Python Guild Team Programming

Semester Project

Instructor: William A Loring

Table of Contents

Software engineering is rarely done alone. It is almost always done as part of a team.

The class will be divided into guilds of 3-5 people each. Guild membership is assigned by the Game Master (the instructor).

Each guild should come up with their own name (let's keep these PG-Rated please). Please let me know your Guild number and name. I will re name your GitHub repository.

Share Your Experience

In any class, some students, depending upon their major, programming experience, artistic talent, etc. may exhibit more proficiency than others on certain aspects of the assigned course work. A Guild gives an opportunity for everyone on the team to work together and share their experience.

This project is as much about working as a team as it is about the assignment. The process of coming together, helping each other out is a huge part of team-based software engineering.

Guild Assignments

Guild assignments are team projects where everyone works on the same code project.

Guild members are encouraged to be resources for the other Guild members. They can lend a hand if someone gets stuck on a problem. A 2nd eye on the code can find a missing semi colon or other minor error.

Individual assignments are not team projects. Everyone does their own work.

Blackboard Groups

Each Guild Team has a Group area in Blackboard under Groups. This is used primarily for grading purposes.

Outlook Groups and Discord

An Outlook Group will be created for each Guild. This is used to share documents, send email, and work collaboratively.

Week 10 Milestone: Project Kickoff and Team Charter

100 points

Time required: 60 minutes

1. Get together with your group using some sort of synchronous communication such as Teams or Zoom. Add a screenshot showing the participants to this assignment.
2. **Read:** Creating Productive Teams
3. **Read:** Elements of a High Performing Team
4. **Read:** Team Charter

Do: Determine how your Guild will handle the following items.

1. What is the name of your Guild?
2. Communication
    a. Synchronous (required)
        i. Teams
        ii. Zoom
        iii. Video conference software of your choice
    b. Asynchronous
        i. Outlook Group Email (Required
3. Collaboration on documents
    a. Outlook Group Files (Required)
4. The Guild will create a list of the skills needed for the project.
    a. Team skills
    b. Computer skills
5. Each member will create a list of the skills they bring to the project.
    a. Team skills
    b. Computer skills
6. Guild leader of the week
    a. The Guild leader is not responsible for doing all the work, only organizing, or getting everyone together.
    b. Rotate between each member for Guild leader of the week.

   c. A recommended practice is to set a schedule for Guild leader rotation at the beginning of the project.
   d. The Guild leader is responsible for submitting the project assignment for that week.

7. Expectations for Guild Members
   a. Set your shared expectations for the Guild
   b. Responsiveness
      i. What are the expectations for response to communication?
   c. This is a collaborative team project.
      i. It is not ok to say to the rest of the team: plan this week without me and let me know what I am supposed to do.
      ii. Collaboration means each team member contributes equally.
   d. Deadlines for individual assignments
   e. Deadline for team assignments
   f. If these expectations are not met, ask the team member why
   g. If there is no response, move on without them
   h. Give an honest evaluation of individual performance with the evaluation

Week 10 Assignment Submission

The Guild Leader submits a Word document detailing this information in Blackboard.

Week 11 Milestone: Git Started with GitHub and Kat's Lemonade Stand

100 points

Time Required: 90 minutes

GitHub

Each Guild will have a separate shared GitHub Repository. This repository should be used to store any code or text documentation pertinent to that assignment.

Agile Software Development

Agile development is one of the current processes for software development and other development activities.

What is Agile?

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals.

https://www.cprime.com/resources/what-is-agile-what-is-scrum

DRY Software Engineering

Don’t **R**epeat **Y**ourself (**DRY**) is a principle of software engineering aimed at reducing repetition of software patterns. It you are repeating any code, there is probably a better solution.

The DRY principle is stated as "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system". This means that there shouldn’t be anything in your code that is duplicated somewhere else.

Violations of DRY are typically referred to as WET solutions, which is commonly taken to stand for "write every time", "write everything twice", "we enjoy typing" or "waste everyone's time".

Create classes, functions or methods with a single purpose or theme. Abstract as much as possible from the application to the classes. The application contains as little logic as possible. It creates and uses objects and their methods.

The Assignment

This assignment will take your Guild through a tutorial about working with GitHub in a shared software engineering environment. This will prepare you for the first shared coding assignment at the end of this tutorial.

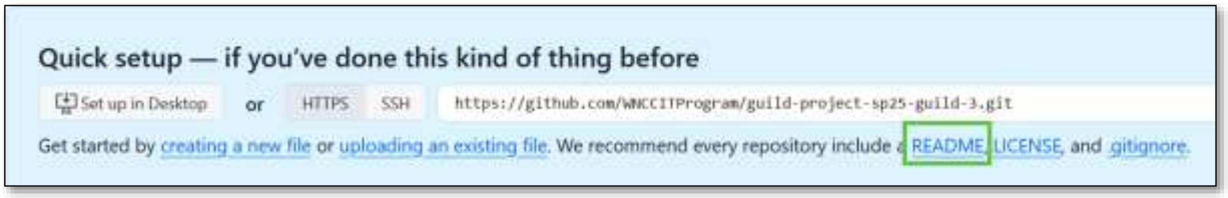
Git Started with the Guild Project

The first video is about how to join the Guild repository and make your initial clone and synchronization of your Guilds repository.

Video walkthrough: Git Started with the Guild Project (You only do this once.)

GitHub Synchronization Issues: How to Fix GitHub Synchronization Issues.

1. Click the link to the assignment → https://classroom.github.com/a/btMVqhy-
2. Join the classroom: **Python Sp25**.
3. **If you are the first person in the GitHub assignment:**
    a. Type in the name of your team: Guild (number) Guild name
    b. For example: Guild 2 Python Persuaders.
    c. Click **Create Team**.
    d. Accept this assignment.
    e. Click on the team's assignment repository.
    f. Click **README**.
    g. Click Commit changes → Click Commit changes. This will create a sample README.md file.

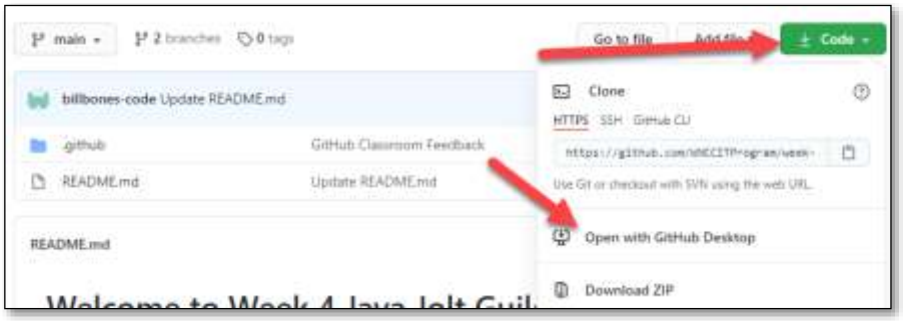

Quick setup — if you've done this kind of thing before

https://github.com/WNCCITProgram/guild-project-sp25-guild-3.git

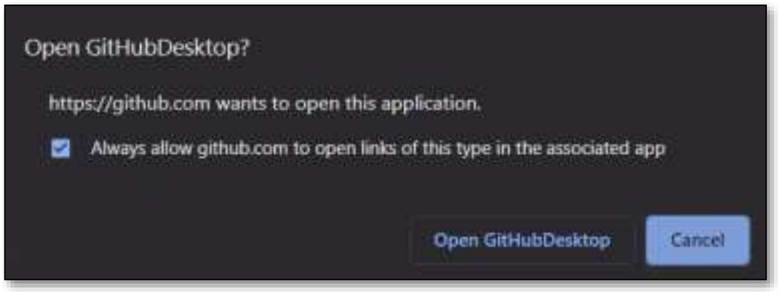
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

    h. Click Code in the upper right to go to the main repository screen
    i. Your repository is complete.
    j. Email the instructor the name of your Guild. Starter code files will be placed in your repository for you to work on the assignment.

4. **Otherwise:** Join your Guild Team.
5. Click on the repository link.
6. Click the green **Code** button → **Open with GitHub Desktop**.


main
2 branches
0 tags
Go to file
Add file
Code
billbones-code Update README.md
.github
GitHub Classroom Feedback
README.md
Update README.md
README.md
Clone
HTTPS
SSH
GitHub CLI
https://github.com/WNCCCITProgram/week-
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

7. Click the checkmark as shown below to speed up your development process. The next time you open a GitHub repository from the web with GitHub Desktop, it will automatically open.

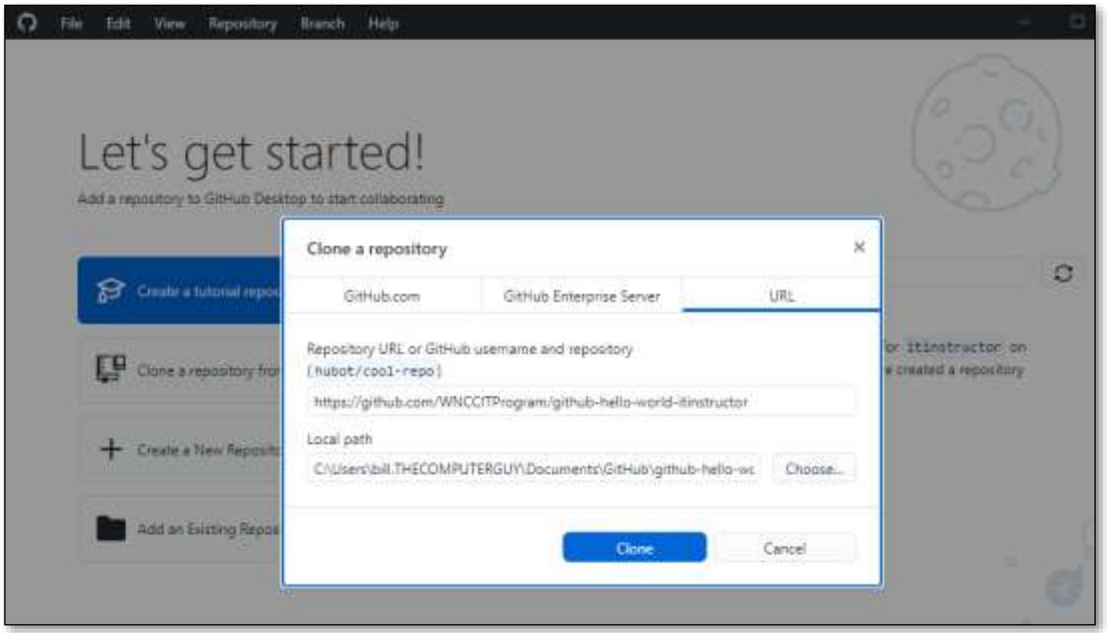


Open GitHubDesktop?

https://github.com wants to open this application.

☑ Always allow github.com to open links of this type in the associated app

8. Click **Open GitHubDesktop**.
9. Make sure that your local path is where you are storing your GitHub repositories.
10. Click **Clone**. This synchronizes the GitHub repository to your local computer repository.



Clone a repository

GitHub.com | GitHub Enterprise Server | URL

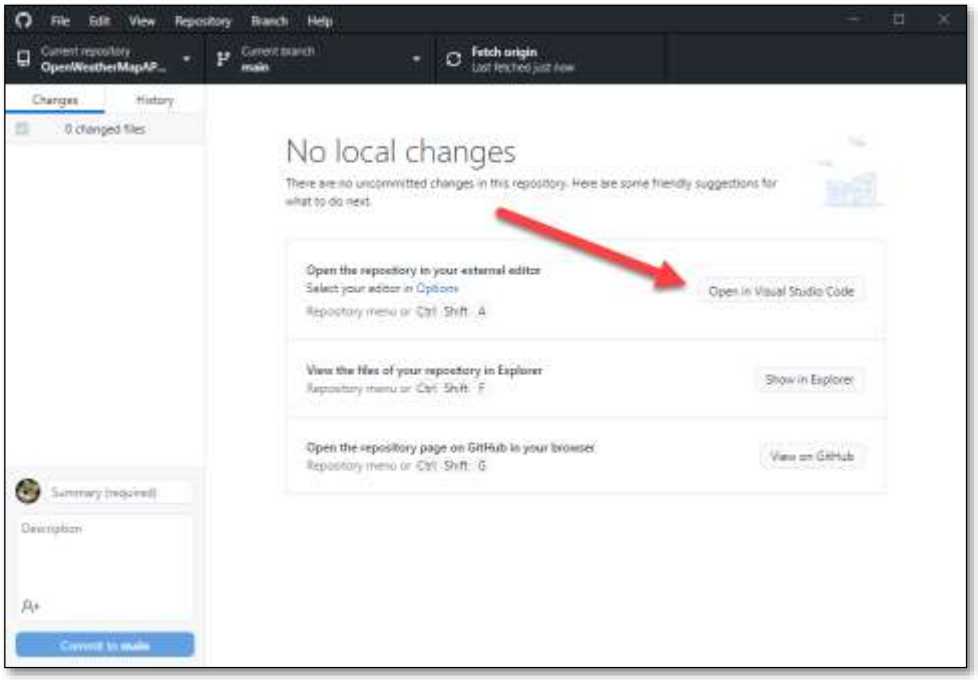
Repository URL or GitHub username and repository
(`hubot/cool-repo`)

https://github.com/WNCCITProgram/github-hello-world-itinstructor

Local path

C:\Users\bill.THECOMPUTERGUY\Documents\GitHub\github-hello-wc

11. On the right-hand side Click: **Open in Visual Studio Code.**



No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Open the repository in your external editor
Select your editor in Options
Repository menu or Ctrl Shift A

View the files of your repository in Explorer
Repository menu or Ctrl Shift F

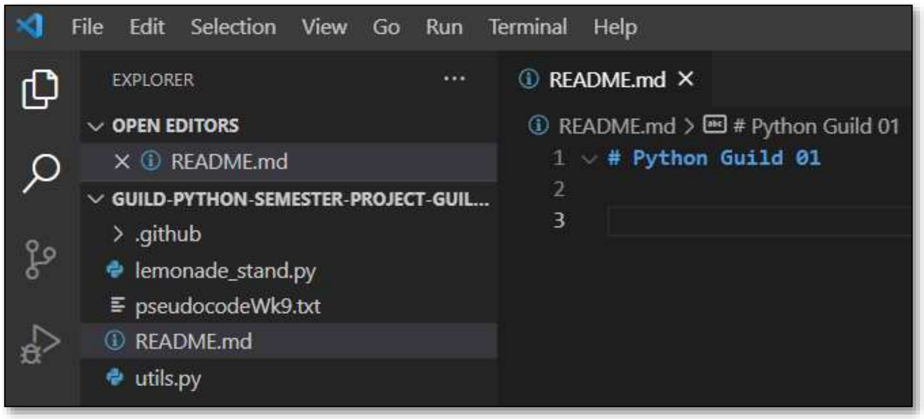
Open the repository page on GitHub in your browser
Repository menu or Ctrl Shift G

12. The files will show up in the left-hand side.

13. Double Click the **README.md** by double clicking it.

14. Add your name and that the change created on your local computer.

15. **Save** and **close** the file. Return to **GitHub Desktop**

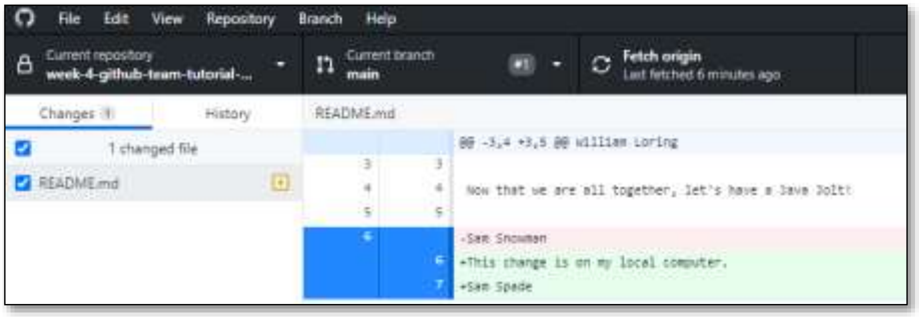


```
# Python Guild 01


```

16. **GitHub Desktop** picked up your changes and tracked them.

    a. Additions are in green.

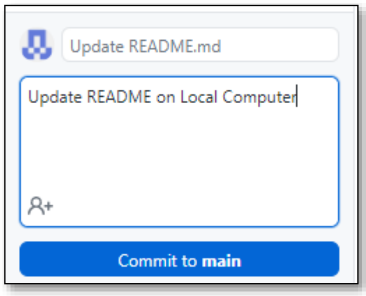
    b. Deletions are in red.



```
@@ -3,4 +3,5 @@ William Loring
 3  3
 4  4  Now that we are all together, let's have a Java Jolt!
 5  5
 6     -Sam Snowman
    6  +This change is on my local computer.
    7  +Sam Spade
```

17. In the lower left side, enter a **Summary** and extended description as shown.

18. Click **Commit to Main**.

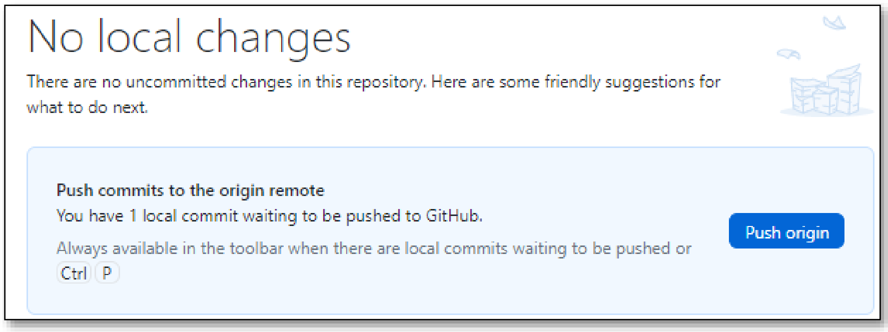


Update README.md

Update README on Local Computer

19. Before you Push origin --> Go to the **Repository** menu **→ Pull**.
    This synchronizes the origin with your local files to make sure there aren't any conflicts.

20. Click **Push origin**.


No local changes

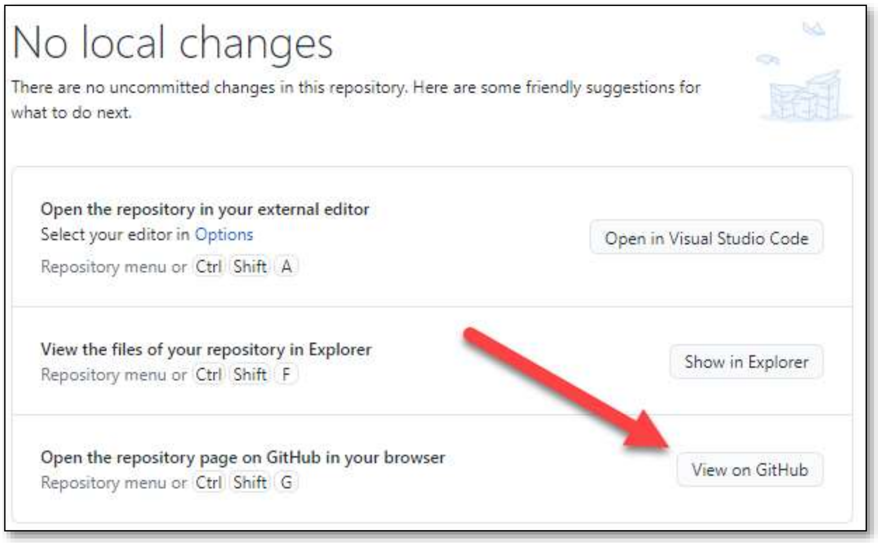
There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Push commits to the origin remote

You have 1 local commit waiting to be pushed to GitHub.

Always available in the toolbar when there are local commits waiting to be pushed or Ctrl P

21. Click **View on GitHub**.


No local changes

There are no uncommitted changes in this repository. Here are some friendly suggestions for what to do next.

Open the repository in your external editor

Select your editor in Options

Repository menu or Ctrl Shift A

View the files of your repository in Explorer

Repository menu or Ctrl Shift F

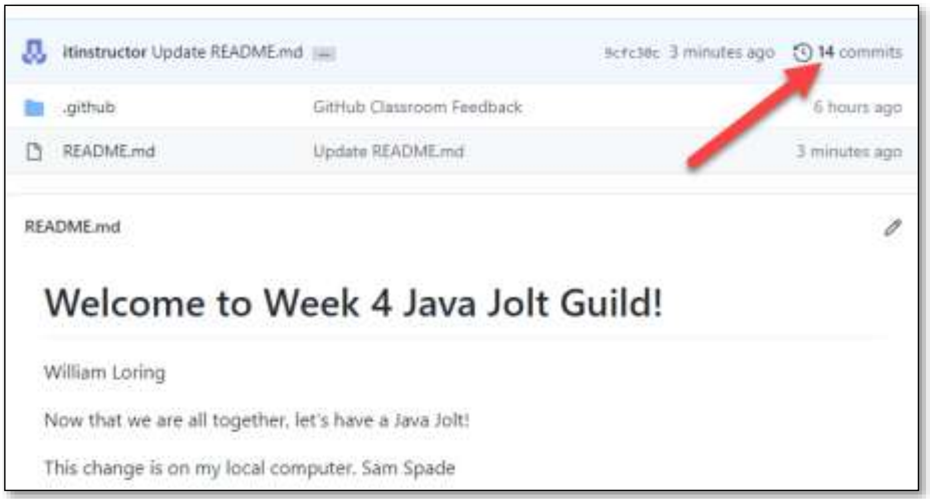
Open the repository page on GitHub in your browser

Repository menu or Ctrl Shift G

22. You should see your changes in GitHub.

23. You can also see how many Commits have been made to your repository.

24. Click **commits.**



itinstructor Update README.md ... 9cfc38c 3 minutes ago 14 commits

.github	GitHub Classroom Feedback	6 hours ago
README.md	Update README.md	3 minutes ago

README.md

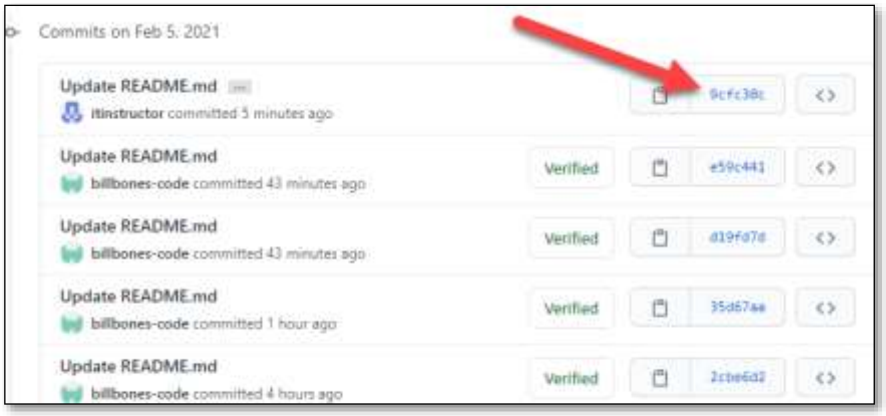
Welcome to Week 4 Java Jolt Guild!

William Loring

Now that we are all together, let's have a Java Jolt!

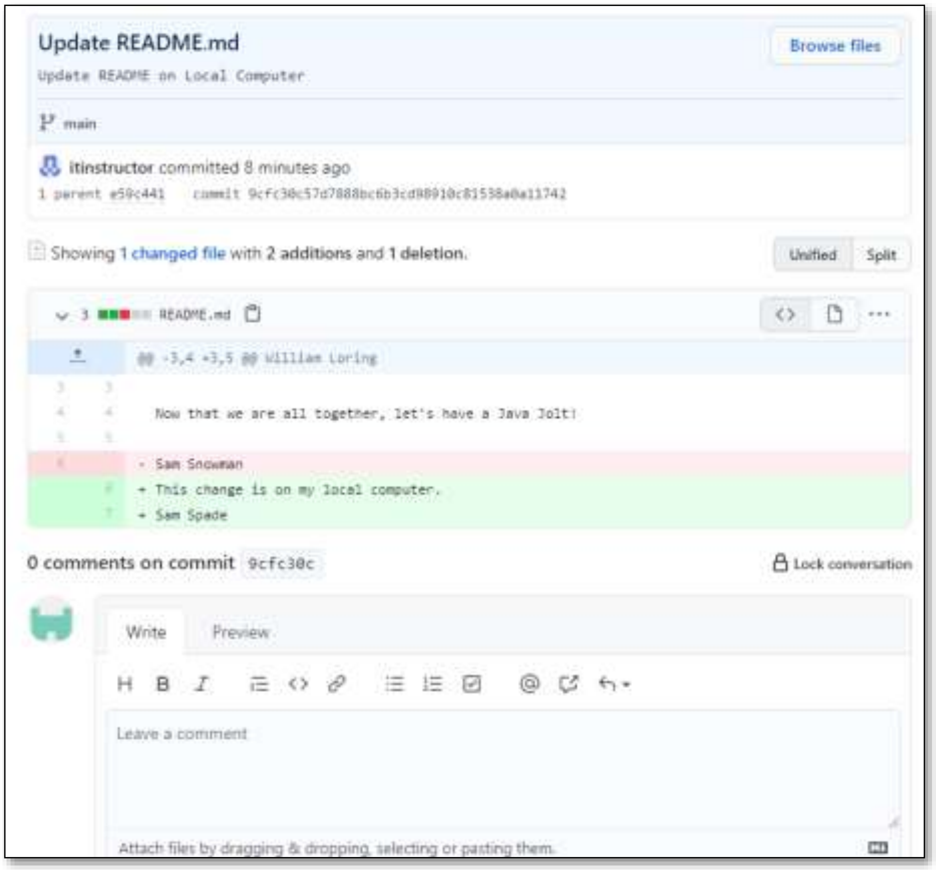
This change is on my local computer. Sam Spade

25. Click the numbers and letters in blue to see what happened with that commit.



Commits on Feb 5, 2021

Update README.md ... itinstructor committed 5 minutes ago		9cfc38c
Update README.md billbones-code committed 43 minutes ago	Verified	e59c441
Update README.md billbones-code committed 43 minutes ago	Verified	d19fd7d
Update README.md billbones-code committed 1 hour ago	Verified	35d67ae
Update README.md billbones-code committed 4 hours ago	Verified	2cbe6d2


Update README.md

Update README on Local Computer

main

itinstructor committed 8 minutes ago

1 parent e59c441 commit 9cfc30c57d7888bc6b3cd98910c81538a0a11742

Showing 1 changed file with 2 additions and 1 deletion.

```
@@ -3,4 +3,5 @@ William Loring

  Now that we are all together, let's have a Java Jolt!

- Sam Snowman
+ This change is on my local computer.
+ Sam Spade
```

0 comments on commit 9cfc30c

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

You have successfully cloned and commited to your Guild repository!

GitHub Project Workflow Steps

Video walkthrough: Guild Project GitHub Workflow

Go through this same workflow every time you work on the project.

Commit Early → Commit Often

This will help keep your GitHub repository synchronized between the team members.

Each time you work on your Guild Project, go through these GitHub workflow steps.

1. Open **GitHub Desktop**.
2. Click **Fetch Origin** or go to **Repository → Pull.**
3. Click **Open in Visual Studio Code.**
4. Work on your code.
5. Go back to **GitHub Desktop**.

6. Go to **Repository → Pull.**
7. Click **Commit to main.**
8. Click **Push origin.**

The Final Frontier: Shared Coding with Kat's Lemonade Stand

Shared Coding Process

1. Pseudocode.
2. Your Guild meets in real time.
3. Pseudocode.
4. Look at the program requirements.
5. Pseudocode.
6. All coding is done locally and committed to GitHub.

There are two major methods of coding for this project.

Synchronous

1. Everyone works on the pseudocode in real time.
2. One person is the driver, the rest are navigators.
3. The driver does the writing of the code.
4. The navigators provide input, look up resources.
5. Divide up the project into parts.
6. Create the basic files and shell of the program

Asynchronous

This is an example of dividing up the coding project.

1. One person declares the variables and input.
2. One person does the calculations.
3. One person does the display.
4. Complete the Project

Finalize Project

The Guild might meet in real time to finalize the project.

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Minimum Program Requirements

You've decided to open a lemonade stand. Grandma will provide the starting capital for the venture as well as a winning recipe for the lemonade.

The recipe requires lemons, sugar, water, and cups.

Let's build a program to purchase these ingredients. We won’t do everything all at once, we will build it one milestone at a time.

- Grandma will give you $50 of venture capital.
- Grandma will provide you with the recipe, water, table, chair, umbrella, and a pitcher for the lemonade.
- You must purchase lemons, sugar, and cups.
- You must make a sign and give your lemonade stand a name.

The program should:

- Use the title function any time you need to display a title.
- Allow you to purchase the needed supplies.

You program doesn’t have to look like the example runs. The results should be the same.

Assignment Requirements

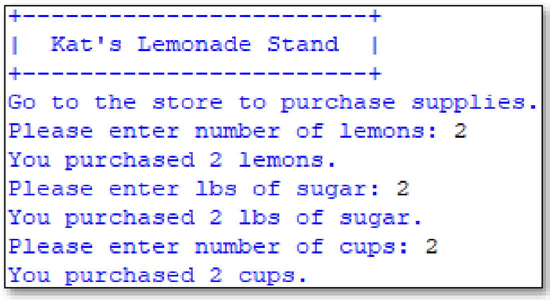
- Create a folder for each week’s work.
- Each week’s folder will have a working version of the program.

There are three Python programs in your GitHub repository.

- **menu.py** is a start for a program that will display menu’s

- **lemonade_stand.py** is a scaffolded program. That means that the outline of the program is already created.
- **utils.py** is imported into **lemonade_stand** and contains three functions: **title()**, **get_int()**, and **get_float()**. If you are not sure what these functions do and how to use them, please read the comments in the module.
- Create three functions.
  Use the **get_int()** function from the utils module where indicated.
  - **get_lemon()**
    - get_int()
    - print results
  - **get_sugar()**
    - get_int()
    - print results
  - **get_cup()**
    - get_int()
    - print results
- The main function will primarily call other functions to do the work.

Example run:



```
+------------------------+
|  Kat's Lemonade Stand  |
+------------------------+
Go to the store to purchase supplies.
Please enter number of lemons: 2
You purchased 2 lemons.
Please enter lbs of sugar: 2
You purchased 2 lbs of sugar.
Please enter number of cups: 2
You purchased 2 cups.
```

Coding Workflow with Github Desktop

1. **In GitHub Desktop → Fetch Origin**
2. Click **Open in Visual Studio Code**
3. Make changes to the code

4. Return to **GitHub Desktop**
5. Type in a summary of your changes.
6. **Fetch origin** again.
7. **Commit to Master**
8. **Push origin**

Assignment Submission

Part of your individual grade will be when, how often, and what you commit to the shared repository.

We are using the Agile development model. We want each week's product to be functional.

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version if the next version has trouble.
2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.
4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Week 12 Milestone: Kat's Lemonade Stand

100 points

Time Required: 90 minutes

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Shared Coding Process

1. Pseudocode first

2. Split up coding tasks
3. KanBan (Read GitHub KanBan Board attached to Week 9 assignment)
4. Code and communicate
5. Commit often
6. Test and submit

Coding Workflow with Github Desktop

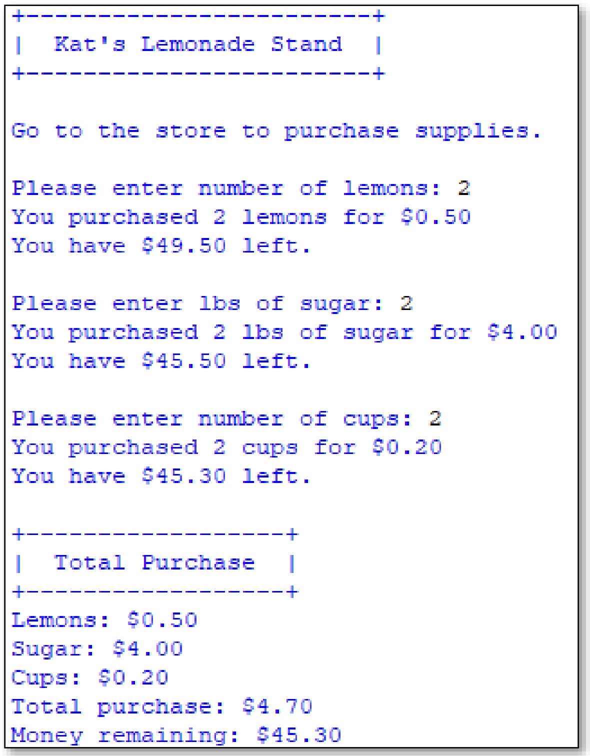
1. **In GitHub Desktop → Fetch Origin**
2. Click **Open in Visual Studio Code**
3. Make changes to the code
4. Return to **GitHub Desktop**
5. Type in a summary of your changes.
6. **Fetch origin** again.
7. **Commit to Master**
8. **Push origin**

Minimum Program Requirements

Kat has become very skilled at buying the ingredients, making, and selling lemonade. She does have trouble keeping track of how much money she has left. She has asked if you could add that to her program.

- Grandma’s initial investment is turned into working capital.
- Keep a running total to track how much working capital there is left after each purchase.
- Add a display function to display the total purchase.
    - Use the **utils.title()** function to title the display
    - Display each individual item cost, then the total purchase

Example run:



```
+------------------------+
|  Kat's Lemonade Stand  |
+------------------------+

Go to the store to purchase supplies.

Please enter number of lemons: 2
You purchased 2 lemons for $0.50
You have $49.50 left.

Please enter lbs of sugar: 2
You purchased 2 lbs of sugar for $4.00
You have $45.50 left.

Please enter number of cups: 2
You purchased 2 cups for $0.20
You have $45.30 left.

+------------------+
|  Total Purchase  |
+------------------+
Lemons: $0.50
Sugar: $4.00
Cups: $0.20
Total purchase: $4.70
Money remaining: $45.30
```

Assignment Submission

Part of your individual grade will be when, how often, and what you commit to the shared repository.

We are using the Agile development model. We want each week's product to be functional.

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version if the next version has trouble.
2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.
4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Week 13 Milestone: Kat's Lemonade Stand

100 points

Time Required: 90 minutes

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Shared Coding Process

1. Pseudocode first
2. Split up coding tasks
3. KanBan
4. Code and communicate
5. Commit often
6. Test and submit

Coding Workflow with Github Desktop

1. **In GitHub Desktop → Fetch Origin**
2. Click **Open in Visual Studio Code**
3. Make changes to the code
4. Return to **GitHub Desktop**
5. Type in a summary of your changes.
6. **Fetch origin** again.
7. **Commit to Master**
8. **Push origin**

Minimum Program Requirements

The profits are starting to roll in. Kat is making more trips to the store. Grandma insists that Kat use special triple filtered Springtime Mountain water for her lemonade. Kat can purchase this special water for $1 per gallon.

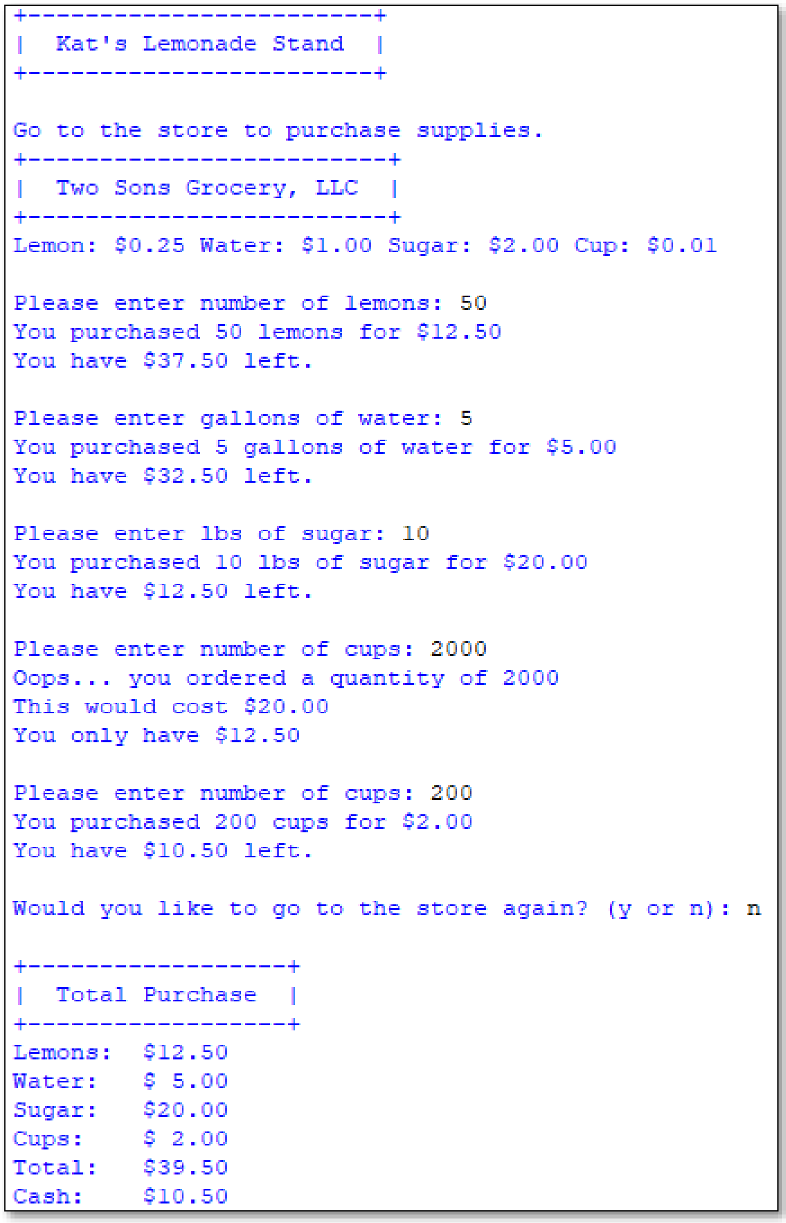
Ingredient and costs from Two Sons Grocery LLC.

- Lemon: 0.25
- Water: 1.00
- Sugar: 2.00
- Cup: .01

Instead of using separate global constants, we will be putting all ingredient costs into a single dictionary.

- Create a costs dictionary to hold the names and prices of the ingredients.
  - The name will be the key, the price will be the value.
    - Key: 'Lemon' Value: .25
  - Use the costs dictionary to access the price for each ingredient used in all calculations.
    - costs.get('Lemon') will access the corresponding Lemon value: .25
- Create a display method that uses a loop to display the prices from the costs dictionary before you go shopping.
- Allow the user to choose whether keep purchasing ingredients or exit the program.
- Prevent buying more ingredients than you have money for.
  - Hint: Use a while True loop in each get function.
  - Create a **check_cost()** function that tests whether there is enough cash left and returns true or false.

Example run:



```
+-------------------------+
|  Kat's Lemonade Stand  |
+-------------------------+

Go to the store to purchase supplies.
+--------------------------+
|  Two Sons Grocery, LLC  |
+--------------------------+
Lemon: $0.25 Water: $1.00 Sugar: $2.00 Cup: $0.01

Please enter number of lemons: 50
You purchased 50 lemons for $12.50
You have $37.50 left.

Please enter gallons of water: 5
You purchased 5 gallons of water for $5.00
You have $32.50 left.

Please enter lbs of sugar: 10
You purchased 10 lbs of sugar for $20.00
You have $12.50 left.

Please enter number of cups: 2000
Oops... you ordered a quantity of 2000
This would cost $20.00
You only have $12.50

Please enter number of cups: 200
You purchased 200 cups for $2.00
You have $10.50 left.

Would you like to go to the store again? (y or n): n

+-------------------+
|  Total Purchase  |
+-------------------+
Lemons:  $12.50
Water:   $ 5.00
Sugar:   $20.00
Cups:    $ 2.00
Total:   $39.50
Cash:    $10.50
```

Assignment Submission

Part of your individual grade will be when, how often, and what you commit to the shared repository.

We are using the Agile development model. We want each week's product to be functional.

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version if the next version has trouble.

2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.
4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Week 14 Milestone: Kat's Lemonade Stand OOP

100 points

Time Required: 90 minutes

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Shared Coding Process

1. Pseudocode first
2. Split up coding tasks
3. KanBan
4. Code and communicate
5. Commit often
6. Test and submit

Coding Workflow with Github Desktop

1. **In GitHub Desktop → Fetch Origin**
2. Click **Open in Visual Studio Code**
3. Make changes to the code
4. Return to **GitHub Desktop**
5. Type in a summary of your changes.

6. **Fetch origin** again.
7. **Commit to Master**
8. **Push origin**

Program Requirements

Kat is rolling in the lemons. She is planning on franchising and taking her lemonade stand worldwide.

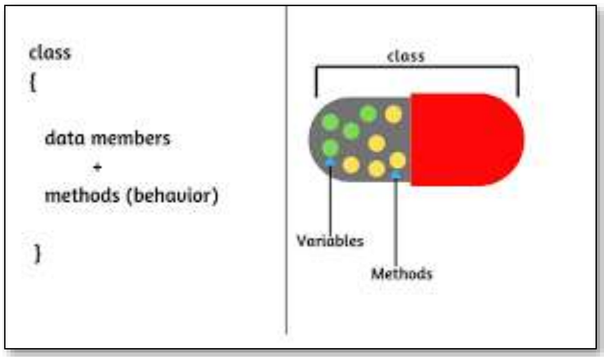
To do that, she would like to add some class to her program. Time for OOP (Object Oriented Programming). OOP will allow us to expand our program as much as we wish.

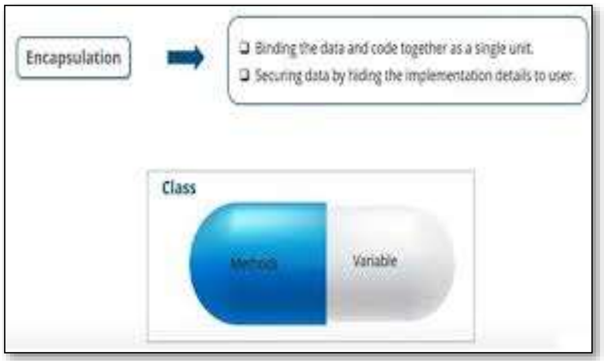
Uncle Vernon made a lemonade stand out of plywood. He also provided a pitcher, an old umbrella, a folding chair, and a straw hat with a flower on it.

Convert Functional to OOP

We are going to convert our functional program to an object-oriented program. This will be much like some tutorials and exercises we have done.

Data hiding and encapsulation. Everything to do with the entity/class (Lemonade Stand) is inside the entity/class.




Encapsulation
Binding the data and code together as a single unit.
Securing data by hiding the implementation details to user.
Class
Methods
Variable

1. Move all functions and variables inside the **LemonadeStand** class.
2. Global variables are not needed. All variables can be inside the **LemonadeStand** as object variables.
3. All object variables (Which includes lists and dictionaries) are referred to as **self._cash** or **self._costs**
   self refers to the private class variables as they belong to the class. This is called encapsulation.

4. The _ (single underscore) hides the class variables from other programs. This is called data hiding.
5. There can be local variables inside the class methods. They would not have the _, they would be named as we have in the past.
6. Use the inventory dictionary to accumulate the quantity of items purchased. A global variable is not needed. An object variable takes the place of a global variable. While the object is in memory, the object variables survive.
7. There is not a need for return statements. We can use object variables. The only method that will return a value is the check costs function which will return true or false.

Create an inventory dictionary.

- Lemon: 0
- Water: 0
- Sugar: 0
- Cup: 0

The inventory dictionary will hold the names and quantity of the ingredients.

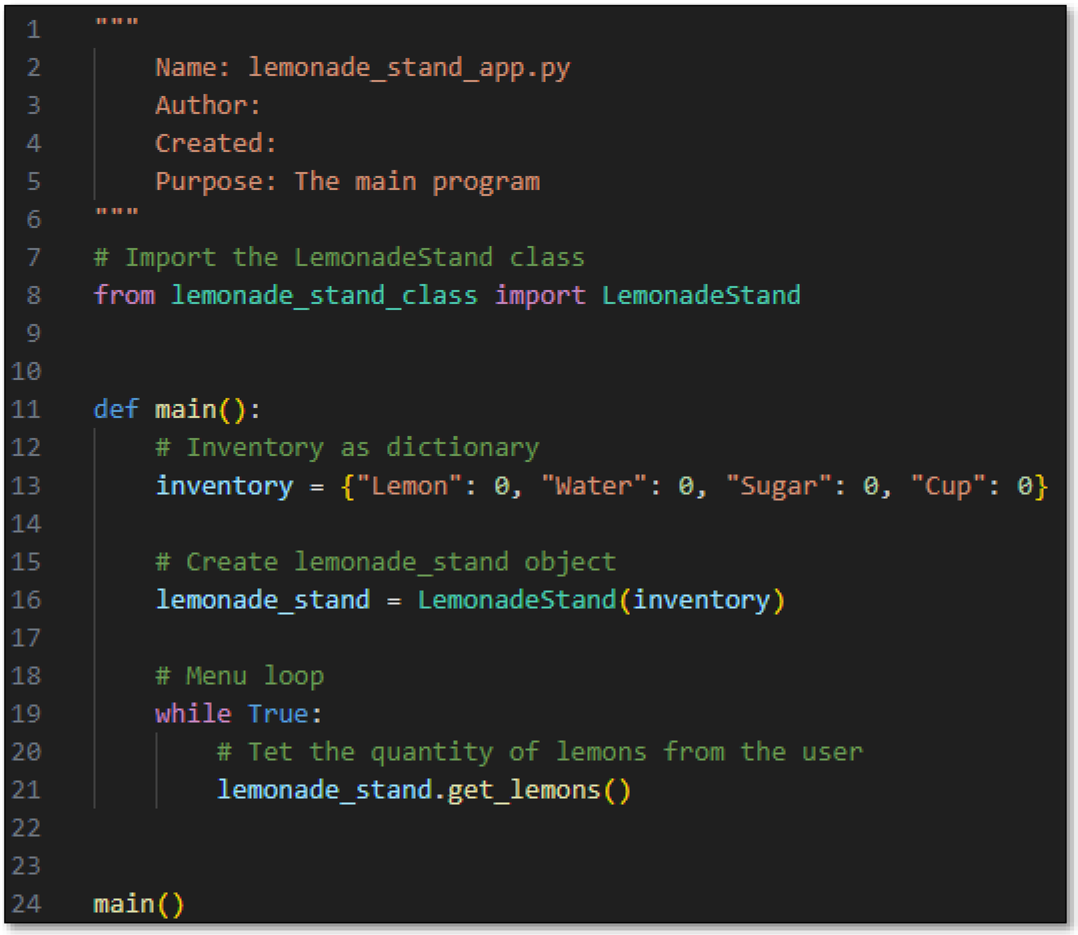
- The name will be the key, the qty will be the value.
    - Key: 'Lemon' Value: 0
- Use the inventory dictionary to keep track of total purchases.
- The print inventory method would print the inventory using a loop as shown in Chapter 6.
- When you are ready to print the final costs, multiply the item qty in the inventory dictionary by the price in the cost dictionary.

Lemonade Stand Inventory OOP Program Example

OOP (Object Oriented Programming) cuts down on parameter passing. It makes for tidier code as our program gets more complex. You may want to divide your code into separate class files.

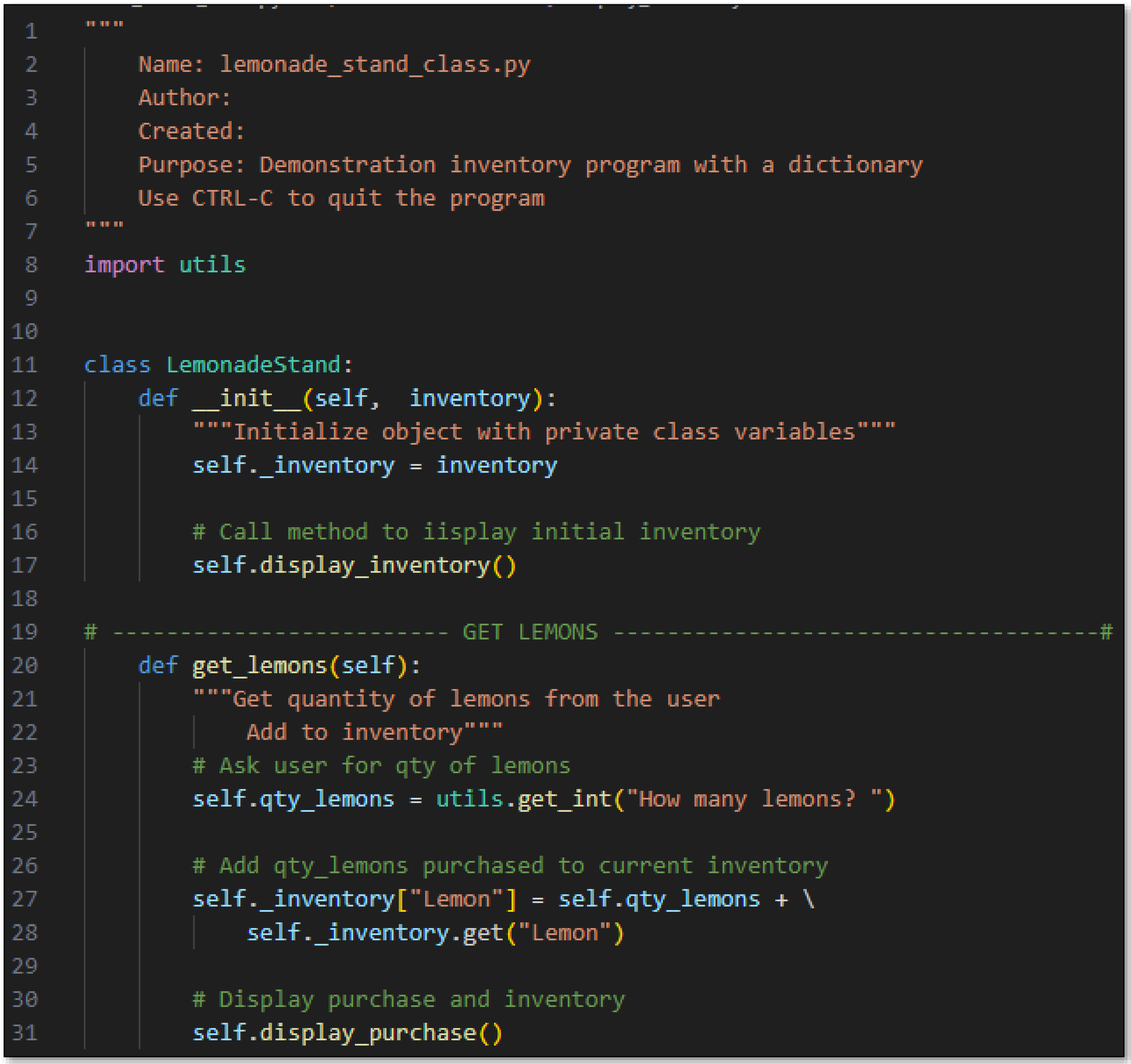
This is an example of how to use inventory, this is not the complete solution for the program. It is a starting point.

The main program file starts the program. The class does all the work.

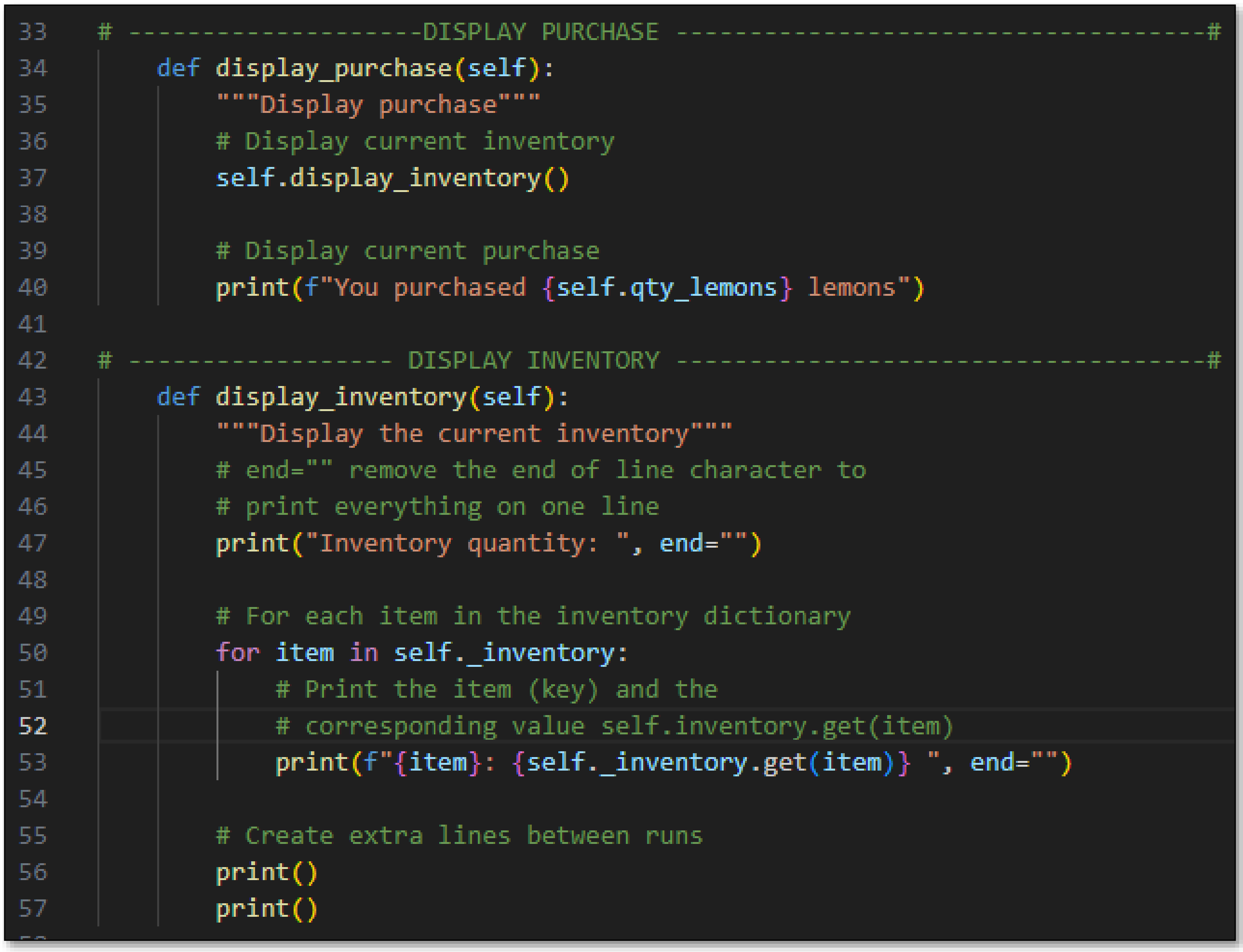


```python
"""
    Name: lemonade_stand_app.py
    Author:
    Created:
    Purpose: The main program
"""
# Import the LemonadeStand class
from lemonade_stand_class import LemonadeStand


def main():
    # Inventory as dictionary
    inventory = {"Lemon": 0, "Water": 0, "Sugar": 0, "Cup": 0}

    # Create lemonade_stand object
    lemonade_stand = LemonadeStand(inventory)

    # Menu loop
    while True:
        # Tet the quantity of lemons from the user
        lemonade_stand.get_lemons()


main()
```

The class does all the work. All variables have **self.** in front of them. No more passing parameters around inside a class.

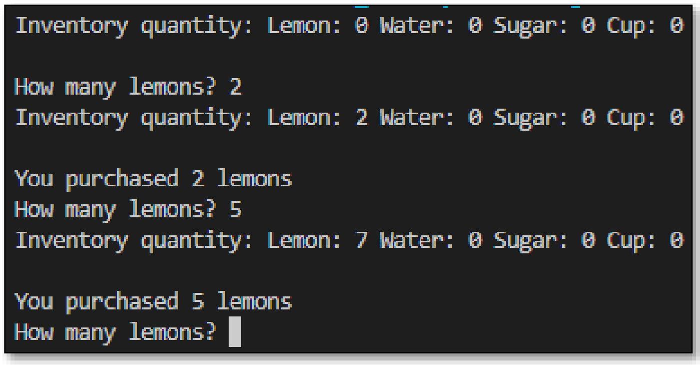


```python
"""
    Name: lemonade_stand_class.py
    Author:
    Created:
    Purpose: Demonstration inventory program with a dictionary
    Use CTRL-C to quit the program
"""
import utils


class LemonadeStand:
    def __init__(self,  inventory):
        """Initialize object with private class variables"""
        self._inventory = inventory

        # Call method to iisplay initial inventory
        self.display_inventory()

# ------------------------ GET LEMONS ----------------------------------#
    def get_lemons(self):
        """Get quantity of lemons from the user
            Add to inventory"""
        # Ask user for qty of lemons
        self.qty_lemons = utils.get_int("How many lemons? ")

        # Add qty_lemons purchased to current inventory
        self._inventory["Lemon"] = self.qty_lemons + \
            self._inventory.get("Lemon")

        # Display purchase and inventory
        self.display_purchase()
```



```python
# --------------------DISPLAY PURCHASE ------------------------------------#
    def display_purchase(self):
        """Display purchase"""
        # Display current inventory
        self.display_inventory()

        # Display current purchase
        print(f"You purchased {self.qty_lemons} lemons")

# ------------------ DISPLAY INVENTORY -----------------------------------#
    def display_inventory(self):
        """Display the current inventory"""
        # end="" remove the end of line character to
        # print everything on one line
        print("Inventory quantity: ", end="")

        # For each item in the inventory dictionary
        for item in self._inventory:
            # Print the item (key) and the
            # corresponding value self.inventory.get(item)
            print(f"{item}: {self._inventory.get(item)} ", end="")

        # Create extra lines between runs
        print()
        print()
```

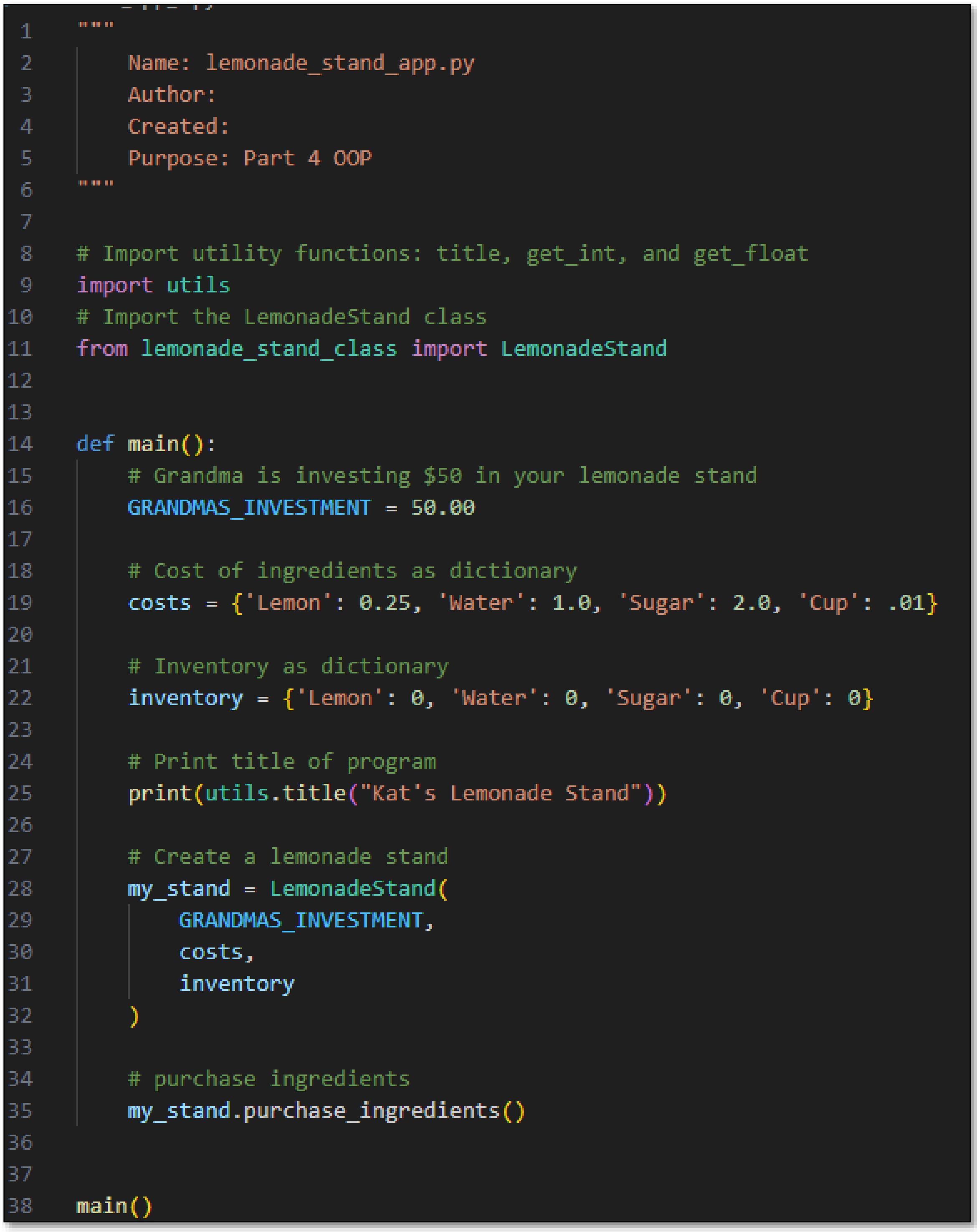
Example run:



```
Inventory quantity: Lemon: 0 Water: 0 Sugar: 0 Cup: 0

How many lemons? 2
Inventory quantity: Lemon: 2 Water: 0 Sugar: 0 Cup: 0

You purchased 2 lemons
How many lemons? 5
Inventory quantity: Lemon: 7 Water: 0 Sugar: 0 Cup: 0

You purchased 5 lemons
How many lemons?
```

Main Function

Once you get your get_lemons() and inventory to work, your main program could look something like this. All the work is done in the Lemonade class.



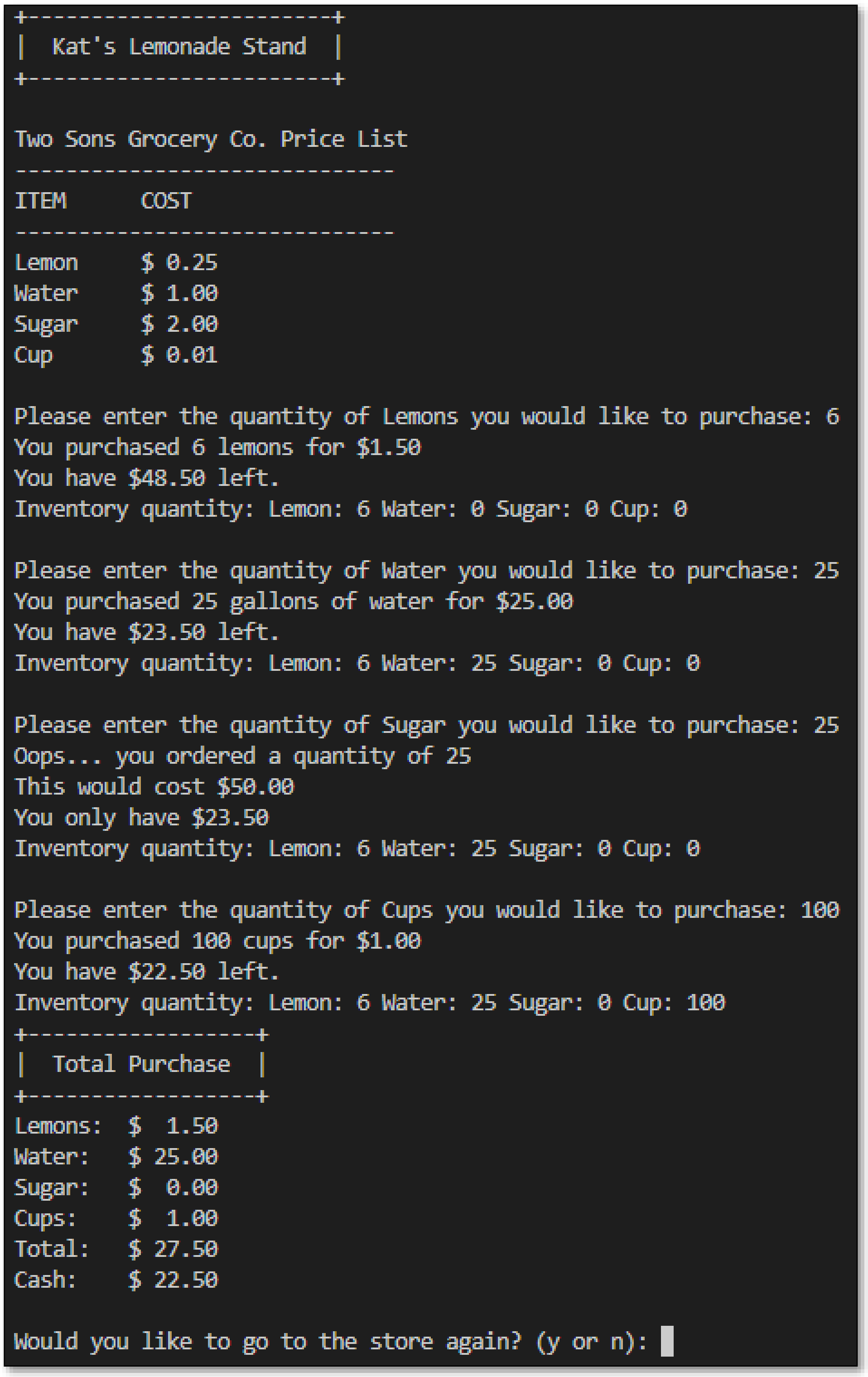
```python
"""
    Name: lemonade_stand_app.py
    Author:
    Created:
    Purpose: Part 4 OOP
"""

# Import utility functions: title, get_int, and get_float
import utils
# Import the LemonadeStand class
from lemonade_stand_class import LemonadeStand


def main():
    # Grandma is investing $50 in your lemonade stand
    GRANDMAS_INVESTMENT = 50.00

    # Cost of ingredients as dictionary
    costs = {'Lemon': 0.25, 'Water': 1.0, 'Sugar': 2.0, 'Cup': .01}

    # Inventory as dictionary
    inventory = {'Lemon': 0, 'Water': 0, 'Sugar': 0, 'Cup': 0}

    # Print title of program
    print(utils.title("Kat's Lemonade Stand"))

    # Create a lemonade stand
    my_stand = LemonadeStand(
        GRANDMAS_INVESTMENT,
        costs,
        inventory
    )

    # purchase ingredients
    my_stand.purchase_ingredients()


main()
```

Lemonade Class Pseudocode

The pseudocode shown is not complete, it is provided to give you a starting place.

```
# Import utility functions: title, get_int, and get_float
import utils
class LemonadeStand:
    def __init__(self, starting_capital, costs, inventory)
    """Initialize object with private class variables"""
    self._cash = starting_capital
    self._costs = costs  # Costs dictionary
    self._inventory = inventory  # Inventory dictionary
    self._current_purchase = 0

    def purchase_ingredients(self):
        """Purchase all ingredients"""
        self.display_costs()
        purchase_again = "y"
        while purchase_again == "y":
            self.get_lemon()
            self.get_water()
            self.get_sugar()
            self.get_cup()
            self.display_purchase()
            purchase_again = input(
                '\nWould you like to go to the store again? (y or n): ')

    # Get the 4 ingredients using the get functions

    def get_lemon(self):
        """Get and display the number of lemons from user"""
        # Get number of lemons from user
        lemon_qty = utils.get_int("Lemons")
        # Calculate lemon cost
        lemon_cost = (lemon_qty * self.costs["Lemon"])

        # If lemon cost is less than cash on hand, make the purchase

        # If lemon cost is more than cash on hand, do not make the purchase

        # Update the current purchase variable

        # Display the current inventory
        def get_water(self):
            pass

        def get_sugar(self):
            pass

        def get_cup(self):
            pass

        def display_purchase(self):
            pass
```

Current Milestone Current Run



```
+------------------------+
|  Kat's Lemonade Stand  |
+------------------------+

Two Sons Grocery Co. Price List
------------------------------
ITEM      COST
------------------------------
Lemon     $ 0.25
Water     $ 1.00
Sugar     $ 2.00
Cup       $ 0.01

Please enter the quantity of Lemons you would like to purchase: 6
You purchased 6 lemons for $1.50
You have $48.50 left.
Inventory quantity: Lemon: 6 Water: 0 Sugar: 0 Cup: 0

Please enter the quantity of Water you would like to purchase: 25
You purchased 25 gallons of water for $25.00
You have $23.50 left.
Inventory quantity: Lemon: 6 Water: 25 Sugar: 0 Cup: 0

Please enter the quantity of Sugar you would like to purchase: 25
Oops... you ordered a quantity of 25
This would cost $50.00
You only have $23.50
Inventory quantity: Lemon: 6 Water: 25 Sugar: 0 Cup: 0

Please enter the quantity of Cups you would like to purchase: 100
You purchased 100 cups for $1.00
You have $22.50 left.
Inventory quantity: Lemon: 6 Water: 25 Sugar: 0 Cup: 100
+------------------+
|  Total Purchase  |
+------------------+
Lemons:  $  1.50
Water:   $ 25.00
Sugar:   $  0.00
Cups:    $  1.00
Total:   $ 27.50
Cash:    $ 22.50

Would you like to go to the store again? (y or n):
```

Assignment Submission

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version if the next version has trouble.
2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.
4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Week 15 Milestone: Kat's Lemonade Stand Make Lemonade

100 points

Time Required: 90 minutes

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.
- Please read all the directions before beginning the assignment.

Shared Coding Process

1. Pseudocode first
2. Split up coding tasks
3. KanBan
4. Code and communicate
5. Commit often
6. Test and submit

Coding Workflow with Github Desktop

1. **In GitHub Desktop → Fetch Origin**
2. Click **Open in Visual Studio Code**
3. Make changes to the code
4. Return to **GitHub Desktop**
5. Type in a summary of your changes.
6. **Fetch origin** again.
7. **Commit to Master**
8. **Push origin**

Minimum Program Requirements

- Implement and test the program one part at a time.
- Don’t commit a program doesn’t compile.

Recipe

To make lemonade, we need a recipe. Grandma says that 1 cup of lemon juice, 4 cups of water, and .5 cup of sugar make 5 servings of Lemonade.

You will need 4 lemons for 1 cup of juice, there are 16 cups in a gallon and 2 cups in a pound.

You will need a dictionary for the recipe.

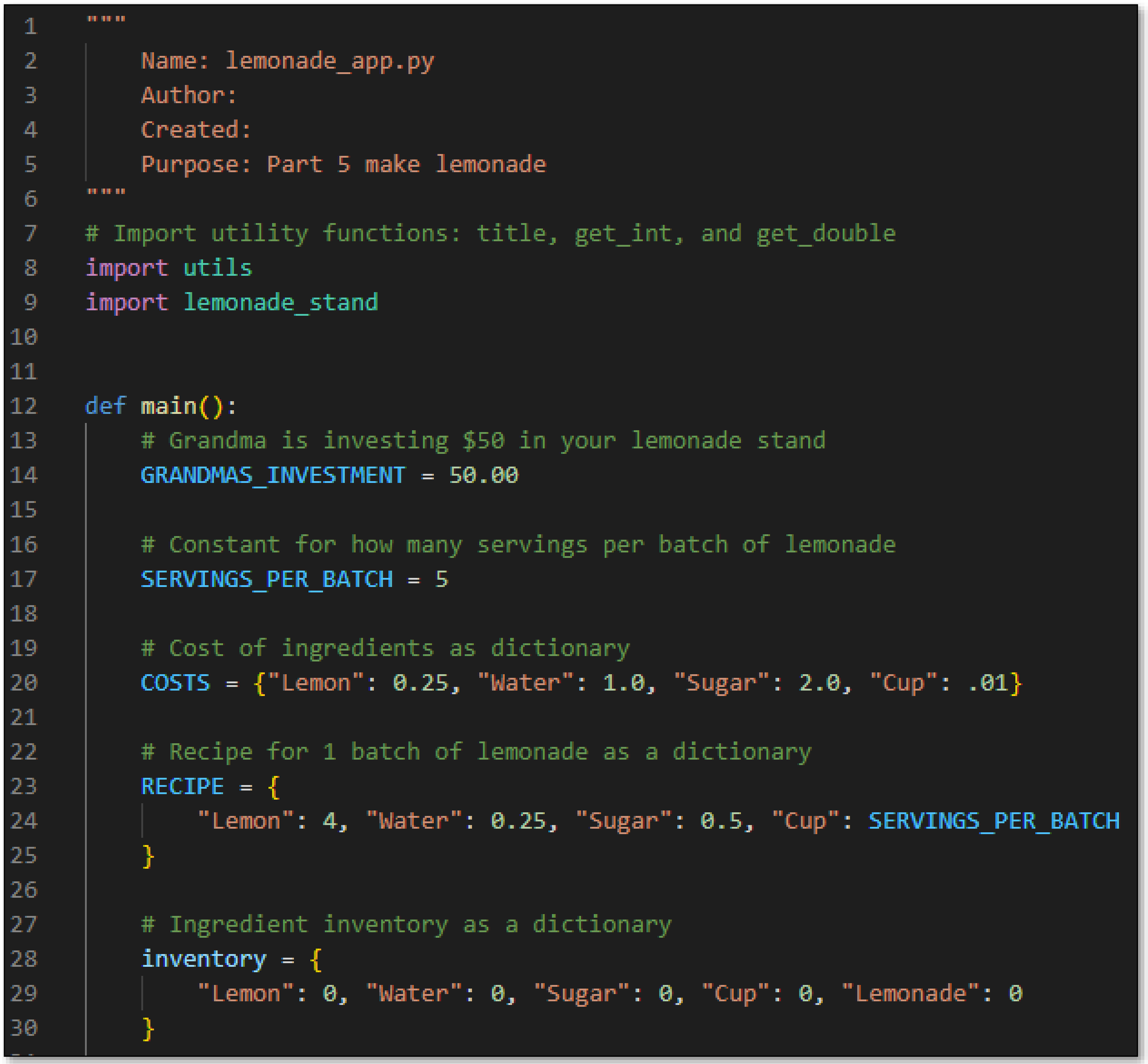
```
# Constant for how many servings per batch of lemonade
SERV_PER_BATCH = 5
```

- lemon: 4
- water: 0.25
- sugar: 0.5
- cup: SERV_PER_BATCH

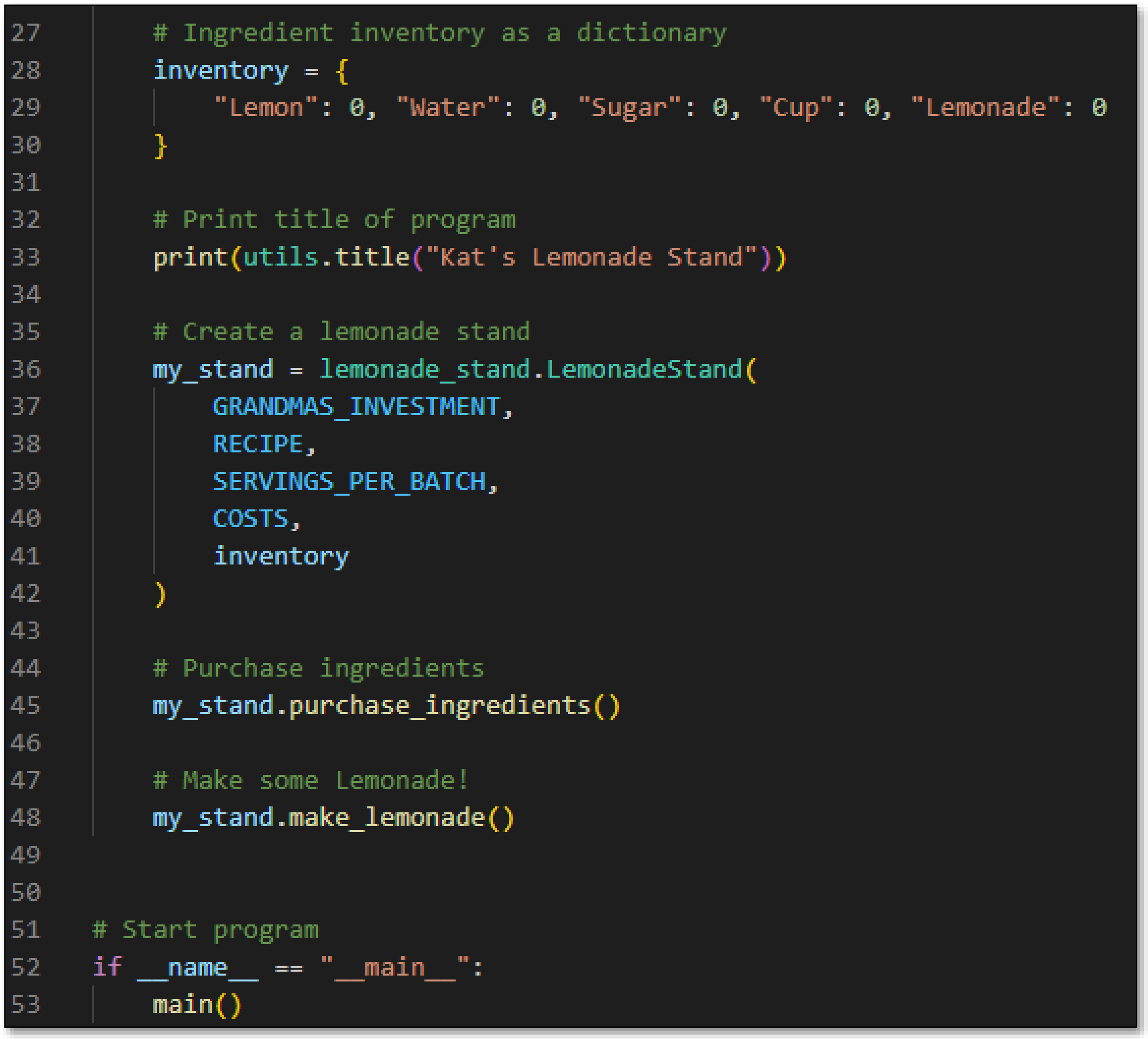
Inventory

Add Lemonade to our dictionary to track how many servings of Lemonade we can make.

Your main method will look something like this.



```python
"""
    Name: lemonade_app.py
    Author:
    Created:
    Purpose: Part 5 make lemonade
"""
# Import utility functions: title, get_int, and get_double
import utils
import lemonade_stand


def main():
    # Grandma is investing $50 in your lemonade stand
    GRANDMAS_INVESTMENT = 50.00

    # Constant for how many servings per batch of lemonade
    SERVINGS_PER_BATCH = 5

    # Cost of ingredients as dictionary
    COSTS = {"Lemon": 0.25, "Water": 1.0, "Sugar": 2.0, "Cup": .01}

    # Recipe for 1 batch of lemonade as a dictionary
    RECIPE = {
        "Lemon": 4, "Water": 0.25, "Sugar": 0.5, "Cup": SERVINGS_PER_BATCH
    }

    # Ingredient inventory as a dictionary
    inventory = {
        "Lemon": 0, "Water": 0, "Sugar": 0, "Cup": 0, "Lemonade": 0
    }
```



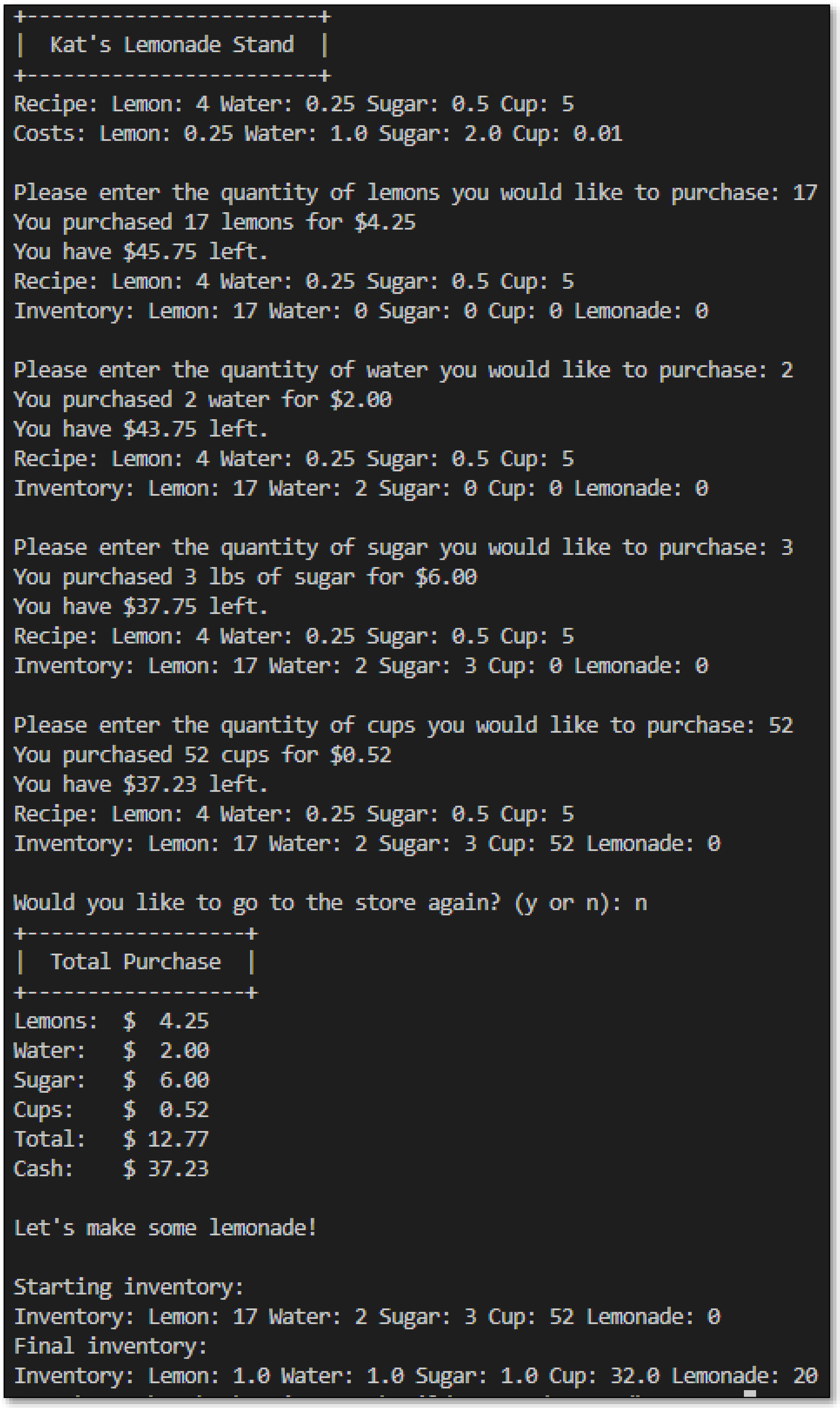
```python
    # Ingredient inventory as a dictionary
    inventory = {
        "Lemon": 0, "Water": 0, "Sugar": 0, "Cup": 0, "Lemonade": 0
    }

    # Print title of program
    print(utils.title("Kat's Lemonade Stand"))

    # Create a lemonade stand
    my_stand = lemonade_stand.LemonadeStand(
        GRANDMAS_INVESTMENT,
        RECIPE,
        SERVINGS_PER_BATCH,
        COSTS,
        inventory
    )

    # Purchase ingredients
    my_stand.purchase_ingredients()

    # Make some Lemonade!
    my_stand.make_lemonade()


# Start program
if __name__ == "__main__":
    main()
```

Pseudocode

The pseudocode shown is not complete, it is provided to give you a starting place.

```
class LemonadeStand:
    def __init__(
        self,
        GRANDMAS_INVESTMENT,
        RECIPE,
        SERVINGS_PER_BATCH,
        COSTS,
        inventory
    ):
    """Initialize object with private class variables"""
        self._cash = GRANDMAS_INVESTMENT
        self._RECIPE = RECIPE
        self._SERVINGS_PER_BATCH = SERV_PER_BATCH
        self._COSTS = COSTS
        self._inventory = inventory
        # Track the amount of the current purchase from the store
        self._current_purchase = 0

    # New functions
def calculate_servings(self):
        # Store the number of servings per ingredient in a list
        serving_count = []
        Use a for loop to go through the recipe by ingredient
            # Calculate number of servings per ingredient
            # // is the modulus operator which returns a whole number
            # We don't want fractions of ingredients
            self.servings = self._inventory[ingredient] // self._recipe[ingredient]

            # Store the number of serving per ingredient in the serving_count list
            serving_count.append(servings)

        # Multiply serving_count by servings per batch
        # min returns the lowest value in the list
        self.servings = min(serving_count) * self.SERV_PER_BATCH

    def make_lemonade(self)
        """Make lemonade with current ingredients"""
        # How many servings of lemonade we can make
        self.calculate_servings()

        # calculate how many batches of lemonade we can make

        # Display starting inventory
        # loop through recipe by ingredient
            inventory -= recipe * batches
        # Add lemonade servings to inventory
        # Display starting inventory
```

Example Run



```
+------------------------+
|  Kat's Lemonade Stand  |
+------------------------+
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Costs: Lemon: 0.25 Water: 1.0 Sugar: 2.0 Cup: 0.01

Please enter the quantity of lemons you would like to purchase: 17
You purchased 17 lemons for $4.25
You have $45.75 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 17 Water: 0 Sugar: 0 Cup: 0 Lemonade: 0

Please enter the quantity of water you would like to purchase: 2
You purchased 2 water for $2.00
You have $43.75 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 17 Water: 2 Sugar: 0 Cup: 0 Lemonade: 0

Please enter the quantity of sugar you would like to purchase: 3
You purchased 3 lbs of sugar for $6.00
You have $37.75 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 17 Water: 2 Sugar: 3 Cup: 0 Lemonade: 0

Please enter the quantity of cups you would like to purchase: 52
You purchased 52 cups for $0.52
You have $37.23 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 17 Water: 2 Sugar: 3 Cup: 52 Lemonade: 0

Would you like to go to the store again? (y or n): n
+------------------+
|  Total Purchase  |
+------------------+
Lemons:  $  4.25
Water:   $  2.00
Sugar:   $  6.00
Cups:    $  0.52
Total:   $ 12.77
Cash:    $ 37.23

Let's make some lemonade!

Starting inventory:
Inventory: Lemon: 17 Water: 2 Sugar: 3 Cup: 52 Lemonade: 0
Final inventory:
Inventory: Lemon: 1.0 Water: 1.0 Sugar: 1.0 Cup: 32.0 Lemonade: 20
```

Assignment Submission

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version if the next version has trouble.
2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.
4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Week 16 Milestone: Kat's Lemonade Stand Sell Lemonade

100 points

Time Required: 90 minutes

Please read all the directions before beginning the assignment.

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

Minimum Program Requirements

Start planning the scope and how to deliver your final application. it can be a console or GUI based.

The program does not have to be complete at this point. Add and test features as you go. Determine your own milestones of function and completeness. When you turn in the Guild assignment, the program should be functional at that milestone (sprint).

It's time to open the lemonade stand and sell some lemonade.

Disaster Strikes!

Add some random events that affect your cash/profit. Here are some ideas.

- The wind blew over your lemonade stand. You have to rebuild.

- The fire inspector visits and gives you a fine for not having a fire extinguisher.
- A relative gives you money to help with your cash flow.

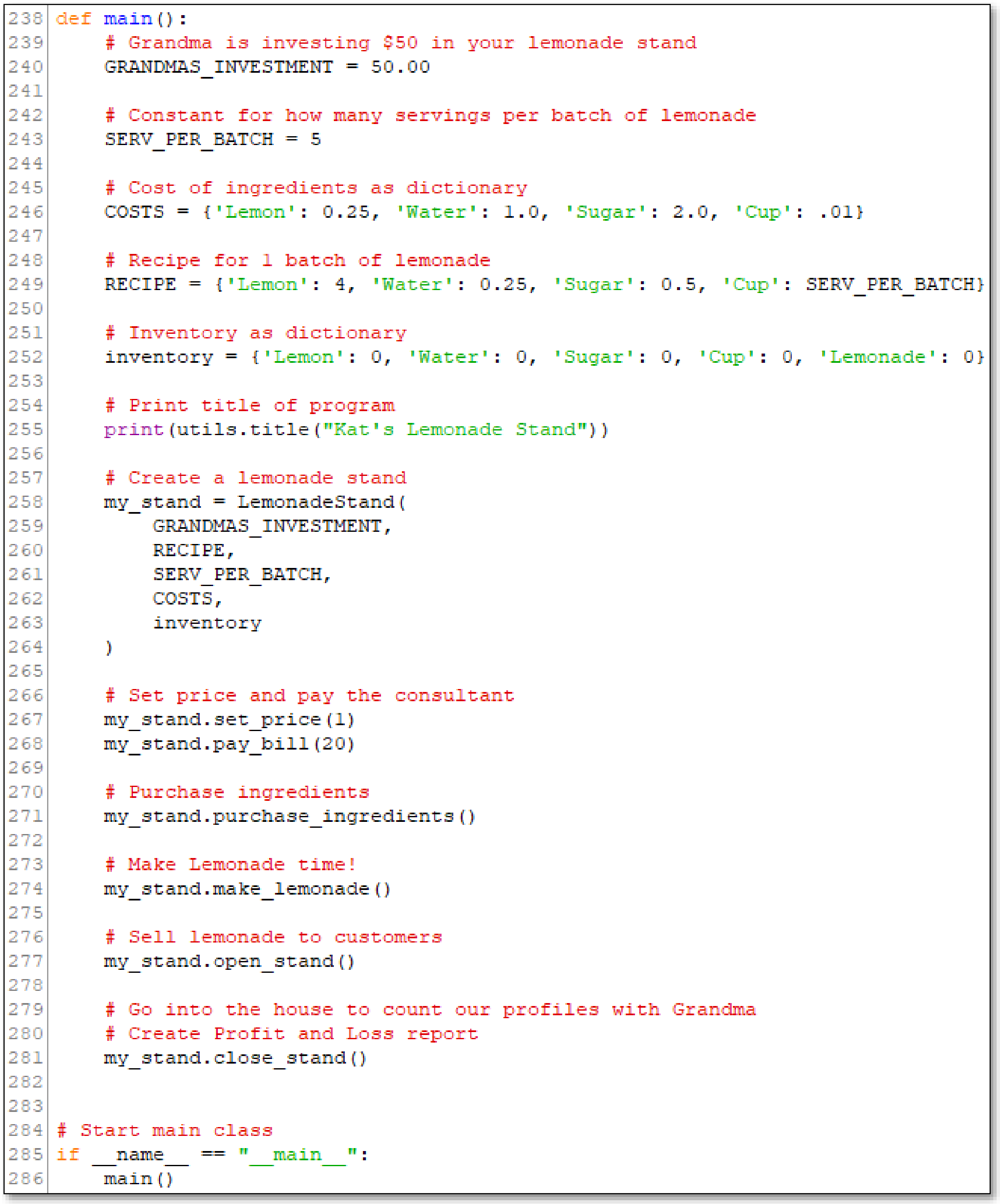
We've hired a lemonade stand consultant to commission a study of the local beverage market.

He advised us that the optimal price point for our lemonade is $1.00 and billed us $20 for his services.

We want to add some more methods.

- Open Stand
- Make Sale
    - You may have multiple customers
- Close Stand
    - End of day profit and loss report

Your main function may look something like this.



```
def main():
    # Grandma is investing $50 in your lemonade stand
    GRANDMAS_INVESTMENT = 50.00

    # Constant for how many servings per batch of lemonade
    SERV_PER_BATCH = 5

    # Cost of ingredients as dictionary
    COSTS = {'Lemon': 0.25, 'Water': 1.0, 'Sugar': 2.0, 'Cup': .01}

    # Recipe for 1 batch of lemonade
    RECIPE = {'Lemon': 4, 'Water': 0.25, 'Sugar': 0.5, 'Cup': SERV_PER_BATCH}

    # Inventory as dictionary
    inventory = {'Lemon': 0, 'Water': 0, 'Sugar': 0, 'Cup': 0, 'Lemonade': 0}

    # Print title of program
    print(utils.title("Kat's Lemonade Stand"))

    # Create a lemonade stand
    my_stand = LemonadeStand(
        GRANDMAS_INVESTMENT,
        RECIPE,
        SERV_PER_BATCH,
        COSTS,
        inventory
    )

    # Set price and pay the consultant
    my_stand.set_price(1)
    my_stand.pay_bill(20)

    # Purchase ingredients
    my_stand.purchase_ingredients()

    # Make Lemonade time!
    my_stand.make_lemonade()

    # Sell lemonade to customers
    my_stand.open_stand()

    # Go into the house to count our profiles with Grandma
    # Create Profit and Loss report
    my_stand.close_stand()


# Start main class
if __name__ == "__main__":
    main()
```

Final Twist

Did you know that this program is going to be a game?

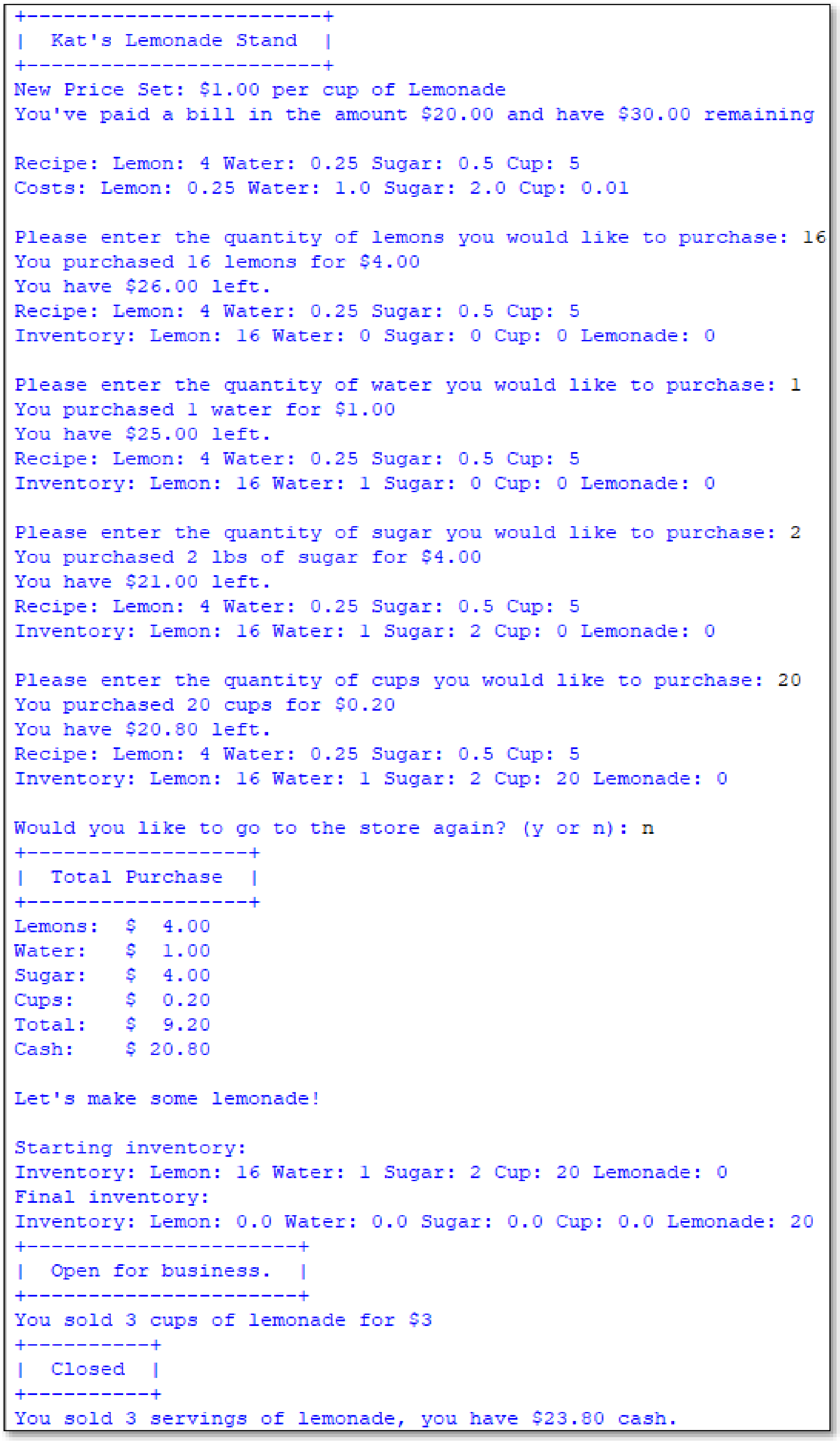
- We want a random number of customers from 1 – 10.
- If we get 0 customers, we go bankrupt.

Pseudocode

The pseudocode shown is not complete, it is provided to give you a starting place.

```
class LemonadeStand:
    def __init__(self, starting_capital, RECIPE, SERV_PER_BATCH, COSTS, 
inventory):
        ''' Initialize object with private class variables '''
        self._cash = starting_capital
        self._RECIPE = RECIPE
        self._SERV_PER_BATCH = SERV_PER_BATCH
        self._COSTS = COSTS
        self._inventory = inventory
        self._current_purchase = 0
        self._price = 0
        self._expenses = 0
        self._revenue = 0
        self._customer_count = 0

    # New functions
    def open_stand(self):

    # You may have multiple customers
    def customer_sale(self):

    # Count our money
    def close_stand(self):
```

Possible example run (This is not complete):



```
+------------------------+
|  Kat's Lemonade Stand  |
+------------------------+
New Price Set: $1.00 per cup of Lemonade
You've paid a bill in the amount $20.00 and have $30.00 remaining

Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Costs: Lemon: 0.25 Water: 1.0 Sugar: 2.0 Cup: 0.01

Please enter the quantity of lemons you would like to purchase: 16
You purchased 16 lemons for $4.00
You have $26.00 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 16 Water: 0 Sugar: 0 Cup: 0 Lemonade: 0

Please enter the quantity of water you would like to purchase: 1
You purchased 1 water for $1.00
You have $25.00 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 16 Water: 1 Sugar: 0 Cup: 0 Lemonade: 0

Please enter the quantity of sugar you would like to purchase: 2
You purchased 2 lbs of sugar for $4.00
You have $21.00 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 16 Water: 1 Sugar: 2 Cup: 0 Lemonade: 0

Please enter the quantity of cups you would like to purchase: 20
You purchased 20 cups for $0.20
You have $20.80 left.
Recipe: Lemon: 4 Water: 0.25 Sugar: 0.5 Cup: 5
Inventory: Lemon: 16 Water: 1 Sugar: 2 Cup: 20 Lemonade: 0

Would you like to go to the store again? (y or n): n
+-------------------+
|  Total Purchase   |
+-------------------+
Lemons:  $  4.00
Water:   $  1.00
Sugar:   $  4.00
Cups:    $  0.20
Total:   $  9.20
Cash:    $ 20.80

Let's make some lemonade!

Starting inventory:
Inventory: Lemon: 16 Water: 1 Sugar: 2 Cup: 20 Lemonade: 0
Final inventory:
Inventory: Lemon: 0.0 Water: 0.0 Sugar: 0.0 Cup: 0.0 Lemonade: 20
+----------------------+
|  Open for business.  |
+----------------------+
You sold 3 cups of lemonade for $3
+----------+
|  Closed  |
+----------+
You sold 3 servings of lemonade, you have $23.80 cash.
```

Assignment Submission

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version if the next version has trouble.
2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.
4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Week 16 Milestone: Kat's Lemonade Stand

100 points

Time Required: 90 minutes

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.
- Please read all the directions before beginning the assignment.

Minimum Program Requirements

Time for a bit of interactivity. Let's create a menu system for our program.

The menu can be in the main function and call object methods. When the menu item is complete, return to the main menu.

Create a **display_menu**() method.

Here is a list of possible menu items.

1. Purchase Ingredients
2. Open Lemonade Stand
3. Make Lemonade

4. Sell Lemonade
5. Close Lemonade Stand

Assignment Submission

1. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.
2. **Guild Team Submission in Blackboard:** Submit a short screencast showing the functionality of the program.
3. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.

Finals Week: Kat's Lemonade Stand

200 points

Time Required: 120 minutes

Requirements

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.
- Please read all the directions before beginning the assignment.

Shared Coding Process

1. Pseudocode first
2. Split up coding tasks
3. KanBan
4. Code and communicate
5. Commit often
6. Test and submit

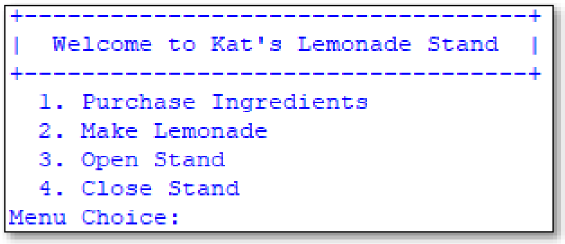
Coding Workflow with Github Desktop

1. **In GitHub Desktop → Fetch Origin**
2. Click **Open in Visual Studio Code**
3. Make changes to the code
4. Return to **GitHub Desktop**
5. Type in a summary of your changes.
6. **Fetch origin** again.
7. **Commit to Master**
8. **Push origin**

Minimum Program Requirements

Time to finish up our program.

We want a menu system of some type to allow the user to choose what they want to do. This is a simple example.



```
+----------------------------------+
|  Welcome to Kat's Lemonade Stand  |
+----------------------------------+
   1. Purchase Ingredients
   2. Make Lemonade
   3. Open Stand
   4. Close Stand
Menu Choice:
```

Add 3 new features to the Lemonade Stand. These should be object methods, functions, or integrate with an existing object method. We want modular code.

Here are some ideas to get you started thinking. You can also come up with your own ideas.

- Grandma just had surgery. She is recovering well. She will need a new wheelchair at the end of summer. She will need her $50 back. You get a loan from the bank and must pay it off. The payment occurs each time you open the Lemonade Stand.
- Incorporate a Yelp rating to help determine how many customers per hour stop by.

Assignment Submission

1. **Milestone Folder:** Create a Milestone folder for each week's successful iteration of your project. Milestone 8, Milestone 9, etc. That will give you a way to easily go back to a known working version is the next version has trouble.

2. **Guild GitHub Assignment:** The Guild pseudocode, KanBan board, and code is created in GitHub.

3. **Guild Team Submission in Blackboard:** Submit a link to a short screencast showing the functionality of the program.

4. **Guild Individual Evaluation:** Each Guild member submits an Individual Guild Evaluation in Blackboard.