

Project Report:

Portfolio Milestone Module 6 – Online Shopping Cart V2

Alexander Ricciardi

Colorado State University Global

CSC500: Principles of Programming

Professor: Dr. Brian Holbert

October 19, 2025

Project Report:

Portfolio Milestone Module 6 – Online Shopping Cart V2

This documentation is part of the Portfolio Milestone Module 6 from CSC500: Principles of Programming at Colorado State University Global. This Project Report is an overview of the program's functionality and testing scenarios, including console output screenshots. The program is coded in Python 3.13, and it is called Portfolio Milestone Module 6 – Online Shopping Cart V2

The Assignment Direction:

Online Shopping Cart

Step 4: Build the ShoppingCart class with the following data attributes and related methods. Note: Some can be method stubs (empty methods) initially, to be completed in later steps

- Parameterized constructor, which takes the customer name and date as parameters
- Attributes
- customer_name (string) - Initialized in default constructor to "none"
- current_date (string) - Initialized in default constructor to "January 1, 2020"
- cart_items (list)
- Methods
- add_item()
 - Adds an item to cart_items list. Has parameter ItemToPurchase. Does not return anything.
- remove_item()
 - Removes item from cart_items list. Has a string (an item's name) parameter. Does not return anything.
 - If item name cannot be found, output this message: Item not found in cart. Nothing removed.
- modify_item()
 - Modifies an item's description, price, and/or quantity. Has parameter ItemToPurchase. Does not return anything.
 - If item can be found (by name) in cart, check if parameter has default values for description, price, and quantity. If not, modify item in cart.
 - If item cannot be found (by name) in cart, output this message: Item not found in cart. Nothing modified.
- get_num_items_in_cart()
 - Returns quantity of all items in cart. Has no parameters.
- get_cost_of_cart()
 - Determines and returns the total cost of items in cart. Has no parameters.
- print_total()
 - Outputs total of objects in cart.
 - If cart is empty, output this message: SHOPPING CART IS EMPTY
- print_descriptions()
 - Outputs each item's description.

Example of print_total() output:

```
John Doe's Shopping Cart - February 1, 2020
    Number of Items: 8
        Nike Romaleos 2 @ $189 = $378
        Chocolate Chips 5 @ $3 = $15
        Powerbeats 2 Headphones 1 @ $128 = $128
            Total: $521
```

Example of print_descriptions() output:

```
John Doe's Shopping Cart - February 1, 2020
    Item Descriptions
        Nike Romaleos: Volt color, Weightlifting shoes
        Chocolate Chips: Semi-sweet
        Powerbeats 2 Headphones: Bluetooth headphones
```

Step 5: In the main section of your code, implement the print_menu() function. print_menu() has a ShoppingCart parameter and outputs a menu of options to manipulate the shopping cart. Each option is represented by a single character. Build and output the menu within the function.

If an invalid character is entered, continue to prompt for a valid choice. *Hint:* Implement Quit before implementing other options. Call print_menu() in the main() function. Continue to execute the menu until the user enters q to Quit.

Example:

```
MENU
    a - Add item to cart
    r - Remove item from cart
    c - Change item quantity
    i - Output items' descriptions
    o - Output shopping cart
    q - Quit
Choose an option:
```

Step 6: Implement Output shopping cart menu option. Implement Output item's description menu option.

Example of shopping cart menu option:

```
OUTPUT SHOPPING CART
John Doe's Shopping Cart - February 1, 2020
    Number of Items: 8
        Nike Romaleos 2 @ $189 = $378
        Chocolate Chips 5 @ $3 = $15
        Powerbeats 2 Headphones 1 @ $128 = $128
            Total: $521
```

Example of item description menu option.

```
OUTPUT ITEMS' DESCRIPTIONS
John Doe's Shopping Cart - February 1, 2020
    Item Descriptions
        Nike Romaleos: Volt color, Weightlifting shoes
```

Chocolate Chips: Semi-sweet
Powerbeats 2 Headphones: Bluetooth headphones

Your program submission materials must include your source code and screenshots of the application executing the code and the results.

My Program Description:

The program is a small terminal app. It is an implementation of an online shopping cart. This version implements Steps 4-6 (ShoppingCart class and menu). It provides a menu for the shopping cart. Users can add, remove, and modify items through the menu.

⚠ 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/Portfolio-Milestone-Module-6>

Figure-1
Code on GitHub

```

My-Academics-Portfolio/MS-in-AI-Machine-and-Learning/CSC200-Principles-of-Programming/Portfolio-Milestone-Module-4/online_shopping_cart_v2.py

# This file contains the logic for the online shopping cart application.
# It reads items from a file and allows users to add items to their cart,
# view the total cost, and print a receipt.

# Import required modules
import os
import sys
import re
import json

# Define ShoppingCart class
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item):
        self.items.append(item)

    def calculate_total_cost(self):
        total_cost = sum(item['cost'] for item in self.items)
        return total_cost

    def print_receipt(self):
        print("Receipt:")
        print("Items in Cart:")
        for item in self.items:
            print(f"- {item['name']} x {item['quantity']} at ${item['cost']}")
        print(f"\nTotal Cost: ${self.calculate_total_cost()}\n")

# Define Item class
class Item:
    def __init__(self, name, quantity, cost):
        self.name = name
        self.quantity = quantity
        self.cost = cost

# Define ShoppingCartValidator class
class ShoppingCartValidator:
    def validate_items(self, items):
        # Implement validation logic here
        pass

# Define ShoppingCartProcessor class
class ShoppingCartProcessor:
    def process(self, items):
        # Implement processing logic here
        pass

# Main function
def main():
    # Read items from file
    with open('items.json') as f:
        items = json.load(f)

    # Create ShoppingCart object
    cart = ShoppingCart()

    # Add items to cart
    for item in items:
        cart.add_item(Item(item['name'], item['quantity'], item['cost']))

    # Calculate total cost
    total_cost = cart.calculate_total_cost()

    # Print receipt
    cart.print_receipt()

if __name__ == "__main__":
    main()

```

Code Snippet 1*Main Project Code: average_rain_and_csu_book.py*

```
#-----
# File: online_shopping_cart_v2.py
# Project:
# Author: Alexander Ricciardi
# Date: 2025-10-15
# File Path: Portfolio-Milestone-Module-6/online_shopping_cart_v2.py
# -----
# Course: CSS-500 Principles of Programming
# Professor: Dr. Brian Holbert
# Fall C-2025
# Sep-Nov 2025
# -----
# Assignment:
# Portfolio Milestone Module 6
#
# Directions:
# Online Shopping Cart - Continue from Portfolio Milestone Module 4
# that implemented steps 1 through 3, this module integrates steps 4 through 6
#
# Step 1: Build the ItemToPurchase class with the following specifications:
#   • Attributes
#   • item_name (string)
#   • item_price (float)
#   • item_quantity (int)
#   • Default constructor
#   • Initializes item's name = "none", item's price = 0, item's quantity = 0
#   • Method
#   • print_item_cost()
# Example of print_item_cost() output:
# Bottled Water 10 @ $1 = $10
#
# Step 2: In the main section of your code, prompt the user for two items
# and create two objects of the ItemToPurchase class.
# Example:
#   Item 1
#   Enter the item name:
#   Chocolate Chips
#   Enter the item price:
#   3
#   Enter the item quantity:
#   1
#
#   Item 2
```

```

# Enter the item name:
# Bottled Water
# Enter the item price:
# 1
# Enter the item quantity:
# 10
#
# Step 3: Add the costs of the two items together and output the total cost.
# Example:
# TOTAL COST
# Chocolate Chips 1 @ $3 = $3
# Bottled Water 10 @ $1 = $10
# Total: $13
#
# -----
# NOTE: Steps 2-3 are NOT included in this version's main() function.
# The program starts with an empty cart and uses the menu system (Steps 5-6) to add items.
# -----
#
# Step 4: Build the ShoppingCart class with the following data attributes and related methods.
# Note: Some can be method stubs (empty methods) initially, to be completed in later steps
#
# Parameterized constructor, which takes the customer name and date as parameters
# Attributes:
# - customer_name (string) - Initialized in default constructor to "none"
# - current_date (string) - Initialized in default constructor to "January 1, 2020"
# - cart_items (list)
#
# Methods:
# - add_item():
#   Adds an item to cart_items list.
#   Has parameter ItemToPurchase.
#   Does not return anything.
#
# - remove_item():
#   Removes item from cart_items list.
#   Has a string (an item's name) parameter.
#   Does not return anything.
#   If item name cannot be found, output this message:
#     Item not found in cart. Nothing removed.
# - modify_item():
#   Modifies an item's description, price, and/or quantity.
#   Has parameter ItemToPurchase. Does not return anything.
#   If item can be found (by name) in cart, check if parameter has default values for
description, price,

```

```

# and quantity. If not, modify item in cart.
# If item cannot be found (by name) in cart, output this message:
#     Item not found in cart. Nothing modified.
#
# - get_num_items_in_cart()
# Returns quantity of all items in cart.
# Has no parameters.
#
# - get_cost_of_cart()
# Determines and returns the total cost of items in cart.
# Has no parameters.
# - print_total()
# Outputs total of objects in cart.
# If cart is empty, output this message:
#     SHOPPING CART IS EMPTY
#
# - print_descriptions()
# Outputs each item's description.
#
# Example of print_total() output:
#     John Doe's Shopping Cart - February 1, 2020
#     Number of Items: 8
#     Nike Romaleos 2 @ $189 = $378
#     Chocolate Chips 5 @ $3 = $15
#     Powerbeats 2 Headphones 1 @ $128 = $128
#     Total: $521
#
# Example of print_descriptions() output:
#     John Doe's Shopping Cart - February 1, 2020
#     Item Descriptions
#     Nike Romaleos: Volt color, Weightlifting shoes
#     Chocolate Chips: Semi-sweet
#     Powerbeats 2 Headphones: Bluetooth headphones
#
# Step 5:
# In the main section of your code, implement
# - the print_menu() function.
#     print_menu() has a ShoppingCart parameter and outputs a menu of options to manipulate the
# shopping cart.
#     Each option is represented by a single character. Build and output the menu within the
# function.
#     If an invalid character is entered, continue to prompt for a valid selection. Hint:
#         Implement Quit before implementing other options.
#         Call print_menu() in the main() function.
#         Continue to execute the menu until the user enters q to Quit.

```

```

#
# Example:
#   MENU
#     a - Add item to cart
#     r - Remove item from cart
#     c - Change item quantity
#     i - Output items' descriptions
#     o - Output shopping cart
#     q - Quit
# Choose an option:
#
# Step 6: Implement Output shopping cart menu option.
# Implement Output item's description menu option.
#
# Example of shopping cart menu option:
#   OUTPUT SHOPPING CART
#   John Doe's Shopping Cart - February 1, 2020
#   Number of Items: 8
#   Nike Romaleos 2 @ $189 = $378
#   Chocolate Chips 5 @ $3 = $15
#   Powerbeats 2 Headphones 1 @ $128 = $128
#   Total: $521
#
# Example of item description menu option.
#   OUTPUT ITEMS' DESCRIPTIONS
#   John Doe's Shopping Cart - February 1, 2020
#   Item Descriptions
#   Nike Romaleos: Volt color, Weightlifting shoes
#   Chocolate Chips: Semi-sweet
#   Powerbeats 2 Headphones: Bluetooth headphones
# -----
# Project:
# Online Shopping Cart
#
# Project description:
# The program is a small console app. that implements the functionality of
# an online shopping cart.
# -----
#
# --- Module Contents Overview ---
# - Class: ItemToPurchase
# - Class: ShoppingCart
# - Function: prompt_for_item_info
# - Function: print_menu
# - Function: main

```

```

# -----
# --- Dependencies / Imports ---
# - Standard Library:
#   - dataclasses (data containers and auto-generated init/repr methods)
#   - copy (copies objects for menu/banner templates when needed)
# - 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
# ----

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

"""

Online Shopping Cart Version 2
The console online shopping cart program is contained in this file
implements the functionality of an online shopping cart,
it displays simple banners and a menu.
This module implements Steps 4-6 (ShoppingCart class and menu system).
It provides a menu for the shopping cart.
Users can add, remove, and modify items through the menu.
"""

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

# Standard library imports
from dataclasses import dataclass
from copy import deepcopy
# Third party library
from colorama import Fore, Style

# Import utilities from same directory
from menu_banner_utilities import Banner, Menu
from validation_utilities import (

```

```

    validate_prompt_string,
    validate_prompt_nonezero_positive_int,
    validate_prompt_nonezero_positive_float,
    validate_prompt_yes_or_no,
    validate_prompt_date,
    wait_for_enter,
)

# =====
# ||| Step 1: Build the ItemToPurchase class |||
# |||
# =====

#
# Classes Definitions
#
# =====

# ItemToPurchase Class Functionality (domain model for line items)
# -----
# ----- class ItemToPurchase

@dataclass
class ItemToPurchase:
    """Represent an item (to be purchased) managed by the shopping cart.

    Attributes:
        item_name: Name of the item
        item_price: Price per unit
        item_quantity: Quantity to purchase
        item_description: Description for display (Step 4 requirement)

    Examples:
        >>> item = ItemToPurchase(item_name="Apples", item_price=1.5, item_quantity=2)
        >>> item.item_name, item.item_price, item.item_quantity
        ('Apples', 1.5, 2)
    """

    # Initializes item variables
    item_name: str = "none"
    item_price: float = 0.0
    item_quantity: int = 0

    # Step 4 requirement (for print_descriptions)
    item_description: str = "none"

    # ----- format_currency()

```

```

# Used by print_item_cost()
@staticmethod
def format_currency(value: float) -> str:
    """Format a numeric value for currency-like display without excess zeros

Notes:
    Used by menu option o - Output shopping cart

Args:
    value: Raw numeric value coming from the backend or router layer.

Returns:
    Currency string trimmed to at most two decimals.
    Ensures CLI and router payloads stay aligned.

Examples:
    >>> ItemToPurchase.format_currency(2.0)
    '2'
    >>> ItemToPurchase.format_currency(2.5)
    '2.5'
    >>> ItemToPurchase.format_currency(2.75)
    '2.75'
    """
    # if the value is a whole number, it drops the decimal portion
    if value == int(value):
        return str(int(value))
    # else format to the two decimal places and remove trailing zeros
    return f"{value:.2f}".rstrip("0")
# -----



# ----- print_item_cost()
def print_item_cost(self, display_name: str | None = None) -> str:
    """Print and return the formatted cost line for this item.

Args:
    display_name: Optional capitalized name to display (if None, uses item_name as-is)

Returns:
    A line-item cost string (for example, "Apples 3 @ $1 = $3")

Examples:
    >>> ItemToPurchase(item_name="W", item_price=2.0,
                           item_quantity=3).print_item_cost()
    'W 3 @ $2 = $6'
    """

```

```

# Use provided display name or default to item_name
name_to_display = display_name if display_name is not None else self.item_name

# compute the item total based on price and quantity
total_cost = self.item_price * self.item_quantity
# Format item info into a string to be displayed
cost_statement = (
    Fore.LIGHTYELLOW_EX +
    f"{name_to_display.title()} {self.item_quantity} @ "
    f"${self.format_currency(self.item_price)} = "
    f"${self.format_currency(total_cost)}"
    + Style.RESET_ALL
)
# Echo the formatted cost to the console for immediate feedback.
print(cost_statement)
# Return the cost so it can be reused or used for testing
return cost_statement
# ----- end class ItemToPurchase

# ===== Step 4: Build the ShoppingCart class =====
# ||| Step 4: Build the ShoppingCart class |||
# ===== ShoppingCart Class Functionality =====
# ----- class ShoppingCart
class ShoppingCart:
    """Manage customer shopping cart items (ItemToPurchase instances).

    This class simulates an online shopping cart functionality

    Attributes:
        customer_name: Name of the customer that owns the cart.
        current_date: Date associated with the shopping session.
        cart_items: Mutable list of tracked items.
    """
    # _____
    # Constructor
    #
    # ----- __init__()

    # Use provided display name or default to item_name
    name_to_display = display_name if display_name is not None else self.item_name

    # compute the item total based on price and quantity
    total_cost = self.item_price * self.item_quantity
    # Format item info into a string to be displayed
    cost_statement = (
        Fore.LIGHTYELLOW_EX +
        f"{name_to_display.title()} {self.item_quantity} @ "
        f"${self.format_currency(self.item_price)} = "
        f"${self.format_currency(total_cost)}"
        + Style.RESET_ALL
    )
    # Echo the formatted cost to the console for immediate feedback.
    print(cost_statement)
    # Return the cost so it can be reused or used for testing
    return cost_statement
# ----- end class ShoppingCart

```

```

def __init__(
    self,
    customer_name: str = "none",
    current_date: str = "January 1, 2020"
) -> None:
    """Initialize cart with customer info.

Args:
    customer_name: customer name associated with the cart
    current_date: date associated with the cart
"""

    self.customer_name = customer_name
    self.current_date = current_date
    self.cart_items: list[ItemToPurchase] = []
# -----


# _____
# Getters
#


# ----- get_num_items_in_cart()
def get_num_items_in_cart(self) -> int:
    """Return total quantity of all items.

Returns:
    Combined quantity count across every tracked item.

Examples:
    >>> cart = ShoppingCart("User", "February 1, 2020")
    >>> cart.add_item(
        ...     ItemToPurchase(item_name="Tomatos",
                           item_description = "fruit",
                           item_quantity=2,
                           item_price=1.5
                           )
        ...
    )
    >>> cart.get_num_items_in_cart()
    2
"""

    return sum(item.item_quantity for item in self.cart_items)
# -----


# ----- get_cost_of_cart()
def get_cost_of_cart(self) -> float:
    """Return total cost of all items.

```

Note:

Used by menu option a - Add item to cart

Returns:

Total cost of all the items in the cart

Examples:

```
>>> cart = ShoppingCart("User", "February 1, 2020")
>>> cart.add_item(
...     ItemToPurchase(item_name="Widget", item_price=2.5, item_quantity=2),
... )
>>> cart.get_cost_of_cart()
5.0
"""
return sum(item.item_price * item.item_quantity for item in self.cart_items)
# -----
```

```
# _____
# Setters
#
# ----- add_item()
def add_item(self, item: ItemToPurchase) -> None:
    """Add item to cart.
```

Notes:

Used by menu option a - Add item to cart

Args:

item: populated item object to add

Examples:

```
>>> cart = ShoppingCart("User", "February 1, 2020")
>>> cart.add_item(
...     ItemToPurchase(item_name="Orange", item_price=5.0, item_quantity=1),
... )
"""
self.cart_items.append(item)
# -----
```

```
# ----- modify_item()
def modify_item(self, item: ItemToPurchase) -> None:
    """Modify an existing item by name, updating only non-default fields.
```

```

Note:
    Used by menu option c - Change item quantity

Args:
    item: Item containing updated values from the caller.

Returns:
    None. Updates occur in place so the API routers and CLI menu share the same state.

Notes:
    Default values to check:
        item_description: "none"
        item_price: 0.0
        item_quantity: 0

    Error message: "Item not found in cart. Nothing modified."
"""

# Format customer name and date into a colorized string
print(Fore.LIGHTYELLOW_EX + f"\n{self.customer_name.title()}'s Shopping Cart - {self.current_date}")

# Loops through all the items present in the cart
# to find an item with a name matching the function parameter name value
# if match is found, modify the item's values from the parameter item object
for cart_item in self.cart_items:
    # Compare names if names match, modify values
    if cart_item.item_name == item.item_name:
        # update description in cart item if the given item description is NOT None
        if item.item_description != "none":
            cart_item.item_description = item.item_description
        # update price in cart item if the given item price is NOT equal to 0
        if item.item_price != 0.0:
            cart_item.item_price = item.item_price
        # update quantity in cart item if the given item quantity is NOT equal to 0
        # to remove item from cart use the ShoppingCart.remove_item()
        if item.item_quantity != 0:
            # Replace old quantity with new one
            cart_item.item_quantity = item.item_quantity
            # Confirm message
            print(Fore.YELLOW + f"\nThe {cart_item.item_name.title()}'s quantity is now: {cart_item.item_quantity}")
    return # exit functions

# Error message
print(Fore.YELLOW + "\nItem not found in cart. Nothing modified.")

```

```

# -----
# ----- remove_item()

def remove_item(self, item_name: str) -> None:
    """Remove item by name and report when the lookup fails.

Args:
    item_name: Name of the item that needs to be removed

Notes:
    Error message: "Item not found in cart. Nothing removed."
"""

# Format customer name and date into a colorized string
print(Fore.LIGHTYELLOW_EX + f"\n{self.customer_name.title()}'s Shopping Cart -\n{self.current_date}")

# Loops through all the items present in the cart
# to find an item with a name matching the function parameter item name value
# if match is found, removes items from cart
# matching the function parameter item name value
for i, item in enumerate(self.cart_items):
    # Check if it is the item that needs to be removed
    if item.item_name == item_name.lower():
        # Store item to be removed in a temp var.
        # to be used by confirmation message
        temp_item = deepcopy(item)
        # Remove item from cart
        self.cart_items.pop(i)
        # Check quantity of the item to be removed
        # to format the confirmation message
        if (temp_item.item_quantity == 1):
            print(Fore.YELLOW + f"\n1 {temp_item.item_name.title()} was removed from\nthe cart")
        return # exit function
    # else
    # Confirm message plural
    print(Fore.YELLOW + f"\n{temp_item.item_quantity}\n{temp_item.item_name.title()}s were removed from cart")
    return # exit function

# Error message
print(Fore.YELLOW + "\nItem not found in cart. Nothing removed.")

# -----
# =====

```

```

# ||
# ||      Step 6: Implement Output item's description menu option      ||
# ||
# =====

# _____
# Printers
#
# ----- print_total()
def print_total(self) -> None:
    """Print cart summary or "SHOPPING CART IS EMPTY" (Step 4).

Notes:
    Used by menu option o - Output shopping cart

Format:
    John Doe's Shopping Cart - February 1, 2020
    Number of Items: 8
    Nike Romaleos 2 @ $189 = $378
    Chocolate Chips 5 @ $3 = $15
    Total: $521

Examples:
    >>> cart = ShoppingCart("User", "February 1, 2020")
    >>> cart.add_item(
    ...     ItemToPurchase(item_name="Widget", item_price=5.0, item_quantity=2),
    ... )
    >>> cart.print_total()
    User Shopping Cart - February 1, 2020
    Number of Items: 2
    Widget 2 @ $5 = $10

    Total: $10
    """
    # Format customer name and date into a colorized string
    print(Fore.LIGHTYELLOW_EX + f"\n{self.customer_name.title()}'s Shopping Cart - "
                                  f"{self.current_date}")

    ---- Assignment requirement ----
    # Check if cart is empty
    if not self.cart_items:
        print(Fore.YELLOW + "\nSHOPPING CART IS EMPTY")

```

```

    return # Exit method

# ---- Number of items in cart
# Get and format total with colors
num_items = Fore.LIGHTYELLOW_EX + f"{self.get_num_items_in_cart()}" + Style.RESET_ALL
print(f"Number of Items: {num_items}")
# Print each item total cost, quantity, description
for item in self.cart_items:
    item.print_item_cost(item.item_name)
# Format cart total into colorized string
total = Fore.LIGHTGREEN_EX+
        f"${ItemToPurchase.format_currency(self.get_cost_of_cart())}" + Style.RESET_ALL
print(f"Total: {total}")

# -----
# ----- print_descriptions()
def print_descriptions(self) -> None:
    """Print item descriptions required in Step 4.

```

Notes:

Used by menu option a i - Output items' descriptions

Returns:

None. Output is directed to stdout just like router-backed renderers

Format (with items):

```

John Doe's Shopping Cart - February 1, 2020
Item Descriptions
Nike Romaleos: Volt color, Weightlifting shoes
Chocolate Chips: Semi-sweet

```

Format

```

John Doe's Shopping Cart - February 1, 2020
Item Descriptions

```

Examples:

```

>>> cart = ShoppingCart("User", "February 1, 2020")
>>> cart.add_item(
...     ItemToPurchase(
...         item_name="Widget",
...         item_description="Blue gadget",
...         item_price=5.0,
...         item_quantity=2,
...     )

```

```

    ... )
>>> cart.print_descriptions()
User Shopping Cart - February 1, 2020
Item Descriptions
Widget: Blue gadget
"""

# Format customer name and date into colorized string
print(Fore.LIGHTYELLOW_EX + f"\n{self.customer_name.title()}'s Shopping Cart - {self.current_date}")

#-- Get   and display description
print("Item Descriptions")
# Loops through all the items present in the cart
# and print each item name and description
for item in self.cart_items:
    # Get item name and description and format with colors
    item_desc = Fore.LIGHTWHITE_EX + item.item_description + Style.RESET_ALL
    print(f"{item.item_name.title()}: {item_desc}")
# ----- print_items_with_quantity()

# ----- print_items_with_quantity()
def print_items_with_quantity(self) -> None:
    """Print all items in cart with their names and quantities.

Format (with items):
    Apples: 3
    Bananas: 2
    Orange Juice: 1

Examples:
    >>> cart = ShoppingCart("User", "February 1, 2020")
    >>> cart.add_item(
        ...     ItemToPurchase(item_name="apples", item_quantity=3, item_price=1.5)
        ... )
    >>> cart.add_item(
        ...     ItemToPurchase(item_name="bananas", item_quantity=2, item_price=0.5)
        ... )
    >>> cart.print_items_with_quantity()
    Apples: 3
    Bananas: 2
"""

# Format customer name and date into a colorized string
print(Fore.LIGHTYELLOW_EX + f"\n{self.customer_name.title()}'s Shopping Cart - {self.current_date}")

```

```

{self.current_date}")

# Loop through all items in cart and print name with quantity
for item in self.cart_items:
    item_name = Fore.LIGHTWHITE_EX + item.item_name.title() + Style.RESET_ALL
    item_qty = Fore.LIGHTYELLOW_EX + str(item.item_quantity) + Style.RESET_ALL
    print(f"{item_name}'s quantity: {item_qty}")

# ----- end class ShoppingCart

#
# -----
# Helper Functions
#
# =====
# Item Prompt Function
# =====
# ----- prompt_for_item_info()

def prompt_for_item_info() -> ItemToPurchase:
    """Collects item info

    Returns:
        ItemToPurchase: populated item object

    Notes:
        Used by menu option a - Add item to cart

    Examples:
        >>> item = prompt_for_item_info()
        >>> item.item_name # doctest: +SKIP
        'Widget'
        """
        # Prompt user to enter the item info.
        name = validate_prompt_string("Enter the item name:\n").lower()

        description = validate_prompt_string("Enter the item description:\n")
        price = validate_prompt_nonzero_positive_float("Enter the item price:\n")
        quantity = validate_prompt_nonzero_positive_int("Enter the item quantity:\n")

        # Create an item with input info and returns it
        return ItemToPurchase(
            item_name=name,
            item_description=description,
            item_price=price,
            item_quantity=quantity,

```

```

        )
# ----- end prompt_for_item_info()

# =====
# ||           Step 5-6: Implement print_menu function
# ||           ||
# ||           ||
# =====
# =====
# Menu Function
# =====
# ----- print_menu()

def print_menu(cart: ShoppingCart) -> None:
    """Display Online Shopping Cart menu (Steps 5-6).

    Args:
        cart: ShoppingCart object

    Menu options:
        a - Add item to cart.
        r - Remove item from cart (shows empty message if cart is empty)
        c - Change item quantity (shows empty message if cart is empty)
        i - Output items' descriptions (shows empty message if cart is empty)
        o - Output shopping cart (shows empty message if cart is empty)
        q - Quit.

    """
    # Create menu with letter-based options using the Menu class
    menu = Menu(
        "MENU",
        [
            "Add item to cart",
            "Remove item from cart",
            "Change item quantity",
            "Output items' descriptions",
            "Output shopping cart",
            "Quit",
        ],
        prefixes=["a", "r", "c", "i", "o", "q"],
    )
    # Render the menu and store it in a string variable to be displayed
    menu_display = Fore.LIGHTCYAN_EX + menu.render()

    # Create a Add Itme banner instance
    add_item_banner = Banner(["Add Itme"])

```

```

# Render the description banner and store it in a string variable to be displayed
add_item_banner_display = Fore.LIGHTCYAN_EX + add_item_banner.render()

desc_banner = Banner(["OUTPUT ITEMS' DESCRIPTIONS"])
# Render the description banner and store it in a string variable to be displayed
desc_banner_display = Fore.LIGHTCYAN_EX + desc_banner.render()

# Create a cart output banner instance
cart_output_banner = Banner(["OUTPUT SHOPPING CART"])
# Render the cart output banner and store it in a string variable to be displayed
cart_output_banner_display = Fore.LIGHTCYAN_EX + cart_output_banner.render()

# Create a Remove item banner instance
remove_banner = Banner(["REMOVE ITEM"])
# Render the Remove banner and store it in a string variable to be displayed
remove_banner_display = Fore.LIGHTCYAN_EX + remove_banner.render()

# Create a Modify Quantity item banner instance
mod_quantity_banner = Banner(["MODIFY QUANTITY"])
# Render the Remove banner and store it in a string variable to be displayed
mod_quantity_banner_display = Fore.LIGHTCYAN_EX + mod_quantity_banner.render()

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

    # Prompt the user for selection and captures it
    selection = input("\nChoose an option: ").strip().lower()

    match selection:
        case 'a': # Launch the add item feature
            print()
            print(add_item_banner_display)
            print()
            # Creat a n item instance and prompt user for item info
            new_item = prompt_for_item_info()
            # Add the new populated item to cart
            cart.add_item(new_item)
            # Confirmation message
            print(Fore.LIGHTGREEN_EX + f"\n{new_item.item_quantity}"
            "\n{new_item.item_name.title()} added to cart.")
            wait_for_enter()

        case 'r': # Launch the remove item feature

```

```

print()
print(remove_banner_display)

# Check if cart is empty
if not cart.cart_items:
    # Format customer name and date into a colorized string
    print(Fore.LIGHTYELLOW_EX + f"\n{cart.customer_name.title()}'s Shopping
          Cart - {cart.current_date}")
    print(Fore.YELLOW + "\nSHOPPING CART IS EMPTY")
else:
    # Display items in cart descriptions
    cart.print_descriptions()
    print()
    # Prompt user for the name of the item to remove
    name = validate_prompt_string("Enter the item name to remove:\n").lower()
    # Remove item from cart
    cart.remove_item(name)

wait_for_enter()

case 'c': # Launch the item change quantity feature
print()
print(mod_quantity_banner_display)

# Check if cart is empty
if not cart.cart_items:
    # Format customer name and date into a colorized string
    print(Fore.LIGHTYELLOW_EX + f"\n{cart.customer_name.title()}'s Shopping
          Cart - {cart.current_date}")
    print(Fore.YELLOW + "\nSHOPPING CART IS EMPTY")
else:
    # Display the items in cart name and quantity
    cart.print_items_with_quantity()
    # Prompt user for the name of the item to change quantity
    name = validate_prompt_string("\nEnter the item name:\n").lower()
    new_qty = validate_prompt_nonzero_positive_int("Enter the new quantity:
                                                 ")
    # Create item with only name and quantity set (others default)
    temp_item = ItemToPurchase(item_name=name, item_quantity=new_qty)
    # Modify item
    cart.modify_item(temp_item)

wait_for_enter()

case 'i': # Launch the item output description feature

```

```

print()
print(desc_banner_display)

# Check if cart is empty
if not cart.cart_items:
    # Format customer name and date into a colorized string
    print(Fore.LIGHTYELLOW_EX + f"\n{cart.customer_name.title()}'s Shopping
          Cart - {cart.current_date}")
    print(Fore.YELLOW + "\nSHOPPING CART IS EMPTY")
else:
    # Output the items' descriptions present in cart
    cart.print_descriptions()

wait_for_enter()

case 'o': # Launch the item Output cart feature
print()
print(cart_output_banner_display)

# # Check if cart is empty
# if not cart.cart_items:
#     # Format customer name and date into a colorized string
#     print(Fore.LIGHTYELLOW_EX + f"\n{cart.customer_name.title()}'s Shopping
#           Cart - {self.current_date}")
#     print(Fore.YELLOW + "\nSHOPPING CART IS EMPTY")
# else:

# Built-in check for empty cart
# Display the formated cart info and total
cart.print_total()

wait_for_enter()

case 'q': # Launch the item quit feature
if (validate_prompt_yes_or_no("Are you sure that you want to exist? ")):
    print("\nThank you for shopping!")
    # Exits while loop
    break

case _:
    # Invalid input
    print(Fore.LIGHTRED_EX + "\nInvalid selection, please enter a, r, c, i, o, or
          q.")
    wait_for_enter()

# ----- end print_menu()

```

```
# _____  
# ----- Main Function -----  
#  
# ======  
# Main Application Flow Functionality (program entry and user interaction)  
# ======  
# ----- main()  
def main() -> None:  
    """Run the shopping cart program (Steps 4-6).  
  
    Program:  
        1. Display header.  
        2. Prompt for customer name and date  
        3. Create ShoppingCart (empty)  
        4. Enter menu loop (Steps 5-6)  
        5. Display exit message  
  
    Examples:  
        >>> main()  
  
        """  
  
        # Display program header  
        header = Banner(["Online Shopping Cart"])  
        print(Fore.LIGHTCYAN_EX + header.render())  
        print()  
  
        # _____  
        # Part 4  
        #  
        # Prompt for customer info  
        customer_name = validate_prompt_string("Enter customer's name:\n")  
        current_date = validate_prompt_date("Enter today's date:\n")  
  
        # Create shopping cart (empty - no items added initially)  
        cart = ShoppingCart(customer_name, current_date)  
  
        # _____  
        # Part 5-6  
        #  
        # The print-menu() function implements  
        # the main functionality of the Online Shopping Cart  
        print_menu(cart)
```

```
# Exit program message
print("\nBye! 🌟\n")

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

See next page

Figure 2
Project Outputs on Terminal – User Flow

```

PS P:\CSC-500\Portfolio-Milestone-Module-6> uv run online_shopping_cart_v2.py
[Online Shopping Cart]

Enter customer's name:
Alex Ricciardi
Enter today's date:
10/16/2025

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: a
[ Add Item ]

Enter the item name:
Ceramic Coffee Mug
Enter the item description:
12 oz stoneware mug with heat-resistant handle
Enter the item price:
12.55
Enter the item quantity:
2
2 "Ceramic Coffee Mug" added to cart.
Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: a
[ Add Item ]

Enter the item name:
Bluetooth Speaker
Enter the item description:
Portable speaker with 10-hour battery life
Enter the item price:
39.99
Enter the item quantity:
1
1 "Bluetooth Speaker" added to cart.
Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: a
[ Add Item ]

Enter the item name:
Notebook Set
Enter the item description:
Pack of three A5 dot-grid notebooks
Enter the item price:
8.75
Enter the item quantity:
4
4 "Notebook Set" added to cart.
Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: i
[ OUTPUT ITEMS' DESCRIPTIONS ]

Alex Ricciardi's Shopping Cart - October 16, 2025
Item Descriptions
Ceramic Coffee Mug: 12 oz stoneware mug with heat-resistant handle
Bluetooth Speaker: Portable speaker with 10-hour battery life
Notebook Set: Pack of three A5 dot-grid notebooks

Press Enter to continue...

```

```

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: o

OUTPUT SHOPPING CART

Alex Ricciardi's Shopping Cart - October 16, 2025
Number of Items: 7
Ceramic Coffee Mug 2 @ $12.55 = $25.1
Bluetooth Speaker 1 @ $39.99 = $39.99
Notebook Set 4 @ $8.75 = $35.0
Total: $100.09

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: r

REMOVE ITEM

Alex Ricciardi's Shopping Cart - October 16, 2025
Item Descriptions
Ceramic Coffee Mug: 12 oz stoneware mug with heat-resistant handle
Bluetooth Speaker: Portable speaker with 10-hour battery life
Notebook Set: Pack of three A5 dot-grid notebooks

Enter the item name to remove:
Ceramic Coffee Mug

Alex Ricciardi's Shopping Cart - October 16, 2025
2 Ceramic Coffee Mugs were removed from cart

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: c

MODIFY QUANTITY

Alex Ricciardi's Shopping Cart - October 16, 2025
Bluetooth Speaker's quantity: 1
Notebook Set's quantity: 4

Enter the item name:
Notebook Set
Enter the new quantity: 2

Alex Ricciardi's Shopping Cart - October 16, 2025
The Notebook Set's quantity is now: 2

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: o

OUTPUT SHOPPING CART

Alex Ricciardi's Shopping Cart - October 16, 2025
Number of Items: 3
Bluetooth Speaker 1 @ $39.99 = $39.99
Notebook Set 2 @ $8.75 = $17.5
Total: $57.49

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: q
Are you sure that you want to exist? [Y/N]: y

Thank you for shopping!
Bye! 🍀

PS P:\CSC-580\Portfolio-Milestone-Module-6>

```

Figure 3

Project Outputs Troubleshooting: The Cart is Empty, Data Type, and Numeric Nonzero Type

```

Windows PowerShell
PS P:\CSC-580\Portfolio-Milestone-Module-6> uv run online_shopping_cart_v2.py
[Online Shopping Cart]

Enter customer's name:
Alex Ricciardi
Enter today's date:
10/16/0
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).
Enter today's date:
25/10/2025
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).
Enter today's date:
09/33/2025
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).
Enter today's date:
November, 2025
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).
Enter today's date:
Hello 12, 2025
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).
Enter today's date:
November, 07 2025
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).
Enter today's date:
November, 07, 2025
Invalid input. Please enter a valid date (e.g., February 01, 2020 or 02/01/2020 or 2020-02-01).

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: r

[ REMOVE ITEM ]

Alex Ricciardi's Shopping Cart - November 7, 2025
SHOPPING CART IS EMPTY

Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: c

[ MODIFY QUANTITY ]

Alex Ricciardi's Shopping Cart - November 7, 2025
SHOPPING CART IS EMPTY

Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: i

[ OUTPUT ITEMS' DESCRIPTIONS ]

Alex Ricciardi's Shopping Cart - November 7, 2025
SHOPPING CART IS EMPTY

Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: o

[ OUTPUT SHOPPING CART ]

Alex Ricciardi's Shopping Cart - November 7, 2025
SHOPPING CART IS EMPTY

Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: 1

Invalid selection, please enter a, r, c, i, o, or q.

Press Enter to continue...

[ MENU ]
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: z

Invalid selection, please enter a, r, c, i, o, or q.

Press Enter to continue...

```

```

    MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: a
Add Item

Enter the item name:
Ceramic Coffee Mug
Enter the item description:
12 oz stoneware mug with heat-resistant handle
Enter the item price:
0.0
Invalid input. Please enter a nonzero positive float (e.g., 12.99).
Enter the item price:
-5
Invalid input. Please enter a nonzero positive float (e.g., 12.99).
Enter the item price:
y
Invalid input. Please enter a nonzero positive float (e.g., 12.99).
Enter the item price:
12.99
Enter the item quantity:
0
Invalid input. Please enter a nonzero positive integer (e.g., 2).
Enter the item quantity:
-5
Invalid input. Please enter a nonzero positive integer (e.g., 2).
Enter the item quantity:
5.9
Invalid input. Please enter a nonzero positive integer (e.g., 2).
Enter the item quantity:
u
Invalid input. Please enter a nonzero positive integer (e.g., 2).
Enter the item quantity:
4

4 "Ceramic Coffee Mug" added to cart.

Press Enter to continue...
    MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: a
Add Item

Enter the item name:
Bluetooth Speaker
Enter the item description:
Portable speaker with 10-hour battery life
Enter the item price:
39.99
Enter the item quantity:
3

3 "Bluetooth Speaker" added to cart.

Press Enter to continue...
    MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: r
REMOVE ITEM

Alex Ricciardi's Shopping Cart - November 7, 2025
Item Descriptions
Ceramic Coffee Mug: 12 oz stoneware mug with heat-resistant handle
Bluetooth Speaker: Portable speaker with 10-hour battery life

Enter the item name to remove:
Notebook Set

Alex Ricciardi's Shopping Cart - November 7, 2025
Item not found in cart. Nothing removed.

Press Enter to continue...
    MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: r
REMOVE ITEM

Alex Ricciardi's Shopping Cart - November 7, 2025
Item Descriptions
Ceramic Coffee Mug: 12 oz stoneware mug with heat-resistant handle
Bluetooth Speaker: Portable speaker with 10-hour battery life

Enter the item name to remove:
Ceramic Coffee Mug

Alex Ricciardi's Shopping Cart - November 7, 2025
4 Ceramic Coffee Mugs were removed from cart

Press Enter to continue...

```

```

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: c

MODIFY QUANTITY

Alex Ricciardi's Shopping Cart - November 7, 2025
Bluetooth Speaker's quantity: 3

Enter the item name:
Ceramic Coffee Mug
Enter the new quantity: 6

Alex Ricciardi's Shopping Cart - November 7, 2025
Item not found in cart. Nothing modified.

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: c

MODIFY QUANTITY

Alex Ricciardi's Shopping Cart - November 7, 2025
Bluetooth Speaker's quantity: 3

Enter the item name:
Bluetooth Speaker
Enter the new quantity: 9

Alex Ricciardi's Shopping Cart - November 7, 2025
The Bluetooth Speaker's quantity is now: 9

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: o

OUTPUT SHOPPING CART

Alex Ricciardi's Shopping Cart - November 7, 2025
Number of Items: 9
Bluetooth Speaker 9 @ $39.39 = $354.51
Total: $354.51

Press Enter to continue...

MENU
a - Add item to cart
r - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option: q
Are you sure that you want to exist? [Y/N]: y

Thank you for shopping!
Bye! 🌟
PS P:\CSC-500\Portfolio-Milestone-Module-6>

3:05 PM
10/16/2025

```

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 pages 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_nonzero_positive_int(): prompt user until a valid nonzero 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_nonzero_positive_float(): prompt user until a valid nonzero 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
# -----


"""
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 nonzero positive integer is entered.

Args:
    prompt: text to display to the user

Returns:
    The validated nonzero positive integer value

Behavior:
    - Re-prompts when the input cannot be validated as a nonzero 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 nonzero 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_nonozero_positive_float()
def validate_prompt_nonozero_positive_float(prompt: str) -> float:
    """Ask the user until a valid nonzero 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 nonzero 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 nonzero positive float
(e.g., 12.99).")
# ----- end validate_prompt_nonzero_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.
"""

# _____
# 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:



"""

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
MINIMUM_WIDTH: int = 10

#
# -----
# Constructor
#
# -----
# ----- __init__()
def __init__(


```

```

    self,
    content: Sequence[object] = DEFAULT_CONTENT,
    inner_width: int = MINIMUM_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())
        =====
    """

```

```

    ||      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

```