, 117.8 0.4, 113.5 0.2, 110.1 0.8, 110.3 6.7, 110.6 6.1, 111.5 2.1, 112.8 4.6, 112.8 7.1, 112.9 3.8, 113.2 3.3, 113.7 1.9, 114.0 3.4, 116.5 7.5, 116.9 2.2, 117.6 2.5, 117.8 5.8, 118
(( 129.5 9.4, 129.1 3.7, 129.1 1.7, 128.6 0.9, 128.4 6.6, 127.7 2.9, 127.4 7.4, 126.6 2.2, 125.7 6.1, 123.4 3.9, 122.3 2.6, 122.1 5.2, 121.9 4.7, 121.6 1.4, 121.2 4.1, 120.6 4.5, 120.1 4.5, 121
```

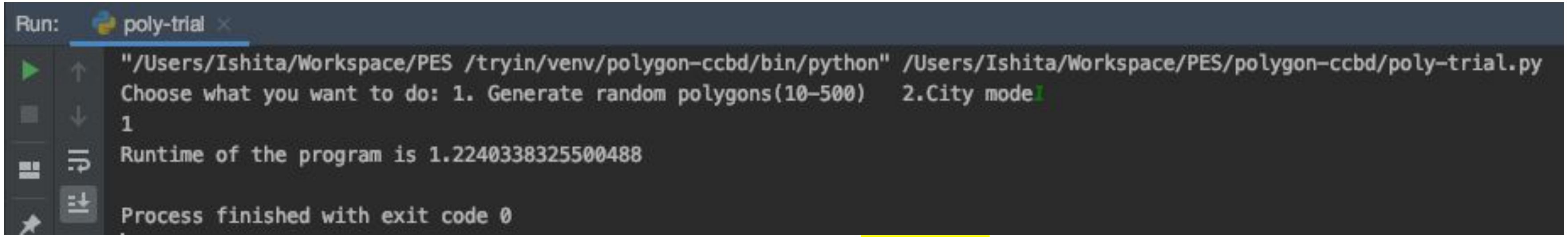
A snippet of our WKT file...



An example displaying some of the polygons generated!

Time taken (bonus question!)

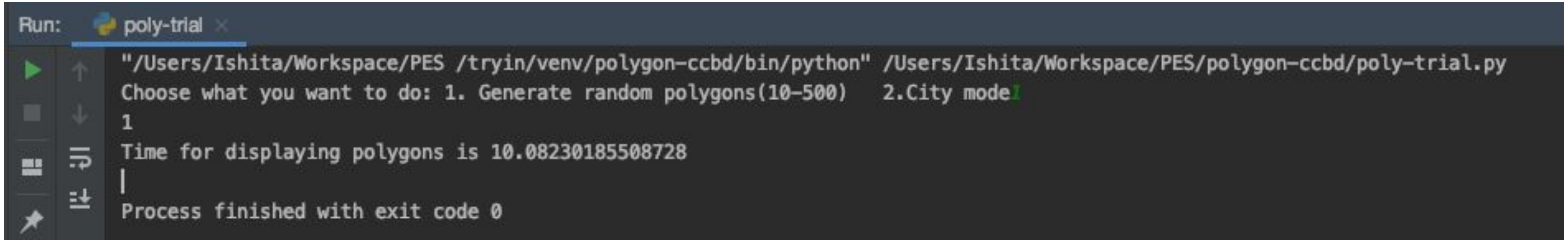
- For generating 10-500 polygons, time taken was- 1.224s



```
"/Users/Ishita/Workspace/PES /tryin/venv/polygon-ccbd/bin/python" /Users/Ishita/Workspace/PES/polygon-ccbd/poly-trial.py
Choose what you want to do: 1. Generate random polygons(10-500)   2.City mode
1
Runtime of the program is 1.2240338325500488

Process finished with exit code 0
```

- For displaying the polygons, time taken- 10.08s

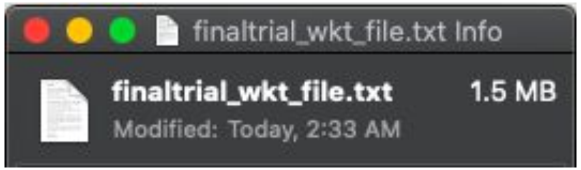


```
"/Users/Ishita/Workspace/PES /tryin/venv/polygon-ccbd/bin/python" /Users/Ishita/Workspace/PES/polygon-ccbd/poly-trial.py
Choose what you want to do: 1. Generate random polygons(10-500)   2.City mode
1
Time for displaying polygons is 10.08230185508728

Process finished with exit code 0
```

File sizes

- For generating polygons with 10-500 vertices, the WKT file size was 1.5 MB


finaltrial_wkt_file.txt Info

finaltrial_wkt_file.txt 1.5 MB
Modified: Today, 2:33 AM

Vertices	Time Taken	File Size
10-100	0.06269502639770508	58 KB
10-200	0.2010810375213623	237 KB
10-300	0.4452328681945801	541 KB
10-400	0.7899413108825684	982 KB
10-500	1.2747068405151367	1.5 MB

Bonus question!

Question: To generate a geographical area with a certain data distribution.

We have generated a random distribution for a small city.

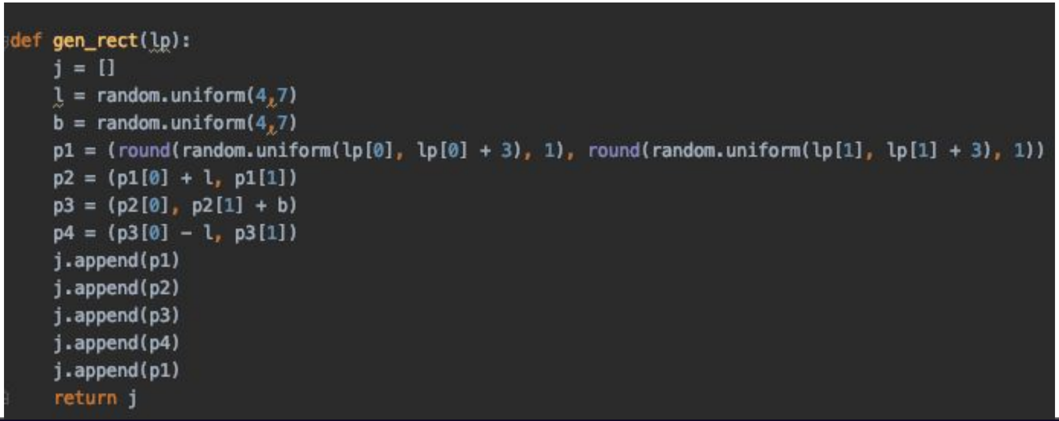
A random distribution is a set of random elements that follow a certain probability density function. We observed that a planned city seems to follow a random distribution, this means that there is a certain probability assigned to different elements(here, plots i.e. building plots, water bodies, parks/gardens etc). So we manually assigned and used these probabilities to create a city using our polygon generator.

- We assigned the probabilities as follows-

Building plots- 0.85	Water bodies-0.05	Parks/Gardens-0.10

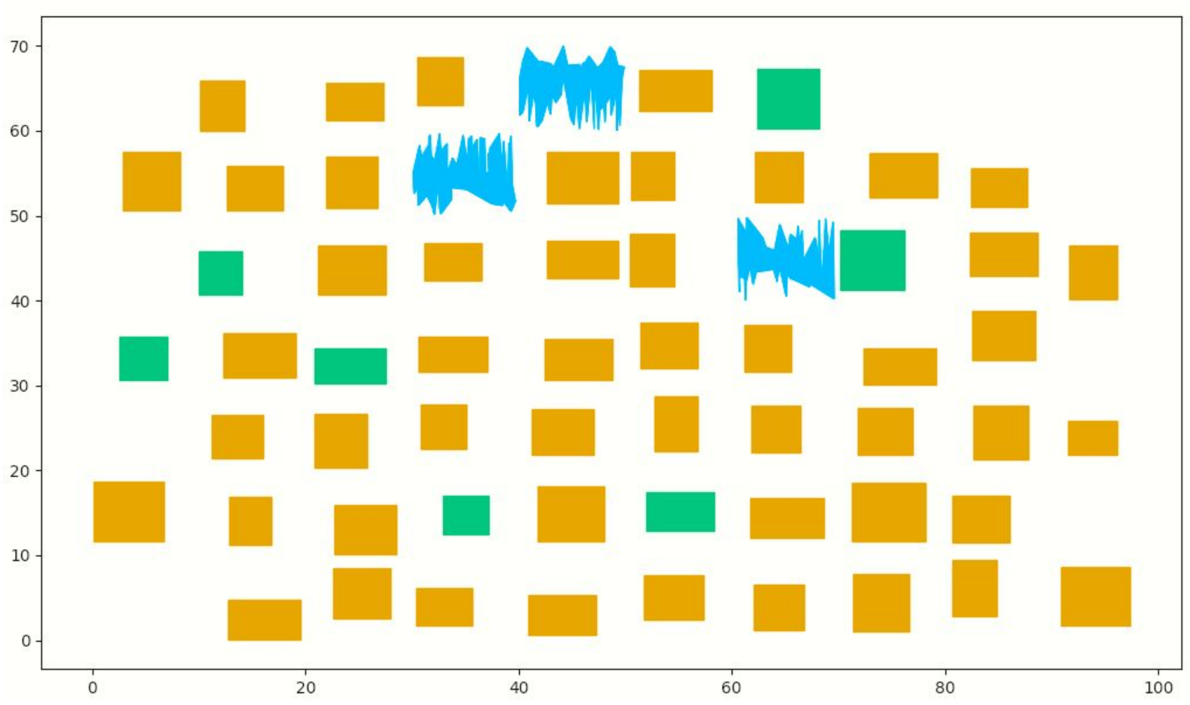
City generation

- Using our polygon generator, we have generated irregular shapes for water bodies.
- We've created a gen_rect() function to generate squares and rectangles that represent the building plots, parks and gardens.
- Additionally, we've assigned appropriate colors to represent them.



```
def gen_rect(lp):
    j = []
    l = random.uniform(4,7)
    b = random.uniform(4,7)
    p1 = (round(random.uniform(lp[0], lp[0] + 3), 1), round(random.uniform(lp[1], lp[1] + 3), 1))
    p2 = (p1[0] + l, p1[1])
    p3 = (p2[0], p2[1] + b)
    p4 = (p3[0] - l, p3[1])
    j.append(p1)
    j.append(p2)
    j.append(p3)
    j.append(p4)
    j.append(p1)
    return j
```

Generated city!



- Building plots- 0.85
- Parks/Gardens- 0.10
- Water bodies-0.05

PES University

Cloud Computing and Big Data

THANK YOU!