Project Report:

Portfolio Milestone Module 4 – Online Shopping Cart

Alexander Ricciardi

Colorado State University Global

CSC500: Principles of Programming

Professor: Dr. Brian Holbert

October 5, 2025

Project Report:

Portfolio Milestone Module 4 – Online Shopping Cart

This documentation is part of the Portfolio Milestone Module 4 from CSC500: Principles of Programming at Colorado State University Global. This Project Report is an overview of the program's functionality and testing scenarios, including console output screenshots. The program is coded in Python 3.13, and it is called Portfolio Milestone Module 4 – Online Shopping Cart.

The Assignment Direction:

Online Shopping Cart

Step 1: Build the ItemToPurchase class with the following specifications:

- Attributes
- item_name (string)
- item_price (float)
- item_quantity (int)
- Default constructor
- Initializes item's name = "none", item's price = 0, item's quantity = 0
- Method
- print_item_cost()

Example of print_item_cost() output:

Bottled Water 10 @ $1 = $10

Step 2: In the main section of your code, prompt the user for two items and create two objects of the ItemToPurchase class.

Example:

Item 1
Enter the item name:
Chocolate Chips
Enter the item price:
3
Enter the item quantity:
1
Item 2
Enter the item name:
Bottled Water
Enter the item price:
1
Enter the item quantity:
10

Step 3: Add the costs of the two items together and output the total cost.

Example:

TOTAL COST
Chocolate Chips 1 @ $3 = $3
Bottled Water 10 @ $1 = $10
Total: $13

Your program submission materials must include your source code and screenshots of the application executing the code and the results. Please refer to the video as a recourse and reference: Python Classes and Objects (With Examples) .

My Program Description:

The program is a small terminal app. it is an implementation of an online shopping cart implementation
The program renders banners and a menu, and it calculates the total cost of items
based on each item's price and quantity. Only two items are asked to be entered in this implementation.

⚠️ My notes:
As I was using some of the same code lines for my critical assignments, I created several small utilities functions and classes that I can reuse; they are found below the comments:

```
# ____________________________________________________________________________
# General Utility Functions
#
```

And

```
# ____________________________________________________________________________
# Utility Classes
#
```

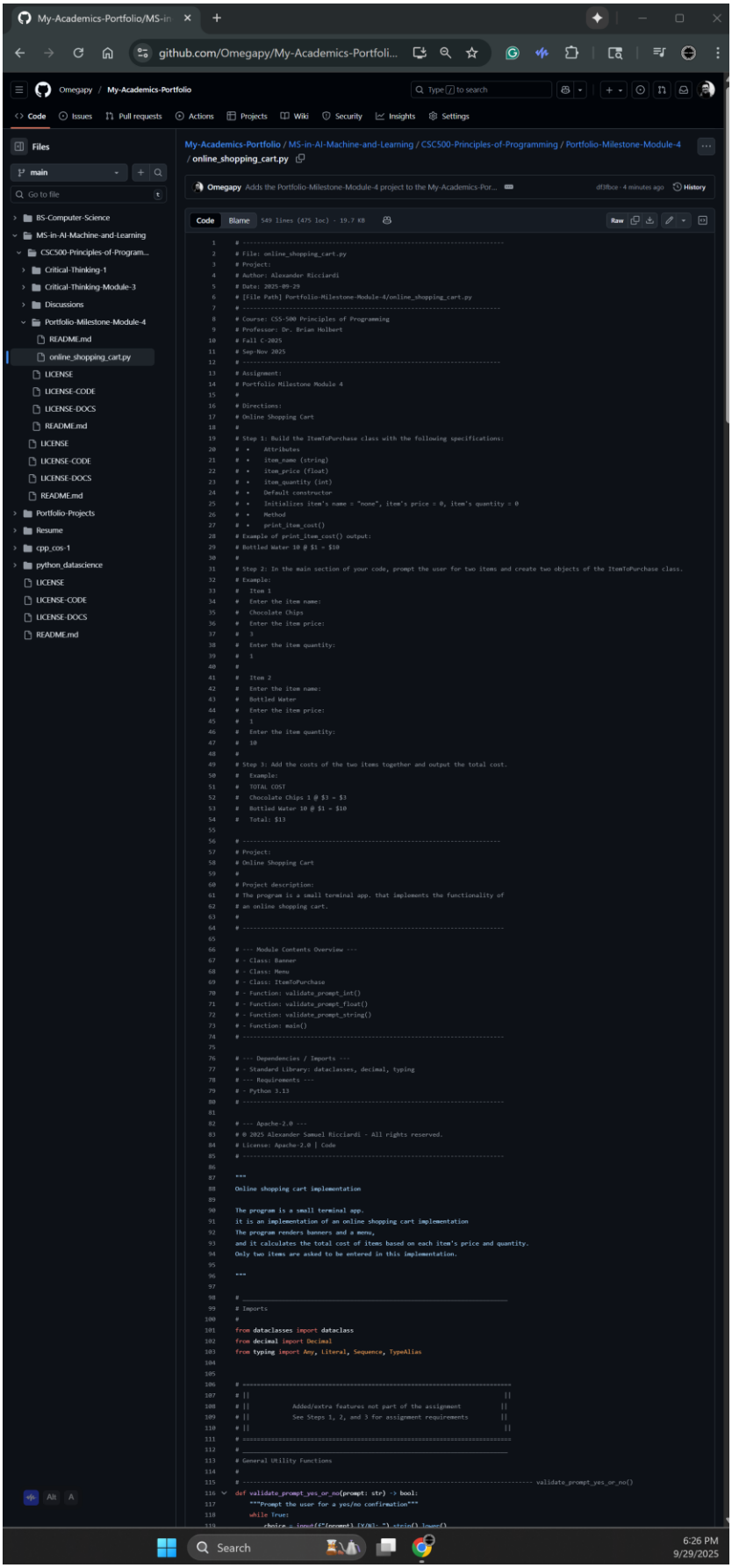
Git Repository:

I use GitHub as my Distributed Version Control System (DVCS).
The following is a link to my GitHub profile, Omega.py.

The link to this specific assignment is:
https://github.com/Omegapy/My-Academics-Portfolio/tree/main/MS-in-AI-Machine-and-Learning/CSC500-Principles-of-Programming/Critical-Thinking-Module-3

Figure-1

Code on GitHub



Code Snippet 1

Project Code in VS Code

```
# ------------------------------------------------------------------------
# File: online_shopping_cart.py
# Project:
# Author: Alexander Ricciardi
# Date: 2025-09-29
# [File Path] Portfolio-Milestone-Module-4/online_shopping_cart.py
# ------------------------------------------------------------------------
# Course: CSS-500 Principles of Programming
# Professor: Dr. Brian Holbert
# Fall C-2025
# Sep-Nov 2025
# ------------------------------------------------------------------------
# Assignment:
# Portfolio Milestone Module 4
#
# Directions:
# Online Shopping Cart
#
# Step 1: Build the ItemToPurchase class with the following specifications:
#  •    Attributes
#  •    item_name (string)
#  •    item_price (float)
#  •    item_quantity (int)
#  •    Default constructor
#  •    Initializes item's name = "none", item's price = 0, item's quantity = 0
#  •    Method
#  •    print_item_cost()
# Example of print_item_cost() output:
# Bottled Water 10 @ $1 = $10
#
# Step 2: In the main section of your code, prompt the user for two items and create two
objects of the ItemToPurchase class.
# Example:
#   Item 1
#   Enter the item name:
#   Chocolate Chips
#   Enter the item price:
#   3
#   Enter the item quantity:
#   1
#
#   Item 2
#   Enter the item name:
```

```
#   Bottled Water
#   Enter the item price:
#   1
#   Enter the item quantity:
#   10
#
# Step 3: Add the costs of the two items together and output the total cost.
#   Example:
#   TOTAL COST
#   Chocolate Chips 1 @ $3 = $3
#   Bottled Water 10 @ $1 = $10
#   Total: $13

# ----------------------------------------------------------------------
# Project:
# Online Shopping Cart
#
# Project description:
# The program is a small terminal app. that implements the functionality of
# an online shopping cart.
#
# ----------------------------------------------------------------------

# --- Module Contents Overview ---
# - Class: Banner
# - Class: Menu
# - Class: ItemToPurchase
# - Function: validate_prompt_int()
# - Function: validate_prompt_float()
# - Function: validate_prompt_string()
# - Function: main()
# ----------------------------------------------------------------------

# --- Dependencies / Imports ---
# - Standard Library: dataclasses, decimal, typing
# --- Requirements ---
# - Python 3.13
# ----------------------------------------------------------------------

# --- Apache-2.0 ---
# © 2025 Alexander Samuel Ricciardi - All rights reserved.
# License: Apache-2.0 | Code
# ----------------------------------------------------------------------

"""
```

```
Online shopping cart implementation

The program is a small terminal app.
it is an implementation of an online shopping cart implementation
The program renders banners and a menu,
and it calculates the total cost of items based on each item's price and quantity.
Only two items are asked to be entered in this implementation.

"""

# ____________________________________________________________________________
# Imports
#
from dataclasses import dataclass
from decimal import Decimal
from typing import Any, Literal, Sequence, TypeAlias


# ============================================================================
# ||                                                                        ||
# ||            Added/extra features not part of the assignment           ||
# ||            See Steps 1, 2, and 3 for assignment requirements         ||
# ||                                                                        ||
# ============================================================================
# ____________________________________________________________________________
# General Utility Functions
#
# ------------------------------------------------------------ validate_prompt_yes_or_no()
def validate_prompt_yes_or_no(prompt: str) -> bool:
    """Prompt the user for a yes/no confirmation"""
    while True:
        choice = input(f"{prompt} [Y/N]: ").strip().lower()
        if choice in ("y", "yes"):
            return True
        if choice in ("n", "no"):
            return False
        print("Invalid input. Please enter 'Y' or 'N'.")

# ---------------------------------------------------------- end validate_prompt_yes_or_no()

# ------------------------------------------------------------------ wait_for_enter()
def wait_for_enter() -> None:
    """Terminal pause: Waits until user presses Enter"""
    input("\nPress Enter to continue...")
# ---------------------------------------------------------------- end wait_for_enter()
```

```
# ____________________________
# User input validation utility functions
#
# ------------------------------------------------------------ validate_prompt_int()
def validate_prompt_int(prompt: str) -> int:
    """Prompt until a valid integer is entered"""
    while True:
        raw = input(f"{prompt}\n").strip()
        try:
            return int(raw)
        except ValueError:
            print("Invalid input. Please enter a whole integer (e.g., 3).")

# ------------------------------------------------------------ end validate_prompt_int()

# ------------------------------------------------------------ validate_prompt_float()
def validate_prompt_float(prompt: str) -> float:
    """Prompt User until a valid float is entered"""
    while True:
        raw = input(f"{prompt}\n").strip()
        try:
            return float(raw)
        except ValueError:
            print("Invalid input. Please enter a valid float (e.g., 12.99).")

# ------------------------------------------------------------ end validate_prompt_float()

# ------------------------------------------------------------ validate_prompt_string()
def validate_prompt_string(prompt: str) -> str:
    """Prompt User until a valid float is entered"""
    while True:
        raw = input(f"{prompt}\n").strip()
        try:
            if raw == "": raise ValueError()
            return str(raw)
        except ValueError:
            print("Invalid input. Please enter a string (e.g., Hello).")

# ------------------------------------------------------------ end validate_prompt_string()

# ____________________________________________________________________________
# Utility Classes
#
# ____________________
```

```
# Terminal Banner
#
# ------------------------------------------------------------------------ class Banner
class Banner:
    """Render Banner object to be on the terminal

    It renders a banner box using box-drawing characters using text alignments
left/center/right

    """

    Alignment: TypeAlias = Literal["left", "center", "right"]

    # ____________________
    #  Class constants
    #
    DEFAULT_TEXT: str = "Banner"
    DEFAULT_ALIGNMENT: str = "center"
    DEFAULT_WIDTH: int = 44


    # ____________________
    # Constructor
    #
    # ------------------------------------------------------------- __init__()
    def __init__(
        self,
        text: str = DEFAULT_TEXT,
        alignment: Alignment = DEFAULT_ALIGNMENT,
        width: int = DEFAULT_WIDTH,
    ) -> None:
        """Initialize a new banner

        Args:
            text: Text in the banner
            alignment: Alignment for the text (left/center/right)
            width: width of the banner content area
        """
        self._lines: list[tuple[str, Alignment]] = [(text, alignment)]
        self.width: int = max(width, len(text), 3)
    # ---------------------------------------------------------- end __init__()

    # ____________________
    # Private Methods
    #
```

```
# ------------------------------------------------------------ _from_lines()
@classmethod
def _from_lines(
    cls,
    lines: Sequence[tuple[str, Alignment]],
    width: int = DEFAULT_WIDTH,
) -> "Banner":
    """Construct a banner using lines of box-drawing characters

    Args:
        lines: Sequence of text/alignment
        width: inner width of the banner

    Returns:
        "Banner" to be render

    Raises:
        ValueError: if lines: is empty
    """
    if not lines:
        raise ValueError("Banner.from_lines requires at least one line.")
    first_text, first_align = lines[0]
    inst = cls(text=first_text, alignment=first_align, width=width)
    inst._lines = list(lines)
    content_width = max(len(t) for t, _ in inst._lines)
    inst.width = max(width, content_width, 3)
    return inst
# ------------------------------------------------------------ end _from_lines()

# ____________________
# Banner constructor helper methods
#

def _top(self) -> str:
    return f"╔{'=' * self.width}╗"

def _divider(self) -> str:
    return f"╠{'=' * self.width}╣"

def _bottom(self) -> str:
    return f"╚{'=' * self.width}╝"

def _text_line(self, text: str, alignment: Alignment = "center") -> str:
    if alignment == "left":
```

```
            return f"‖{text.ljust(self.width)}‖"
        if alignment == "right":
            return f"‖{text.rjust(self.width)}‖"
        return f"‖{text.center(self.width)}‖"

    # ________________________
    # Render banner
    #
    # ------------------------------------------------------------- render()
    def render(self) -> str:
        """Render the banner text as a single string that can be displayed

        Returns:
            The rendered banner in the form of a string
        """
        output = [self._top()]
        for text, align in self._lines:
            output.append(self._text_line(text, align))
        output.append(self._bottom())
        return "\n".join(output)
    # ---------------------------------------------------------- end render()

# ------------------------------------------------------------------------ end class Banner


# ________________________
# Terminal app Menu
#
# ------------------------------------------------------------------------- class Menu
class Menu:
    """Render a Menu object to be on the terminal using the Banner class"""

    # ------------------------------------------------------------- __init__()
    def __init__(
        self,
        title: str,
        options: Sequence[str],
        width: int = Banner.DEFAULT_WIDTH,
    ) -> None:
        """Create a new menu

        Args:
            title: Menu title
            options: Menu options, sequence tarting at 1.
            width: inner Banner width
```

```
        Raises:
            ValueError: if options is empty.
        """
        if not options:
            raise ValueError("Menu requires at least one option.")
        self.title = title
        self._options = list(options)
        self._width = width
        self._numbered_options = [
            f"{index}. {label}" for index, label in enumerate(self._options, start=1)
        ]
        self._banner_lines = [
            (self.title, "center"),
            *[(option, "left") for option in self._numbered_options],
        ]
        self._banner = Banner._from_lines(self._banner_lines, width=self._width)
    # ------------------------------------------------------------ end __init__()

    # ________________________
    # Menu constructor helper methods
    #

    def _choices(self) -> list[str]:
        """Turn index choices number to strings (e.g., ["1", "2"])."""
        return [str(index) for index in range(1, len(self._options) + 1)]

    def _choice_index_range(self) -> str:
        """Turn choice (str) index into a displayable range (e.g., "1-3" or "1")
        see _choices()
        """
        options = self._choices()
        if len(options) == 1:
            return options[0] # "1"
        return f"{options[0]}-{options[-1]}" # (e.g., "1-3")

    def _choice_index_list(self) -> str:
        """lists available index choices for errors message (e.g., "1, 2, or 3")."""
        options = self._choices()
        if len(options) == 1:
            return options[0]
        return ", ".join(options[:-1]) + f", or {options[-1]}"

    # ------------------------------------------------------------ render()
    def render(self) -> str:
        """Render the menu as a banner string
```

```
        Returns:
            A string of the fully form Menu
        """
        sections = [
            self._banner._top(),
            self._banner._text_line(self.title, alignment="center"),
            self._banner._divider(),
        ]
        sections.extend(
            self._banner._text_line(option, alignment="left") for option in
                                                                    self._numbered_options
        )
        sections.append(self._banner._bottom())
        return "\n".join(sections)
    # ------------------------------------------------------------ end render()

# ------------------------------------------------------------------------ end class Menu

# ============================================================================
# ||                                                                        ||
# ||                Step 1: Build the ItemToPurchase class                  ||
# ||                                                                        ||
# ============================================================================
# ____________________________________________________________________________
# App Classes Definitions
#
# ------------------------------------------------------------------ class ItemToPurchase
@dataclass
class ItemToPurchase:
    """Represent a purchasable item with its name, price, and quantity.

    This is the minimal domain model used by the console backend to compute
    line-item and total costs. It does not perform I/O and holds only data
    relevant to pricing and quantities.
    """
    #_____________________________________
    # Assignment requirement
    # -- Step 1 Initializes item variables
    item_name: str = "none"
    item_price: float = 0.0
    item_quantity: int = 0

    #_____________________________________
    # Assignement requirement
```

```
    # -- Step 1 part of the print_item_cost() output format
    # ------------------------------------------------------------ format_currency()
    @staticmethod
    def format_currency(value: float) -> str:
        """Return a formatted float (e.g., 10.00 = 10) remove unnecessary trailing zero
        Example:
            2.0 -> "2"; 2.5 -> "2.5"; 2.75 -> "2.75".
        """
        if value == int(value):
            return str(int(value))
        return f"{value:.2f}".rstrip("0")  # Strip unnecessary Os
    # ------------------------------------------------------------ end format_currency()

    #_____________________________________
    # Assignment requirement
    # -- Step 1 print_item_cost() method
    # ------------------------------------------------------------ print_item_cost()
    def print_item_cost(self) -> str:
        """Print and return cost for this item

        Returns:
            a cost line (e.g., "Apples 3 @ $1 = $3")
        """
        total_cost = self.item_price * self.item_quantity
        cost_statement = (
            f"{self.item_name} {self.item_quantity} @ "
            f"${self.format_currency(self.item_price)} = "
            f"${self.format_currency(total_cost)}"
        )
        print(cost_statement)
        return cost_statement
    # ------------------------------------------------------------ end print_item_cost()

# ------------------------------------------------------------------ end class ItemToPurchase


# ____________________________________________________________________________
# -------------- Main Function --------------
#
# ------------------------------------------------------------------------------ main()
def main() -> None:
    """Run the shopping cart program"""
    # _________________________
    # Embedded Helper Functions
    #
```

```python
# ------------------------------------------------------------ build_banner()
def build_banner(text: str) -> str:
    """Create a banner string

    Args:
        text: The text inside the banner box

    Returns:
        A string, the full banner, with borders
    """
    banner = Banner(text=text, alignment="center", width=Banner.DEFAULT_WIDTH)
    return banner.render()
# ------------------------------------------------------------ end build_banner()

# ------------------------------------------------------------ prompt_for_item()
def prompt_for_item(item_number: int) -> ItemToPurchase:
    """Prompt the user a new item values
    and from it creates a ItemToPurchase` object

    Args:
        item_number: the item sequenced number of creation

    Returns:
        ItemToPurchase object instance with name, price, and quantity entered

    Raises:
        ValueError: if user input is not valide
    """
    print(f"Item {item_number}")
    item_name = validate_prompt_string("Enter the item name:")
    item_price = validate_prompt_float("Enter the item price:")
    item_quantity = validate_prompt_int("Enter the item quantity:")
    return ItemToPurchase(item_name=item_name, item_price=item_price,
                                                        item_quantity=item_quantity)

# ------------------------------------------------------------ end prompt_for_item()

# _________________________
# Instance Banner and Menu objects
#
HEADER = build_banner("Online Shopping Cart")
ENTRY_BANNER = build_banner("Provide the item information")
RESULTS_BANNER = build_banner("TOTAL COST")
MAIN_MENU = Menu("Menu", ["Calculate total for two items", "Exit"])
```

```
print(HEADER)

# ____________________
# Main Loop
#
while True:
    print()
    print(MAIN_MENU.render())
    choice = input(f"Select an option ({MAIN_MENU._choice_index_range()}): ").strip()

    if choice == "1":
        # ============================================================
        # ||                                                        ||
        # ||    Step 2: prompt the user for two items and create    ||
        # ||            two objects of the ItemToPurchase class     ||
        # ||                                                        ||
        # ============================================================
        print()
        print(ENTRY_BANNER)
        first_item = prompt_for_item(1)
        print()
        second_item = prompt_for_item(2)
        # ============================================================
        # ||                                                        ||
        # ||    Step 3: Add the costs of the two items together     ||
        # ||            and output the total cost                   ||
        # ||                                                        ||
        # ============================================================
        total_cost = (
            first_item.item_price * first_item.item_quantity
            + second_item.item_price * second_item.item_quantity
        )

        print()
        print(RESULTS_BANNER)
        first_item.print_item_cost()
        second_item.print_item_cost()
        print(f"Total: ${ItemToPurchase.format_currency(total_cost)}")
        wait_for_enter()

    elif choice == "2":
        if (validate_prompt_yes_or_no("Are you sure that you want to exist? ")):
            print("\nBye! 👌\n")
            break
    else:
```

```
            print(f"\nInvalid selection. Please enter {MAIN_MENU._choice_index_list()}.")

# ------------------------------------------------------------------------------ end main()


# ____________________________________________________________________________
# Module Initialization / Main Execution Guard (if applicable)
#
# ------------------------------------------------------------------------ main_guard
if __name__ == "__main__":
    main()
# ---------------------------------------------------------------------------- end main_guard


 # ____________________________________________________________________________
 # End of File
 #
```

Continue next page

Figure 2

Project Outputs in the VS Code Terminal

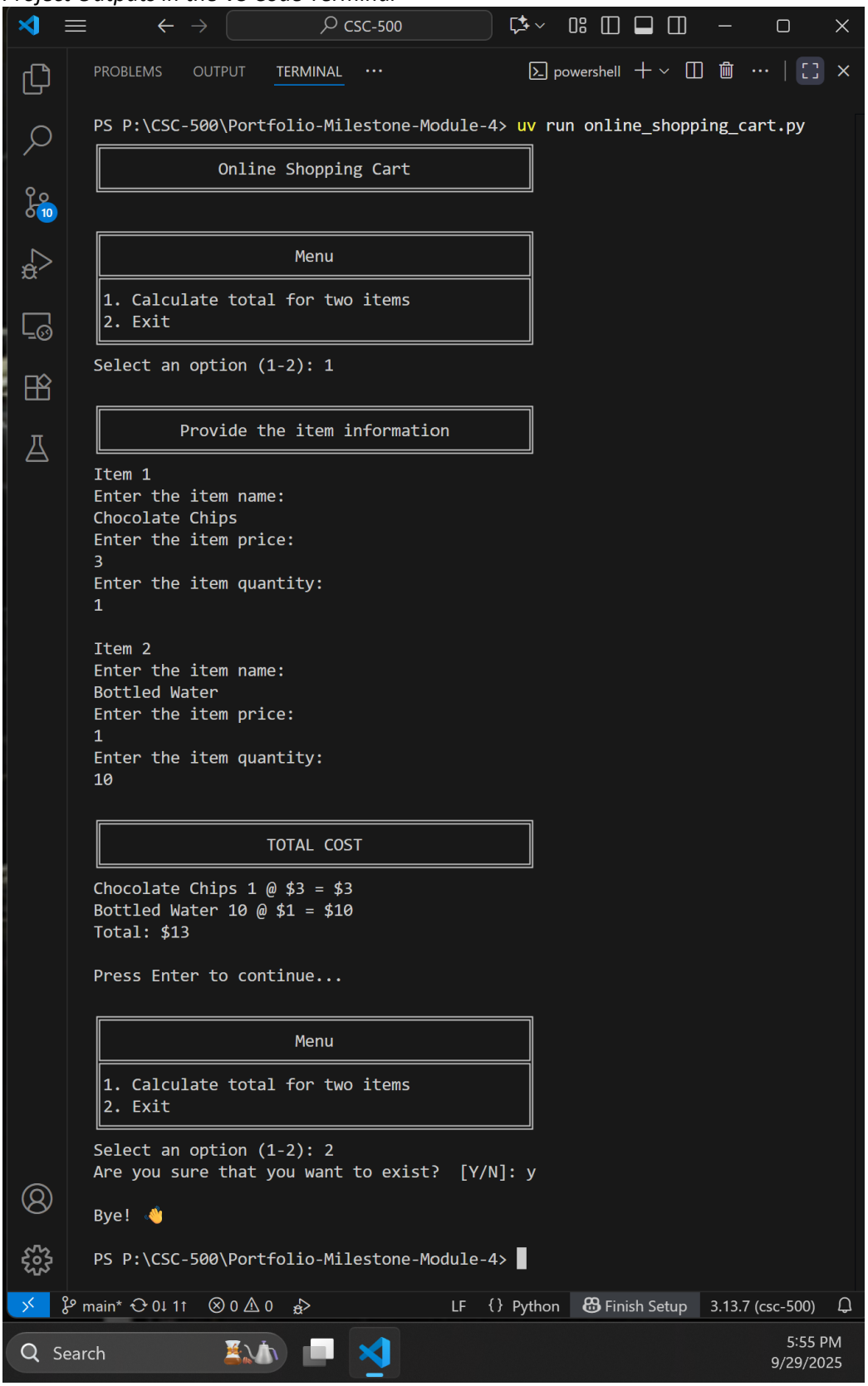
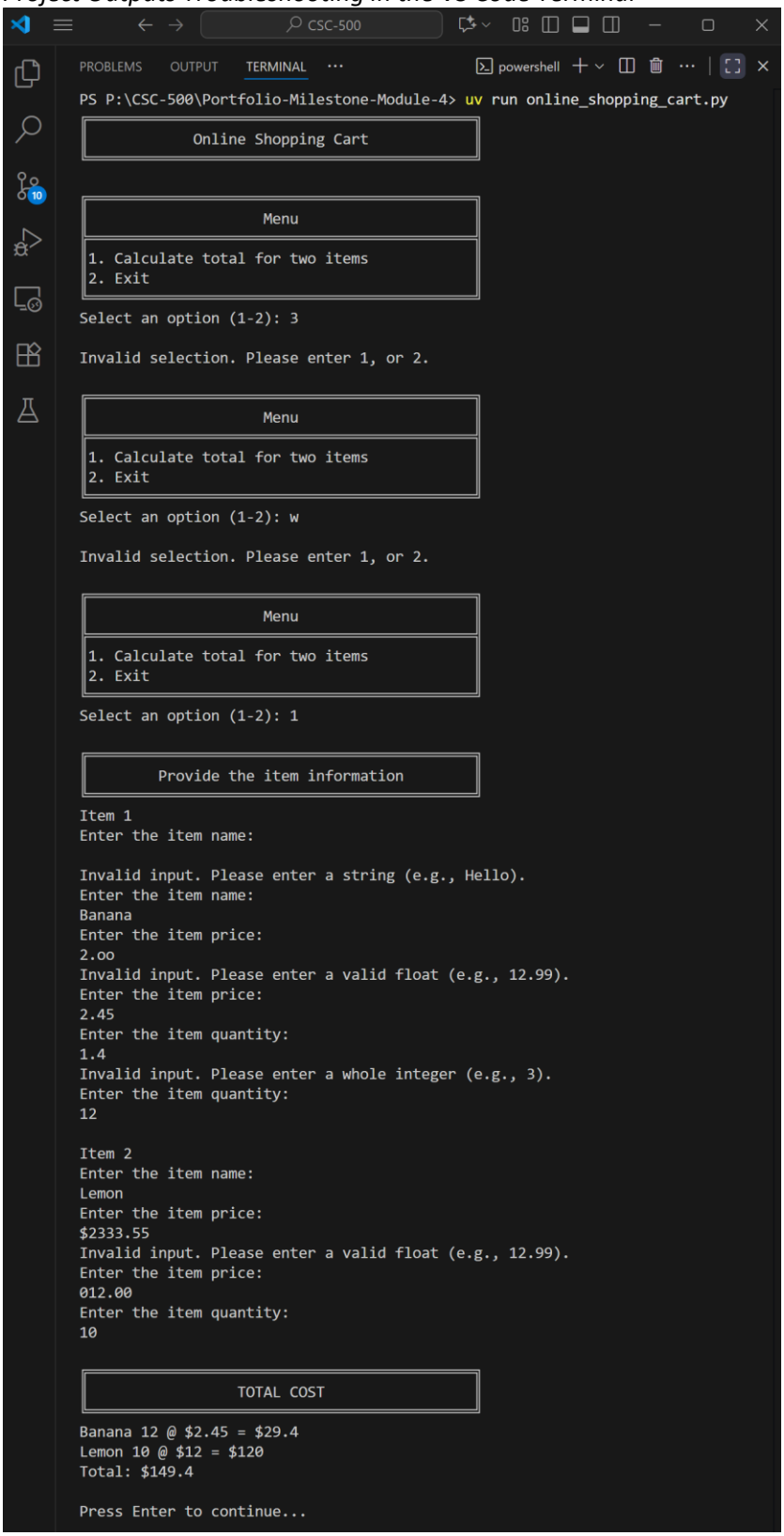
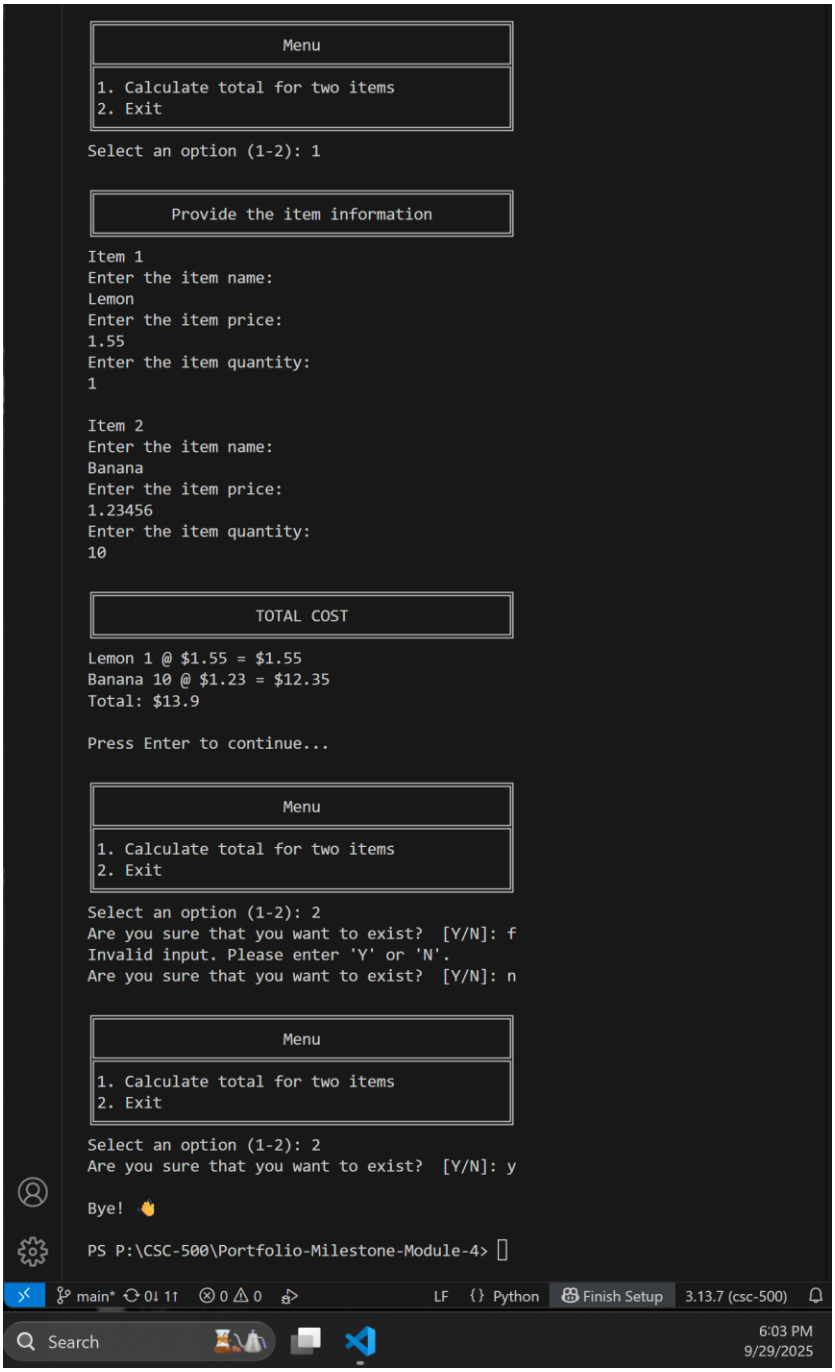


Figure 3

Project Outputs Troubleshooting in the VS Code Terminal





As shown in Figures 2, 3, and Code Snippet 1, the program runs without any issues, displaying the correct outputs as expected.