**The Development Document**

**Course : Introduction to Programming**

**Tutor : Ms. Fakhra**

**By Adeena kalim**

Table of Contents

[**Specification:** 3](#_Toc188373972)

[**System Flowchart :** 4](#_Toc188373973)

[**Illustration** 4](#_Toc188373974)

[**Technical Description & Walkthrough** 5](#_Toc188373975)

[**Critical Reflection** 7](#_Toc188373976)

[**Appendix** 8](#_Toc188373977)

# **Specification:**

The goal of this project is to develop and run a Python software that simulates the operation of a vending machine. This program will have an easy-to-use menu interface, support financial transactions, check the availability of selected items, dispense those products, and calculate the change to be returned to the user. It will include extensive error handling tools to handle erroneous inputs and the ability to restock items as needed. The vending machine's design allows for a smooth user experience while also keeping efficient operating operations. The whole code repository, including the application and its documentation, is available at this GitHub link: [<https://github.com/Adeenaklm/Vending-Machine>].

\

## **System Flowchart:**

## Illustration

The flowchart represents the vending machine's logical operation. It begins with the display of available items and prices. When a user picks an item, the system checks stock availability and conducts the payment. Upon successful completion, the machine dispenses the item and inquires if the user wishes to purchase additional items. Clear decision-making paths ensure a smooth and intuitive user experience (Lucidchart, 2025). Following successful completion, the machine dispenses the item and asks whether the user wants to buy more.

# **Technical Description & Walkthrough**

Walkthrough Link: [Vending Machine - Video recording](https://drive.google.com/file/d/1-wsJnx-t0jeKZ56AiMvp1cz701O3b4pv/view?usp=drive_link)

# **Overview**

This Python application uses a straightforward text-based interface to mimic how a vending machine might operate. Viewing the available things grouped into "Cold Drinks," "Snacks," and "Healthy Options," making choices, processing payments, getting change, and viewing item suggestions are all made possible by it. The vending machine manages the inventory of items, verifies that user input is accurate, and offers an interactive user experience.

**Core Functionality**

**Initialization and item management**

* Predefined item categories, each including a collection of goods, are used to initialize the vending machine.
* Every item has a name, price, stock level, and unique code ensuring easy access to the item details.
* With categories serving as the top-level keys, the objects are organized in a dictionary fashion.

def \_\_init\_\_(self):

self.items = {

"Cold Drinks": {

"C1": {"name": "Coffee", "price": 1.50, "stock": 8},

**Interactive user interface**

* **Displaying the menu(display\_menu method)**

The application outputs the item code, name, price, and stock availability for every item that is offered, arranged by category. It provides the consumer with an easy-to-use interface for browsing possibilities.

* **Item selection(prompt\_user\_for\_selection method)**

The user is prompted to enter an item code (such as "C1", "S1", or "H1") using the prompt\_user\_for\_selection method. It determines whether the input matches a legitimate item code. The function makes the selection if the item is in stock; if not, it asks the user once again. It asks the user to try again if they input an incorrect code.

* **Payment and change(prompt\_for\_payment method)**

The application manages payment input, verifies that the user has supplied enough money, and computes any amount of change that is owed. The application responds with an error message if a value that is not a number is entered.

**Stock Management and Suggestions**

* **Stock update**

To prevent overselling, the stock is updated whenever an item is bought. Following a successful purchase, this update takes place in the run method.

item["stock"] -= 1

* **Item suggestions(suggest\_items method)**

After a purchase, the algorithm suggests up to two additional things from the machine to promote continued use based on available stock. This increases user engagement with the computer by assisting users in exploring more stuff they might find interesting. (Real Python, 2025).

**Techniques used**

**Minimizing code duplication**

To cut down on repetitious logic in the code, the application makes good use of procedures like get\_item(), prompt\_user\_for\_selection(), and prompt\_for\_payment(). Every time an item lookup is required, the get\_item() method eliminates duplication by centralizing the logic for fetching an item.

def get\_item(self, code):

for category in self.items.values():

if code in category:

return category[code]

return None

**Effective use of functions that pass and return values**

Specific activities are carried out by functions like calculate\_change() and suggest\_items(), which return results that are utilized by other software components. Functionality becomes more modular as a result of this decoupling. Calculate\_change() returnvalue is used to guarantee correct transactions and report the precise amount of change.

def calculate\_change(self, payment, item\_price):

return round(payment - item\_price, 2)

**Object oriented programming (OOP techniques)**

The Vending Machine class contains the vending machine functionality, which is encapsulated in the application using Object-Oriented Programming (OOP) concepts.

In addition to maintaining the item inventory, the class has methods for user interaction, payment processing, stock updating, and item suggestion.

By grouping comparable functionalities together, OOP makes it easy to expand the application by adding new goods, categories, or features (like payment methods).

**Insights into the development process**

The vending machine software was developed utilizing a methodical methodology that concentrated on:

* Creating the Item Database: Dictionary-based item storage was selected due to its ease of use and adaptability, which makes it possible to easily change and retrieve item data.
* Ensuring User-Friendliness: The user is guided through the procedure by clear prompts and error handling in the interface, which was meant to be straightforward.
* Reducing Code Duplication: To keep the code DRY (Don't Repeat Yourself), a number of routine tasks, such item selection and payment processing, were divided into reusable functions.
* Modular Design: The code is simple to comprehend, maintain, and expand since functions were created to carry out certain tasks

# **Critical Reflection**

In conclusion, the vending machine software provides a strong basis, demonstrating important programming ideas like OOP, modular design, and error handling. Although the application is well-structured and works effectively, there is still much space for improvement. Some ideas include adding a GUI for a more engaging user experience or integrating a database for ongoing stock management. I could enhance the vending machine application and take it to a higher level of usefulness by developing my abilities in database administration, GUI design, and interacting with external APIs.

# **Appendix**

# Initializing the vending machine with item categories, their codes, names, prices, and stock

def \_\_init\_\_(self):

self.items = {

"Cold Drinks": {

"C1": {"name": "Coffee", "price": 1.50, "stock": 8},

"C2": {"name": "Tea", "price": 1.00, "stock": 10},

},

"Snacks": {

"S1": {"name": "Chips", "price": 1.20, "stock": 5},

"S2": {"name": "Chocolate", "price": 1.80, "stock": 7},

},

"Healthy Options": {

"H1": {"name": "Granola Bar", "price": 2.50, "stock": 4},

"H2": {"name": "SunBites Olive & Oregano", "price": 3.50, "stock": 9},

}

}

def display\_menu(self):

print("Welcome to the Ultimate Vending Machine Experience!")

print("Where your cravings meet instant satisfaction!")

print("\nHere's what we have for you today:")

for category, items in self.items.items():

print(f"\n♦ {category}:")

for code, details in items.items():

print(

f" {code}: {details['name']} - AED {details['price']:.2f} "

f"({details['stock']} in stock)"

)

def get\_item(self, code):

for category in self.items.values():

if code in category:

return category[code]

return None

def prompt\_user\_for\_selection(self):

while True:

user\_input = input("\nEnter the item code (e.g., C1, S1, H1): ").upper()

item = self.get\_item(user\_input)

if item:

if item["stock"] > 0:

return user\_input, item

else:

print(f"Sorry, {item['name']} is out of stock.")

else:

print("Invalid code. Please try again.")

def prompt\_for\_payment(self, item\_price):

while True:

try:

payment = float(input(f"The price is AED {item\_price:.2f}. Enter payment: AED "))

if payment >= item\_price:

return payment

else:

print("Insufficient payment. Please enter at least the item price.")

except ValueError:

print("Invalid input. Please enter a numeric value.")

def calculate\_change(self, payment, item\_price):

return round(payment - item\_price, 2)

def suggest\_items(self, purchased\_item\_code):

suggestions = []

for category, items in self.items.items():

for code, details in items.items():

if code != purchased\_item\_code and details["stock"] > 0:

suggestions.append(details["name"])

return suggestions[:2] # Suggest up to 2 items.

def run(self):

while True:

self.display\_menu()

selected\_code, item = self.prompt\_user\_for\_selection()

# Prompt for payment

payment = self.prompt\_for\_payment(item["price"])

change = self.calculate\_change(payment, item["price"])

print(f"\nDispensing {item['name']}... Enjoy!")

# Update stock

item["stock"] -= 1

# Provide change if applicable

if change > 0:

print(f"Returning AED {change:.2f} in change.")

# Suggest additional items

suggestions = self.suggest\_items(selected\_code)

if suggestions:

print(f"\nHow about trying: {', '.join(suggestions)}?")

# Check if the user wants to buy more

continue\_choice = input("\nWould you like to purchase another item? (yes/no): ").lower()

if continue\_choice != "yes":

print("Thanks for using the Ultimate Vending Machine! See you soon!")

break

if \_\_name\_\_ == "\_\_main\_\_":

vending\_machine = VendingMachine()

vending\_machine.run()

Copy of code

<https://zzzcode.ai/python/code-generator?id=8aa4e41d-be11-4e9e-b46b-e73d11127501>

Screenshots

for category, items in self.items.items():

print(f"\n♦ {category}:")

for code, details in items.items():

print(

f" {code}: {details['name']} - AED {details['price']:.2f} "

f"({details['stock']} in stock)"

)

def get\_item(self, code):

"""Fetches item details by code."""

for category in self.items.values():

if code in category:

return category[code]

return None

def prompt\_user\_for\_selection(self):

"""Prompts user to select an item and validates the selection."""

while True:

user\_input = input("\nEnter the item code (e.g., C1, C2, C3 | S1, S2, S3| H1, H2, H3 : ").upper()

if user\_input == "EXIT":

return None, None

item = self.get\_item(user\_input)

if item:

if item["stock"] > 0:

print(f"You selected: {item['name']} ✓")

return user\_input, item

else:

print(f" Sorry, {item['name']} is out of stock. Please choose another item.")

else:

print("❌ Invalid code. Please try again.")

def prompt\_for\_payment(self, item\_price):

"""Handles payment process and ensures sufficient payment is made."""

while True:

try:

payment = float(input(f"The price is AED {item\_price:.2f}. Enter payment: AED "))

if payment >= item\_price:

return payment

else:

print("Insufficient payment. Please enter at least the item price.")

except ValueError:

print("❌ Invalid input. Please enter a numeric value.")

def calculate\_change(self, payment, item\_price):

"""Calculates change to be returned."""

return round(payment - item\_price, 2)

def suggest\_items(self, purchased\_item\_code):

"""Suggests additional items, excluding the purchased item."""

suggestions = []

for category, items in self.items.items():

for code, details in items.items():

if code != purchased\_item\_code and details["stock"] > 0:

suggestions.append(details["name"])

return suggestions[:2] # Suggest up to 2 items

def run(self):

"""Main method to execute the vending machine process."""

while True:

try:

                payment = float(input(f"The price is AED {item\_price:.2f}. Enter payment: AED "))

                if payment >= item\_price:

                    return payment

                else:

                    print("Insufficient payment. Please enter at least the item price.")

            except ValueError:

                print("❌ Invalid input. Please enter a numeric value.")

    def calculate\_change(self, payment, item\_price):

        """Calculates change to be returned."""

        return round(payment - item\_price, 2)

    def suggest\_items(self, purchased\_item\_code):

        """Suggests additional items, excluding the purchased item."""

        suggestions = []

        for category, items in self.items.items():

            for code, details in items.items():

                if code != purchased\_item\_code and details["stock"] > 0:

                    suggestions.append(details["name"])

        return suggestions[:2]  # Suggest up to 2 items

    def run(self):

        """Main method to execute the vending machine process."""

        while True:

            self.display\_menu()  # Display available items

            # Prompt user for item selection

            selected\_code, item = self.prompt\_user\_for\_selection()

            if not selected\_code:

                print("\nThank you for using the Vending Machine. Goodbye!")

                break

            # Handle payment

            payment = self.prompt\_for\_payment(item["price"])

            # Calculate and return change

            change = self.calculate\_change(payment, item["price"])

            print(f"\n ✔ Dispensing {item['name']}... Enjoy!")

            if change > 0:

                print(f"↩ Returning AED {change:f} in change.")

            # Update stock

            item["stock"] -= 1

            # Suggest additional items

            suggestions = self.suggest\_items(selected\_code)

            if suggestions:

                print(f"\n How about trying: {', or  '.join(suggestions)}?")

            # Ask if the user wants to purchase another item

            continue\_choice = input("\nWould you like to purchase another item ? (yes/no): ").lower()

            if continue\_choice != "yes":

                print("\nThank you for using the Vending Machine. Goodbye!")

                break

if \_\_name\_\_ == "\_\_main\_\_":

    vending\_machine = VendingMachine()

    vending\_machine.run()

**References**

Programiz. (2025). *Object-Oriented Programming in Python*. Available at: <https://www.programiz.com/python-programming/object-oriented-programming> [Accessed 17 Jan. 2025].

Here’s the formatted citation for the source you provided:

Real Python. (2025). *Python Input Validation*. Available at: <https://realpython.com/python-input/> [Accessed 19 Jan. 2025].

Stack Overflow. (2025). *Python: How to Manage Inventory*. Available at: <https://stackoverflow.com/questions/4328129/python-how-to-manage-inventory> [Accessed 19 Jan. 2025].

Lucidchart. (2025). *How to Create Flowcharts for System Workflows*. Available at: <https://www.lucidchart.com/pages/flowchart> [Accessed 12 Jan. 2025].