Project Report:

Critical Thinking Module 5 – Rainfall Average Calculator and Bookstore Points

Alexander Ricciardi

Colorado State University Global

CSC500: Principles of Programming

Professor: Dr. Brian Holbert

October 12, 2025

Project Report:

Critical Thinking Module 5 – Rainfall Average Calculator and Bookstore Points

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 Rainfall Average Calculator and Bookstore Points.

The Assignment Direction:

Creating Python Programs

Part 1:
Write a program that uses nested loops to collect data and calculate the average rainfall over a period of years. The program should first ask for the number of years. The outer loop will iterate once for each year. The inner loop will iterate twelve times, once for each month. Each iteration of the inner loop will ask the user for the inches of rainfall for that month. After all iterations, the program should display the number of months, the total inches of rainfall, and the average rainfall per month for the entire period.

Part 2:
The CSU Global Bookstore has a book club that awards points to its students based on the number of books purchased each month. The points are awarded as follows:

- If a customer purchases 0 books, they earn 0 points.
- If a customer purchases 2 books, they earn 5 points.
- If a customer purchases 4 books, they earn 15 points.
- If a customer purchases 6 books, they earn 30 points.
- If a customer purchases 8 or more books, they earn 60 points.

Write a program that asks the user to enter the number of books that they have purchased this month and then displays the number of points awarded.

Submission:
Compile and submit your pseudocode, source code, and screenshots of the application executing the code from Parts 1 and 2, the results and GIT repository in a single document (Word is preferred).

My Program Description:

The program is a small console-based program consisting of 2 parts.

- Part 1 - Rainfall Average Calculator captures rainfall data from user inputs and calculates the rainfall average from the inputted per-month rainfall for the inputted number of years. Then display the results in the console.

- Part 2 - Bookstore Points calculates the Bookstore club points based on a tier system from the user inputted number of books purchased and displays the results.

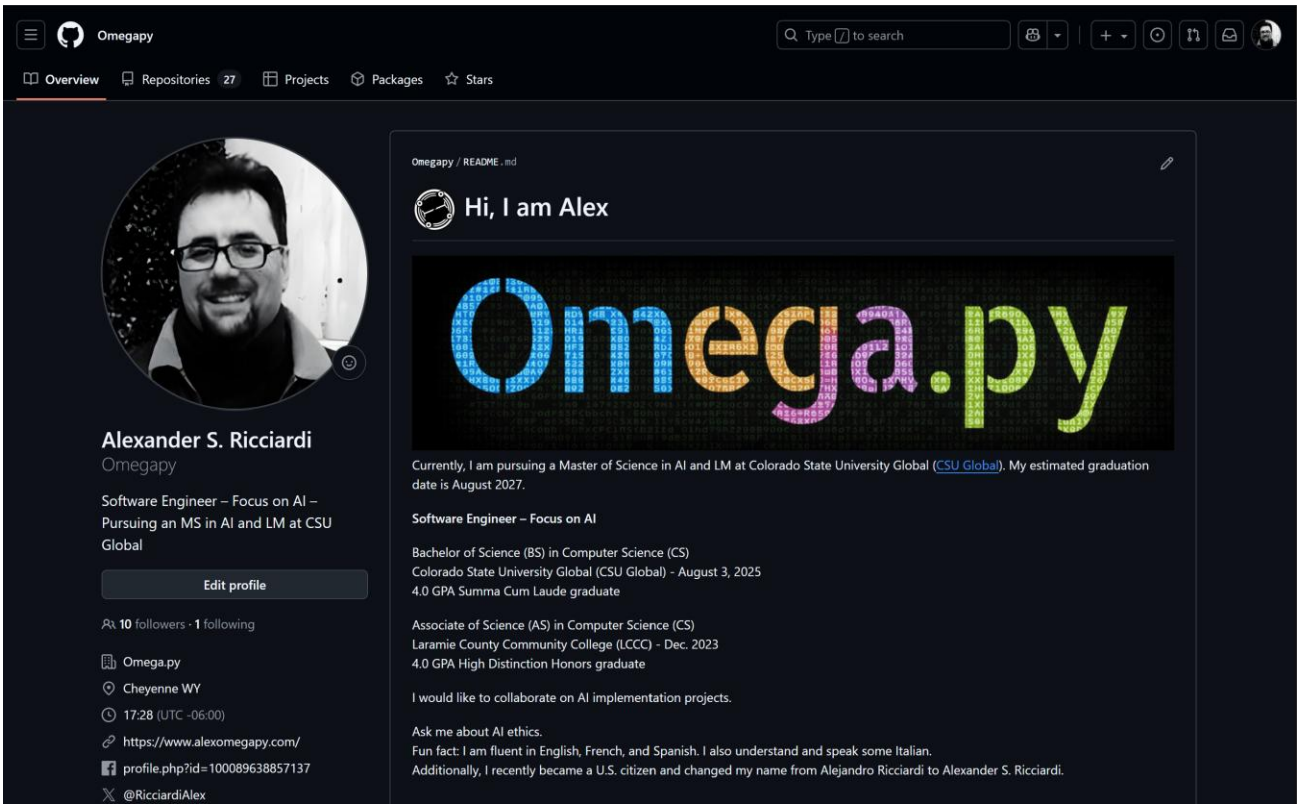
⚠️ My notes:
As I was using some of the same code lines for my critical assignments, I created several small utility functions and classes that I can reuse; they are found in the following Python files:

- validation_utilities.py: user input prompt and validation functions
- menu_banner_utilities.py: ASCII banner and menu UI classes

See the code for the utility functions and classes can be found at the end of this document

Git Repository:

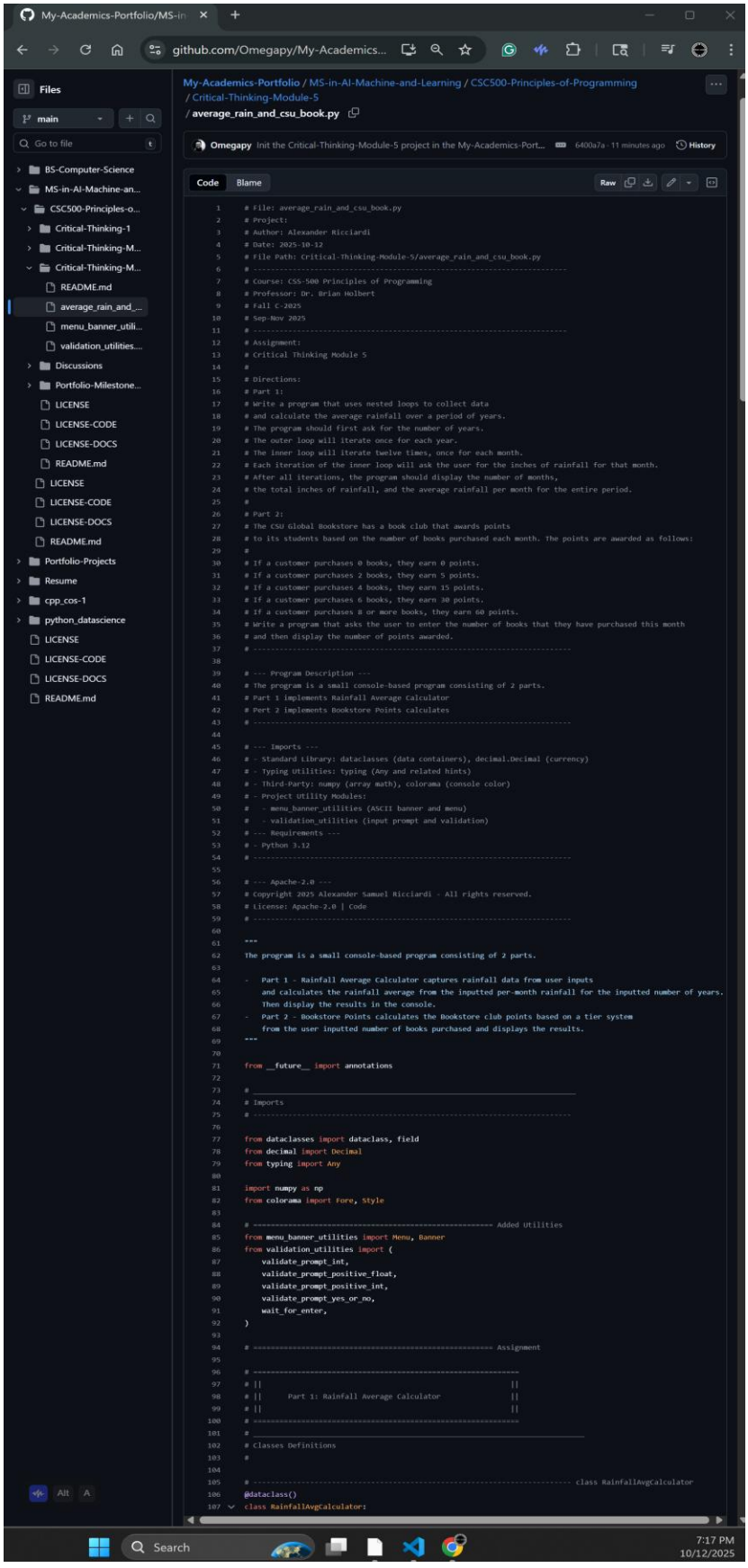
I use GitHub as my Distributed Version Control System (DVCS).
The following is a link to my GitHub profile, Omega.py.



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-5

See next page

Figure-1
Code on GitHub



Code Snippet 1

Main Project Code: average_rain_and_csu_book.py

```
# File: average_rain_and_csu_book.py
# Project:
# Author: Alexander Ricciardi
# Date: 2025-10-12
# File Path: Critical-Thinking-Module-5/average_rain_and_csu_book.py
# ------------------------------------------------------------------------
# Course: CSS-500 Principles of Programming
# Professor: Dr. Brian Holbert
# Fall C-2025
# Sep-Nov 2025
# ------------------------------------------------------------------------
# Assignment:
# Critical Thinking Module 5
#
# Directions:
# Part 1:
# Write a program that uses nested loops to collect data
# and calculate the average rainfall over a period of years.
# The program should first ask for the number of years.
# The outer loop will iterate once for each year.
# The inner loop will iterate twelve times, once for each month.
# Each iteration of the inner loop will ask the user for the inches of rainfall for that
month.
# After all iterations, the program should display the number of months,
# the total inches of rainfall, and the average rainfall per month for the entire period.
#
# Part 2:
# The CSU Global Bookstore has a book club that awards points
# to its students based on the number of books purchased each month. The points are awarded as
follows:
#
# If a customer purchases 0 books, they earn 0 points.
# If a customer purchases 2 books, they earn 5 points.
# If a customer purchases 4 books, they earn 15 points.
# If a customer purchases 6 books, they earn 30 points.
# If a customer purchases 8 or more books, they earn 60 points.
# Write a program that asks the user to enter the number of books that they have purchased
this month
# and then display the number of points awarded.
# ------------------------------------------------------------------------

# --- Program Description ---
# The program is a small console-based program consisting of 2 parts.
```

```
# Part 1 implements Rainfall Average Calculator
# Pert 2 implements Bookstore Points calculates
# ------------------------------------------------------------------------

# --- Imports ---
# - Standard Library: dataclasses (data containers), decimal.Decimal (currency)
# - Typing Utilities: typing (Any and related hints)
# - Third-Party: numpy (array math), colorama (console color)
# - Project Utility Modules:
#   - menu_banner_utilities (ASCII banner and menu)
#   - validation_utilities (input prompt and validation)
# --- Requirements ---
# - Python 3.12
# ------------------------------------------------------------------------

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

"""
The program is a small console-based program consisting of 2 parts.

-   Part 1 - Rainfall Average Calculator captures rainfall data from user inputs
    and calculates the rainfall average from the inputted per-month rainfall for the inputted
    number of years.
    Then display the results in the console.
-   Part 2 - Bookstore Points calculates the Bookstore club points based on a tier system
    from the user inputted number of books purchased and displays the results.
"""

from __future__ import annotations

# ____________________________________________________________________________
# Imports
# ------------------------------------------------------------------------

from dataclasses import dataclass, field
from decimal import Decimal
from typing import Any

import numpy as np
from colorama import Fore, Style

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

```python
from menu_banner_utilities import Menu, Banner
from validation_utilities import (
    validate_prompt_int,
    validate_prompt_positive_float,
    validate_prompt_positive_int,
    validate_prompt_yes_or_no,
    wait_for_enter,
)

# ========================================================== Assignment

# ============================================================
# ||                                                        ||
# ||      Part 1: Rainfall Average Calculator               ||
# ||                                                        ||
# ============================================================
# ____________________________________________________________________________
# Classes Definitions
#

# ------------------------------------------------------------------------ class
RainfallAvgCalculator
@dataclass()
class RainfallAvgCalculator:
    """Store rainfall data and compute average.

    Attributes:
        years: int, number of years
        data: Numpy array, stores rainfall by year (rows) and month (columns)

    Example:
        >>> rain_avg = RainfallAvgCalculator(years=2)
        >>> rain_avg.record(0, 0, 2.5)
        >>> rain_avg.data.shape
        (2, 12)
    """

    years: int
    # Used to store rainfall by year and month
    data: np.ndarray = field(init=False, repr=False)

    #_____________________________________
    # constructor setup
    # ------------------------------------------------------------  __post_init__()
    def __post_init__(self) -> None:
```

```
        """Initialize the two-dimensional rainfall array, year (rows) and month (columns)"""

        # Populate the array with zeros, year = num. (rows) of years and month = 12 (columns)
        self.data = np.zeros((self.years, 12), dtype=float)
    # ------------------------------------------------------------

    #_____________________________________
    # Attribute functions
    #

    # ------------------------------------------------------------  total_months()
    @property
    def total_months(self) -> int:
        """Return the total number of months"""
        return self.years * 12
    # ------------------------------------------------------------

    # ------------------------------------------------------------  total_inches()
    @property
    def total_inches(self) -> float:
        """Return the total rainfall based on all recorded months"""
        return float(np.sum(self.data))
    # ------------------------------------------------------------

    #_____________________________________
    # Attribute functions
    #

    # ------------------------------------------------------------ record()
    def record(self, year_index: int, month_index: int, value: float) -> bool:
        """Store a single month's rainfall data in an array

        Args:
            year_index: year index corresponding to the year
            month_index: month index corresponding to the month
            value: Rainfall in inches

        Returns:
        none

        raise:
        IndexError and ValueError

        Example:
            >>> rain_avg = RainfallAvgCalculator(years=1)
```

```
        >>> rain_avg.record(0, 0, 1.25)
        >>> rain_avg.data[0, 0]
        1.25
    """
    # Safe guard, just in case..
    try:
        if not 0 <= year_index < self.years:
            raise IndexError("year_index is out of range for the configured years")
        if not 0 <= month_index < 12:
            raise IndexError("month_index must be between 0 and 11 inclusive")
        if value < 0:
            raise ValueError("value must be non-negative")
    except (IndexError, ValueError) as exc:
        print(f"Invalid rainfall entry: {exc}")
        raise

    # Record the rainfall value in an array
    self.data[year_index, month_index] = value
# ------------------------------------------------------------

# ------------------------------------------------------------ average()
def average(self) -> float:
    """Calculate the average rainfall from stored data.

    Returns:
        Average rainfall in inches.

    Example:
        >>> rain_avg = RainfallAvgCalculator(years=1)
        >>> rain_avg.record(0, 0, 3.0)
        >>> round(rain_avg.average(), 2)
        0.25
    """

    # safeguard, check if it is no data recorded
    if self.data.size == 0:
        return 0.0
    # computes the mean from the data and returns it
    return float(np.mean(self.data))
# ------------------------------------------------------------

# ------------------------------------------------------------ summary()
def summary(self) -> str:
    """Compute formatted rainfall statistics for display.
```

```
        Returns:
            Formatted string storing the total months, total inches, and per-month average.

        Example:
            >>> rainfall_avg = RainfallAvgCalculator(years=1)
            >>> rainfall_avg.record(0, 0, 12.0)
            >>> "Average rainfall per month" in rainfall_avg.summary()
            True
        """

        # Format computed data values into colorized string
        total_inches = Fore.LIGHTMAGENTA_EX + f"{self.total_inches:.2f}" + Style.RESET_ALL
        total_months = Fore.LIGHTMAGENTA_EX + f"{self.total_months}" + Style.RESET_ALL
        avg = Fore.LIGHTMAGENTA_EX + f"{self.average():.2f}" + Style.RESET_ALL

        # Return a formatted string with computed results
        return (
            f"Number of months: {total_months}\n"
            f"Total inches of rainfall: {total_inches}\n"
            f"Average rainfall per month: {avg}"
        )
    # ------------------------------------------------------------

# ------------------------------------------------------------ end class RainfallAvgCalculator

# ------------------------------------------------------------ run_rainfall_avg_calculator()
def run_rainfall_avg_calculator() -> None:
    """Collect rainfall data across years and display aggregate statistics.

    Returns:
        None.

    Example:
        >>> run_rainfall_avg_calculator()
    """

    while True:
        # Prompt user to enter the number of years
        years = validate_prompt_positive_int("Enter the number of years to analyze: ")
        # if the positive value of years is equal to 0, it displays an error message
        if years == 0:
            print("years must be a nonzero positive integer")
        else:
            break # exits while loop, years is a nonzero positive integer
```

```
    # Create a RainfallAvgCalculator object to store rainfall inputted data based on the
      inputted num. of years
    rain_avg = RainfallAvgCalculator(years=years)

    # --- Assignment requirement, nested loops to collect data
    # and calculate the average rainfall over a period of years ---
    # Number of years loop
    for year_index in range(1, years + 1):
        # Months loop capture monthly rainfall input (12 months)
        print(f"\n---- Year {year_index} ----")
        for month_index in range(1, 13):
            # Format year and month values into colorize strings
            year_display = Fore.LIGHTMAGENTA_EX + f"{year_index}" + Style.RESET_ALL
            month_display = Fore.LIGHTMAGENTA_EX + f"{month_index}" + Style.RESET_ALL
            rainfall_prompt = (
                "Enter rainfall (in inches) for "
                f"Year {year_display}, Month {month_display}: "
            )
            # Prompt user to enter rainfall amount in inches for a given month in a given year
            # validate and capture rainfall input
            rainfall = validate_prompt_positive_float(rainfall_prompt)
            # Store each month's inputted data for a given year
            rain_avg.record(year_index - 1, month_index - 1, rainfall)

    # Computer average rainfall from the rainfall data and display results
    print()
    print(rain_avg.summary())
    wait_for_enter()
# ------------------------------------------------------ end run_rainfall_avg_calculator()

# ------------------------------------------------------------- run_rainfall_menu()
def run_rainfall_menu() -> None:
    """Render a Rainfall Average Calculator submenu

    Example:
        >>> run_rainfall_menu()
        ╔═══════════════════════════════════════════════╗
        ║          Rainfall Average Calculator          ║
        ╠═══════════════════════════════════════════════╣
        ║ 1. Enter rainfall and show summary            ║
        ║ 2. Back                                       ║
        ╚═══════════════════════════════════════════════╝

    Returns:
        None.
```

```
    """

    options = [
        "Enter rainfall and show summary",
        "Back",
    ]
    # Create a Menu object for the Rainfall Average Calculator menu
    menu_rainfall = Menu("Rainfall Average Calculator", options)
    # Render the menu and store it in a string variable to be displayed
    menu_rainfall_display = Fore.LIGHTCYAN_EX + menu_rainfall.render()

    print(menu_rainfall_display)

    # Menu loop
    while True:
        # Prompt the user for selection and captures it
        selection = validate_prompt_positive_int("Please enter your selection: ")
        print()
        match selection:
            case 1: # Launch the Rainfall Average Calculator, Part 1 of the assignment
                run_rainfall_avg_calculator()
                print()
                print(menu_rainfall_display)
            case 2:
                return # Ends run_rainfall_menu() - back to main menu
            case _:
                 # the input was not a recognized menu choice; guide the user
                print(
                    Fore.LIGHTRED_EX
                    + f"Invalid selection. Please enter {menu_rainfall._choice_index_list()}."
                )

# ------------------------------------------------------------ end run_rainfall_menu()

# ============================================================
# ||                                                        ||
# ||                Part 2: Bookstore Points                ||
# ||                                                        ||
# ============================================================

# ------------------------------------------------------------ run_bookstore_points()
def run_bookstore_points() -> None:
    """Prompt for books purchased and report the awarded points.

    Returns:
```

```
        None.

    Example:
        >>> # Requires interactive input; launch within a console session.
        >>> run_bookstore_points()
    """

    points = 0


    # Prompt user to enter the number of books purchased
    # validate and capture num. books input
    books = validate_prompt_positive_int("\nThe number of books: ")

    # -- Assignment requirement --
    # If a customer purchases 0 books, they earn 0 points
    # next tier is triggered when the customer purchases 2 books
    # so if the customer purchases 1 book, they also earn 0 points
    # The tiers are:
    # - Tier-1 = 0 to 1 books -> 0 points
    # - Tier-2 = 2 to 3 books -> 5 points
    # - Tier-3 = 4 to 5 books -> 15 points
    # - Tier-4 = 6 to 7 books -> 30 points
    # - Tier-5 = 8 or more books -> 60 points
    if books <=1: # Tier-1
        points = 0
    elif books <= 3: # Tier-2
        points = 5
    elif books <= 5: # Tier-3
        points = 15
    elif books <= 7: # Tier-4
        points = 30
    else:            # Tier-5
        points = 60

    # Format the number of books and points values into colorized strings
    books_str = Fore.LIGHTMAGENTA_EX + f"{books}" + Style.RESET_ALL
    points_str = Fore.LIGHTMAGENTA_EX + f"{points}" + Style.RESET_ALL

   # Display the num. of books and the related points earned
    print(f"\nBooks purchased: {books_str}")
    print(f"Points awarded: {points_str}")
    wait_for_enter()
# ------------------------------------------------------------- end run_bookstore_points()
```

```
# ------------------------------------------------------------ run_bookstore_menu()
def run_bookstore_menu() -> None:
    """Render a Bookstore Points submenu

    Example:
        >>> run_bookstore_menu()
        ╔═══════════════════════════════════════╗
        ║             Bookstore Points          ║
        ╠═══════════════════════════════════════╣
        ║ 1. Compute points for books purchased ║
        ║ 2. Back                               ║
        ╚═══════════════════════════════════════╝

    Returns:
        None.
    """

    options = [
        "Compute points for books purchased",
        "Back",
    ]
    # Create a Menu object for the Bookstore Points menu
    menu_bookstore = Menu("Bookstore Points", options)
    # Render the menu a store it in a string variable to be displayed
    menu_bookstore_display = Fore.LIGHTCYAN_EX + menu_bookstore.render()

    print(menu_bookstore_display)

    # Menu loop
    while True:
         # Prompt the user for selection and capture it
        selection = validate_prompt_positive_int("Please enter your selection: ")
        match selection:
            case 1: # Launch the Bookstore Points calculator, Part 2 of the assignment
                run_bookstore_points()
                print()
                print(menu_bookstore_display)
            case 2:
                return # Ends run_bookstore_menu() - back to main menu
            case _:
                 # the input was not a recognized menu choice; guide the user
                print(
                    Fore.LIGHTRED_EX
                    + f"Invalid selection. Please enter
{menu_bookstore._choice_index_list()}."
```

```
                )
# ------------------------------------------------------------ end run_bookstore_menu()

# ____________________________________________________________________________
# ------------- Main Function -------------
#____________________________________________________________________________
# ============================================================================
# Main Application Flow Functionality (program entry and user interaction)
# ============================================================================

# ------------------------------------------------------------ main()
def main() -> None:
    """Program entry point runs the program, displays the main menu

    Example:
        ╔═══════════════════════════════════════╗
        ║                Main Menu              ║
        ╠═══════════════════════════════════════╣
        ║ 1. Part 1: Rainfall Average Calculator ║
        ║ 2. Part 2: Bookstore Points           ║
        ║ 3. Exit                               ║
        ╚═══════════════════════════════════════╝

    Returns:
        None.
    """

    # Create the program banner
    main_banner = Banner([("Rainfall Average Calculator & Book Points")])
    # Render the banner and store it in a string variable to be displayed
    main_banner_display = Fore.LIGHTGREEN_EX + main_banner.render()

    options = [
        "Part 1: Rainfall Average Calculator",
        "Part 2: Bookstore Points",
        "Exit",
    ]

    # Create a Menu object for the Main Menu
    menu_main = Menu("Main Menu", options)
    # Render the menu a store it in a string variable to be displayed
    menu_main_display = Fore.YELLOW + menu_main.render()


    print(main_banner_display)
```

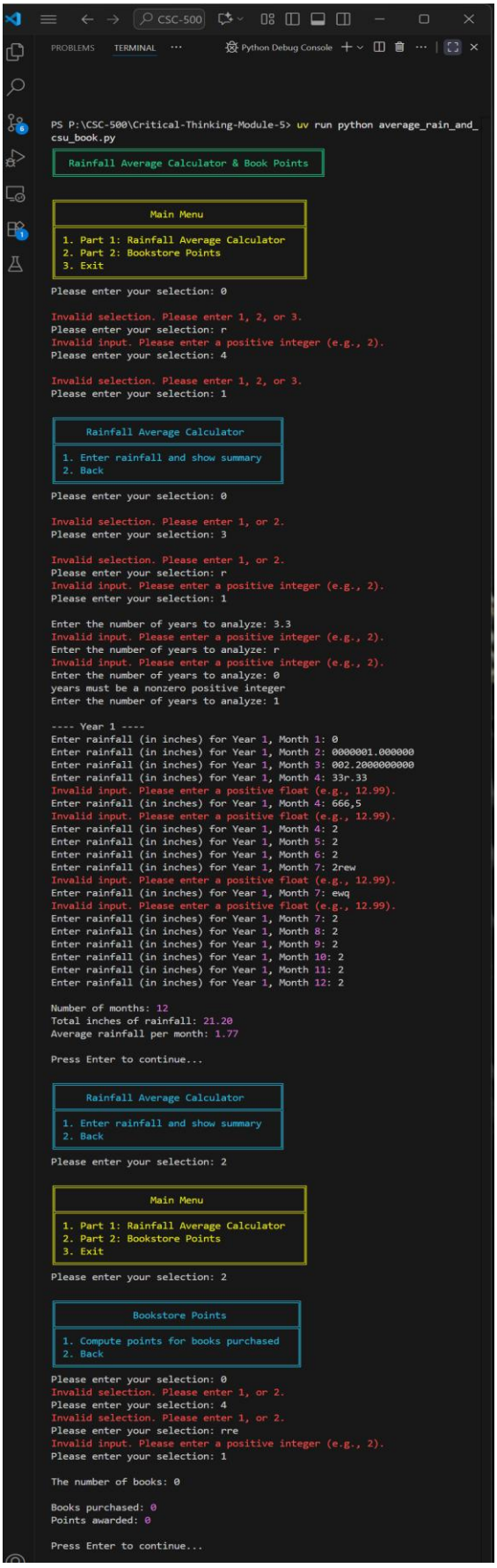
```
    print()
    print(menu_main_display)

    # Main menu loop
    while True:
        # Prompt the user for selection and capture it
        selection = validate_prompt_positive_int("Please enter your selection: ")
        print()
        match selection:
            # ============================================================
            # ||                Part 1: Rainfall Average Calculator      ||
            # ============================================================
            case 1: # Launch the Rainfall Average Calculator, Part 1 of the assignment
                run_rainfall_menu()
                print(menu_main_display)
            # ============================================================
            # ||                 Part 2: Bookstore Points                ||
            # ============================================================
            case 2: # Launch the Bookstore Points calculator, Part 2 of the assignment
                run_bookstore_menu()
                print(menu_main_display)
            case 3:
                if (validate_prompt_yes_or_no("Are you sure that you want to exist? ")):
                    print("\nBye! 👌\n")
                    return # Ends main() function - exits program
            case _:
                 # the input was not a recognized menu choice; guide the user
                print(
                    Fore.LIGHTRED_EX
                    + f"Invalid selection. Please enter {menu_main._choice_index_list()}."
                )
# ------------------------------------------------------------ end main()

# ____________________________________________________________________________
# Module Initialization / Main Execution Guard
# ----------------------------------------------------------------------------

# ------------------------------------------------------------ main_guard
if __name__ == "__main__":
    main()
# ---------------------------------------------------------------

# ____________________________________________________________________________
# End of File
#
```

Figure 2

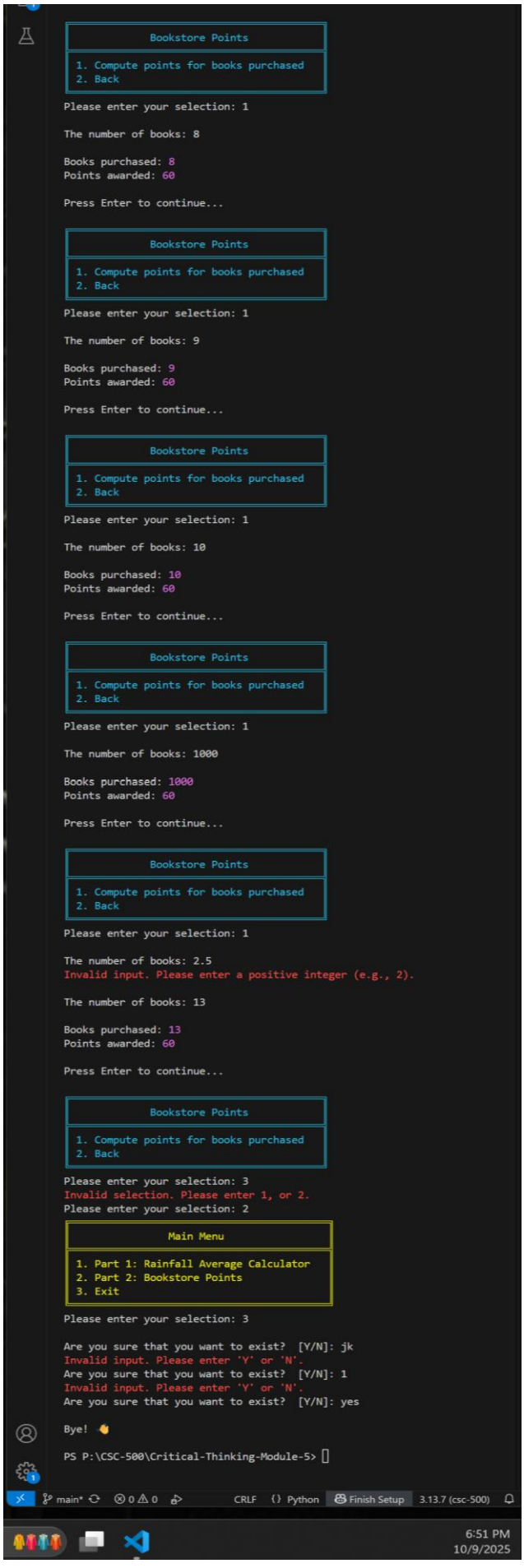
Project Outputs in the VS Code Terminal

Figure 3

Project Outputs Troubleshooting in the VS Code Terminal




Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 1
Books purchased: 1
Points awarded: 0
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 2
Books purchased: 2
Points awarded: 5
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 3
Books purchased: 3
Points awarded: 5
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 4
Books purchased: 4
Points awarded: 15
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 5
Books purchased: 5
Points awarded: 15
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 6
Books purchased: 6
Points awarded: 30
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 7
Books purchased: 7
Points awarded: 30
Press Enter to continue...


Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 8
Books purchased: 8
Points awarded: 60
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 9
Books purchased: 9
Points awarded: 60
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 10
Books purchased: 10
Points awarded: 60
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 1000
Books purchased: 1000
Points awarded: 60
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 1
The number of books: 2.5
Invalid input. Please enter a positive integer (e.g., 2).
The number of books: 13
Books purchased: 13
Points awarded: 60
Press Enter to continue...
Bookstore Points
1. Compute points for books purchased
2. Back
Please enter your selection: 3
Invalid selection. Please enter 1, or 2.
Please enter your selection: 2
Main Menu
1. Part 1: Rainfall Average Calculator
2. Part 2: Bookstore Points
3. Exit
Please enter your selection: 3
Are you sure that you want to exist? [Y/N]: jk
Invalid input. Please enter 'Y' or 'N'.
Are you sure that you want to exist? [Y/N]: 1
Invalid input. Please enter 'Y' or 'N'.
Are you sure that you want to exist? [Y/N]: yes
Bye!
PS P:\CSC-500\Critical-Thinking-Module-5>
main*
CRLF
Python
Finish Setup
3.13.7 (csc-500)
6:51 PM
10/9/2025

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

Code Snippet 2

User Input Validation Functions: validation_utilities.py

```
# ------------------------------------------------------------------------
# File: validation_utilities.py
# Author: Alexander Ricciardi
# Date: 2025-10-05
# ------------------------------------------------------------------------
# Course: CSS-500 Principles of Programming
# Professor: Dr. Brian Holbert
# Fall C-2025
# Sep.-Nov. 2025

# --- Module Functionality ---
# The file provides user input validation utility functions.
# ------------------------------------------------------------------------

# --- Functions ---
# - validate_prompt_yes_or_no(): request a yes/no confirmation, until a valid input is entered
# - wait_for_enter(): pause program until the user presses Enter
# - validate_prompt_int(): prompt user until a valid integer is entered
# - validate_prompt_positive_int(): prompt user until a valid integer is entered
# - validate_prompt_float(): prompt user until a valid float is entered
# - validate_prompt_positive_float(): prompt user until a valid float is entered
# - validate_prompt_string(): prompt user until a non-empty string is entered

# --- Imports ---
# - __future__.annotations to simplify forward typing references.
# - colorama (Fore, init) for cross-platform colored terminal
# ------------------------------------------------------------------------

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

"""
The file provides input validation utility functions.

Each function contains a validation loop for prompting user, capturing input,
and validating input (ints, floats, or strings). The loop will loop until a valid
input is entered, and then the function will validate the input.
"""
```

```python
# _________________
# Imports
#

from __future__ import annotations

from colorama import Fore, init
# Initialize colorama primarily to make it work on Windows
init(autoreset=True)

# ____________________________________________________________________________
#  Miscellaneous user-input validation 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(Fore.LIGHTRED_EX + "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...")
# ------------------------------------------------------------------

# ____________________________________________________________________________
# Integer validation 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 an integer:")
            Enter an integer:
            three
            Invalid input. Please enter an integer (e.g., 2).
            Enter the integer entered is: 2
            >>> value
            2
    """
    # Keep asking until a valid int is entered
```

```
    while True:
        raw = input(f"{prompt}").strip()
        try:
            return int(raw)
        except ValueError:
            # Invalid input error message
            print(Fore.LIGHTRED_EX + "Invalid input. Please enter an integer (e.g., 2).")
# ------------------------------------------------------------ end validate_prompt_int()

# ------------------------------------------------------- validate_prompt_positive_int()
def validate_prompt_positive_int(prompt: str) -> int:
    """Ask the user until a valid positive integer is entered.

    Args:
        prompt: text to display to the user

    Returns:
        The validated positive integer value

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

    Examples:
       user input shown after [prompts]:

            >>> value = validate_prompt_positive_int("Enter the item quantity:")
            Enter the item quantity:
            three
            Invalid input. Please enter a positive integer (e.g., 2).
            Enter the item quantity: 2
            >>> value
            2
    """
    # Keep asking until a valid positive int is entered
    while True:
        raw = input(f"{prompt}").strip()
        try:
            if int(raw) >= 0: # 0 is both + and -
                return int(raw)
            raise  ValueError
        except ValueError:
            # Invalid input error message
            print(Fore.LIGHTRED_EX + "Invalid input. Please enter a positive integer (e.g.,
                                     2).")
# ---------------------------------------------------------- end validate_prompt_positive_int()
```

```
# ____________________________________________________________________________
# Float validation functions
#

# ------------------------------------------------------------ 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 valid float is entered
    while True:
        raw = input(f"{prompt}").strip()
        try:
            return float(raw)
        except ValueError:
            # Invalid input error message
            print(Fore.LIGHTRED_EX + "Invalid input. Please enter a valid float (e.g.,
                  12.99).")
# ------------------------------------------------------------ end validate_prompt_float()

# ---------------------------------------------------- validate_prompt_positive_float()
def validate_prompt_positive_float(prompt: str) -> float:
    """Ask the user until a valid positive integer is entered.

    Args:
```

```
        prompt: text to display to the user

    Returns:
        The validated positive float value

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

    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 positive int is entered
    while True:
        raw = input(f"{prompt}").strip()
        try:
            if float(raw) >= 0.0: # 0.0 is both + and -
                return float(raw)
            raise  ValueError
        except ValueError:
            # Invalid input error message
            print(Fore.LIGHTRED_EX + "Invalid input. Please enter a positive float (e.g.,
                  12.99).")
# ------------------------------------------------- end validate_prompt_positive_float()

# ____________________________________________________________________________
# String validation functions
#

# ------------------------------------------------------------ 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}").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(Fore.LIGHTRED_EX + "Invalid input. Please enter a string (e.g., Hello).")
# ---------------------------------------------------------------- end validate_prompt_string()

# ____________________________________________________________________________
# End of File
#
```

See next page

Code Snippet 3

Banner and Menu Classes: validation_utilities.py

```
# ------------------------------------------------------------------------
# File: menu_banner_utilities.py
# Author: Alexander Ricciardi
# Date: 2025-10-05
# ------------------------------------------------------------------------
# Course: CSS-500 Principles of Programming
# Professor: Dr. Brian Holbert
# Fall C-2025
# Sep.-Nov. 2025

# --- Module Functionality ---
# Provides console UI classes that render bordered banners and numbered menus.
# ------------------------------------------------------------------------

# --- Classes ---
# - Banner: Creates an ASCII-styled boxes with alignment, dividers, and sizing functionality.
# - Menu: Creates a console menus using Banner.
# ------------------------------------------------------------------------

# --- Imports ---
# - __future__.annotations for postponed evaluation of type hints.
# - typing (Literal, Sequence, TypeAlias) to annotate alignment options and content.
# ------------------------------------------------------------------------

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

"""
The file provides a console banner and a menu console-based UI.

The Banner and Menu use ASCII formatting to display UI on the console.
The Banner renders banner-style titles, and the Menu class uses the Banner class
to render menus.
"""
# ________________
# Imports
#

from __future__ import annotations

from typing import Literal, Sequence, TypeAlias
```

```
# ____________________________________________________________________________
# Utility Classe 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: Sequence[object] = DEFAULT_CONTENT,
        inner_width: int = MINIUM_WIDTH,
    ) -> None:
        """Construct and initialize a new banner with default values if none are entered

        Args:
            text: text lines inside the banner
            alignment: Alignment of text lines ("left", "center", or "right")
            inner_width: the minimum inner width 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 isDividerr
                ╚═══════════════╝


        """
        # Initialize the banner lines to default content
        self._lines: list[object] = list(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
          inner_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]),inner_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()
    # Top border line for the banner
    def _top(self) -> str:
        """Build top banner border.

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

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

```
    # ------------------------------------------------------------ _divider()
    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()
    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 lines 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 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

# ____________________________________________________________________________
# Utility Classe Menu
#
# ============================================================================
# Menu Class Functionality (uses the Banner class)
# ============================================================================
# ---------------------------------------------------------------------- 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())
            ╔══════════════════════╗
```

```
        ╠═══════════════════════╣
        ╠═══════════════════════╣
        ╠═══════════════════════╣
        ╚═══════════════════════╝
"""
# ____________________
# 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 Variables 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)]
    # ------------------------------------------------------------

    # ------------------------------------------------------------- _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]}"
    # ------------------------------------------------------------

    # ------------------------------------------------------------- _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]}"
    # ------------------------------------------------------------

    # ------------------------------------------------------------ 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 class Menu
```