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.
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 utility functions and classes that I can reuse; they are found below the comments:

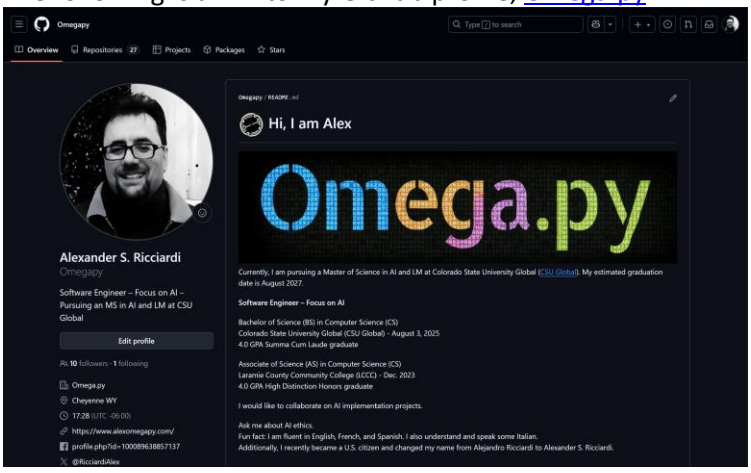
```
# ================================================================ Added Utilities
```

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

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

Git Repository:

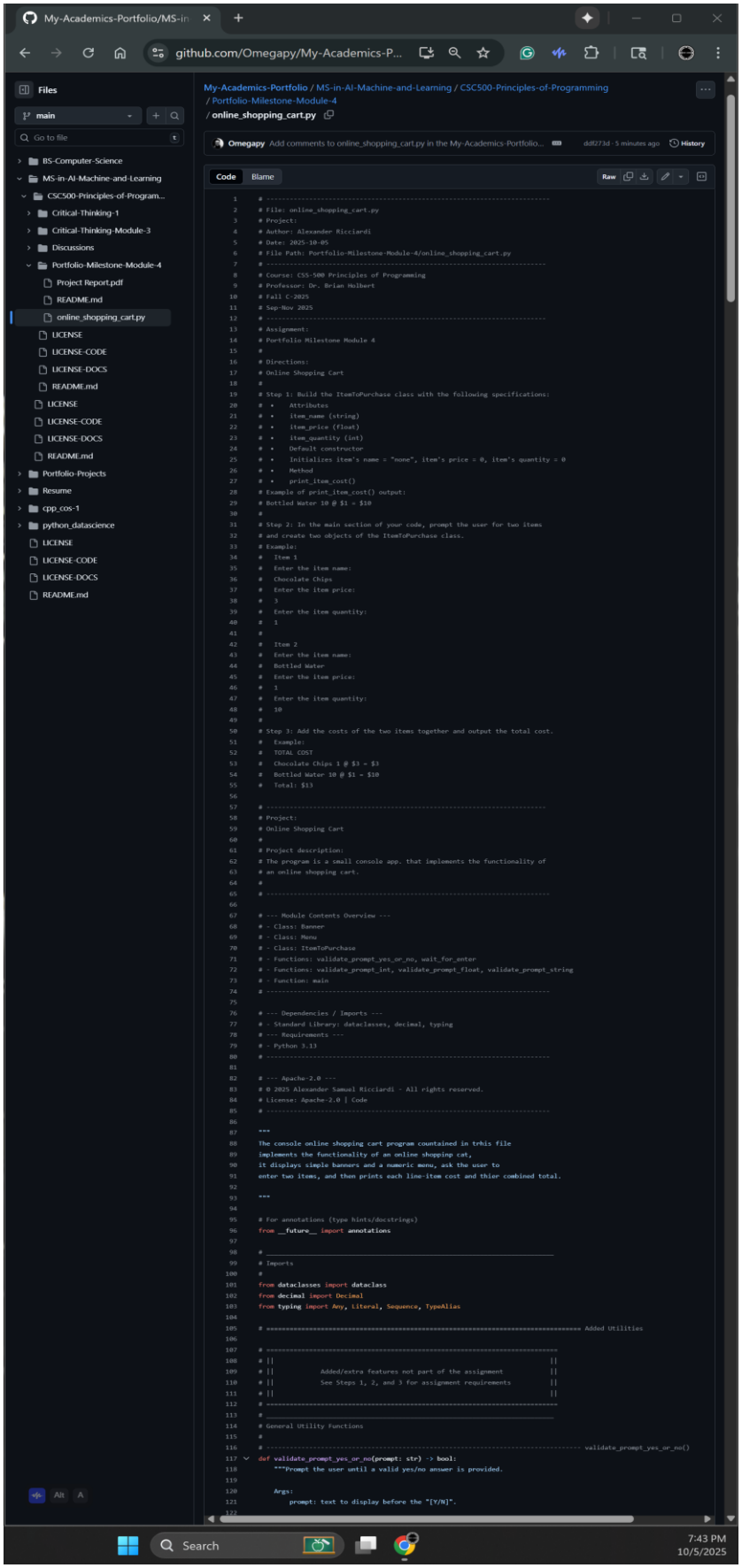
I use GitHub as my Distributed Version Control System (DVCS).
The following is a link to my GitHub profile, Omega.py.



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/Portfolio-Milestone-Module-4

Figure-1

Code on GitHub



Code Snippet 1

Project Code in VS Code

```
# ------------------------------------------------------------------------
# File: online_shopping_cart.py
# Project:
# Author: Alexander Ricciardi
# Date: 2025-10-05
# 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 console app. that implements the functionality of
# an online shopping cart.
#
# ----------------------------------------------------------------------

# --- Module Contents Overview ---
# - Class: Banner
# - Class: Menu
# - Class: ItemToPurchase
# - Functions: validate_prompt_yes_or_no, wait_for_enter
# - Functions: validate_prompt_int, validate_prompt_float, 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
# ----------------------------------------------------------------------

"""
The console online shopping cart program contained in this file
```

```
implements the functionality of an online shopping cart,
it displays simple banners and a numeric menu, asks the user to
Enter two items, and then print each line-item cost and their combined total.

"""

# For annotations (type hints/docstrings)
from __future__ import annotations

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

# ============================================================================ Added Utilities

# ============================================================================
# ||                                                                        ||
# ||            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 until a valid yes/no answer is provided.

    Args:
        prompt: text to display before the "[Y/N]".

    Returns:
        True if the user selects yes ("y"/"yes"); False if no ("n"/"no").

    Examples:
        user input shown after [prompts]:

            >>> result = validate_prompt_yes_or_no("Continue?")
            Continue? [Y/N]: maybe
            Invalid input. Please enter 'Y' or 'N'.
            Continue? [Y/N]: y
            >>> result
```

```
            True
    """
    # Loop until a yes/no input is provided.
    while True:
        choice = input(f"{prompt} [Y/N]: ").strip().lower()
        # confirmations
        if choice in ("y", "yes"):
            return True
        # reinforce confirmations
        if choice in ("n", "no"):
            return False
        # invalid input message
        print("Invalid input. Please enter 'Y' or 'N'.")

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

# ------------------------------------------------------------------ wait_for_enter()
def wait_for_enter() -> None:
    """Pause execution until the user presses Enter.

    Examples:
            >>> wait_for_enter()

            Press Enter to continue...
            <user presses Enter>
    """
    input("\nPress Enter to continue...")
# ------------------------------------------------------------------ end wait_for_enter()

# ____________________________
# User input validation utility functions
#
# ============================================================================
# Input Validation Utility Functions
# ============================================================================
# ------------------------------------------------------------ validate_prompt_int()
def validate_prompt_int(prompt: str) -> int:
    """Ask the user until a valid integer is entered.

    Args:
        prompt: text to display to the user

    Returns:
        The validated integer value
```

```
    Behavior:
        - re-prompts when the input cannot be validated as an integer

    Examples:
       user input shown after [prompts]:

            >>> value = validate_prompt_int("Enter the item quantity:")
            Enter the item quantity:
            three
            Invalid input. Please enter a whole integer (e.g., 3).
            Enter the item quantity:
            3
            >>> value
            3
    """
    # Keep asking until a valid int is entered
    while True:
        raw = input(f"{prompt}\n").strip()
        try:
            return int(raw)
        except ValueError:
            # Invalid input error message
            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:
    """Asks the user until a valid float is entered.

    Args:
        prompt: text to display to the user

    Returns:
        The validated float value

    Behavior:
        - Re-prompts when the input cannot be validated as a float

    Examples:
        user input shown after [prompts]:

            >>> price = validate_prompt_float("Enter the item price:")
            Enter the item price:
            price
```

```
            Invalid input. Please enter a valid float (e.g., 12.99).
            Enter the item price:
            12.99
            >>> price
            12.99
    """
    # Keep asking until a valid float is entered
    while True:
        raw = input(f"{prompt}\n").strip()
        try:
            return float(raw)
        except ValueError:
            # Invalid input error message
            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:
    """Ask the user until a non-empty string is entered.

    Args:
        prompt: text to display to the user

    Returns:
        A validated string value

    Behavior:
        - when the input cannot be validated as a non-empty string

    Examples:
        user input shown after [prompts]:

            >>> name = validate_prompt_string("Enter the item name:")
            Enter the item name:

            Invalid input. Please enter a string (e.g., Hello).
            Enter the item name:
            Apples
            >>> name
            'Apples'
    """
    # Keep asking until a non-empty string is entered
    while True:
        raw = input(f"{prompt}\n").strip()
```

```
        try:
            if raw == "":
                # raise error if input string is empty
                raise ValueError()
            return str(raw)
        except ValueError:
            # Keep asking until a none-empty is entered
            print("Invalid input. Please enter a string (e.g., Hello).")

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

# ____________________________________________________________________________
# Utility Classes
#
# ______________________
# console Banner
#
# ============================================================================
# Banner Class (box headers)
# ============================================================================
# ------------------------------------------------------------------------ class Banner
class Banner:
    """Instantiate box banner for the console from one or more text lines

    The banner consists of a top border, one header line
    and maybe more text lines with alignment (left/center/right), and a bottom border.
    The inner_width of the box automatically expands to fit the text length.

    Examples:
                ╔═══════════════╗
                ║  First line   ║
                ╚═══════════════╝
    """

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

    # ______________________
    #  Class constants
    #
    # Default title text when no content is provided
    DEFAULT_TEXT = "Banner"
    # Default alignment
    DEFAULT_ALIGNMENT = "center"
    # Whether divider is applied to the line
    DEFAULT_ISDIVIDER = False
```

```
    # Default banner content tuple
    DEFAULT_CONTENT: list[tuple[str, Alignment, bool]] = [(DEFAULT_TEXT, DEFAULT_ALIGNMENT,
DEFAULT_ISDIVIDER)]
    # Minimum Banner inner width
    MINIUM_WIDTH: int = 10

    # _________________________
    # Constructor
    #
    # ------------------------------------------------------------ __init__()
    def __init__(
        self,
        content: list[tuple(*str, *Alignment, *bool)] = DEFAULT_CONTENT,
        inner_width: int = MINIUM_WIDTH,
    ) -> None:
        """Construct and initialize a new banner with default values if none is entered

        Args:
            text: text lines inside the banner
            alignment: Alignment of text lines ("left", "center", or "right")
            inner_width: the minimum inner with of the banner (auto-expands for longer text)

        example:
             >>> banner_1 = Banner([("First line"),
                                    (),
                                    ("Third Line", "left", True),
                                    ("Fourth Line", "Right")
                                ])
            >>> print(banner_1.render())

                ╔═══════════════╗
                ║  First line   ║ # Default alignment and isDivider
                ║               ║ # Second Line empty
                ║ Third Line    ║ # Aligns to the left and adds a divider
                ╠═══════════════╣
                ║ Fourth Line   ║ # Aligns to the right and default isDivider
                ╚═══════════════╝


        """
        # Initialize the banner lines to default content
        self._lines: list [tuple(str, Alignment, bool)] = content
        # Check if the first line tuple is a default to string,
        # as a tuple with just one element, and if the element is a string defaults to a
        # string type
        # if the line tuple is a string, the inner width is the maximum comparison between
```

```
        # MINIUM_WIDTH and the string length
        if isinstance(self._lines[0], str):
            # Compare and return the largest value + 4
            self.inner_width = max(
                                    len(self._lines[0]), self.MINIUM_WIDTH
                                ) + 4 # inner padding
        else:
            # Compare lines text elements and return the text element largest length value + 4
            self.inner_width = max(
                                    # Compare and return the largest value
                                    max(# if the line tuple is empty it returns 0
                                        # else it  iterates through all the line tuple text
                                        #  elements
                                        # and return the length of each line tuple text
                                        # element
                                        (0 if not t else len(t[0])) for t in self._lines
                                    ),
                                    self.MINIUM_WIDTH
                                ) + 4
    # ------------------------------------------------------------ end __init__()

    # _____________________
    # Banner render helper methods

    # -----------------
    # line render helper method
    def _text_line(self, text: str, alignment: Alignment) -> str:
        """Build a text line, aligned inside the banner borders

        Examples:
            ‖  First line (Header)  ‖
        """
        # Left align the text by adding spaces to the right of the text
        if alignment == "left":
            return f"‖ {text.ljust(self.inner_width -2)} ‖"
        # Right align the text by adding spaces to the left of the text
        if alignment == "right":
            return f"‖ {text.rjust(self.inner_width -2)} ‖"
        # Center align the text by adding spaces on both sides of the text
        return f"‖ {text.center(self.inner_width - 2)} ‖"

    # -----------------
    # Banner borders render helper methods

    # Top border line for the banner
```

```
    def _top(self) -> str:
        """Build top banner border.

        Examples:
            ╔═══════════════════════╗
        """

        return f"╔{'=' * self.inner_width}╗"

    # Horizontal divider used between sections
    def _divider(self) -> str:
        """Build borderline divider after the first text line.

        Notes: if the Banner has one line it gets replaced by
            the Banner bottom line in the render method

        Examples:
            ╠═══════════════════════╣
        """
        return f"╠{'=' * self.inner_width}╣"

    # Bottom border line for the banner
    def _bottom(self) -> str:
        """Return the bottom border line for the banner.

        Examples:
            ╚═══════════════════════╝
        """
        return f"╚{'=' * self.inner_width}╝"

    # _________________________
    # Render banner to one string
    #
    # ------------------------------------------------------------ render()
    def render(self) -> str:
        """Render the full banner as a single string, including first lines, other line if
any,
            border elements.

        Example:
                ╔═══════════════╗
                ║  First line   ║ # Default alignment and isDivider
                ║               ║ # Second Line empty
                ║ Third Line    ║ # Aligns to the left and adds a divider
                ╚═══════════════╝
```

```
                ╟ Fourth Line  ╢ # Aligns to the right and default isDivider
                ╚═══════════════╝
        """
        # Add top border (e.g., "╔═══╗") banner string
        banner = [self._top()]
        # For each Loop, loops over the line tuple list (_lines)
        for line in self._lines:
            # Empty line tuple e.g., ()
            if not line:
                txt = ""
                align = self.DEFAULT_ALIGNMENT
                isDiv = self.DEFAULT_ISDIVIDER
            # Check if the line tuple has defaulted to string,
            # as a tuple with just one element and if the element is a string defaults to a
            # string type
            # ("Option")
            elif isinstance(line, str):
                txt = line
                align = self.DEFAULT_ALIGNMENT
                isDiv = self.DEFAULT_ISDIVIDER
            # Line tuple with more than one element set
            # e.g. ("Option", "left") or ("Option", "left", True)
            else:
                txt = line[0]
                align = line[1]
                # set the divider flag to the default value if no flag value was set, else to
                # the set value
                isDiv = self.DEFAULT_ISDIVIDER if len(line) < 3 else line[2]
            # add text line (e.g., "╟  First line  ╢" ) to banner string
            banner.append(self._text_line(txt, align))
            # add border divider (e.g., "╟═══╢") if flag is true to banner string
            if isDiv: banner.append(self._divider())
        # add add bottom (e.g.,"╚═══╝") border to banner string
        banner.append(self._bottom())
        return "\n".join(banner) # add a return in the front of banner string

    # ------------------------------------------------------------ end render()

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

# ________________________
# console app Menu
#
# ========================================================================
# Menu Class Functionality (numbered options UI)
```

```
# ============================================================================
# ---------------------------------------------------------------------- class Menu
class Menu:
    """The menu class renders a console menu using the Banner class.

    Examples:
        >>> menu = Menu("Menu Example", ["Option", "Option", "Option"])
        >>> print(menu.render())
            ╔═══════════════════════╗
            ║     Menu Example      ║
            ╠═══════════════════════╣
            ║ 1. Option             ║
            ║ 2. Option             ║
            ║ 3. Option             ║
            ╚═══════════════════════╝
    """
    # ________________________
    # Constructor
    #
    # ------------------------------------------------------------ __init__()
    def __init__(
        self,
        title: str = "Menu",
        options: Sequence[str] = ["Option", "Option", "Option"],
        inner_width: int = 10,
    ) -> None:
        """Create a new menu.

        Args:
            title: title text displayed in the menu header
            options: option lines
            inner_width: the minimum inner banner width

        Examples:
            >>> menu = Menu("Menu", ["Start", "Exit"], inner_width=25)
        """
        # Validate we have at least one option; otherwise selection makes no sense
        if not options:
            raise ValueError("Menu requires at least one option.")
        # Initialize Varaibles with entered values
        self._title = title
        self._options = list(options)
        self._inner_width = inner_width
        # Add indices to the option using the list index, to be used as a selection choice
        self._numbered_options = [
```

```
            f"{index}. {text}" for index, text in enumerate(self._options, start=1)
        ]

        # Initialize banner first line by adding the menu header
        self._menu_lines = [ # First line of the Banner object
                (self._title, "center", True)
            ]
        # Initialize banner additional line by adding the menu options
        for option in self._numbered_options:
            self._menu_lines.append((option, "left"))

        # Instantiate the menu as a Banner object
        self._menu = Banner(self._menu_lines)

    # ------------------------------------------------------------ end __init__()

    # _________________________
    # Menu constructor helper methods
    #
    # ------------------------------------------------------------ _choices()
    def _choices(self) -> list[str]:
        """Return available choice indices as strings (e.g., ["1", "2"])

        Examples:
            >>> Menu("Menu Range", ["Option-1", "Obtion-2"])._choices()
            ['1', '2']
        """
        return [str(index) for index in range(1, len(self._options) + 1)]


    # ------------------------------------------------------------ End _choices()

    # ------------------------------------------------------------ _choice_index_range()
    def _choice_index_range(self) -> str:
        """Return a string of the index range (e.g., "1-3" or "1")

        Can be used to prompt user to enter a number related to the wanted option

        Examples:
            >>> Menu("Menu List", ["Option"])._choice_index_range()
            "1"
            >>> Menu("Menu List", ["Option-1", "Obtion-2"])._choice_index_range()
            "1-3"
        """
        options = self._choices() # List of choice indices
```

```
        # If there is only one option
        if len(options) == 1:
            return options[0] # "1"
        # Else range (e.g., "1-3")
        return f"{options[0]}-{options[-1]}"

    # ------------------------------------------------------------ End _choice_index_range()

    # ------------------------------------------------------------ _choice_index_list()
    def _choice_index_list(self) -> str:
        """Return a list of choices based on option indices (e.g., "1, 2, or 3")

        Can be used to prompt to enter a number related to the wanted option

        Examples:
            >>> menu = Menu("Menu list", ["Option"])._choice_index_list()
            "1"
            >>> Menu("Pair", ["First", "Second"])._choice_index_list()
            "1, or 2"
        """
        options = self._choices() # List of choice indices
        # only one choice index exists
        if len(options) == 1:
            return options[0] # "1"
        # Else list (e.g., "1, 2, or 3")
        return ", ".join(options[:-1]) + f", or {options[-1]}"
    # ------------------------------------------------------------ End _choice_index_list()

    # ------------------------------------------------------------ render()
    def render(self) -> "Banner Rendered":
        """Render the menu including title and numbered options

        Examples:
            >>> menu = Menu("Menu Example", ["Option", "Option", "Option"])
            >>> print(menu.render())
            ╔═══════════════════════╗
            ║     Menu Example      ║
            ╠═══════════════════════╣
            ╟ 1. Option             ╢
            ╟ 2. Option             ╢
            ╟ 3. Option             ╢
            ╚═══════════════════════╝
        """
        return self._menu.render()
```

```
    # ------------------------------------------------------------ end render()

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

# ========================================================================== End Added Utilities

# ================================================================================== Assignment
# ============================================================================
# ||                                                                        ||
# ||                 Step 1: Build the ItemToPurchase class                 ||
# ||                                                                        ||
# ============================================================================
# ____________________________________________________________________________
# App Classes Definitions
#
# ============================================================================
# ItemToPurchase Class Functionality (domain model for line items)
# ============================================================================
# ------------------------------------------------------------------- class ItemToPurchase
@dataclass() # data type of class
class ItemToPurchase:
    """The data class represents an item with name, price, and quantity.

    Examples:
        >>> item = ItemToPurchase(item_name="Apples", item_price=1.5, item_quantity=2)
        >>> item.item_name, item.item_price, item.item_quantity
        ('Apples', 1.5, 2)
    """
    #_____________________________________
    # Assignment requirement
    # -- Step 1 Initializes item variables
    item_name: str = "none"
    item_price: float = 0.0
    item_quantity: int = 0

    #_____________________________________
    # Assignment requirement
    # -- Step 1 part of the print_item_cost() output format
    # ------------------------------------------------------------ format_currency()
    @staticmethod
    def format_currency(value: float) -> str:
        """Format a numeric value for currency-like display without excess zeros.

        Examples:
            >>> ItemToPurchase.format_currency(2.0)
```

```
            '2'
            >>> ItemToPurchase.format_currency(2.5)
            '2.5'
            >>> ItemToPurchase.format_currency(2.75)
            '2.75'
        """
        # if the value is a whole number, it drops the decimal portion
        if value == int(value):
            return str(int(value))
        # else format to the two decimal places and remove trailing zeros
        return f"{value:.2f}".rstrip("0")

    # ------------------------------------------------------------ end format_currency()

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

        Returns:
            The line-item cost string (e.g., "Apples 3 @ $1 = $3").

        Examples:
            >>> ItemToPurchase(item_name="W", item_price=2.0,
item_quantity=3).print_item_cost()
            'W 3 @ $2 = $6'
        """
        # compute the item total based on price and quatity
        total_cost = self.item_price * self.item_quantity
        # render string to be displayed using required format
        cost_statement = (
            f"{self.item_name} {self.item_quantity} @ "
            f"${self.format_currency(self.item_price)} = "
            f"${self.format_currency(total_cost)}"
        )
        # Echo the formatted cost to the console for immediate feedback.
        print(cost_statement)
        # Return the string so callers can reuse or test it if desired.
        return cost_statement

    # ------------------------------------------------------------ end print_item_cost()

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

```
# ____________________________________________________________________________
# -------------- Main Function --------------
#
# ============================================================================
# Main Application Flow Functionality (program entry and user interaction)
# ============================================================================
# ------------------------------------------------------------------------------ main()
def main() -> None:
    """Run the shopping cart program.

    To simulate an online shopping cart, the Program uses a while loop to display a menu with
the two choices:
    "1. Calculate total for two items" and "2. exit". It prompts the user to enter two items'
information: item name, price, and quantity.
    validates input. Then it computes and prints the total costs for each item and the
combined total of the items.

    Examples:
            Online Shopping Cart
            ____________________

            1. Calculate total for two items
            2. Exit
            Select an option (1-2): 1

            Provide the item information
            Item 1
            Enter the item name:
            Apples
            Enter the item price:
            1.5
            Enter the item quantity:
            2

            Item 2
            Enter the item name:
            Water
            Enter the item price:
            2
            Enter the item quantity:
            3

            TOTAL COST
            Apples 2 @ $1.5 = $3
            Water 3 @ $2 = $6
```

```
        Total: $9
    """
    # ____________________
    # Embedded Helper Functions
    #

    # ------------------------------------------------------------ prompt_for_item()
    def prompt_for_item(item_number: int) -> ItemToPurchase:
        """Collect item details from the user and build an ``ItemToPurchase``.

        Args:
            item_number: The ordinal position of the item (1-based) used for display.

        Returns:
            A new ItemToPurchase object

        Examples:
            user input shown after prompts:

                Item 1
                Enter the item name:
                Apples
                Enter the item price:
                1.5
                Enter the item quantity:
                2
        """
        # item is being processed
        print(f"Item {item_number}")
        # prompt user, gather, and validate each entered input
        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
    #
    # Pre-build banners objects
    HEADER = Banner(["Online Shopping Cart"])
    ITEMS_BANNER = Banner(["Provide two items information"])
```

```
RESULTS_BANNER = Banner(["TOTAL COST"])
MAIN_MENU = Menu("Menu", ["Calculate total for two items", "Exit"])

# Render the prebuild objects
HEADER_RENDERED = HEADER.render()
ITEMS_BANNER_RENDERED = ITEMS_BANNER.render()
RESULTS_BANNER_RENDERED = RESULTS_BANNER.render()
MAIN_MENU_RENDERED = MAIN_MENU.render()

#_____________________
# Console display
#

print(HEADER_RENDERED)

# ------------------
# Main Loop
#
# Main event loop: show menu, accept a choice, and act on it.
while True:
    print()
    print(MAIN_MENU_RENDERED)
    choice = input(f"Select an option ({MAIN_MENU._choice_index_range()}): ").strip()

    if choice == "1":
        # If the user chose option 1, prompt for two items.
        # ============================================================
        # ||                                                        ||
        # ||    Step 2: prompt the user for two items and create    ||
        # ||            two objects of the ItemToPurchase class     ||
        # ||                                                        ||
        # ============================================================
        print()
        print(ITEMS_BANNER_RENDERED)
        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                   ||
        # ||                                                        ||
        # ============================================================
        # Compute (price * quantity) for each item
        # and sum the results into a total cost
```

```
            total_cost = (
                first_item.item_price * first_item.item_quantity
                + second_item.item_price * second_item.item_quantity
            )

            print()
            # Display the header for the total cost
            print(RESULTS_BANNER_RENDERED)
            # display the cost line for each item
            first_item.print_item_cost()
            second_item.print_item_cost()
            # display the combined item total
            print(f"Total: ${ItemToPurchase.format_currency(total_cost)}")
            # Pause so the user can review the results before continuing.
            wait_for_enter()

        elif choice == "2":
            # if the user chose option 2, confirm exit.
            if (validate_prompt_yes_or_no("Are you sure that you want to exist? ")):
                print("\nBye! 👌\n")
                break
        else:
            # else the input was not a recognized menu index; guide the user
            print(f"\nInvalid selection. Please enter {MAIN_MENU._choice_index_list()}.")

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

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

 # ___________________________________________________________________________
 # End of File
 #
```

Figure 2

Project Outputs in the VS Code Terminal

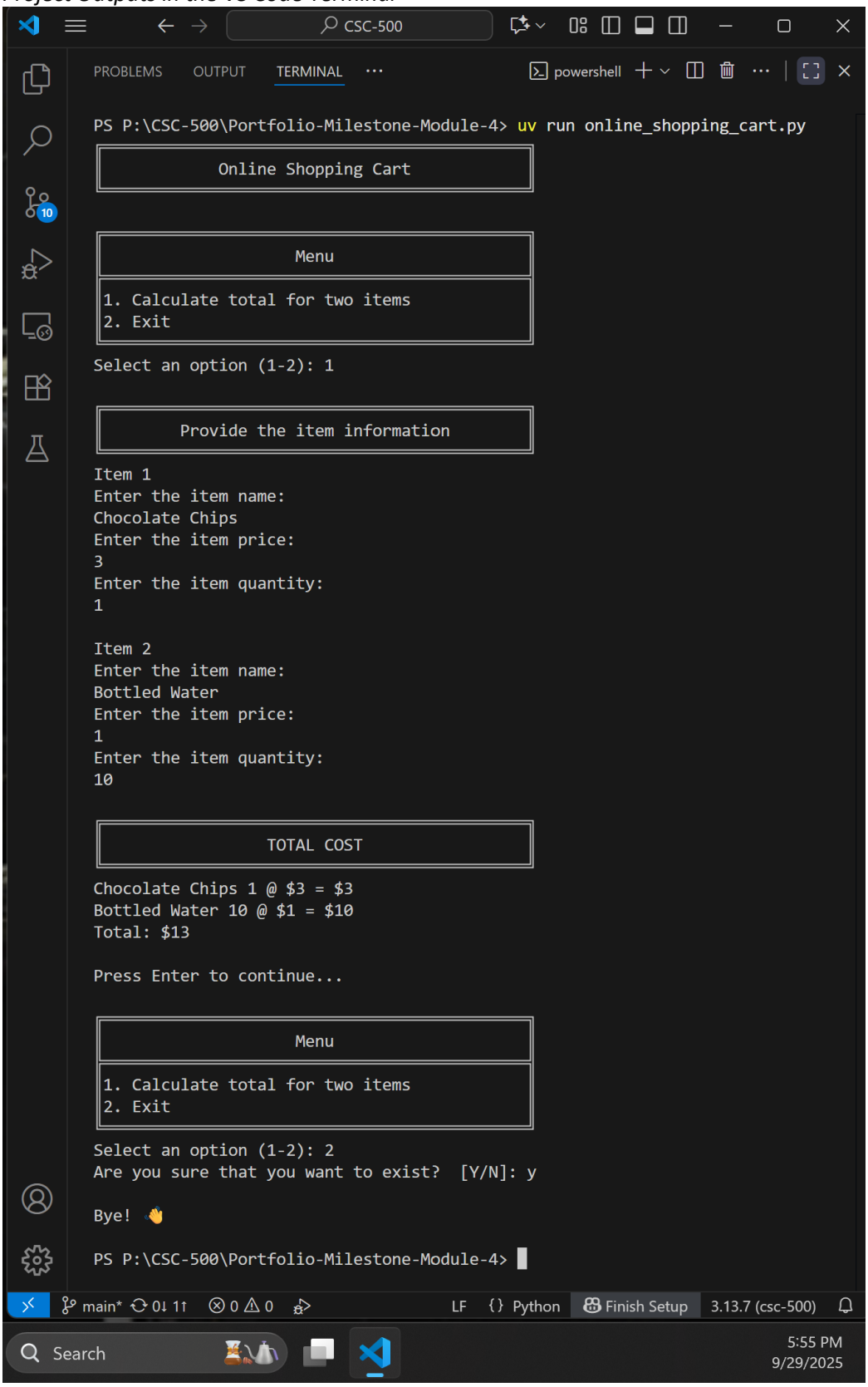
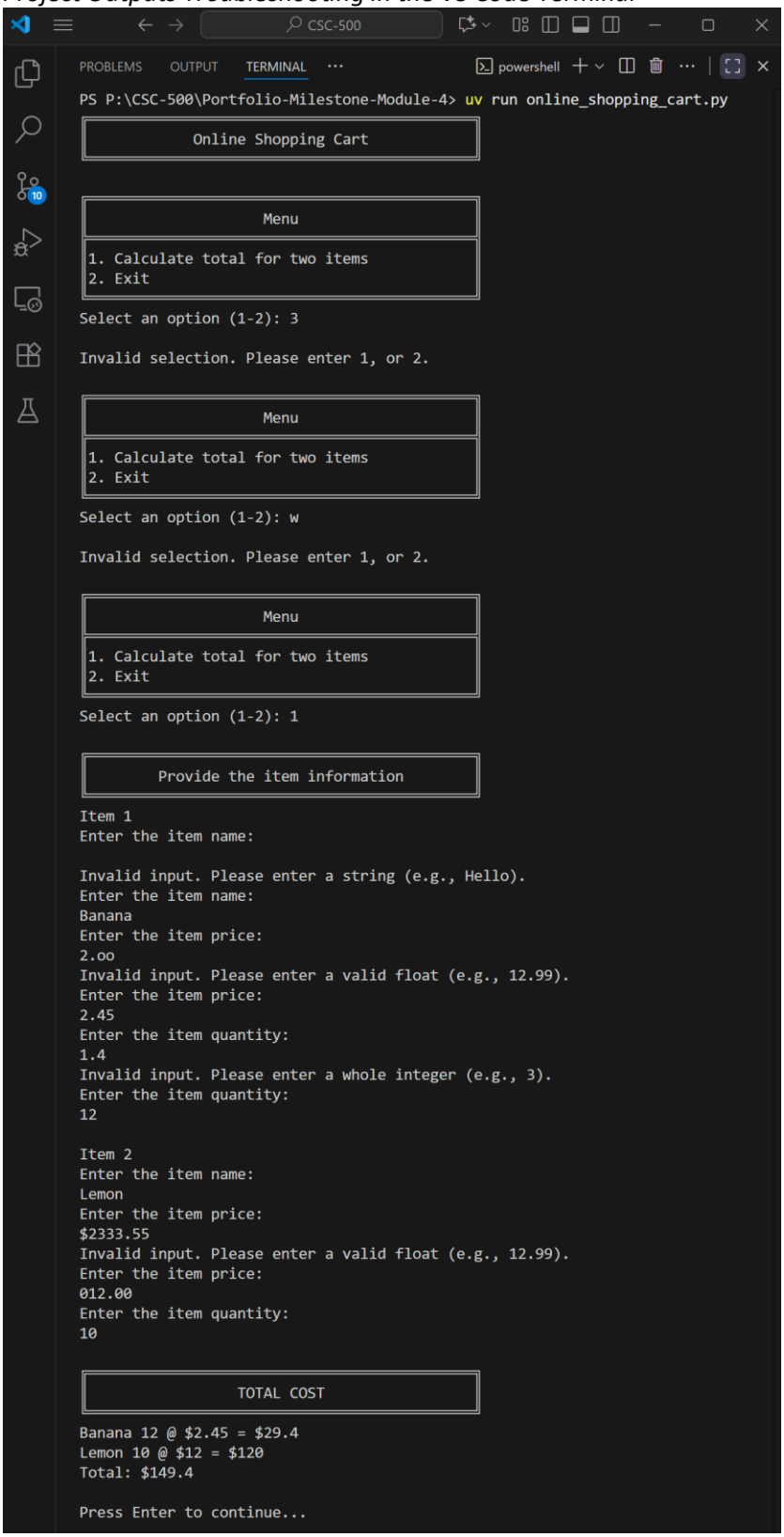
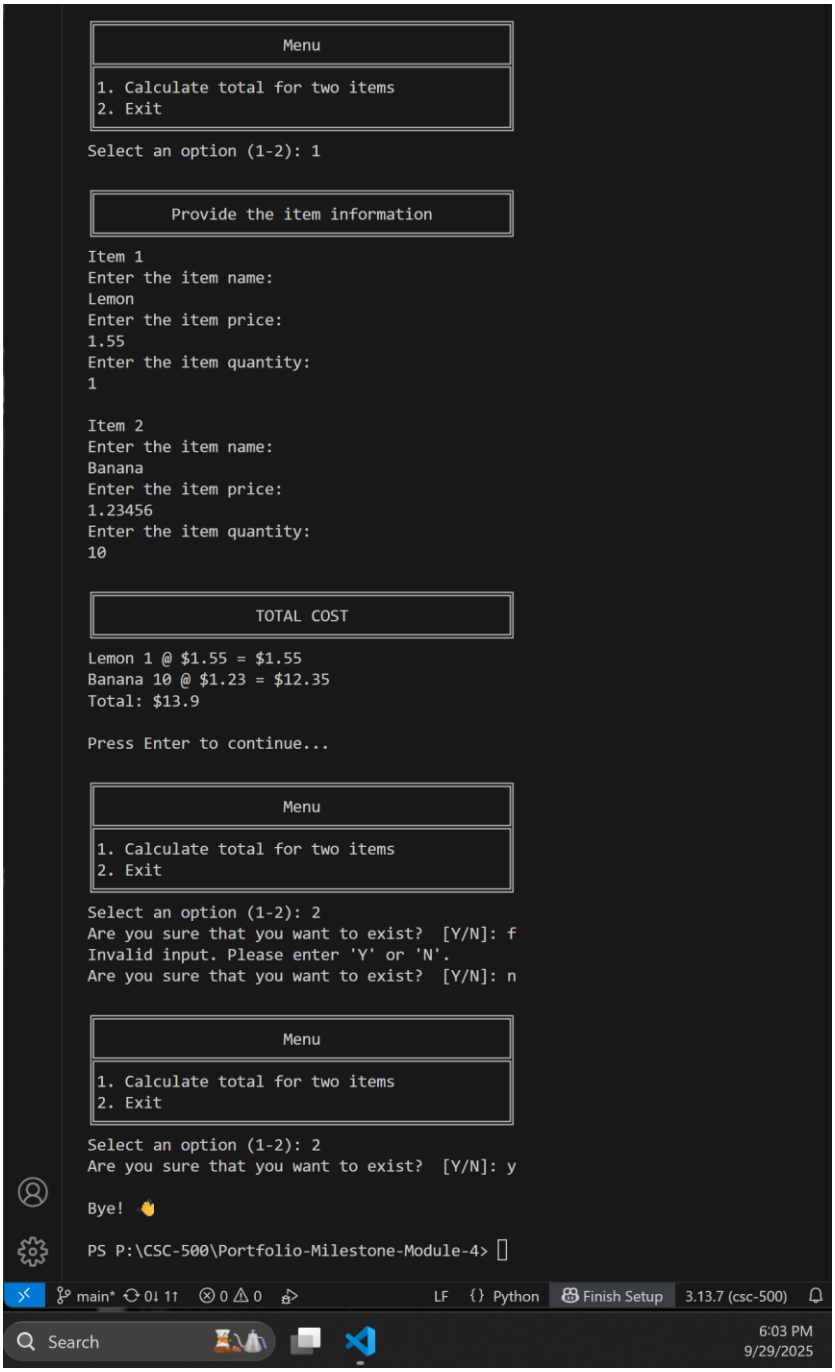


Figure 3

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.