1. 13 selfies, from step 1 above [if you don't submit these, you will LOSE 2 points!]

- No points for selfie but deduct if
    - -2 if 0-4 selfies are submitted
    - -1 if 4-8 selfies are submitted

Check if the locations map to the locations given in the KML file (-0.25 if any 1 or more point is not mapped correctly.)

Note: The students can take selfies with half-covered faces. It's to prove that they have gone to the location and taken a selfie.

*2. Your .kml file from step 5 above - with the placemarks, convex hull and nearest-neighbor line segments (**1 point**)*

- Should have 13 different GPS locations. 1 should be the home location and the rest can be anywhere (-0.25 for each missing location)
- Should have a convex hull mapped. The convex hull should contain all 13 points and a few should be on the boundary. (-0.25 deductions if it isn't)
- Should have 4 nearest neighbors from their home location.(-0.25 if not)
- Should have a line segment in .kml file: line(home,nearest_neighbor) (-0.25 if not)
- If the convex hull and nearest neighbors are split into two separate files, it's fine.
- If either the convex hull or the nearest neighbor is missing, -0.25 points each.

*3. Text file (.txt or .sql) with two queries from step 5 - table creation commands and the queries themselves (**2 points**)*

- compute the convex hull (0.75 points)
    - You basically can call ST_CONVEXHULL (in Postgres) to generate a convex hull for points.
    - -0.5 point If only table creation and data insertion commands are present without the rest of the query
    - -0.25 if convex hull generated is not convex (i.e. there are points outside the boundary)
    - If you are not sure about the correctness of queries, please run them.

- Nearest Neighbors (0.75 points)
    - You basically can call ST_Distance (in Postgres) to order points based on their distance.

- -0.5 point If table creation and data insertion commands are present without the rest of query
- If the result should be limited to the nearest neighbors (recommended), if the first row is not the nearest neighbor -0.5
- If you are not sure about the correctness of queries, please run them
- Please note that:
  - it is okay to hardcode points, in the above queries!
  - It is okay to create and use a table to store your 13 points in it, then write queries against the table.
  - It is okay to report the entire table row, name, ID, coordinates or other unique attributes as the results for the nearest neighbors.

Sample Queries:

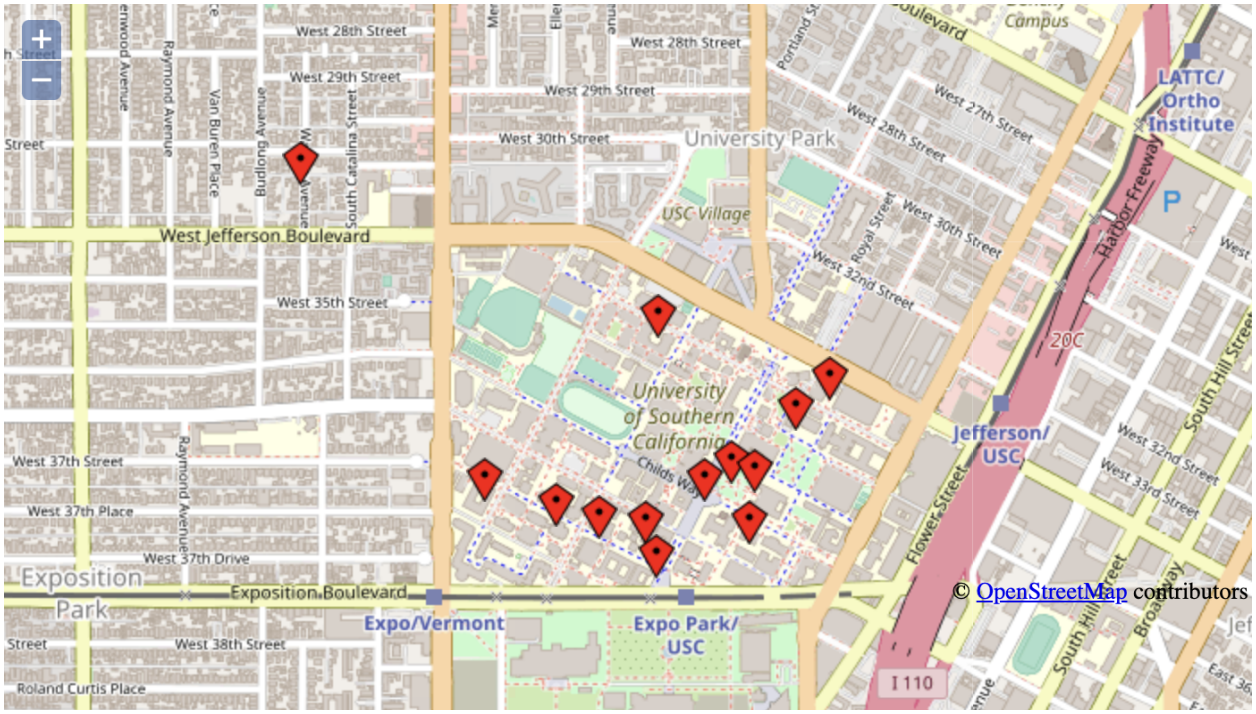
```
# Create table
CREATE TABLE MyNeighborhood (name VARCHAR(1000), gps GEOMETRY);

# Insert points
INSERT INTO MyNeighborhood VALUES
('Home',ST_GeomFromText('POINT(-118.291565 34.035194)')),
...
('LOCATION13',ST_GeomFromText('POINT(-118.243432, 34.2324)'));

# Calculate convex hull
CREATE TABLE ConvexHull AS (SELECT
ST_CONVEXHULL(ST_MULTI(ST_COLLECT(gps))) Hull from
MyNeighborhood);

SELECT ST_ASTEXT(Hull) Hull FROM ConvexHull;

# Calculate the nearest neighbor
SELECT nn.name, ST_ASTEXT(local.gps) as Home, ST_ASTEXT(nn.gps) as
Neighbor
FROM MyNeighborhood local, MyNeighborhood nn
WHERE local.name='Home' and nn.name <> 'Home'
ORDER BY ST_Distance(local.gps, nn.gps) ASC LIMIT 1;
```

4. *screenshots from steps 3,5 (**1 point**)*
    - Convex Hull and nearest neighbors can be separated into different images, or can be integrated into one image. However, images for the convex hull and nearest neighbors should contain all 13 locations, as the homework instruction says that: *Verify this looks correct, using Google Earth, take a snapshot.* Required computation cannot be verified easily without all location information.
    - Google Earth/Map screenshots with sampled locations (0.2 points)
        - -0.1 If 7-12 locations show on the map
        - -0.2 If 0-6 locations show on the map.
        - Deduct at most 0.2 points for the above two items.
    - Convex Hull (0.4 points)
        - -0.2 If the hull is not convex.
        - -0.2 If not all 13 locations are inside or on the edge of the hull.
        - -0.2 If the image for the convex hull does not contain 13 locations.
        - Deduct at most 0.4 points for the above three items.
    - Nearest 4 Neighbors line (0.4 points)
        - -0.1 For **each** missing or wrongly calculated line, -0.4 at most in this item.
        - -0.2 If the image for the image does not contain 13 locations.
        - Deduct at most 0.4 points for above the two items.

*5. A .html file (with the OpenLayers code) from step 6, or a CodePen/jsfiddle link (**1 point**)*

- Locations on the HTML map (0.5 points)
    - -0.25 If 7-12 locations show on the map
    - -0.5 If 0-6 locations show on the map.
- Usage of localStorage (0.5 points)
    - -0.2 If localStorage contains data less than 13 locations' coordinates.
    - -0.4 If localStorage is used but contains no location data.
    - -0.5 If localStorage is not or wrongly used in code.
    - Deduct at most 0.5 points for the above three items.

Sample overview for the html file:



Sample code for Javascript part:

```
<script>
    // here's how to plot points on a map, using OpenLayers:
    // [this is code I mildly modified, from an existing source]
    function initMap() {
        map = new OpenLayers.Map('map');
        basemap = new OpenLayers.Layer.OSM("Simple OSM Map");
        map.addLayer(basemap);
        markers = new OpenLayers.Layer.Markers("Markers");
        map.addLayer(markers);
    } // initMap()

    function addMarker(latitude, longitude) {
        var lonLat = new OpenLayers.LonLat(longitude, latitude)
            .transform(
                new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
                map.getProjectionObject() // to Spherical Mercator Projection
            );
        var point = new OpenLayers.Marker(lonLat);
        markers.addMarker(point);
        map.setCenter(lonLat, 18); // second arg is zoom level
        //console.log(latitude + ", " + longitude);
    } // addMarker()


    // load and setup map layers
    initMap();

    var locations_raw_data = {
        "data": [{
```

```
        "name": "Walton",
        "latitude": 34.0264741,
        "longitude": -118.2949476
    },
    {
        "name": "RanNg FTN",
        "latitude": 34.0201498,
        "longitude": -118.2905534
    },
    {
        "name": "Shumway FTN",
        "latitude": 34.0201631,
        "longitude": -118.2852835
    },
    {
        "name": "TownGown FTN",
        "latitude": 34.019299,
        "longitude": -118.2842066
    },
    {
        "name": "Youth FTN",
        "latitude": 34.0205144,
        "longitude": -118.2846141
    },
    {
        "name": "Doglas FTN",
        "latitude": 34.0234501,
        "longitude": -118.286398
    },
    {
        "name": "SocialWork FTN",
        "latitude": 34.0222003,
        "longitude": -118.2822416
    },
    {
        "name": "Viterbi LIB",
        "latitude": 34.0196655,
        "longitude": -118.2888366
    },
    {
        "name": "Helen LIB",
        "latitude": 34.0194153,
        "longitude": -118.2877934
    },
    {
        "name": "BIX LIB",
        "latitude": 34.0193054,
        "longitude": -118.2866988
    },
    {
        "name": "Hoose LIB",
        "latitude": 34.0186285,
        "longitude": -118.2864389
    },
    {
```

```
                "name": "Doheny LIB",
                "latitude": 34.0203524,
                "longitude": -118.2840793
            },
            {
                "name": "Leavey LIB",
                "latitude": 34.0215809,
                "longitude": -118.2830563
            }
        ]
    };
    var LOCATIONS = 'dysprosium_locations';
    localStorage.setItem(LOCATIONS, JSON.stringify(locations_raw_data));
    const locations_ls_data = JSON.parse(localStorage.getItem(LOCATIONS));

    for (let loc of locations_ls_data.data) {
        addMarker(loc.latitude, loc.longitude);
    }
</script>
```

*6. Spirograph point generation code, the resulting .kml file ("spiro.kml"), shapefile (this needs to be a .zip) and a screenshot (**1 point**)*

- Spirograph point generation code (0.25 points)
    - Code can be in Python, Java, JS, Excel...
    - Starter code in JavaScript is given. You might want to run it once and check if you doubt the code (unless they have written it in a weird language that you can't compile and run). You can use https://www.codechef.com/ide (or other online IDEs) to run it and see if you don't have a particular compiler or interpreter.
    - Parameters in parametric equations should be: R=6, r=1, a=8.
    - -0.1 for each wrong value to R / r / a, -0.25 at most in this item.
- A .kml file (pdf is also acceptable) (0.25 points)
    - If the spirograph image is wrong then it is because of the KML file, check the KML file and spirograph point generation file. If both are wrong deduct 0.25 for each.
- A .zip file for Shapefile (0.25 points)
    - You don't need to unzip the file, if a zip file is present giving the allotted marks.
- Screenshot with Spirograph visualized using ArcGIS Online / Equator. (0.25 points)
    - Either screenshot from ArcGIS Online or Equator is acceptable.
    - Do not deduct points if there is no point near Tommy Trojan.
- Allowed corner cases:
    - If students didn't multiply a factor to scale the graph size, it would go into the Pacific ocean.

- If the curve in KML is above the ground by some distance.

Sample point generation code in python:

spiro-22fall.py

```python
import math
import numpy as np


def compute_coords():
    # x(t) = (R+r)*cos((r/R)*t) - a*cos((1+r/R)*t)
    # y(t) = (R+r)*sin((r/R)*t) - a*sin((1+r/R)*t)
    R, r, a = 6, 1, 8
    n, pi = 16, math.pi
    tt_x, tt_y = 34.020686374889145, -118.28545753100134
    ratio = 8000.0
    for t in np.arange(0, n * pi, 0.01):
        x = ((R + r) * math.cos((r / R) * t) - a * math.cos((1 + r / R) * t)) / ratio + tt_x
        y = ((R + r) * math.sin((r / R) * t) - a * math.sin((1 + r / R) * t)) / ratio + tt_y
        print(str(y) + ',' + str(x) + ',20')


if __name__ == '__main__':
    compute_coords()

```

Sample KML file without computed coordinates:

step7-22fall.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
    <Document>
        <Style id="z1">
            <IconStyle>
                <Icon>
                    <href>http://www.google.com/intl/en_us/mapfiles/ms/micons/blue-dot.png</href>
                </Icon>
            </IconStyle>
        </Style>
        <Style id="redLine">
            <LineStyle>
                <color>ff0000ff</color>
                <width>4</width>
            </LineStyle>
        </Style>
        <Placemark>
            <name>TT</name>
            <styleUrl>#z1</styleUrl>
            <Point>
                <coordinates>-118.28545753100134,34.020686374889145</coordinates>
            </Point>
        </Placemark>
        <Placemark>
            <name>Spirograph</name>
            <styleUrl>#redLine</styleUrl>
            <LineString>
                <extrude>1</extrude>
                <tessellate>1</tessellate>
                <coordinates>
                    <!-- computed coordinates here -->
                </coordinates>
            </LineString>
        </Placemark>
    </Document>
</kml>
```

Sample screenshot with ArcGIS and Equator: