

TRAVELLING SALESMAN PROBLEM IMPLEMENTATION USING PYTHON

Project report

Submitted to:

ISHAN KUMAR SIR

SCHOOL OF COMPUTER SCIENCE,

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA,PUNJAB



Submitted By:

Divya
12110470

Sriya Ranga 12101402

Prabin Teli

12101991

THE TRAVELING SALESMAN PROBLEM

The Traveling Salesman Problem (TSP) is a very commonly known mathematical problem that asks for the most efficient route possible with a given set of points (cities) that must all be visited. The traveling salesman problem is very important because this problem is a general optimization, which means that its solution can be applied to many fields like transportation and in science.

THE PROBLEM STATED

A traveling salesman wishes to go to a certain number of destinations in order to sell objects. He wants to travel to each destination exactly once and return home taking the shortest total route.

Each voyage can be represented as a graph $G = (V, E)$ where each destination, including his home, is a vertex, and if there is a direct route that connects two distinct destinations then there is an edge between those two vertices. The traveling salesman problem is solved if there exists a shortest route that visits each destination once and permits the salesman to return home. (This route is called a Hamiltonian Cycle)

Though we are not all traveling salesman, this problem interests those who want to optimize their routes, either by considering distance, cost, or time. If one has four people in their car to drop off at their respective homes, then one automatically tries to think about the shortest distance possible. In this case, distance is minimized. If one is traveling to different parts of the city using the public transportation system, then minimizing distance might not be the goal, but rather minimizing cost.

SOME KNOWN ALGORITHMS

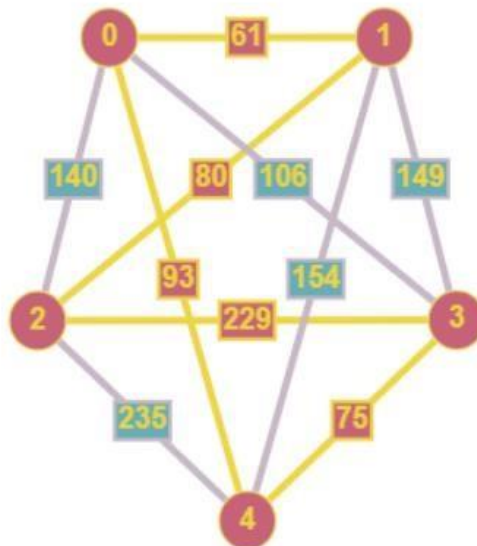
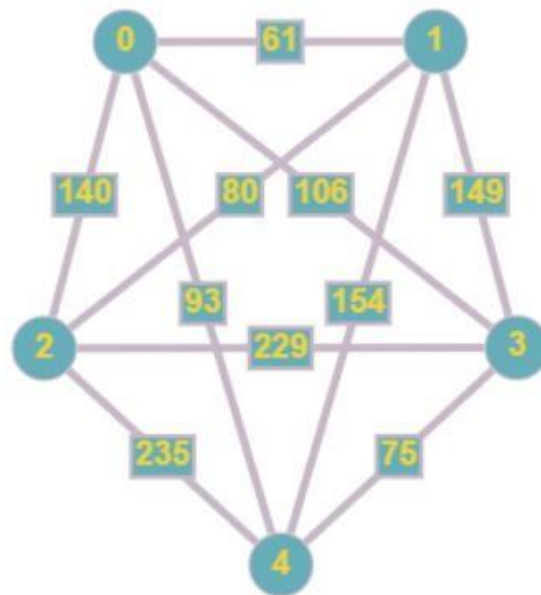
- **GREEDY APPROACH** : A greedy algorithm is a general term for algorithms that try to add the lowest cost possible in each iteration, even if they result in sub-optimal combinations. In this example, all possible edges are sorted by distance, shortest to longest. Then the shortest edge that will neither create a vertex with more than 2 edges, nor a cycle with less than the total number of cities is added. This is repeated until we have a cycle containing all of the cities.

- **NEAREST NEIGHBOUR APPROACH** : The nearest neighbour heuristic is another greedy algorithm, or what some may call naive. It starts at one city and connects with the closest unvisited city. It repeats until every city has been visited. It then returns to the starting city.
- **BRUTE FORCE APPROACH** : The Brute Force approach, also known as the Naive Approach, calculates and compares all possible permutations of routes or paths to determine the shortest unique solution. To solve the TSP using the Brute-Force approach, you must calculate the total number of routes and then draw and list all the possible routes. Calculate the distance of each route and then choose the shortest one—this is the optimal solution. Its Time complexity is $O(n!)$.
- **DYNAMIC PROGRAMMING APPROACH** : Dynamic programming algorithms are often used for smoothing out. A dynamic programming calculation will take a gander at the as of sub issues and will combine their responses for offer the best response for the given issue. In correlation, a greedy algorithm regards the solution as some grouping of steps furthermore, picks the locally ideal decision at each progression. Utilizing a greedy algorithm does not generally ensure an ideal solution, though a dynamic programming algorithm does, due to the way that picking locally ideal choices might achieve an awful global arrangement.

WE ARE USING SOLVING TSP USING DYNAMIC PROGRAMMING

Dynamic programming (normally alluded to as DP) is an exceptionally incredible procedure to tackle a specific class of issues. It requests rich definition of the methodology and basic reasoning. The thought is extremely basic, If you have tackled a issue with the given input, at that point, saves the result for future reference, to do whatever it takes not to handle a comparative issue again. If the given issue can be isolated in to more humble sub-issues and

these more unobtrusive sub issues are consequently apportioned in to regardless more unassuming ones, and in this cycle, if you notice some over-lapping subproblems, its a significant piece of information for DP. Also, the best responses for the subproblems add to the best arrangement of the given issue



MODULES

LOGIN MODULE

The *Login Module* is a portal module that allows users to type a user name and password to log in. You can add this module on any module tab to allow users to log in to the system.

TSP Module

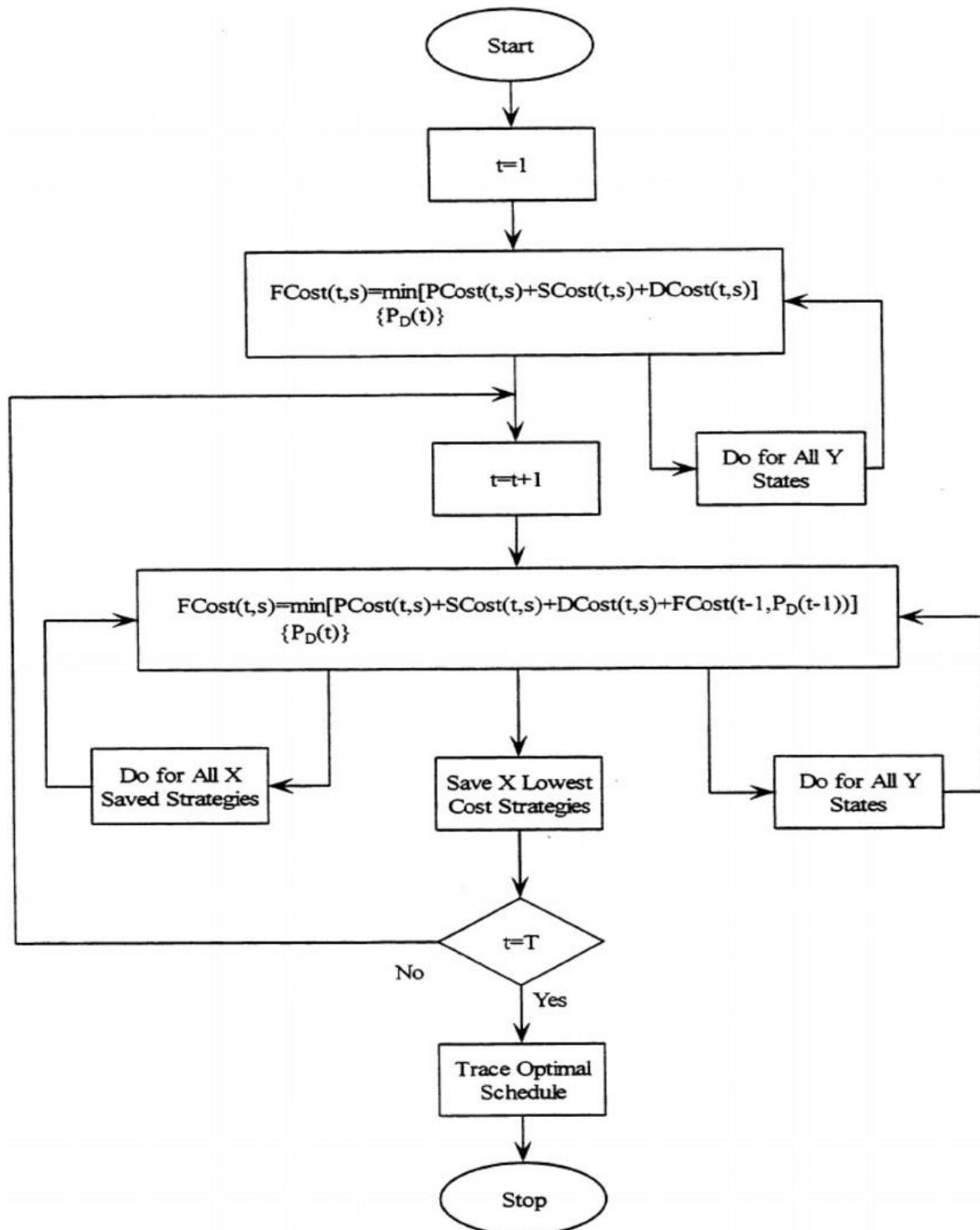
WE ARE USING SOLVING TSP USING DYNAMIC PROGRAMMING

Algorithm

- a) Let the given arrangement of vertices be $\{1, 2, 3, 4, \dots, n\}$. Allow us to think about 1 as beginning and finishing point of yield. For each and every other vertex I (other than 1).
- b) We track down the base expense way with 1 as the beginning stage, I as the consummation point and all vertices showing up precisely once. Leave the expense of this way alone $cost(i)$, the expense of comparing Cycle would be $cost(i) + dist(i, 1)$ where $dist(i, 1)$ is the separation from I to 1.

To figure $cost(i)$ utilizing Dynamic Programming, we need to have some recursive connection as far as sub-issues. Allow us to characterize a term $C(S, I)$ be the expense of the base expense way visiting every vertex in set S precisely once, beginning at 1 and finishing at

I. d) We return the base of all $[cost(i) + dist(i, 1)]$ qualities.



ADVANTAGES OF USING DP FOR SOLVING TSP

When we utilize the dynamic programming algorithm for tracking down an ideal answer for a travelling salesman algorithm, we have the accompanying benefits:

- 1) We don't utilize direct programming procedures.

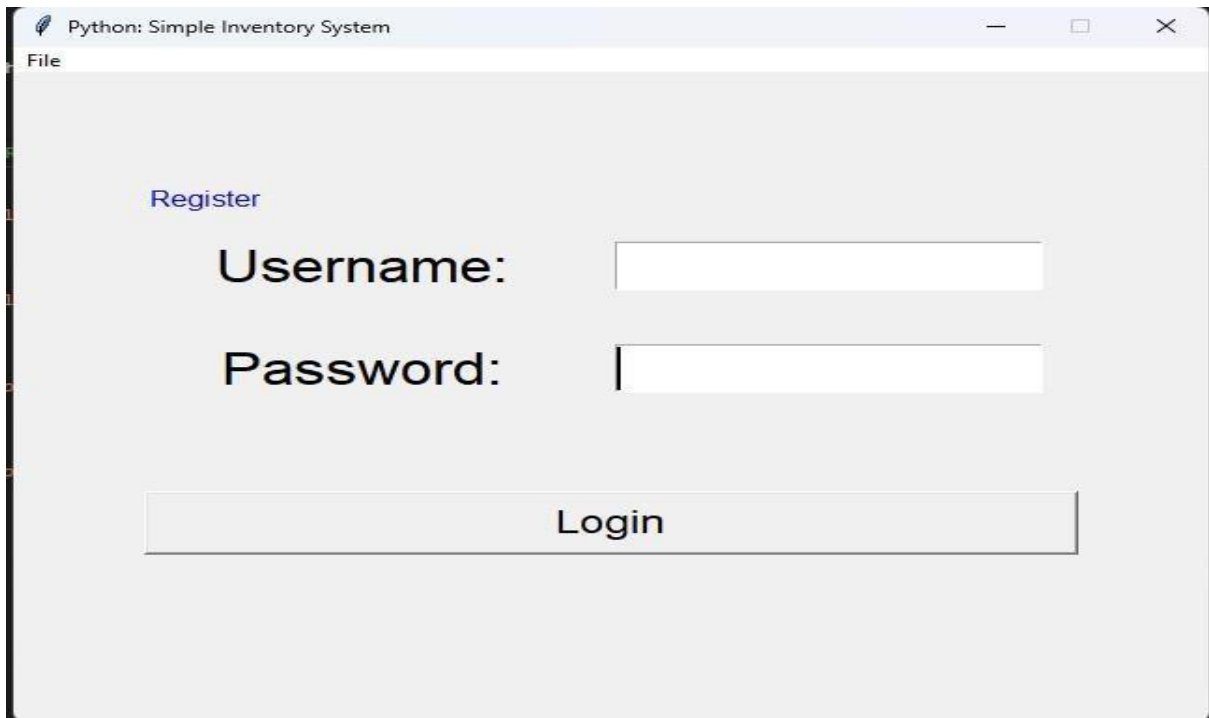
- 2) Utilizing a greedy algorithm does not generally ensure an ideal solution, though a dynamic programming algorithm does.
- 3) The proposed technique is straightforward and apply.
- 4) It has less time complexity than Brute Force Algorithm , i.e , Brute Force has time complexity $O(n!)$ whereas this approach has time complexity $O(n^2 \cdot 2^n)$.

SCREENSHOTS OF PROJECT

OUTPUT OF PROGRAM

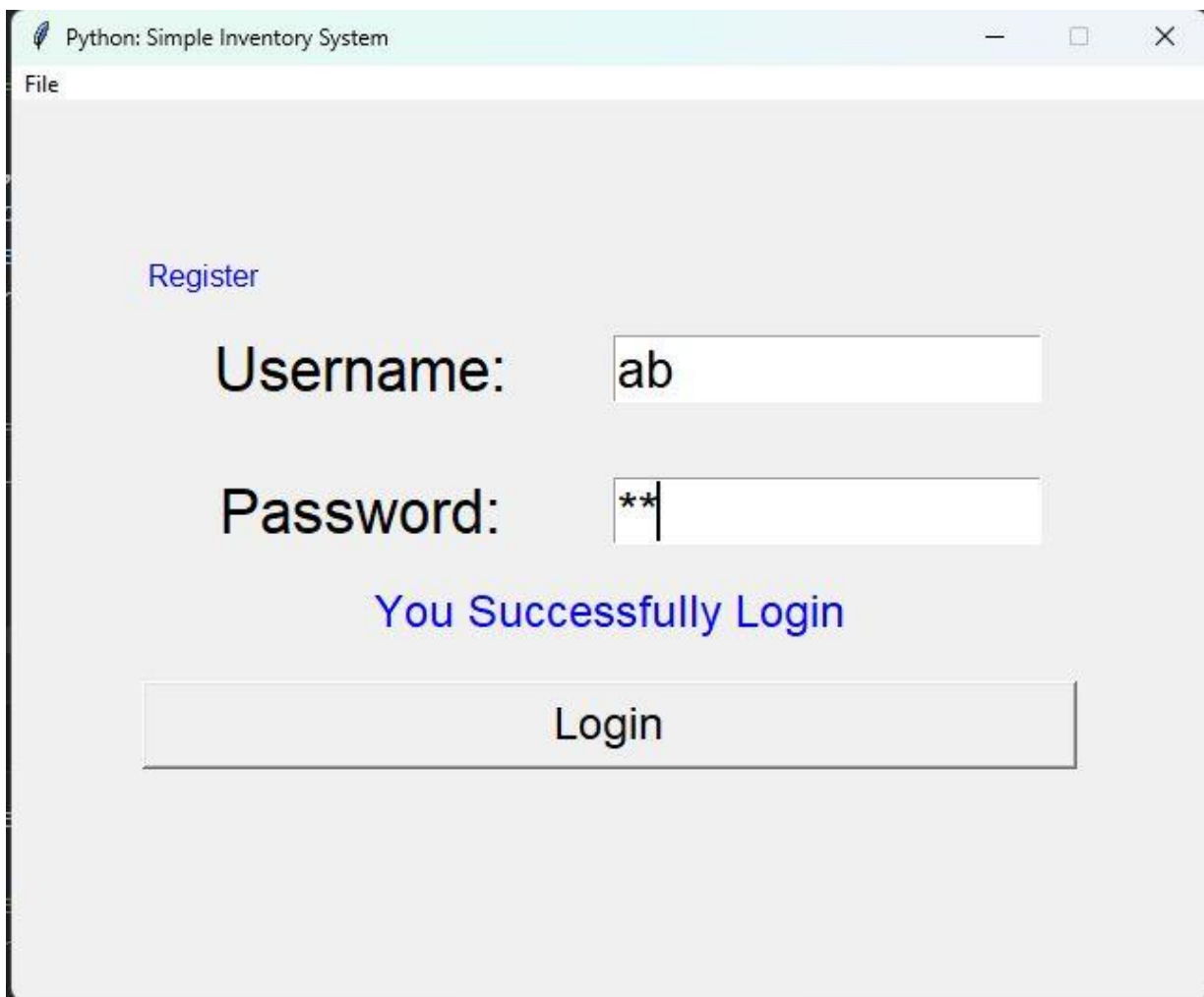
```
The shortest path covered while traveling =  
537
```

LOGIN PAGE



The screenshot shows a window titled "Python: Simple Inventory System" with a standard macOS-style title bar (minimize, maximize, close buttons). Below the title bar is a menu bar with "File". The main content area has a light gray background. At the top left, there is a blue link labeled "Register". Below it, the text "Username:" is followed by a white text input field. Underneath that, the text "Password:" is followed by a white text input field. At the bottom, there is a wide, light gray button with the text "Login" centered on it.

SUCCESSFUL LOGIN



The image shows a screenshot of a Python application window titled "Python: Simple Inventory System". The window has a menu bar with "File" and a toolbar with standard window controls. The main content area has a light gray background. At the top left, there is a blue link labeled "Register". Below it, the "Username:" label is followed by a text input field containing "ab". The "Password:" label is followed by a text input field containing two asterisks "**". Below the password field, the text "You Successfully Login" is displayed in blue. At the bottom, there is a large, light gray button labeled "Login".

Python: Simple Inventory System

File

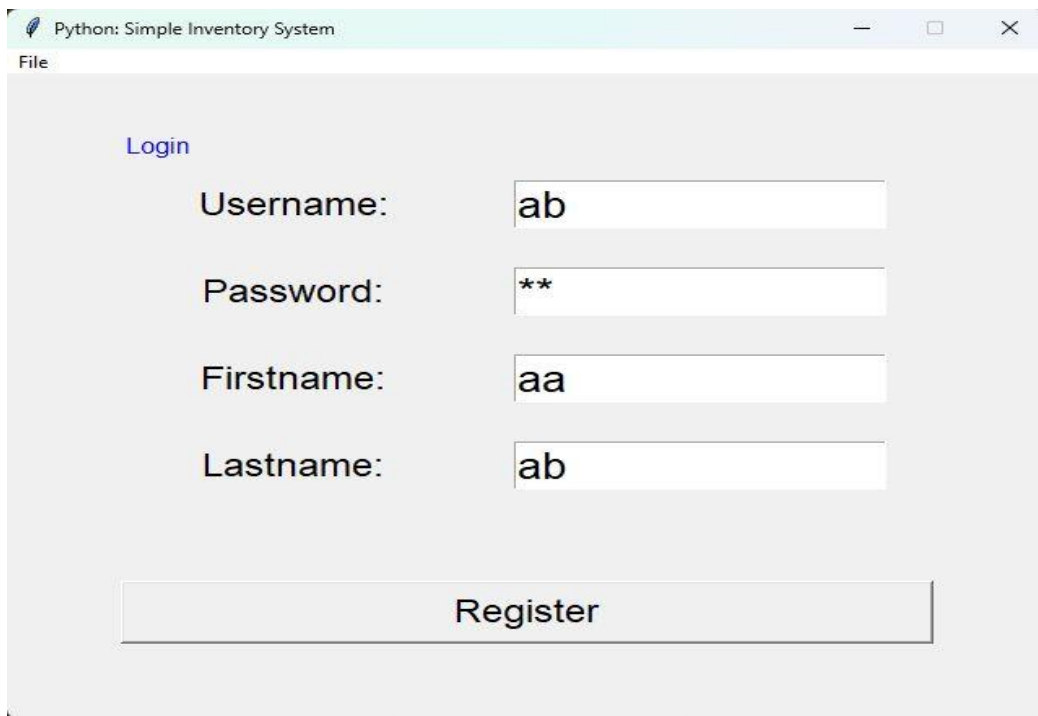
[Register](#)

Username:

Password:

[You Successfully Login](#)

REGISTER PAGE



The screenshot shows a window titled "Python: Simple Inventory System" with a "File" menu. The main content area has a "Login" link in blue. Below it are four input fields: "Username:" with the value "ab", "Password:" with the value "**", "Firstname:" with the value "aa", and "Lastname:" with the value "ab". At the bottom is a large "Register" button.

Python: Simple Inventory System

File

Login

Username: ab

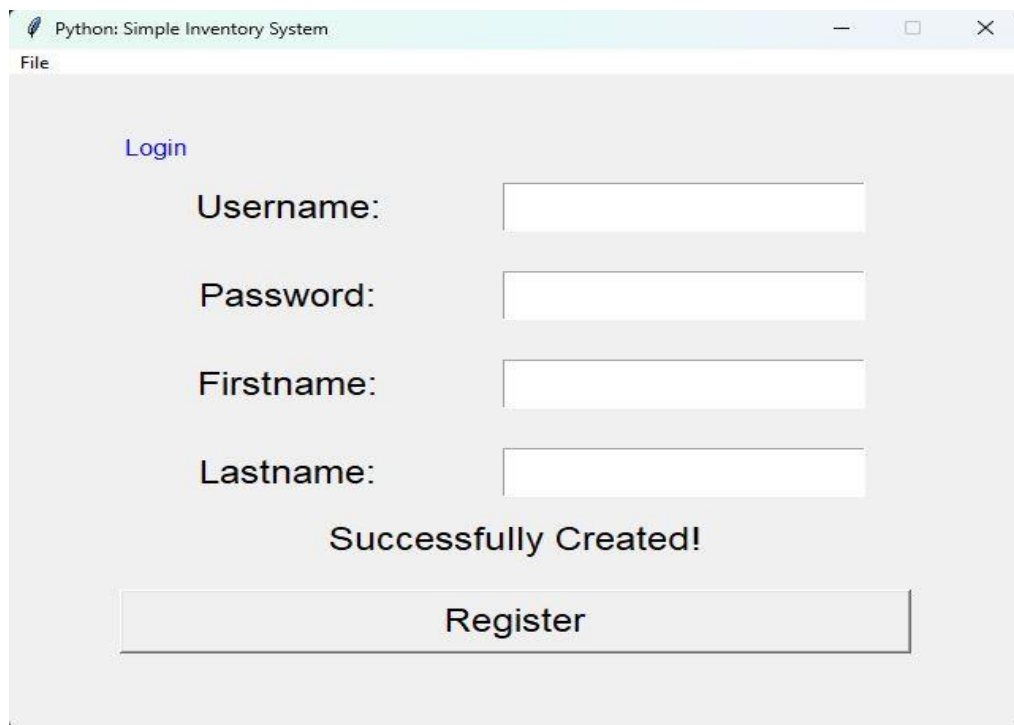
Password: **

Firstname: aa

Lastname: ab

Register

SUCCESSFUL REGISTER



The screenshot shows the same window as before, but the "Register" button is now disabled (grayed out). The input fields are empty. A message "Successfully Created!" is displayed in the center of the form area.

Python: Simple Inventory System

File

Login

Username:

Password:

Firstname:

Lastname:

Successfully Created!

Register