UI/UX CONTRIBUTION TO PUBLIC LABS AND CREATION OF GRAPH LIBRARY

OUR TEAM

Manaswee Koli - 5019128

Aryan Koul - 5019129

Mithun Kuthully - 5019131

Jignesh Mathure - 5019137

CONTENTS

PROBLEM STATEMENT

Task 1

Contributing to open source project of organization called PublicLab, in the field of UI/UX. (First time contribution).

Task 2

Creating Python library based on Graph data structure and algorithm, and then hosting on pypi platform, thus helping users to import it by simply using pip installation commands.

WHAT IS AN OPEN SOURCE PROJECT?

- Open-source software is code that is designed to be publicly accessible anyone can see, modify, and distribute the code as they see fit.
- Open-source software is developed in a decentralized and collaborative way, relying on peer review and community production.
- Open-source software is often cheaper, more flexible, and has more longevity than its proprietary peers because it is developed by communities rather than a single author or company.

WHAT IS PUBLIC LAB?

- Public Lab is a community for DIY environmental investigation and especially welcomes contributions from people and groups underrepresented in free and open-source software.

- It has a welcoming "workflow" that supports you in getting started, which eventually helps you reach out to welcome someone else in turn.

- Its origin is discussed in a paper authored by Rey-Mazón, Pablo, et al., which is titled "Public lab: Community-based approaches" and it helps people to understand open-source.

WORKFLOW

1.

Try an issue

Public labs have prepared extra-welcoming "first-timers-only" issues for beginners.

2.

Help someone else out

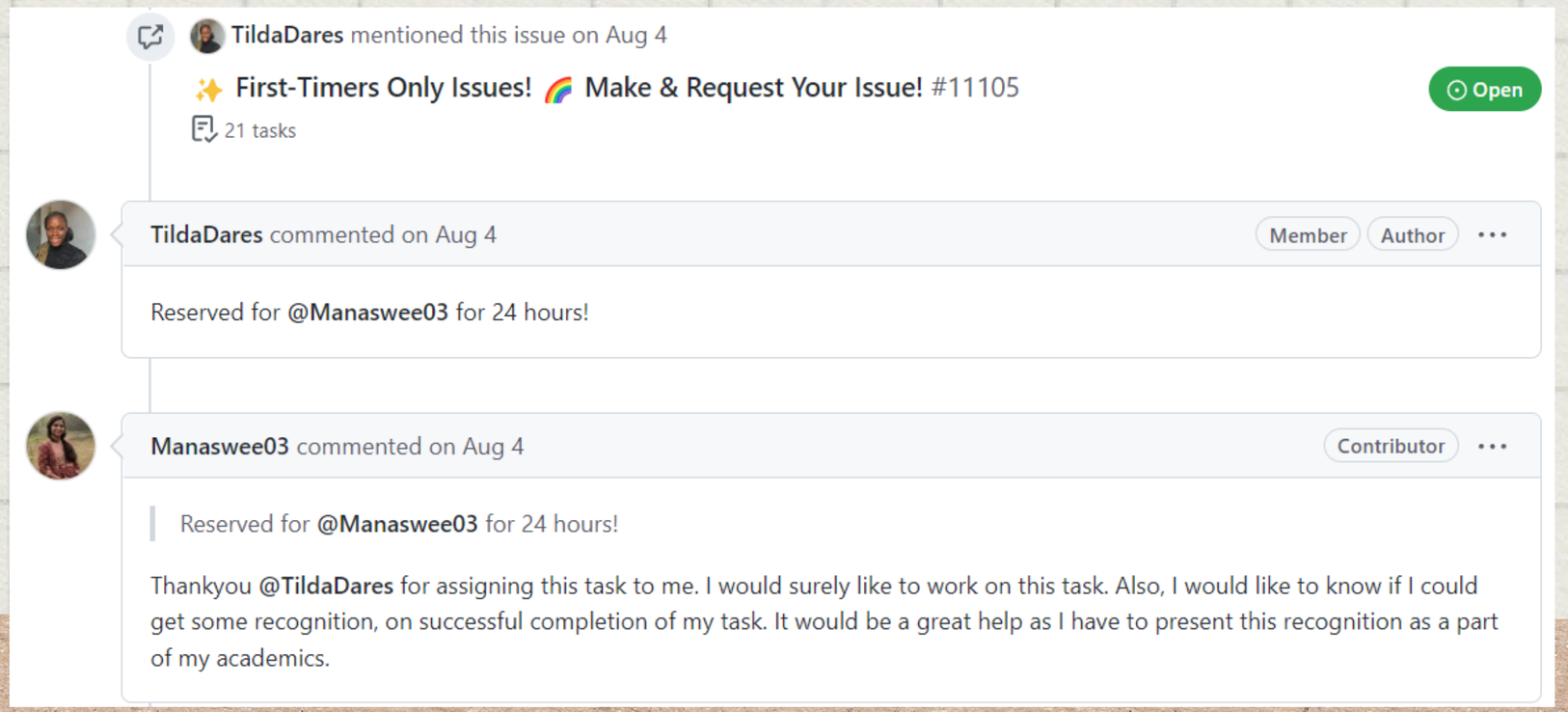
Use your new skills to assist someone else who's just starting out. Browse issues below to see who needs help!

3.

Create a welcoming issue for someone else

Assigned Issue

Add unique ids for comments table row



Task Assignment

SOLUTION

The steps followed to solve the issue were;

1. Claim the issue
2. Fork the repository
3. Update the file app/views/comments/_comments.html.erb in the plots2 repository.
4. Commit the changes
5. Start a pull request
6. Get the approval from the master

RESULTS

Manaswee03 commented on Aug 8 — Contributor

I have successfully resolved the issue of "Adding unique ids for comments table below" by making the required changes in the _comments.html.erb file.

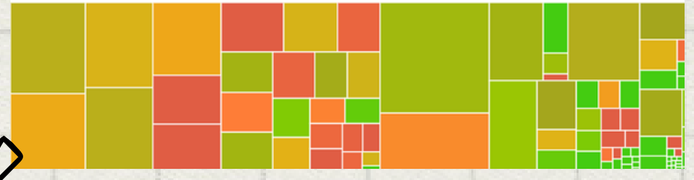
```
16
17    <% comments.each do |comment| %>
18     <tr id="c<%= comment.cid %>">
19       <td>
20         <a href="/profile/<%= comment.author.username %>"> <%=raw strip_tags(sanitize(RDiscount.new(comment.author.username).to_html)).truncate(150) %></a>
21       </td>
22       <td>
23         <% if comment.comment_via == 1 %>
```

Added unique ids for comments table row ✕ 9195293

Manaswee03 requested a review from **a team** as a code owner last month

Codecov Report

Merging #11338 (9195293) into main (3b7222e) will **decrease** coverage by 0.13% .
The diff coverage is n/a .



Add unique ids for comments table row #11323 — Closed

Update _comments.html.erb #11328 — Closed

Update _comments.html.erb #11326 — Closed

Update _comments.html.erb #11324 — Closed

Added unique ids for comments table row #11327 — Closed

Task Completion

RECOGNITION

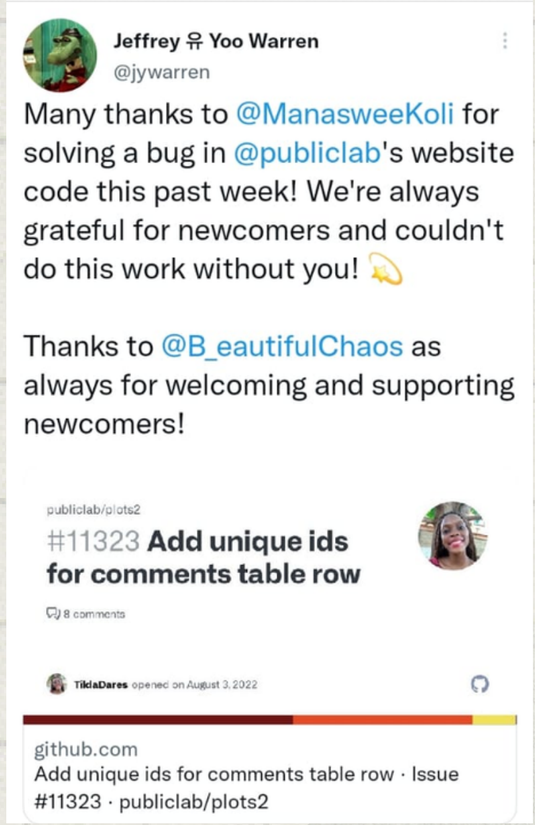
TildaDares commented on Aug 8

Congratulations on merging your PR **@Manasweee03** 🎉

❤️ 1

welcome bot commented on Aug 8 ...

Congrats on merging your first pull request! 🙌🎉⚡
Your code will likely be published to PublicLab.org in the next few days, but first it will be published to https://stable.publiclab.org/ (it will take some minutes for this to load, and until then you may see logs from the build process). Please test out your work on this testing server and report back with a comment that all has gone well!
Do join our weekly check-in to share your this week goal and the awesome work you did 😀. Please find the link to our latest check-in here 📝
Now that you've completed this, you can help someone else take their first step!
Reach out to someone else working on theirs on Public Lab's code welcome page (where you'll now be featured as a recent contributor!). Thanks!



GRAPH LIBRARY

What is a Library?

A library is a collection of related modules. It contains bundles of code that can be used repeatedly in different programs. It makes Programming simpler and convenient for the programmer.

What is a Graph data structure?

A graph is a non-linear data structure, which consists of vertices(or nodes) connected by edges(or arcs) where edges may be directed or undirected. A graph is generally denoted by G(V,E).

Application of Graph Library

- Google maps
- Facebook
- Operating System
- World Wide Web

RELATED WORK

EasyGraph

Mobile Systems and Networking Group, https://github.com/easy-graph/Easy-Graph

EasyGraph is an open-source graph processing library. It is written in Python and supports analysis for undirected graphs and directed graphs. It covers advanced graph processing methods in structural hole spanners detection, graph embedding and several classic methods

Grafalgo

Turner, Jonathan. "Grafalgo-a library of graph algorithms and supporting data structures." (2015).

It includes implementations of algorithms for a variety of classical graph optimization problems like minimum spanning tree problem, the shortest path problem, the max flow problem, etc. The Grafalgo library uses index-based data structures.

NetworkX

Hagberg, Aric, and Drew Conway. "NetworkX: Network Analysis with Python." (2020)

It is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. It is implemented based on NumPy and SciPy and therefore supports all common platforms.

IMPLEMENTATION DETAILS

HARDWARE

- Desktop
- RAM: 4 GB or more.
- Processor: i3 or higher.
- Hard disk: Minimum 10 GB.

SOFTWARE

- Python >= 3.7
- IDE (PyCharm or VS Code)
- Matplotlib library
- OS library

THE METHOD

1. Defining the Graph Function

2. Plotting the Graph

3. Defining the Adjacency Matrix

4. Defining the Adjacency List

5. Test the library on different Use Cases

GRAPH FUNCTION DEFINITON

```python
#*****************Graph************************
class Graph:
    def __init__(self, num_of_nodes, directed=True):
        self.num_of_nodes = num_of_nodes
        self.directed = directed

        # Different representations of a graph
        self.list_of_edges = []

    # Add edge to a graph
    def add_edge(self, node1, node2, weight=1):
        # Add the edge from node1 to node2
        self.list_of_edges.append([node1, node2, weight])

        # If a graph is undirected, add the same edge,
        # but also in the opposite direction
        if not self.directed:
            self.list_of_edges.append([node1, node2, weight])

    # Print a graph representation
    def print_edge_list(self):
        num_of_edges = len(self.list_of_edges)
        for i in range(num_of_edges):
            print("edge ", i+1, ": ", self.list_of_edges[i])
```

```
Graph:
edge  1 :  [0, 1, 25]
edge  2 :  [1, 2, 5]
edge  3 :  [2, 3, 3]
edge  4 :  [3, 4, 1]
edge  5 :  [4, 0, 15]
```

Result

```python
#******************Adjacency Matrix*****************************
class Adj_matrix:
    def __init__(self, num_of_nodes, directed=True):
        self.num_of_nodes = num_of_nodes
        self.directed = directed

        # Initialize the adjacency matrix
        # Create a matrix with `num_of_nodes` rows and columns
        self.adj_matrix = [[0 for column in range(num_of_nodes)]
                           for row in range(num_of_nodes)]

    def add_edge_adj_matrix(self, node1, node2, weight=1):
        self.adj_matrix[node1][node2] = weight

        if not self.directed:
            self.adj_matrix[node2][node1] = weight

    def print_adj_matrix(self):
        for i in range(0,len(self.adj_matrix)):
            print(self.adj_matrix[i])
```

```
Adjaceny Matrix:
[0, 25, 0, 0, 0]
[0, 0, 5, 0, 0]
[0, 0, 0, 3, 0]
[0, 0, 0, 0, 1]
[15, 0, 0, 0, 0]
```

Result

ADJACENCY MATRIX DEFINITON

```python
#*******************Adjacency List*****************************
class Adj_list:
    def __init__(self, directed=True):
        self.directed = directed

        # Initialize the adjacency list
        # Create a dictionary
        self.adj_list = {}

    def add_edge_adj_list(self, node1, node2):
        try:
            if node1 in self.adj_list:
                self.adj_list[node1].append(node2)

                if not self.directed:
                    if node2 in self.adj_list:
                        self.adj_list[node2].append(node1)
                    else:
                        self.adj_list[node2] = [node1]
            else:
                self.adj_list[node1] = [node2]

                if not self.directed:
                    if node2 in self.adj_list:
                        self.adj_list[node2].append(node1)
                    else:
                        self.adj_list[node2] = [node1]
```

ADJACENCY LIST DEFINITON

```
Adjaceny List:
1 : [2, 8]
2 : [3]
3 : [4]
4 : [5, 1]
5 : [1, 1, 4, 5, 6, 7]
```

Result

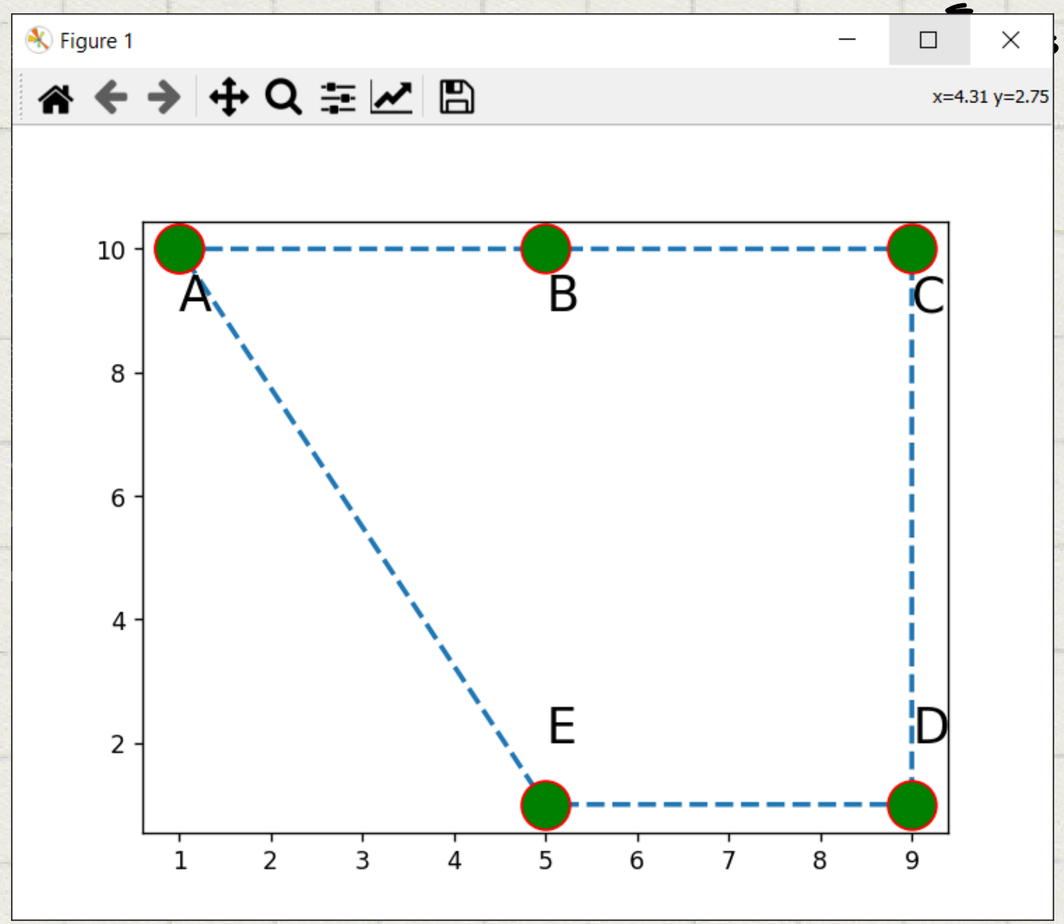
```python
#*******Graph Plotting*****************
import matplotlib.pyplot as plt

class GraphPlot:
    def plotGraph(self, n):
        e = n/2
        f = n//2

        if(e == f):
            l = f

        else:
            l = int(e) + 1

        x = 1
        y = 0
        node1 = []
        node2 = []
        x_values = []
        y_values = []

        for i in range(0, l):
            point1 = [x, 10]
            node1.append(point1)
            if(x>y):
                y = x
            x += 4
```

```python
for i in range(l, n):
    point2 = [y, 1]
    node2.append(point2)
    y -= 4
node2.append([1,10])

f_node = node1 + node2

for i in range(0, n+1):
    x_values.append(f_node[i][0])
    y_values.append(f_node[i][1])

text = []
for i in range(65, 65+n):
    text.append(chr(i))

for i in range(0,len(x_values)-1):
    if(y_values[i]==10):
        plt.text(x_values[i], y_values[i]-1, text[i], fontsize = 20)
    else:
        plt.text(x_values[i], y_values[i]+1, text[i], fontsize = 20)

plt.plot(x_values, y_values, marker='o', markersize=20,
markeredgecolor="red", markerfacecolor="green", linestyle="dashed", linewidth=2)
plt.show()
```

Plotting of Graph

```python
from New_Graph_lib import *

n = int(input("Enter the number of nodes you want: "))

#Graph
graph = Graph(n)

graph.add_edge(0, 1, 25)
graph.add_edge(1, 2, 5)
graph.add_edge(2, 3, 3)
graph.add_edge(3, 4, 1)
graph.add_edge(4, 0, 15)

print()
graph.print_edge_list()

#Plotting the graph
pltgraph = GraphPlot()
pltgraph.plotGraph(n)

#Adjacency Matrix
adj_matrix = Adj_matrix(n)

adj_matrix.add_edge_adj_matrix(0, 0, 25)
adj_matrix.add_edge_adj_matrix(0, 1, 5)
adj_matrix.add_edge_adj_matrix(0, 2, 3)
adj_matrix.add_edge_adj_matrix(1, 3, 1)
adj_matrix.add_edge_adj_matrix(1, 4, 15)
adj_matrix.add_edge_adj_matrix(4, 2, 7)
adj_matrix.add_edge_adj_matrix(4, 3, 11)

print()
adj_matrix.print_adj_matrix()
```

Test Case



Result

GANTT CHART

Tasks	Week 1	Week 2	Week 3	Week 4
July			Issue Selection 20/07/2022	Solving Issue 27/07/2022
August	Sending Pull Request 03/08/2022	Task 1 Submission 08-11 /08/2022	Task 2 Selection 17/08/2022	Working on Task 2 24/08/2022
September	Creating Graph 07/09/2022	Adjacency Matrix 14/09/2022	Displaying Graph 21/09/2022	Module Connection 28/09/2022
October	Displaying Graph 05/10/2022	Documentation 12 /10/2022	Task 2 Submission 08-11 /08/2022	

CONCLUSION

- We contributed to two open-source projects which included the UI/UX contribution on Public Lab and the hosting of Graph Library on PyPi.
- We were able to provide unique ids to the website of Public Labs for comments table row and our pull request was successfully merged into the master branch of the repository.
- We successfully developed and uploaded our Graph library on PyPi.org. The library can be installed using the command "pip install dsagraph". It can be used for creating a graph, plotting the graph and creating the adjacency list and matrix of the graph.

THANK YOU..