**Project Report:**

**Critical Thinking Module 7 – Course Information**

Alexander Ricciardi

Colorado State University Global

CSC500: Principles of Programming

Professor: Dr. Brian Holbert

October 26, 2025

**Project Report:**

**Critical Thinking Module 7 – Course Information**

This documentation is part of the Portfolio Milestone Module 7 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 Critical Thinking Module 7 – Course Information.

**The Assignment Direction:**

Creating Python Programs
Write a program that creates a dictionary containing course numbers and the room numbers of the rooms where the courses meet. The dictionary should have the following key-value pairs:

**Key-Value Pairs: Room Number**

| Course Number (key) | Room Number (value) |
|---|---|
| CSC101 | 3004 |
| CSC102 | 4501 |
| CSC103 | 6755 |
| NET110 | 1244 |
| COM241 | 1411 |

The program should also create a dictionary containing course numbers and the names of the instructors that teach each course. The dictionary should have the following key-value pairs:

**Key-Value Pairs: Instructors**

| Course Number (key) | Instructor (value) |
|---|---|
| CSC101 | Haynes |
| CSC102 | Alvarado |
| CSC103 | Rich |
| NET110 | Burke |
| COM241 | Lee |

The program should also create a dictionary containing course numbers and the meeting times of each course. The dictionary should have the following key-value pairs:

**Key-Value Pairs: Meeting Time**

| Course Number (key) | Meeting Time (value) |
|---|---|
| CSC101 | 8:00 a.m. |
| CSC102 | 9:00 a.m. |
| CSC103 | 10:00 a.m. |
| NET110 | 11:00 a.m. |
| COM241 | 1:00 p.m. |

The program should let the user enter a course number and then it should display the course's room number, instructor, and meeting time.

**Submission:**

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

**My Program Description:**

The program is a small terminal app that allows a user to view an 'university' course(s) information (course number, room, instructor, and time) by entering a course number. The course information (data) is stored in three dictionaries, and the course numbers are used as keys within the dictionaries.

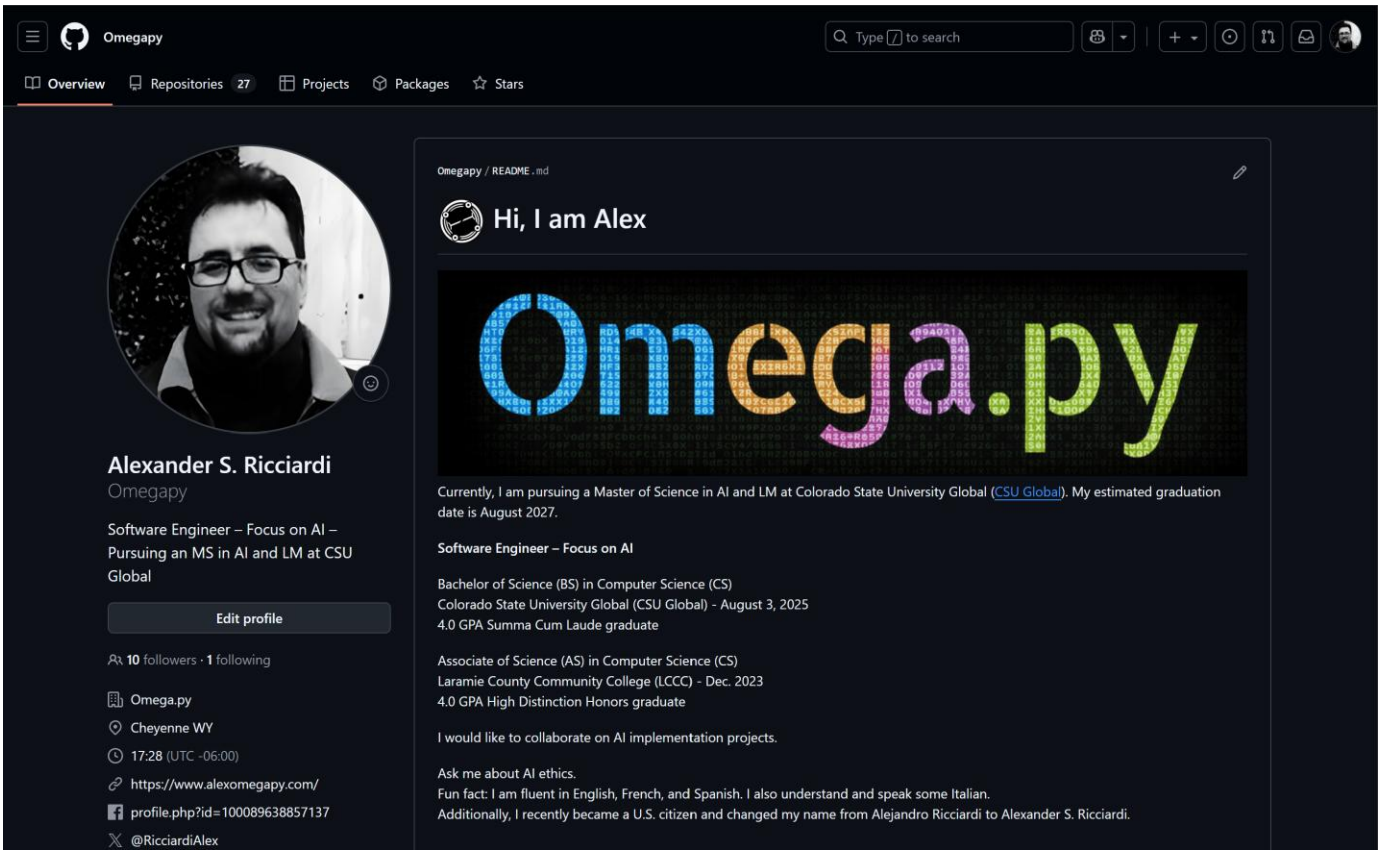
⚠️ My notes:
As I was using some of the same code lines for my critical assignments, I created several small utility functions and classes that I can reuse; they are found in the following Python files:

- 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, which can be found at the end of this document

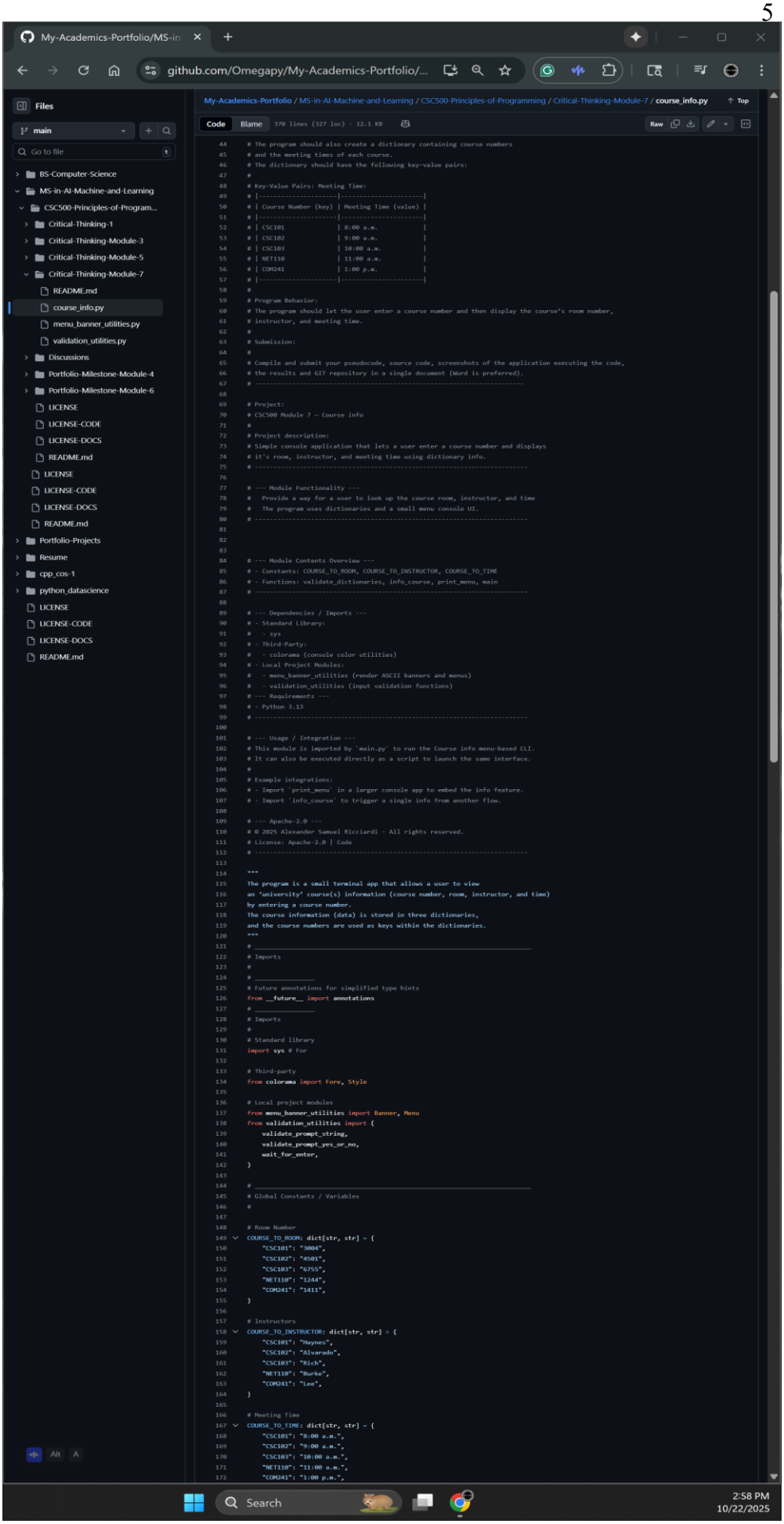
**Git Repository:**

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-7

**Figure-1**
*Code on GitHub*



**Code Snippet 1**

*Main Project Code: average_rain_and_csu_book.py*

```
# ------------------------------------------------------------------------
# File: course_info.py
# Project:
# Author: Alexander Ricciardi
# Date: 2025-10-26
# File Path: Critical-Thinking-Module-7/course_info.py
# ------------------------------------------------------------------------
# Course: CSS-500 Principles of Programming
# Professor: Dr. Brian Holbert
# Fall C-2025
# Sep-Nov 2025
# ------------------------------------------------------------------------
# Assignment:
# Write a program that creates a dictionary containing course numbers
# and the room numbers of the rooms where the courses meet.
# The dictionary should have the following key-value pairs:
#
# Key-Value Pairs: Room Number:
# |---------------------|---------------------|
# | Course Number (key) | Room Number (value) |
# |---------------------|---------------------|
# | CSC101              | 3004                |
# | CSC102              | 4501                |
# | CSC103              | 6755                |
# | NET110              | 1244                |
# | COM241              | 1411                |
# |---------------------|---------------------|
#
# The program should also create a dictionary containing course numbers
# and the names of the instructors that teach each course.
# The dictionary should have the following key-value pairs:
#
# Key-Value Pairs: Instructors:
# |---------------------|--------------------|
# | Course Number (key) | Instructor (value) |
# |---------------------|--------------------|
# | CSC101              | Haynes             |
# | CSC102              | Alvarado           |
# | CSC103              | Rich               |
# | NET110              | Burke              |
# | COM241              | Lee                |
# |---------------------|--------------------|
#
```

```
# The program should also create a dictionary containing course numbers
# and the meeting times of each course.
# The dictionary should have the following key-value pairs:
#
# Key-Value Pairs: Meeting Time:
# |---------------------|----------------------|
# | Course Number (key) | Meeting Time (value) |
# |---------------------|----------------------|
# | CSC101              | 8:00 a.m.            |
# | CSC102              | 9:00 a.m.            |
# | CSC103              | 10:00 a.m.           |
# | NET110              | 11:00 a.m.           |
# | COM241              | 1:00 p.m.            |
# |---------------------|----------------------|
#
# Program Behavior:
# The program should let the user enter a course number and then display the course's room
number,
# instructor, and meeting time.
#
# Submission:
#
# Compile and submit your pseudocode, source code, screenshots of the application executing
the code,
# the results and GIT repository in a single document (Word is preferred).
# ------------------------------------------------------------------------

# Project:
# CSC500 Module 7 – Course info
#
# Project description:
# Simple console application that lets a user enter a course number and displays
# it's room, instructor, and meeting time using dictionary info.
# ------------------------------------------------------------------------

# --- Module Functionality ---
#   Provide a way for a user to look up the course room, instructor, and time
#   The program uses dictionaries and a small menu console UI.
# ------------------------------------------------------------------------

# --- Module Contents Overview ---
# - Constants: COURSE_TO_ROOM, COURSE_TO_INSTRUCTOR, COURSE_TO_TIME
# - Functions: validate_dictionaries, info_course, print_menu, main
# ------------------------------------------------------------------------
```

```
# --- Dependencies / Imports ---
# - Standard Library:
#   - sys
# - Third-Party:
#   - colorama (console color utilities)
# - Local Project Modules:
#   - menu_banner_utilities (render ASCII banners and menus)
#   - validation_utilities (input validation functions)
# --- Requirements ---
# - Python 3.13
# ------------------------------------------------------------------------

# --- Usage / Integration ---
# This module is imported by `main.py` to run the Course info menu-based CLI.
# It can also be executed directly as a script to launch the same interface.
#
# Example integrations:
# - Import `print_menu` in a larger console app to embed the info feature.
# - Import `info_course` to trigger a single info from another flow.

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

"""
The program is a small terminal app that allows a user to view
an ‘university’ course(s) information (course number, room, instructor, and time)
by entering a course number.
The course information (data) is stored in three dictionaries,
and the course numbers are used as keys within the dictionaries.
"""
# ____________________________________________________________________________
# Imports
#
# __________________
# Future annotations for simplified type hints
from __future__ import annotations
# __________________
# Imports
#
# Standard library
import sys # For

# Third-party
```

```
from colorama import Fore, Style

# Local project modules
from menu_banner_utilities import Banner, Menu
from validation_utilities import (
    validate_prompt_string,
    validate_prompt_yes_or_no,
    wait_for_enter,
)

# ____________________________________________________________________________
# Global Constants / Variables
#

# Room Number
COURSE_TO_ROOM: dict[str, str] = {
    "CSC101": "3004",
    "CSC102": "4501",
    "CSC103": "6755",
    "NET110": "1244",
    "COM241": "1411",
}

# Instructors
COURSE_TO_INSTRUCTOR: dict[str, str] = {
    "CSC101": "Haynes",
    "CSC102": "Alvarado",
    "CSC103": "Rich",
    "NET110": "Burke",
    "COM241": "Lee",
}

# Meeting Time
COURSE_TO_TIME: dict[str, str] = {
    "CSC101": "8:00 a.m.",
    "CSC102": "9:00 a.m.",
    "CSC103": "10:00 a.m.",
    "NET110": "11:00 a.m.",
    "COM241": "1:00 p.m.",
}


# ____________________________________________________________________________
# Standalone Function Definitions
#
```

```
# ________________________
# Helper Functions
#
# ============================================================================
# Program logic function, user interface and query
# ============================================================================

# ---------------------------------------------------------------------------- info_course()
def info_course() -> None:
    """Prompt the user to enter a course number
       and display the course number and associated room, instructor, and time.

    Returns:
        None

    Examples:
        >>> info_course()
        Enter course number (e.g., CSC101):
        CSC101
        Course: CSC101
        Room: 3004
        Instructor: Haynes
        Time: 8:00 a.m.
    """
    # Prompt user and validate user a string
    user_input = validate_prompt_string("Enter course number (e.g.,
                                                        CSC101):\n").strip().upper()

    # Check if course exists
    if user_input not in COURSE_TO_ROOM:
        print(Fore.LIGHTRED_EX + "\nCourse not found." + Style.RESET_ALL)
        wait_for_enter()
        return # return None if the inputted course does not exist

    # get course info
    room = COURSE_TO_ROOM[user_input]
    instructor = COURSE_TO_INSTRUCTOR[user_input]
    time = COURSE_TO_TIME[user_input]

    # Create a Course Information banner instance
    results_banner = Banner(["Course Information"])
    print()
    # Render and display banner
    print(Fore.LIGHTCYAN_EX + results_banner.render())
```

```
    # Display course number using colors
    print(
        Fore.LIGHTYELLOW_EX
        + "\nCourse: "
        + Fore.LIGHTWHITE_EX
        + user_input
        + Style.RESET_ALL,
    )
    # Display course room number using colors
    print(
        Fore.LIGHTYELLOW_EX + "Room: " + Fore.LIGHTWHITE_EX + room + Style.RESET_ALL,
    )
    # Display course instructor last name using colors
    print(
        Fore.LIGHTYELLOW_EX
        + "Instructor: "
        + Fore.LIGHTWHITE_EX
        + instructor
        + Style.RESET_ALL,
    )
    # Display course meeting time using colors
    print(
        Fore.LIGHTYELLOW_EX + "Time: " + Fore.LIGHTWHITE_EX + time + Style.RESET_ALL,
    )

    wait_for_enter()
# ------------------------------------------------------------------------------

#---- Menu ----
# ------------------------------------------------------------------------ print_menu()
def print_menu() -> None:
    """Render the menu

    Menu options:
        l - Look Up Course: Execute course info
        q - Quit: Exit program with confirmation

    Returns:
        None

    Examples:
        >>> print_menu()  # doctest: +SKIP
        ╔═══════════════════════╗
        ║         MENU          ║
        ╠═══════════════════════╣
```

```
    ╟ l - Look Up Course   ╢
    ╟ q - Quit             ╢
    ╚════════════════════════╝

    Choose an option:
"""
# Create menu instance using the Menu class
menu = Menu(
    "MENU",
    [
        "Lookup Course Information",
        "Quit",
    ],
    prefixes=["l", "q"],
)
# Render the main menu and store it in a string variable to be displayed
# to be used in the program's main while loop
menu_display = Fore.LIGHTCYAN_EX + menu.render()

# Create info banner instance
info_banner = Banner(["COURSE INFORMATION"])
# Render the course info banner and store it in a string variable to be displayed
# to be used in the program's main while loop
info_banner_display = Fore.LIGHTCYAN_EX + info_banner.render()

# Main program Menu loop
while True:
    print()
    print(menu_display)

    # Prompt and capture the user for selection (1 or q)
    selection = input("\nChoose an option: ").strip().lower()

    match selection:
        # Launch the course info feature
        case "l":
            print()
            print(info_banner_display)
            print()
            # Launch course info query functionality
            info_course()
        # Launch the quit/existing program feature
        case "q":
            if validate_prompt_yes_or_no("Are you sure you want to exit?"):
                print("\nThank you for using Course info System!")
```

```
                    # Exit while loop/program
                    break

            case _:
                # Invalid selection input message
                print(
                    Fore.LIGHTRED_EX + "\nInvalid selection. Please enter 'l' or 'q'.",
                )
                wait_for_enter()

# ------------------------------------------------------------------------------

# _____________________
# Program Entrypoint
# Main function
# ------------------------------------------------------------------------------ main()
def main() -> None:
    """Run the course info program

    Program Flow:
        1. UTF-8 encoding (Windows compatibility)
        2. header banner
        4. menu loop (user can info courses or quit)
           program logic

    Examples:
        >>> main()
        ╔═══════════════════════════╗
        ╠   Course info System    ╠
        ╚═══════════════════════════╝
        ...menu loop...
            user interaction
            program logic
        Goodbye! 👌
    """
    # Configure UTF-8 encoding for console (Windows compatibility)
    sys.stdout.reconfigure(encoding="utf-8")

    # Display program header banner
    header = Banner(["Course info System"])
    print(Fore.LIGHTCYAN_EX + header.render())
    print()

    # Enter menu loop - contain program logic
    print_menu()
```

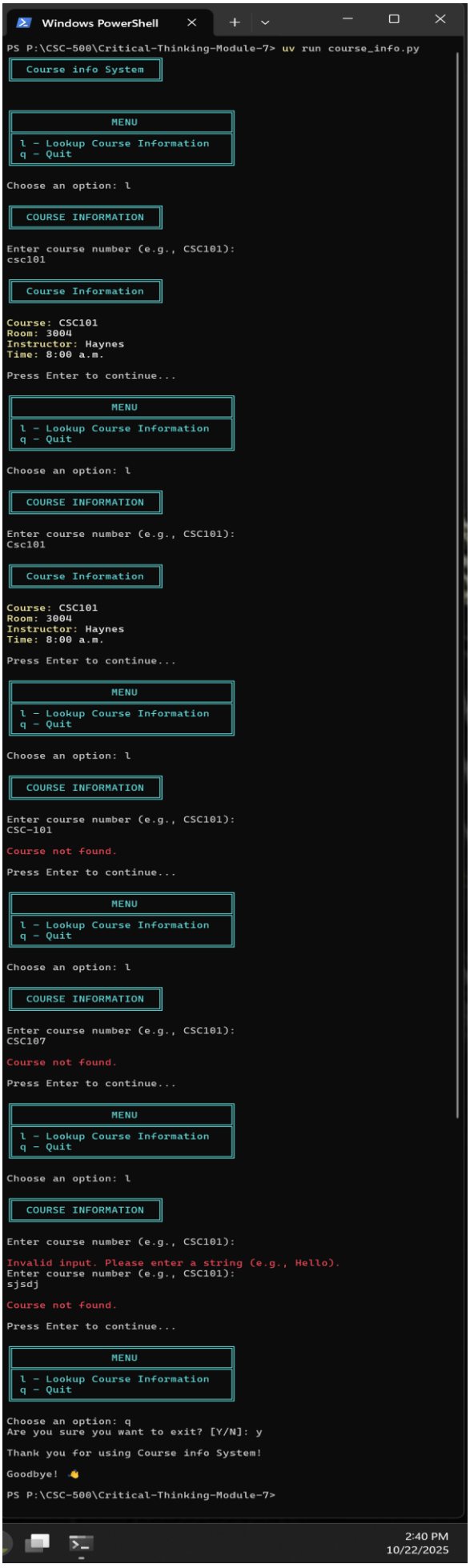
```
    # Exit message (after menu loop ends)
    print("\nGoodbye! 👋\n")
# ------------------------------------------------------------------------------

# ______________________________________________________________________________
# Module Initialization / Main Execution Guard (if applicable)
#
if __name__ == "__main__":
    main()
# ______________________________________________________________________________
# End of File
#
```

*See next page*

**Figure 2**
*Project Outputs on Terminal – User Flow*

**Figure 3**
*Project Outputs Troubleshooting*



All other input types of validation have been troubleshooted in previous assignments

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

*See the next,  for added functions and classes utilities.*

**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 positive integer is entered
# - validate_prompt_nonezer_positive_int(): prompt user until a valid nonezero positive
                                                                    integer is entered
# - validate_prompt_float(): prompt user until a valid float is entered
# - validate_prompt_positive_float(): prompt user until a valid positive float is entered
# - validate_prompt_nonezero_positive_float(): prompt user until a valid nonezero positive
                                                                        float is entered.
# - validate_prompt_string(): prompt user until a non-empty string is entered
# - validate_prompt_date(): prompt user until a valid date 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
```

```python
# ---------------------------------------------------------------------

"""
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.
"""

# _________________
# 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()

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

    Args:
        prompt: text to display to the user

    Returns:
        The validated nonezero positive integer value

    Behavior:
        - Re-prompts when the input cannot be validated as a nonezero 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:
                return int(raw)
            raise  ValueError
        except ValueError:
            # Invalid input error message
            print(Fore.LIGHTRED_EX + "Invalid input. Please enter a nonezero positive integer
                                                                        (e.g., 2).")
```

```
# ------------------------------------------- end validate_prompt_nonezero_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()

# ------------------------------------------------ validate_prompt_nonezero_positive_float()
def validate_prompt_nonezero_positive_float(prompt: str) -> float:
    """Ask the user until a valid nonezero positive float 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 nonzero positive 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 nonezero positive float is entered
    while True:
        raw = input(f"{prompt}").strip()
        try:
            if float(raw) > 0.0:
                return float(raw)
            raise  ValueError
        except ValueError:
            # Invalid input error message
            print(Fore.LIGHTRED_EX + "Invalid input. Please enter a nonezero positive float
                                                                                (e.g., 12.99).")
# ---------------------------------------- end validate_prompt_nonezero_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()

# ____________________________________________________________________________
# Date validation functions
#

# ------------------------------------------------------------ validate_prompt_date()
def validate_prompt_date(
    prompt: str,
    *,
    formats: list[str] | None = None,
    normalize_format: str = "%B %d, %Y"
) -> str:
    """Prompt user until valid date is entered.

    Args:
        prompt: Text to display to the user
        formats: List of accepted input formats -> (default: ["%B %d, %Y", "%m/%d/%Y", "%Y-%m-
                                                                                    %d"])
        normalize_format: Output format for the date string (default: "%B %d, %Y")

    Returns:
        Date string in normalize_format (e.g., "February 1, 2020")

    Examples:
```

```
        user input shown after [prompts]:

            >>> date = validate_prompt_date("Enter date:\\n")
            Enter date:
            2/1/2020
            >>> date
            'February 1, 2020'

            >>> date = validate_prompt_date("Enter date:\\n")
            Enter date:
            February 1, 2020
            >>> date
            'February 1, 2020'

            >>> date = validate_prompt_date("Enter date:\\n")
            Enter date:
            2020-02-01
            >>> date
            'February 1, 2020'

            >>> date = validate_prompt_date("Enter date:\\n")
            Enter date:
            invalid
            Invalid input. Please enter a valid date (e.g., February 1, 2020 or 2/1/2020 or
                                                                                2020-02-01).
            Enter date:
            2/1/2020
            >>> date
            'February 1, 2020'
    """
    # Default formats if none provided
    if formats is None:
        formats = ["%B %d, %Y", "%m/%d/%Y", "%Y-%m-%d"]

    # Build example string from formats
    examples = []
    try:
        example_date = datetime(2020, 2, 1)
        for fmt in formats[:3]:  # Show up to 3 examples
            examples.append(example_date.strftime(fmt))
    except Exception:
        examples = ["February 1, 2020", "2/1/2020", "2020-02-01"]

    example_str = " or ".join(examples)
```

```
    while True:
        raw = input(f"{prompt}").strip()

        # Try each format until one works
        date_obj = None
        for fmt in formats:
            try:
                date_obj = datetime.strptime(raw, fmt)
                break
            except ValueError:
                continue

        if date_obj is not None:
            # Successfully parsed - return normalized format
            # Use %-d for non-zero-padded day on Unix/Mac, %#d on Windows
            # But %B %d, %Y with manual formatting is more portable
            formatted = date_obj.strftime(normalize_format)
            # Remove leading zero from day if present (e.g., "February 01" -> "February 1")
            parts = formatted.split()
            if len(parts) >= 2 and parts[1].endswith(',') and parts[1][:-1].startswith('0'):
                parts[1] = parts[1][1:]  # Remove leading zero
            return ' '.join(parts)
        else:
            # No format matched - show error
            print(Fore.LIGHTRED_EX + f"Invalid input. Please enter a valid date (e.g.,
                                                                    {example_str}).")
# ------------------------------------------------------------ end validate_prompt_date()

# ____________________________________________________________________________
# End of File
#
```

*See next page for classes*

**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.
"""
```

```python
# _________________
# 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
```

```python
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 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
```