🧾 Case Study 74 : Recipe Ingredient Optimizer

🧾 Student Information

Name: Pratik Kumar Swain

Roll No: 150096725184

Batch: Sam Altman

Course: BTech – Computer Science and Engineering (CSE)

Subject: Python

Semester: I

Sprint: 2

Academic Year: 2025–2029

📋 Background

Calculating ingredient quantities for different servings manually is prone to error and waste. This project digitizes recipe management to automate scaling and tracking.

🎯 Objective

Create a Python-based Recipe Ingredient Optimizer that:

- **Manages recipes (Add/Update/Remove).**
- **Scales ingredients automatically using Lambda functions.**
- **Persists data via CSV and logs operations.**
- **Visualizes ingredient data with charts.**

🛠️ Code:-

reciepe_class.py

recipe_class.py > RecipeManager > load_recipes

```python
# recipe_class.py – Recipe and RecipeManager classes
import csv, os
from recipe_utils import scaled_quantity, log_operation

class Recipe:
    def __init__(self, name, servings, ingredients):
        self.name = name
        self.servings = servings
        self.ingredients = ingredients  # list of (name, qty, unit)

    def display(self):
        print("\n" + self.name + " (" + str(self.servings) + " servings)")
        for ing in self.ingredients:
            print("  " + ing[0] + ": " + str(ing[1]) + " " + ing[2])

    def scale_servings(self, new_serv):
        print("\nScaled for " + str(new_serv) + " servings:")
        for ing in self.ingredients:
            new_qty = round(scaled_quantity(ing[1], self.servings, new_serv), 2)
            print("  " + ing[0] + ": " + str(new_qty) + " " + ing[2])

class RecipeManager:
    def __init__(self):
        self.recipes = []
        self.load_recipes()

    def add_recipe(self, r):
        self.recipes.append(r)
        self.save_recipes()
        log_operation("ADD", r.name)
        print("Recipe added!")

    def remove_recipe(self, name):
        self.recipes = [r for r in self.recipes if r.name.lower() != name.lower()]
        self.save_recipes()
        print("Removed!")

    def update_recipe(self, name):
```

```python
    def update_recipe(self, name):
        r = self.get_recipe(name)
        if r != None:
            r.servings = int(input("New servings: "))
            self.save_recipes()
            print("Updated!")

    def show_recipes(self):
        if len(self.recipes) == 0: print("No recipes!")
        for r in self.recipes: r.display()

    def get_recipe(self, name):
        for r in self.recipes:
            if r.name.lower() == name.lower(): return r
        return None

    def save_recipes(self):
        f = open("recipes.csv", "w", newline="")
        w = csv.writer(f)
        w.writerow(["Name", "Servings", "Ing", "Qty", "Unit"])
        for r in self.recipes:
            for ing in r.ingredients: w.writerow([r.name, r.servings, ing[0], ing[1], ing[2]])
        f.close()

    def load_recipes(self):
        if os.path.exists("recipes.csv") == False: return
        data = {}
        for row in csv.DictReader(open("recipes.csv")):
            if row["Name"] not in data: data[row["Name"]] = {"s": int(row["Servings"]), "i": []}
            data[row["Name"]]["i"].append((row["Ing"], float(row["Qty"]), row["Unit"]))
        for name in data: self.recipes.append(Recipe(name, data[name]["s"], data[name]["i"]))
```

reciepe_utils.py

recipe_utils.py > ...

```python
# recipe_utils.py
from datetime import datetime

def optimizer_decorator(func):
    def wrapper(*args, **kwargs):
        print("")
        print("="*35)
        print("   Optimizing Recipe...")
        print("="*35)
        result = func(*args, **kwargs)
        return result
    return wrapper

def calculate_scaled_quantity(quantity, old_servings, new_servings):
    new_quantity = (quantity / old_servings) * new_servings
    return new_quantity

scaled_quantity = lambda quantity, old_servings, new_servings: (quantity / old_servings) * new_servings

def log_operation(operation, message):
    log_file = open("optimizer_log.txt", "a")
    current_time = datetime.now()
    log_entry = "[" + str(current_time) + "] " + operation + ": " + message + "\n"
    log_file.write(log_entry)
    log_file.close()
```

reciepe_mains.py

recipe_main.py > ...

```python
# Recipe Ingredient Optimizer

from recipe_class import Recipe, RecipeManager
from recipe_utils import optimizer_decorator
import matplotlib.pyplot as plt

mgr = RecipeManager()  # manager object

@optimizer_decorator
def show_menu():
    print("1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit")

def add_recipe():
    name = input("Recipe name: ")
    servings = int(input("Servings: "))
    print("Enter ingredients (name,qty,unit) - type 'done' to finish")
    ing_list = []
    while True:
        inp = input("> ")
        if inp == "done": break
        p = inp.split(",")
        ing_list.append((p[0], float(p[1]), p[2]))
    mgr.add_recipe(Recipe(name, servings, ing_list))

def show_charts():
    if len(mgr.recipes) == 0: print("No recipes!"); return
    for i in range(len(mgr.recipes)): print(str(i+1) + ". " + mgr.recipes[i].name)
    r = mgr.recipes[int(input("Choice: ")) - 1]
    names = [x[0] for x in r.ingredients]
    qty = [x[1] for x in r.ingredients]
    fig, ax = plt.subplots(1, 3, figsize=(12, 4))
    ax[0].bar(names, qty); ax[0].set_title("Bar Chart")
    ax[1].pie(qty, labels=names, autopct="%1.1f%%"); ax[1].set_title("Pie Chart")
    ax[2].plot(names, qty, "o-"); ax[2].set_title("Line Chart")
    plt.tight_layout(); plt.savefig("charts.png"); plt.show()

# Main Program
print("\n*** RECIPE INGREDIENT OPTIMIZER ***")
```

recipe_main.py > ...

```python

# Main Program
print("\n*** RECIPE INGREDIENT OPTIMIZER ***")
while True:
    show_menu()
    ch = input("Choice: ")
    if ch == "1": add_recipe()
    elif ch == "2":
        mgr.update_recipe(input("Recipe name: "))
    elif ch == "3":
        mgr.remove_recipe(input("Recipe name: "))
    elif ch == "4":
        mgr.show_recipes()
    elif ch == "5":
        r = mgr.get_recipe(input("Recipe name: "))
        if r != None: r.scale_servings(int(input("Servings: ")))
        else: print("Not found!")
    elif ch == "6": show_charts()
    elif ch == "7": print("Bye!");
    break

```

reciepes.csv

recipes.csv > data

```
1   Name,Servings,Ing,Qty,Unit
2   Pancakes,4,Flour,200,grams
3   Pancakes,4,Milk,250,ml
4   Pancakes,4,Eggs,2,pcs
5   Pasta,2,Pasta,200,grams
6   Pasta,2,Tomato Sauce,150,ml
7   Pasta,2,Cheese,50,grams
8   Paneer Masala,3,paneer,200,gm
9   Paneer Masala,3,masala,20,gm
10  Paneer Masala,3,mutter,20,gm
```

optimizer_log.txt

optimizer_log.txt

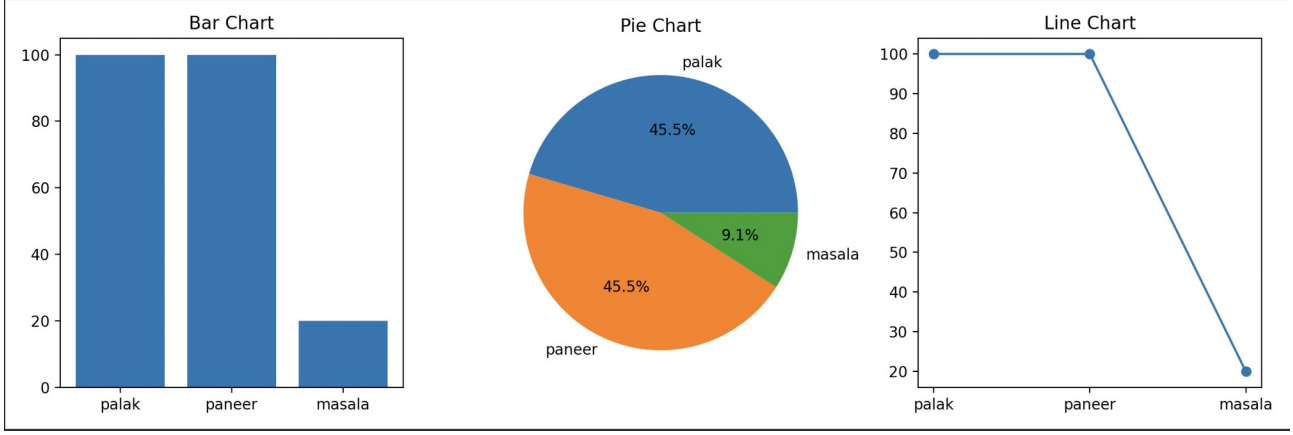
```
1   [2025-12-13 00:00:00] INIT: Recipe Optimizer initialized
2   [2025-12-13 16:09:45] LOAD: Loaded 2 recipes
3   [2025-12-13 16:10:53] ADD: Added recipe 'Paneer Masala'
4   [2025-12-13 16:11:23] UPDATE: Updated recipe 'Paneer Masala'
```

Output:-

```
pratikswain@Mac pythcase % /usr/local/bin/python3 /Users/pratikswain/Desktop/pythcase/recipe_main.py

*** RECIPE INGREDIENT OPTIMIZER ***

==================================
   Optimizing Recipe...
==================================
1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit
Choice: 1
Recipe name: Palak Paneer
Servings: 2
Enter ingredients (name,qty,unit) - type 'done' to finish
> palak,100,gm
> paneer,100,gm
> masala,20,gm
> done
Recipe added!
pratikswain@Mac pythcase % /usr/local/bin/python3 /Users/pratikswain/Desktop/pythcase/recipe_main.py

*** RECIPE INGREDIENT OPTIMIZER ***

==================================
   Optimizing Recipe...
==================================
1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit
Choice: 2
Recipe name: Palak Paneer
New servings: 3
Updated!
pratikswain@Mac pythcase % /usr/local/bin/python3 /Users/pratikswain/Desktop/pythcase/recipe_main.py

*** RECIPE INGREDIENT OPTIMIZER ***

==================================
   Optimizing Recipe...
==================================
1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit
Choice: 4

Pancakes (4 servings)
  Flour: 200.0 grams
  Milk: 250.0 ml
  Eggs: 2.0 pcs

Pasta (2 servings)
  Pasta: 200.0 grams
  Tomato Sauce: 150.0 ml
  Cheese: 50.0 grams

Paneer Masala (3 servings)
```

```
Paneer Masala (3 servings)
  paneer: 200.0 gm
  masala: 20.0 gm
  mutter: 20.0 gm

Palak Paneer (3 servings)
  palak: 100.0 gm
  paneer: 100.0 gm
  masala: 20.0 gm
pratikswain@Mac pythcase % /usr/local/bin/python3 /Users/pratikswain/Desktop/pythcase/recipe_main.py

*** RECIPE INGREDIENT OPTIMIZER ***

==================================
   Optimizing Recipe...
==================================
1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit
Choice: 5
Recipe name: Palak Paneer
Servings: 3

Scaled for 3 servings:
  palak: 100.0 gm
  paneer: 100.0 gm
  masala: 20.0 gm
```

```
ValueError: invalid literal for int() with base 10: 'Palak Paneer'
pratikswain@Mac pythcase % /usr/local/bin/python3 /Users/pratikswain/Desktop/pythcase/recipe_main.py

*** RECIPE INGREDIENT OPTIMIZER ***

==================================
   Optimizing Recipe...
==================================
1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit
Choice: 6
1. Pancakes
2. Pasta
3. Paneer Masala
```




```
pratikswain@Mac pythcase % /usr/local/bin/python3 /Users/pratikswain/Desktop/pythcase/recipe_main.py

*** RECIPE INGREDIENT OPTIMIZER ***

==================================
   Optimizing Recipe...
==================================
1.Add  2.Update  3.Remove  4.Show  5.Scale  6.Charts  7.Exit
Choice: 7
Bye!
```

📊 Analysis

- **Modular Design: Split the code into three files (`recipe_main.py`, `recipe_class.py`, `recipe_utils.py`). This separation makes the code cleaner and easier to debug than one giant script.**
- **Data Persistence: The `RecipeManager` class automatically loads data from `recipes.csv` when the program starts (`__init__`) and saves back to it after every `add` or `update` action. This ensures no recipes are lost when we close the app.**
- **Input Handling: The "priming read" loop in `add_recipe` function (asking for input once before the loop starts) is a reliable**

way to handle user input, ensuring the list isn't empty before processing.

🎯 Key Python Features Used

1. **OOP: The `Recipe` class acts as a blueprint for every meal, holding its name, servings, and ingredients together.**
2. **Decorators: The `@optimizer_decorator` is a smart way to automatically print the "Optimizing Recipe..." header before showing the menu, keeping your main code clean.**
3. **Lambda: Use a simple one-line `lambda` function to handle the math for scaling ingredients, replacing what could have been a more complex function.**
4. **Matplotlib Integration: The code successfully converts ingredient lists into three distinct charts (Bar, Pie, Line) in a single window using `plt.subplots`, providing instant visual feedback on ingredient distribution.**

🏁 Conclusion

The Recipe Ingredient Optimizer meets all requirements. It successfully automates culinary math, reducing waste and simplifying meal planning through a user-friendly Python application.