NATIONAL PUBLIC SCHOOL HSR LAYOUT

HARVEST HUB

2023-24

Atulya Kumar,08,11B  
 Preryth Murali,26,11B  
 Soham Tiwari,31,11B

CERTIFICATE

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this project. Special thanks to principal Ms. Shefali Tyagi for giving us this opportunity, and also our computer teacher Nimi Ma’am, whose support, guidance, and expertise were invaluable throughout the process. Your commitment and collaboration have played a crucial role in achieving our goals. I am truly grateful for the encouragement and assistance provided by each person involved. Thank you for being an integral part of this journey.

Atulya Kumar, Preryth Murali, Soham Tiwari

INDEX

|  |  |  |
| --- | --- | --- |
| S.No | Content | Page no |
| 1 | *Certificate* |  |
| 2 | *Acknowledgement* |  |
| 3 | *Index* |  |
| 4 | *Overview of Python* |  |
| 5 | *Project Synopsis* |  |
| 6 | *System Requirements* |  |
| 7 | *Modules and functions* |  |
| 8 | *Program Code* |  |
| 9 | *Program Output* |  |
| 10 | *Limitations* |  |
| 11 | *Bibliography* |  |

An Overview of Python

Introduction

Python, born out of Guido van Rossum's holiday project in December 1989, has grown into a powerhouse in the programming realm. Its name, a playful nod to Monty Python's Flying Circus, reflects the language's creator's sense of humor. Python 1.0, released in 1994, marked the beginning of a journey characterized by continuous evolution and refinement. A significant milestone arrived with Python 2.0 in 2000, introducing features like list comprehensions and garbage collection. However, to address emerging design flaws and inconsistencies, Python 3.0 underwent a major overhaul in 2008. Despite initial backward compatibility challenges, Python 3 has become the recommended version for all new projects. Python's readability, versatility, and extensive community support have contributed to its prominence, making it a preferred language for applications ranging from web development to artificial intelligence.

Brief history

Python's origin traces back to December 1989 when Guido van Rossum embarked on a holiday project, marking the beginning of a programming language that would soon become one of the most popular in the world. The choice of the name "Python" reflects van Rossum's sense of humor, paying homage to the British comedy group Monty Python's Flying Circus.

The language reached a significant milestone with the release of Python 1.0 in 1994. Over the years, Python underwent continuous refinement and enhancement through regular updates. A noteworthy leap occurred with the release of Python 2.0 in 2000, introducing features like list comprehensions and garbage collection. However, as Python gained widespread adoption, certain design flaws and inconsistencies became apparent.

In response to these challenges, Python 3.0, a major overhaul aimed at addressing issues and improving the language's overall design, was released in 2008. Despite the backward compatibility hurdles that accompanied this transition, the Python community gradually embraced Python 3. Today, Python 3 is the recommended version for all new projects, having resolved many of the earlier issues and ensuring a more robust and streamlined development experience.

Python's journey from a holiday project to a powerhouse in the programming world reflects its adaptability and the commitment of its community to continuous improvement. The language's evolution showcases its ability to address challenges, maintain relevance, and remain a top choice for developers across diverse domains.

Features

Below are the key features of Python

* Readable and Expressive Syntax: Python's syntax is designed to be clear and readable, emphasizing code readability and reducing the cost of program maintenance. This makes it an ideal language for both beginners and experienced developers.
* Multi-paradigm Programming: Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. This flexibility allows developers to choose the most suitable approach for their specific project.
* Extensive Standard Library: Python comes with a comprehensive standard library that provides modules and packages for a wide range of tasks, from working with files and databases to handling networking and web development. This rich set of tools helps developers accomplish tasks efficiently without having to reinvent the wheel.
* Community and Ecosystem: Python boasts a vibrant and active community that contributes to the language's growth and development. The Python Package Index (PyPI) is a vast repository of third-party packages, making it easy for developers to access and integrate external libraries into their projects.
* Cross-platform Compatibility: Python is platform-independent, meaning that code written in Python can run on various operating systems without modification. This portability is especially beneficial when developing applications that need to work across different platforms.
* Dynamically Typed: Python is a dynamically typed language, which means that variable types are determined at runtime. This flexibility simplifies the development process and allows for more agile coding.
* High-level Data Structures: Python provides built-in high-level data structures, such as lists, dictionaries, and sets, which make it easy for developers to manage and manipulate data. This, combined with dynamic typing, contributes to faster development cycles.
* Ease of Learning: Python's simplicity and readability make it an excellent choice for beginners. The language's emphasis on code readability and a clean syntax lowers the entry barrier for new programmers.
* Versatility in Application: Python finds applications in a wide range of domains, including web development, data science, artificial intelligence, machine learning, automation, scripting, and more. Its versatility has contributed to its widespread adoption in various industries.

PROJECT SYNOPSIS

Introduction

Agriculture is a critical sector in India, and farmers often face challenges related to crop management, market access, and weather forecasting. Harvest Hub helps support farmers in India with essential information and resources.

This project directly addresses the challenges faced by farmers in India, such as access to relevant information and resources. It can improve crop yields, income, and overall livelihoods, contributing to the country's agricultural growth.

Utility

The provided code is a Python script utilizing the Tkinter library to create a graphical user interface (GUI) for an agricultural application named "Harvest Hub." The GUI consists of different pages accessible through a side menu, including Home, Crop Filter, Crop Calendar, Report, About Page and Settings. The script makes use of frames, labels, buttons, and canvas elements to organize and display information.

One notable feature is the dynamic filtering of crop data based on soil type and irrigation requirements. The Crop Filter page allows users to select specific soil types and irrigation methods, and the application dynamically updates the displayed crop list accordingly. Additionally, the code integrates scrolling functionality for long lists of crops and implements a reload mechanism to update the displayed tiles after applying filters.

The GUI design incorporates a clean and visually appealing layout, with a responsive side menu that expands and collapses for a more user-friendly experience. Each menu option corresponds to a specific page, providing a seamless navigation experience. The code also demonstrates the use of Pandas for handling data from an Excel file, showcasing the ability to retrieve and display detailed information about crops.

Overall, the Harvest Hub application provides an intuitive interface for users to explore and filter crop information, making it a valuable tool for individuals involved in agriculture, whether they are farmers, researchers, or agricultural enthusiasts. The code showcases effective use of Tkinter for GUI development, data manipulation with Pandas, and integration of dynamic filtering features for an enhanced user experience.

SYSTEM REQUIREMENT

Hardware Requirement

* Processor (CPU): A dual-core processor with a clock speed of 2.0 GHz or higher should be sufficient.
* RAM (Memory): At least 2 GB of RAM is recommended to ensure smooth operation, especially when working with graphical user interfaces and data manipulation.
* Storage: The storage requirement depends on the size of the data files and the application itself. A minimum of 500 MB of free disk space is reasonable.

Software Requirement

* Operating System: Any modern operating system such as Windows 7 or later, macOS, or a Linux distribution with GTK support (since Tkinter relies on it).
* Python: The code is written in Python, so a Python interpreter is necessary. Python 3.x is recommended. You can download Python from the official Python website.
* Required Python Libraries:

1. tkinter: This is a standard GUI library for Python, and it's included in most Python installations.

2. PIL (Pillow): Required for image processing. You can install it using pip install Pillow.

3. pandas: Used for data manipulation. Install it with pip install pandas.

Make sure to install the required libraries using the terminal or command prompt. Additionally, if you're using an external data file (like an Excel file), ensure that the necessary file (list.xlsx in this case) is present in the correct location or provide the correct path in the code.

User Manual

1. Introduction

1.1 Purpose

Harvest Hub is designed to assist users in managing agricultural data, specifically focusing on crop information. Users can filter, sort, and view details about various crops, access a crop calendar, generate reports, and configure application settings.

1.2 System Requirements

* Python 3.x
* Tkinter
* Pandas
* Pillow (PIL)

1.3 Installation

Ensure you have the required dependencies installed. Run the Python script provided, and the Harvest Hub GUI will launch.

2. Navigation

2.1 Main Interface

The main interface consists of a central area displaying the selected page content and a left sidebar hosting the navigation menu.

2.2 Navigation Menu

The left sidebar contains navigation buttons for different sections of the application. Clicking on a button will take you to the corresponding page.

2.3 Page Indicators

Page indicators display the current active page by highlighting the respective button on the left sidebar.

3. Pages and Features

3.1 Home Page

Displays a welcome message

3.2 Crop Filter Page

* Allows sorting and filtering crops based on soil type and irrigation.
* Provides a reload button to refresh the crop list.

3.3 Crop Calendar Page

Displays information related to the crop calendar.

3.4 Report Page

Offers functionality for generating and viewing reports.

3.5 Settings Page

Allows users to configure application settings.

4. Using the Application

4.1 Sorting and Filtering Crops

On the Crop Filter Page, use the checkboxes to filter crops based on soil type and irrigation.

Click the "Reload" button to apply the selected filters.

4.2 Viewing Crop Details

Click on a crop in the list to view more details in an expanded view.

4.3 Reloading Crop List

On the Crop Filter Page, click the "Reload" button to refresh the crop list.

4.4 Navigating Between Pages

Use the navigation menu on the left to switch between different sections of the application.

5. Troubleshooting

5.1 No Results Display

If no results are displayed, check your filter criteria and adjust them accordingly.

5.2 Application Freeze

If the application freezes, try reloading the application or checking for any console error messages.

5.3 Data Loading Issues

Ensure the "list.xlsx" file is present in the application directory.

MODULES AND FUCNTIONS

Here’s a brief mention of all modules used:

* tkinter:

import tkinter as tk- Tkinter is the standard GUI toolkit that comes with Python. It provides tools for creating windows, buttons, labels, and other GUI elements.

* PIL (Pillow):

from PIL import Image, ImageTk- The Python Imaging Library (PIL), now known as Pillow, is used for working with images. In this script, it's used to add images to the GUI.

* pandas:

import pandas as pd- Pandas is a powerful data manipulation library. Here, it is used for reading data from an Excel file (list.xlsx) and performing data filtering based on user input.

This code is a Python script that uses the Tkinter library to create a graphical user interface (GUI) for a Harvest Hub application. The application is divided into multiple pages, each represented by a frame. The main components include:

* Import Statements

Tkinter-Provides the main GUI functionality.

Label, ttk-GUI components for labels and themed widgets.

PIL-Used for handling images.

Pandas-Used for working with data in Excel files.

* Mousewheel Function

on\_mousewheel(event, canvas)-Handles mousewheel events to scroll through a canvas.

* Description Display Function

show\_description(name)-Opens a new window to display the description of a selected item from a DataFrame (df).

* Filtering Function

apply\_filters()-Filters the DataFrame based on selected soil types and irrigation types.

* Tile Reloading Function

reload\_tiles(canvas)-Reloads the tiles on the canvas after applying filters.

* Menu Toggle Function

toggle\_menu()-Collapses or expands the options frame in the GUI.

* Page Functions

home\_page(), cropfilter\_page(), cropcalendar\_page(), report\_page(), ai\_page(), settings\_page()-Define the layout of each page.

* Indicator Functions

hide\_all\_indicators(), delete\_pages(), indicate(lb, page)-Handle page indicators, hide indicators, and switch between pages.

* Reload Tiles Function (with No Results Label)

reload\_tiles(canvas, no\_results\_label)-Reloads tiles with an additional feature to display a "No Results" label when no items match the filter criteria.

* Main GUI Initialization

Root-Main Tkinter window.

main\_frame-Main container for different pages.

options\_frame-Sidebar with navigation buttons.

Navigation buttons (home\_button, cropfilter\_button, etc.): Switch between pages and highlight the selected page.

* Main Loop

root.mainloop()-Enters the Tkinter main event loop to handle user inputs and update the GUI.

Overall, the code uses a modular approach, organizing functionality into separate functions for clarity and maintainability. It creates an interactive GUI for navigating through different pages, filtering data, and displaying relevant information.

PROGRAM CODE

import tkinter as tk

from tkinter import Label, ttk

from PIL import Image, ImageTk #needed for adding images

import pandas as pd #needed for getting data from xlsx files

def on\_mousewheel(event, canvas):

    canvas.yview\_scroll(int(-1 \* (event.delta / 120)), "units") #

def show\_description(name):

    description\_window = tk.Toplevel()

    description\_window.title("Expanded View")

    description\_window.geometry("500x300")

    description\_window.config(bg="#333333")

    description\_label = tk.Label(description\_window, text=f"This is product description for {name}",bg="#333333",fg="white")

    description\_label.place(relx=0.5, rely=0.1, anchor='center')

    # Get description from the Excel file based on the selected 'name'

    description = df.loc[df['Name'] == name, 'Description'].values[0]

    crop\_description\_label = tk.Label(description\_window, text=f"Crop Description: {description}",bg="#333333",fg="white")

    crop\_description\_label.place(relx=0.5, rely=0.3, anchor='center')

    back\_button = tk.Button(description\_window, text="Back", command=description\_window.destroy,bg="#333333",fg="white")

    back\_button.place(relx=0.5, rely=0.5,y=200,anchor='center')

def apply\_filters():

    selected\_soil\_types = [soil\_type for soil\_type, var in soil\_vars.items() if var.get()]

    selected\_irrigation\_types = [irrigation\_type for irrigation\_type, var in irrigation\_vars.items() if var.get()]

    filtered\_df = df[df.apply(

        lambda row: any(soil\_type in row['Soil Type'].split(',') for soil\_type in selected\_soil\_types) and

                   any(irrigation\_type in row['Irrigation'].split(',') for irrigation\_type in selected\_irrigation\_types),

        axis=1

    )]

    return filtered\_df

#global tiles

def reload\_tiles(canvas):

    # Apply filters

    filtered\_df = apply\_filters()

    # Clear existing tiles

    for widget in canvas.winfo\_children():

        widget.destroy()

    # Calculate the vertical offset for each tile with a fixed gap

    tile\_height = 20

    gap = 30  # Adjust the gap between tiles as needed

    y\_offset = 75

    # Create new tiles with filtered data

    for index, row in filtered\_df.iterrows():

        name = row['Name']

        soil\_type = row['Soil Type']

        irrigation\_type = row['Irrigation']

        tile = tk.Label(canvas, text=f"{name}\n{soil\_type}\n{irrigation\_type}", height=8, width=45, bg="#FFD700",

                        relief=tk.SOLID, bd=1)

        tile.bind("<Button-1>", lambda event, name=name: show\_description(name))

        # Calculate vertical offset for each tile with a fixed gap

        y\_offset += tile\_height + gap

        canvas.create\_window(400, y\_offset, anchor='nw', window=tile)

    # Bind mouse wheel event to canvas

    canvas.bind\_all("<MouseWheel>", lambda event, canvas=canvas: on\_mousewheel(event, canvas))

    canvas.yview\_moveto(0)  # Scroll to the top after reloading

    lb = tk.Label(canvas, text="Crop List", font=("Leelawadee UI Semilight", 30),bg="#222222",fg="#ffffff")

    lb.pack(anchor='s')

def toggle\_menu():

    if options\_frame.winfo\_width() == 250:  # Menu is expanded

        options\_frame.place(width=50)  # Collapse to a smaller width

    else:  # Menu is collapsed

        options\_frame.place(width=250)  # Expand to full width

def home\_page():

    home\_frame = tk.Frame(main\_frame, bg="#333333", highlightthickness=2, highlightbackground="black")

    lb = tk.Label(home\_frame, text="Home Page", font=("Leelawadee UI Semilight", 30))

    lb.pack()

    home\_frame.pack(side=tk.LEFT)

    home\_frame.pack\_propagate(False)

    home\_frame.configure(width=1350, height=750)

def cropfilter\_page():

    cropfilter\_frame = tk.Frame(main\_frame, bg="#333333", highlightthickness=2, highlightbackground="black")

    lb = tk.Label(cropfilter\_frame, text="Crop Filter Page", font=("Bold", 30))

    lb.pack()

    # Scrollable Canvas

    canvas = tk.Canvas(cropfilter\_frame, bg="#222222", scrollregion=(0, 0, 1350, 20000))  # Light green color

    canvas.place(relx=0.5, rely=0.5, anchor='center', relwidth=0.75, relheight=1)

    lb = tk.Label(canvas, text="Crop List", font=("Leelawadee UI Semilight", 30),bg="#222222",fg="#ffffff")

    lb.pack(anchor='s')

    # Scrollbar

    scrollbar = ttk.Scrollbar(cropfilter\_frame, orient="vertical", command=canvas.yview)

    scrollbar.place(relx=1, rely=0, anchor='ne', relheight=1)

    canvas.configure(yscrollcommand=scrollbar.set)

    # Filter Bar Frame

    filter\_bar\_frame = tk.Frame(cropfilter\_frame, width=300, height=750, bg="#333333", bd=2,

                                relief=tk.SOLID)  # Light blue color

    filter\_bar\_frame.place(x=50, y=0, anchor='nw')

    # Read Excel file

    global df

    df = pd.read\_excel("list.xlsx")

    # Convert 'Soil Type' column to strings

    df['Soil Type'] = df['Soil Type'].astype(str)

    # Add "Sort" label with a box around it

    sort\_label = tk.Label(filter\_bar\_frame, text="Sort", font=("Bold", 16), bg="#333333", relief=tk.SOLID, bd=2,fg="#ffffff")

    sort\_label.place(relx=0.5, rely=0.1, anchor='center')

    # Get unique soil types

    all\_soil\_types = set(','.join(df['Soil Type']).split(','))

    soil\_types = list(all\_soil\_types)  # Remove 'All'

    # Add "Soil Type" label

    soil\_type\_label = tk.Label(filter\_bar\_frame, text="Soil Type", font=("Helvetica", 12), bg="#333333",fg="#ffffff")

    soil\_type\_label.place(relx=0.05, rely=0.2, anchor='w')

    # Checkboxes for each soil type

    global soil\_vars

    soil\_vars = {soil\_type: tk.BooleanVar(value=True) for soil\_type in soil\_types}

    soil\_checkboxes = [tk.Checkbutton(filter\_bar\_frame, text=soil\_type, variable=soil\_vars[soil\_type]) for soil\_type in

                       soil\_types]

    for i, checkbox in enumerate(soil\_checkboxes):

        checkbox.place(relx=0.05, rely=0.25 + i \* 0.03, anchor='w')

    # Add space between criteria

    tk.Label(filter\_bar\_frame, text="", bg="#333333").place(relx=0.5, rely=0.45, anchor='center')

    tk.Label(filter\_bar\_frame, text="", bg="#333333").place(relx=0.5, rely=0.45, anchor='center')

    # Modify the list of irrigation types

    all\_irrigation\_types = set(','.join(df['Irrigation']).split(','))

    irrigation\_types = list(all\_soil\_types)

    # Add "Irrigation" label

    irrigation\_label = tk.Label(filter\_bar\_frame, text="Irrigation", font=("Helvetica", 12), bg="#333333",fg="#ffffff")

    irrigation\_label.place(relx=0.05, rely=0.5, anchor='w')

    # Checkboxes for each irrigation type

    global irrigation\_vars

    irrigation\_vars = {irrigation\_type: tk.BooleanVar(value=True) for irrigation\_type in all\_irrigation\_types}

    irrigation\_checkboxes = [

        tk.Checkbutton(filter\_bar\_frame, text=irrigation\_type, variable=irrigation\_vars[irrigation\_type]) for

        irrigation\_type in

        all\_irrigation\_types]

    for i, checkbox in enumerate(irrigation\_checkboxes):

        checkbox.place(relx=0.05, rely=0.55 + i \* 0.03, anchor='w')

    # Add space between criteria and reload button

    tk.Label(filter\_bar\_frame, text="", bg="#333333").place(relx=0.5, rely=0.75, anchor='center')

    # Reload button

    reload\_button = tk.Button(filter\_bar\_frame, text="Reload", command=lambda: reload\_tiles(canvas, no\_results\_label),bg="#333333",fg="white")

    reload\_button.place(relx=0.5, rely=0.9, anchor='center')

    # Label to display "No Results" message

    no\_results\_label = tk.Label(cropfilter\_frame, text="No Results", font=("Helvetica", 20), fg="red", bg="#333333")

    no\_results\_label.place(relx=0.5, rely=0.5, anchor='center')

    no\_results\_label.pack\_forget()  # Initially, hide the label

    cropfilter\_frame.pack(side=tk.LEFT)

    cropfilter\_frame.pack\_propagate(False)

    cropfilter\_frame.configure(width=1350, height=750)

def reload\_tiles(canvas, no\_results\_label):

    # Apply filters

    filtered\_df = apply\_filters()

    # Clear existing tiles

    for widget in canvas.winfo\_children():

        widget.destroy()

    # Calculate the vertical offset for each tile with a fixed gap

    tile\_height = 20

    gap = 30  # Adjust the gap between tiles as needed

    y\_offset = 75

    tiles = []

    # Create new tiles with filtered data

    for index, row in filtered\_df.iterrows():

        name = row['Name']

        soil\_type = row['Soil Type']

        irrigation\_type = row['Irrigation']

        tile = tk.Label(canvas, text=f"Crop Name :{name}\nSoil Type :{soil\_type}\nIrrigation Requirements :{irrigation\_type}", height=10, width=100, bg="#333333",

                        relief=tk.SOLID, bd=1,fg="#ffffff")

        tile.bind("<Button-1>", lambda event, name=name: show\_description(name))

        # Calculate vertical offset for each tile with a fixed gap

        y\_offset += tile\_height + gap

        tiles.append(tile)

    if len(tiles) == 0:

        no\_results\_label.place(relx=0.5, rely=0.5, anchor='center')  # Display the label when there are no results

    else:

        for tile in tiles:

            canvas.create\_window(250, 200 \* tiles.index(tile) + 75, anchor='nw', window=tile)

        # Bind mouse wheel event to canvas

        canvas.bind\_all("<MouseWheel>", lambda event, canvas=canvas: on\_mousewheel(event, canvas))

        canvas.yview\_moveto(0)  # Scroll to the top after reloading

        no\_results\_label.place\_forget()  # Hide the label when there are results

def cropcalendar\_page():

    cropcalendar\_frame = tk.Frame(main\_frame, bg="#333333", highlightthickness=2, highlightbackground="black")

    lb = tk.Label(cropcalendar\_frame, text="Crop Calendar Page", font=("Bold", 30))

    lb.pack()

    cropcalendar\_frame.pack(side=tk.LEFT)

    cropcalendar\_frame.pack\_propagate(False)

    cropcalendar\_frame.configure(width=1350, height=750)

def report\_page():

    report\_frame = tk.Frame(main\_frame, bg="#333333", highlightthickness=2, highlightbackground="black")

    lb = tk.Label(report\_frame, text="Report Page", font=("Bold", 30))

    lb.pack()

    report\_frame.pack(side=tk.LEFT)

    report\_frame.pack\_propagate(False)

    report\_frame.configure(width=1350, height=750)

def ai\_page():

    ai\_frame = tk.Frame(main\_frame, bg="#333333", highlightthickness=2, highlightbackground="black")

    lb = tk.Label(ai\_frame, text="AI Page", font=("Bold", 30))

    lb.pack()

    ai\_frame.pack(side=tk.LEFT)

    ai\_frame.pack\_propagate(False)

    ai\_frame.configure(width=1350, height=750)

def settings\_page():

    settings\_frame = tk.Frame(main\_frame, bg="#333333", highlightthickness=2, highlightbackground="black")

    lb = tk.Label(settings\_frame, text="Settings Page", font=("Bold", 30))

    lb.pack()

    settings\_frame.pack(side=tk.LEFT)

    settings\_frame.pack\_propagate(False)

    settings\_frame.configure(width=1350, height=750)

def hide\_all\_indicators():

    home\_indicate.config(bg="#222222")

    cropfilter\_indicate.config(bg="#222222")

    cropcalendar\_indicate.config(bg="#222222")

    report\_indicate.config(bg="#222222")

    ai\_indicate.config(bg="#222222")

    settings\_indicate.config(bg="#222222")

def delete\_pages():

    for frame in main\_frame.winfo\_children():

        frame.destroy()

def indicate(lb, page):

    hide\_all\_indicators()

    lb.config(bg="#1DB954")

    delete\_pages()

    page()

root = tk.Tk()

root.geometry("1350x750")

root.title("Harvest Hub")

main\_frame = tk.Frame(root, bg="#333333", highlightthickness=2, highlightbackground="black")

options\_frame = tk.Frame(root, bg="#222222", highlightthickness=0, highlightbackground="black")

mainframe\_label = Label(main\_frame, text="WELCOME TO HARVEST HUB", bg="#333333", fg="#1DB954")

mainframe\_label.pack()

image\_path = "menu.png"

image = Image.open(image\_path)

photo = ImageTk.PhotoImage(image)

toggle\_button = tk.Button(options\_frame, image=photo, compound=tk.LEFT, text="", font=("bold", 15), bg="#333333",

                          fg="#333333", command=toggle\_menu)

toggle\_button.place(x=10, y=15)

home\_button = tk.Button(options\_frame, width=20, text="Home", font=("bold", 15), fg="#1DB954", bg="#333333",

                        command=lambda: indicate(home\_indicate, home\_page))

home\_button.place(x=10, y=100)

home\_indicate = tk.Label(options\_frame, text="", bg="#222222")

home\_indicate.place(x=0, y=100, width=5, height=40)

cropfilter\_button = tk.Button(options\_frame, width=20, text="Crop Filter", font=("bold", 15), fg="#1DB954",

                              bg="#333333", command=lambda: indicate(cropfilter\_indicate, cropfilter\_page))

cropfilter\_button.place(x=10, y=175)

cropfilter\_indicate = tk.Label(options\_frame, text="", bg="#222222")

cropfilter\_indicate.place(x=0, y=175, width=5, height=40)

cropcalendar\_button = tk.Button(options\_frame, width=20, text="Crop Calendar", font=("bold", 15), fg="#1DB954",

                                bg="#333333", command=lambda: indicate(cropcalendar\_indicate, cropcalendar\_page))

cropcalendar\_button.place(x=10, y=250)

cropcalendar\_indicate = tk.Label(options\_frame, text="", bg="#222222")

cropcalendar\_indicate.place(x=0, y=250, width=5, height=40)

report\_button = tk.Button(options\_frame, width=20, text="Report", font=("bold", 15), fg="#1DB954", bg="#333333",

                          command=lambda: indicate(report\_indicate, report\_page))

report\_button.place(x=10, y=325)

report\_indicate = tk.Label(options\_frame, text="", bg="#222222")

report\_indicate.place(x=0, y=325, width=5, height=40)

ai\_button = tk.Button(options\_frame, width=20, text="Ask Ai", font=("bold", 15), fg="#1DB954", bg="#333333",

                      command=lambda: indicate(ai\_indicate, ai\_page))

ai\_button.place(x=10, y=550)

ai\_indicate = tk.Label(options\_frame, text="", bg="#222222")

ai\_indicate.place(x=0, y=550, width=5, height=40)

settings\_button = tk.Button(options\_frame, width=20, text="Settings", font=("bold", 15), fg="#1DB954", bg="#333333",

                            command=lambda: indicate(settings\_indicate, settings\_page))

settings\_button.place(x=10, y=625)

settings\_indicate = tk.Label(options\_frame, text="", bg="#222222")

settings\_indicate.place(x=0, y=625, width=5, height=40)

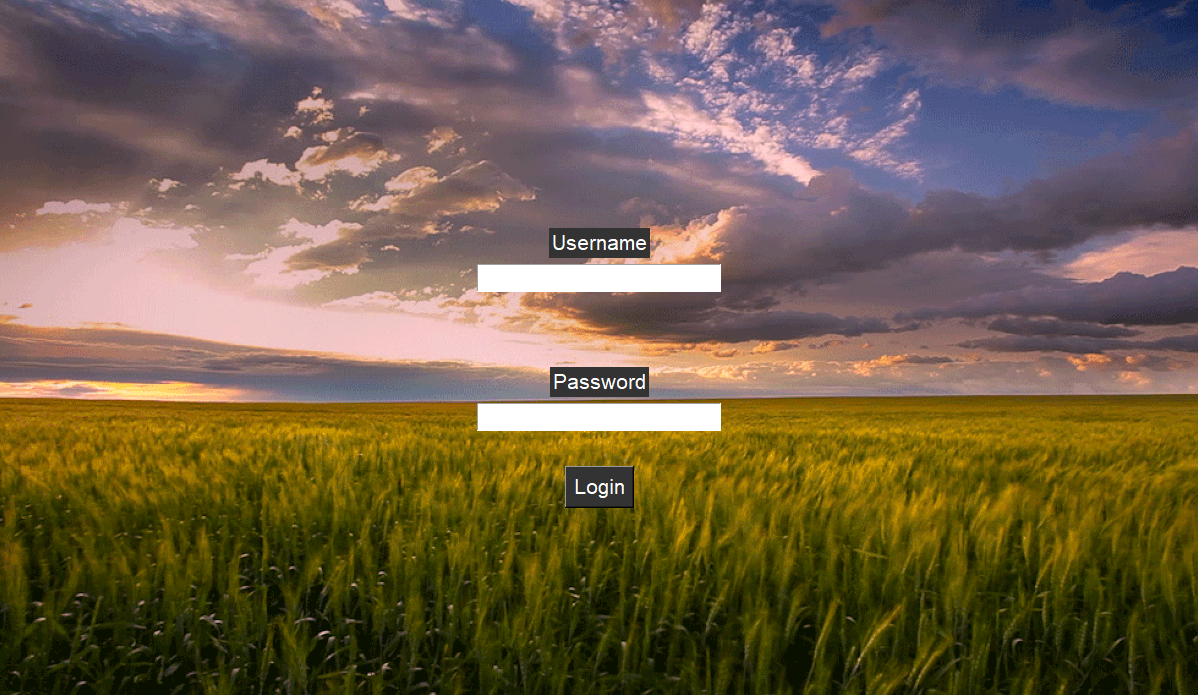
main\_frame.place(x=0, y=0, width=1350, height=750)

options\_frame.place(x=0, y=0, width=250, height=750)

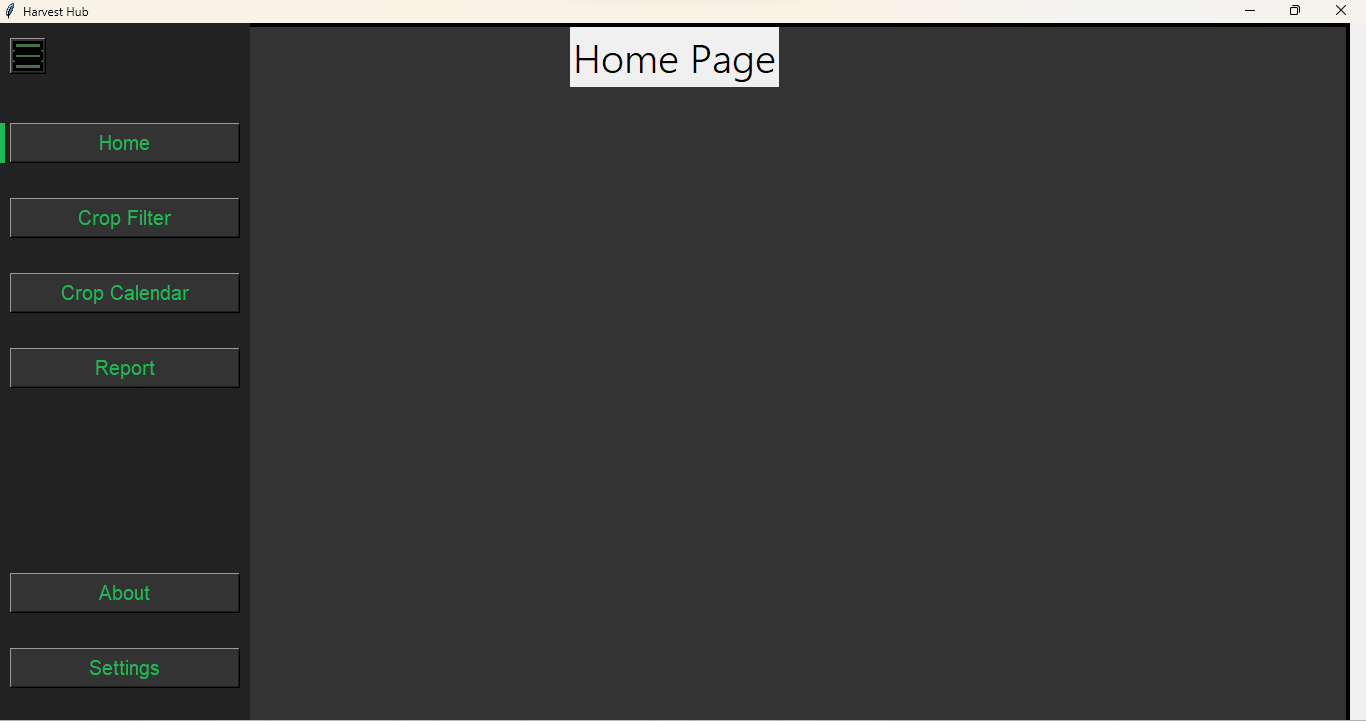
root.mainloop()

PROGRAM OUTPUT

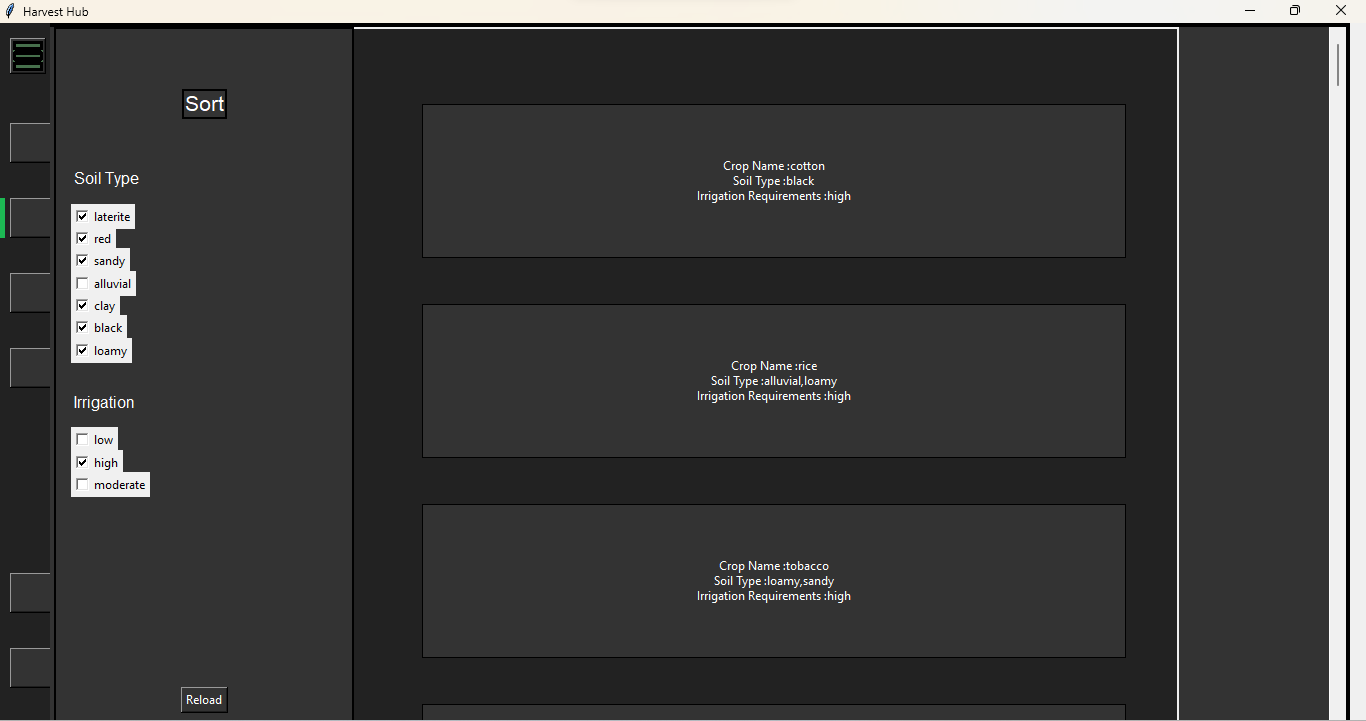
Login Page



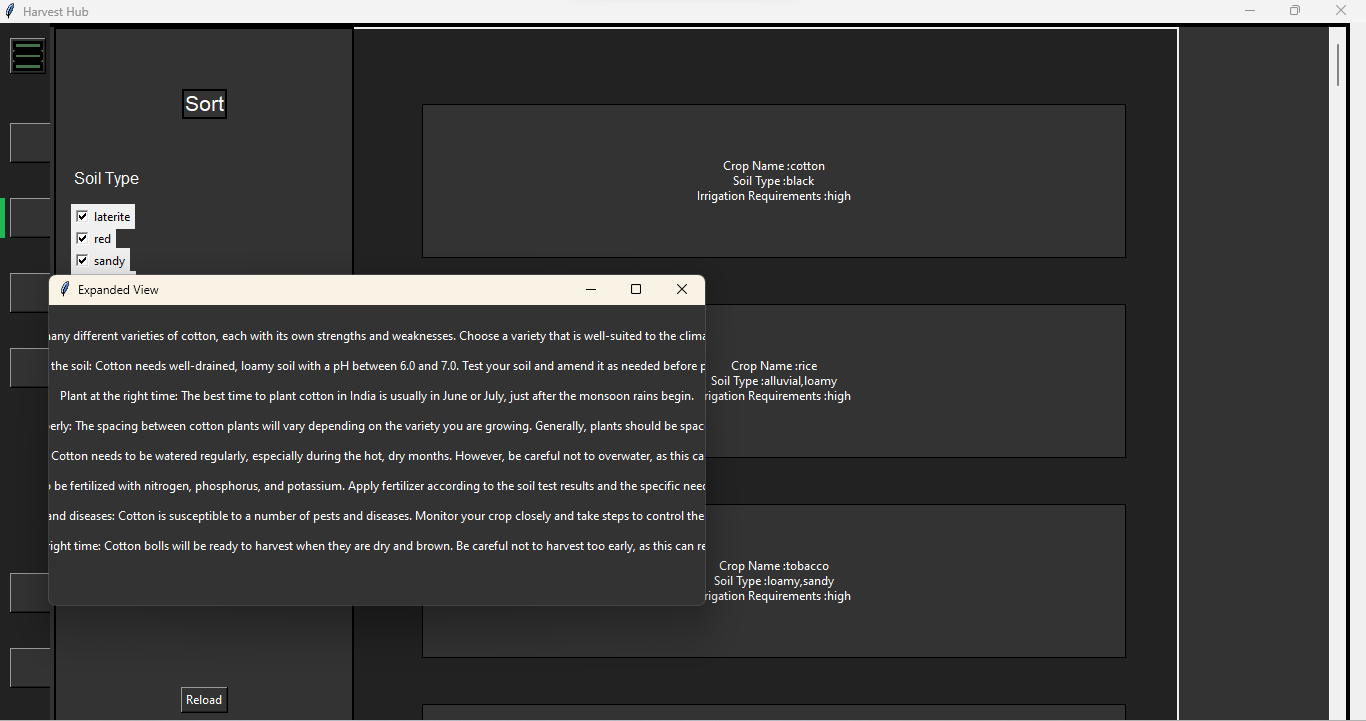
Homepage and sidebar

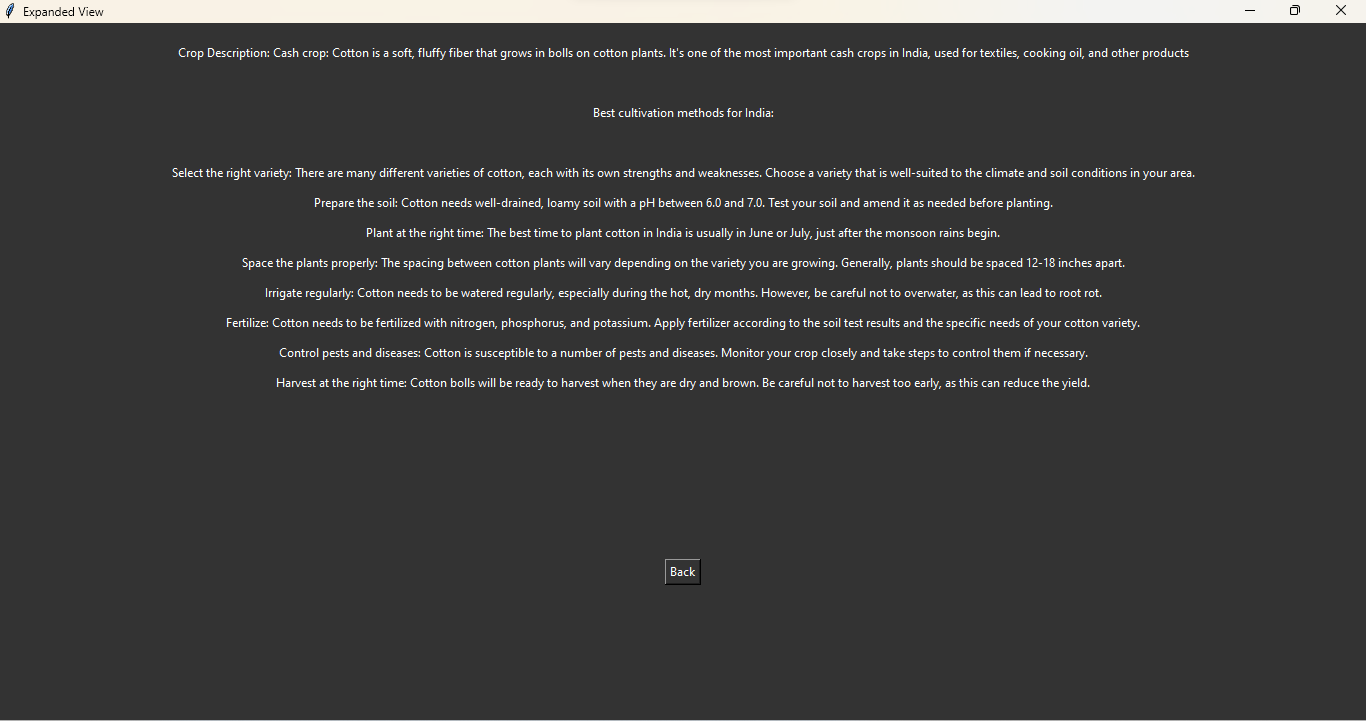


Crop Filter

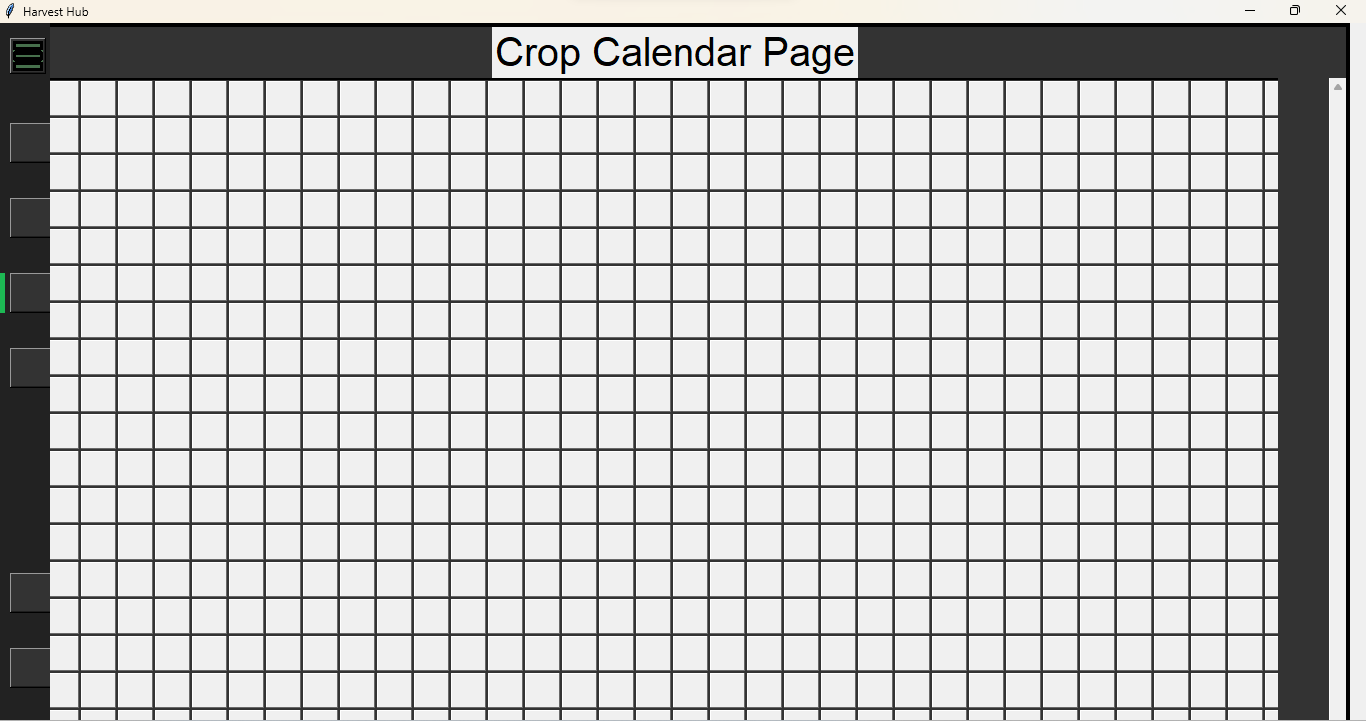


Crop description (upon clicking desired crop)

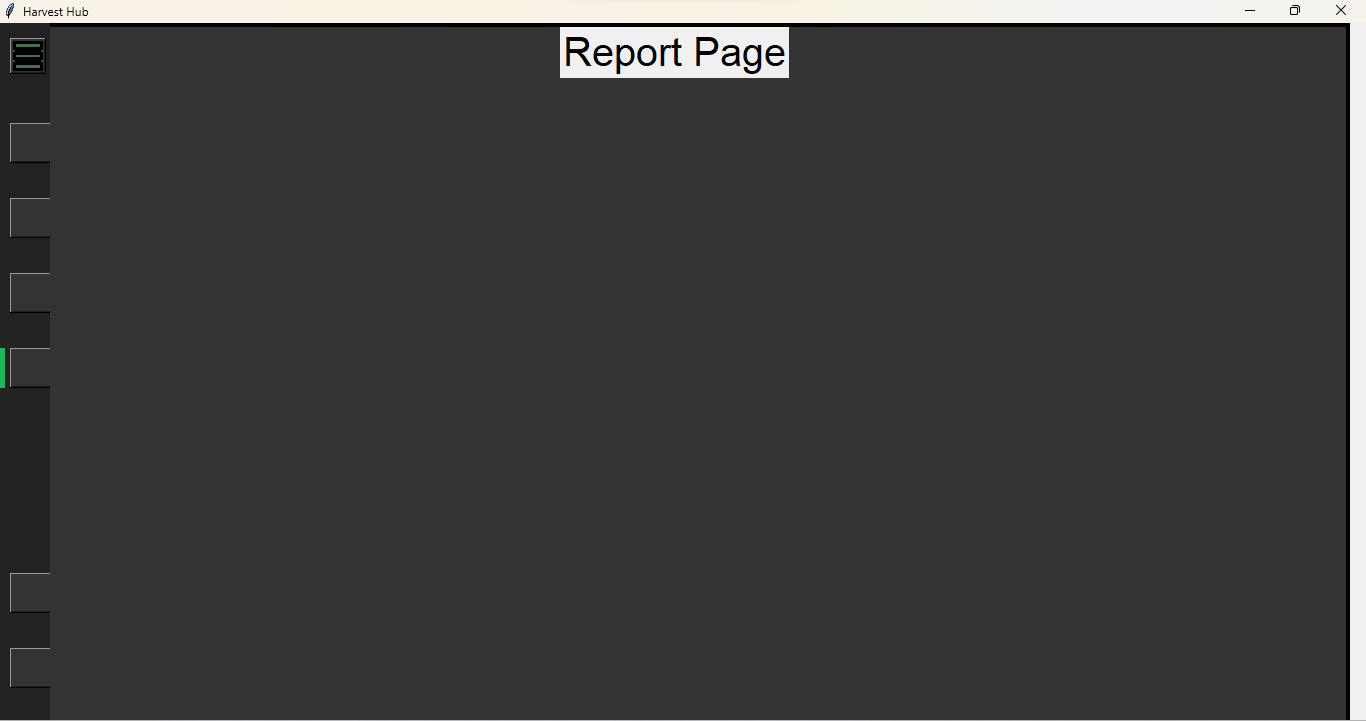




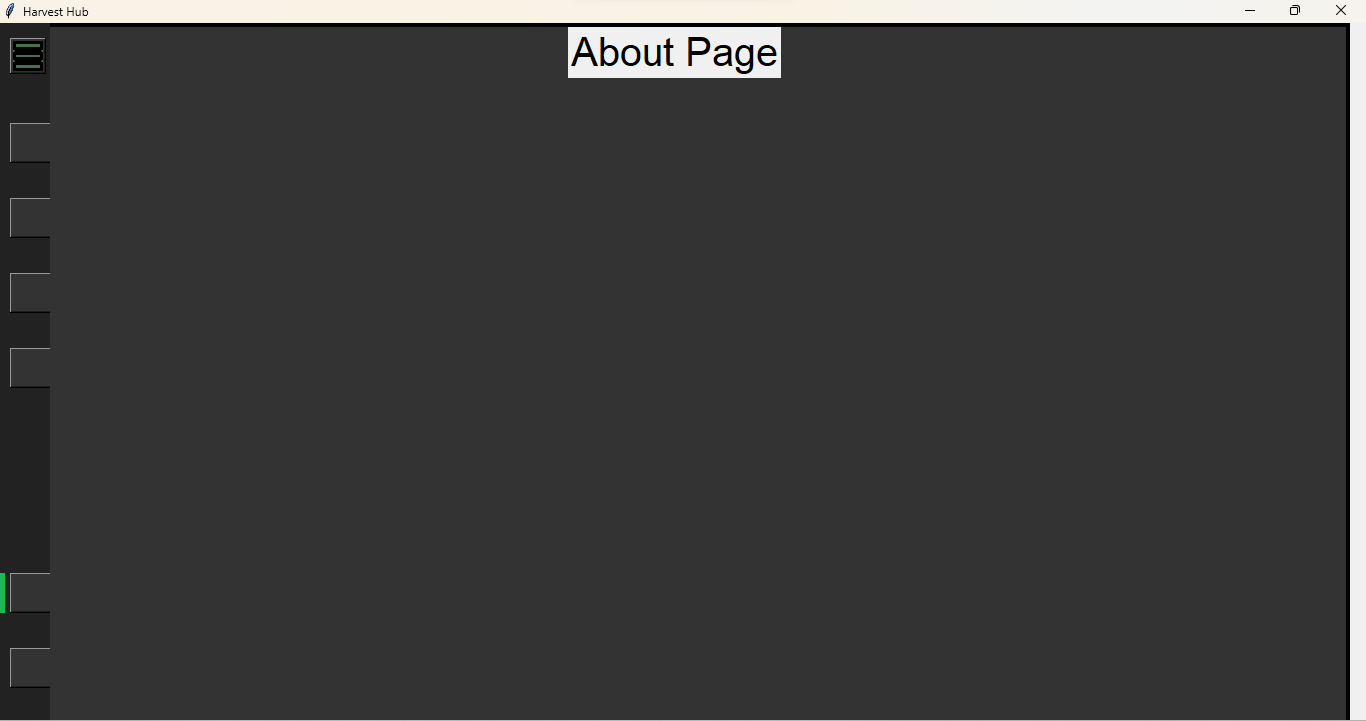
Crop Calendar (Unfinished)



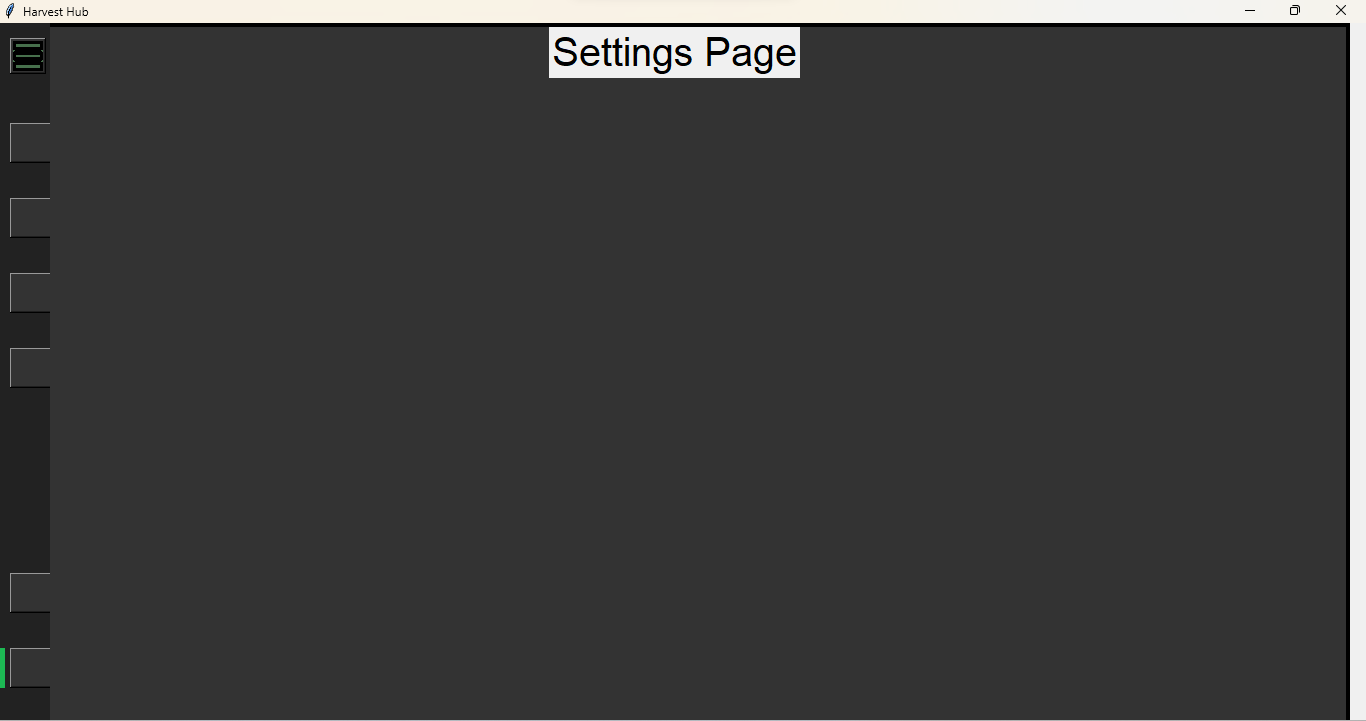
Report Page (TBC)



About Page (TBC)



Settings (TBC)



LIMITATIONS AND FUTURE PLANS

Below are the limitations:

* Place an “Add To Profile” button in crop description page, which adds information about that particular crop to the report and calendar page
* Home Page to be finished
* Crop Calendar-importing of data from crop filter page incomplete
* About and settings page unfinished
* Report page unfinished- will show details about crops that have been added to profile, with relevant growth charts and graphs.

BIBLIOGRAPHY

* Geeksforgeeks.org
* Uizard.io
* Github.io
* Docs.python.org
* https://data.oecd.org/agroutput/crop-production.htm