**GIT Repository**

<https://github.com/yanngilbert-one/CSC500-1/tree/main/Assignment%20Week%204>

**Pseudocode**

[main.py](http://main.py)

BEGIN main

SET item\_list

SET next\_item\_input = yes

WHILE next\_item\_input = yes

item = resources.get\_user\_input

item\_list.append(item)

GET next\_item\_input

DISPLAY items\_in\_list

DISPLAY total\_cost

END main

[resources.py](http://resources.py)

FUNCTION get\_user\_item

SET valid\_input = False

GET item\_name

WHILE NOT valid\_input

TRY

GET item\_price

IF item\_price < 0

DISPLAY error\_message

ELSE

valid\_input = True

break

CATCH

DISPLAY error\_message

SET valid\_input = False

WHILE NOT valid\_input

TRY

GET item\_quantity

IF item\_quantity<0

DISPLAY error\_message

ELSE

valid\_input = True

break

CATCH

DISPLAY error\_message

RETURN ItemToPurchase(item\_name, item\_price, item\_quantity)

item\_to\_purchase.py

CLASS ItemToPurchase

FUNCTION \_\_init\_\_(self, item\_name = “none”, item\_price = 0, item\_quantity = 0)

self.item\_name = item\_name

self.item\_price = item\_price

self.item\_quantity = item\_quantity

FUNCTION print\_item\_cost(self)

CALCULATE total\_cost

DISPLAY item\_name, item\_quantity, item\_price, total-cost

RETURN total\_cost

**Source Code**

[main.py](http://main.py)

import resources

#initialize item list array as well as validation for the loop handling if user would like to continue or stop inputing items

item\_list = []

next\_item\_input = "yes"

#loop to build item list array

while next\_item\_input.lower() == "yes":

#get user item data from user

item = resources.get\_user\_item()

#add item to list of items

item\_list.append(item)

#ask user if continuing to input items

next\_item\_input = input("Would you like to add another item? (yes/no): ")

#print total cost of cart

print("\n--- Items in your list: ---")

total\_cost = sum(item.print\_item\_cost() for item in item\_list)

print(f"\nTotal cost of all ({len(item\_list)}) items in cart: ${total\_cost:.2f}")

[resources.py](http://resources.py)

#import ItemToPurchase class

from item\_to\_purchase import ItemToPurchase

#function to get and validate all item data from user

def get\_user\_item():

#input validation variable

valid\_input = False

#get item name, no validation needed here. In a real scenario we would put a character limit of some sort here and security features like SQL injection protection

item\_name = input("Enter the item name: ")

#loop until data input is valid

while not valid\_input:

try:

#converting input to float

item\_price = float(input("Enter the item price: "))

#if item price is less than 0, it must be negative and therefor not acceptable input

if item\_price < 0:

print("Price cannot be negative, please enter a positive value for the price.")

else:

#validate input and exit loop

valid\_input = True

break

#if value conversion to float is not valid, throw error and display error message

except ValueError:

print("Invalid value. Please enter a positive numerical value.")

#input validation variable

valid\_input = False

#loop until data input is valid

while not valid\_input:

try:

#converting input to int

item\_quantity = int(input("Enter the item quantity: "))

#if item quantity is less than 0, it must be negative and therefor not acceptable input

if item\_quantity < 0:

print("Quantity cannot be negative, please enter a positive value for the quantity.")

else:

#validate input and exit loop

valid\_input = True

break

#if value conversion to int is not valid, throw error and display error message

except ValueError:

print("Invalid value. Please enter a positive integer without any decimals, quantity cannot be a decimal value, you cannot buy a third of an item!")

#return item with its attributes

return ItemToPurchase(item\_name, item\_price, item\_quantity)

item\_to\_purchase.py

#initialize class

class ItemToPurchase:

#constructor

def \_\_init\_\_(self, item\_name = "none", item\_price = 0, item\_quantity = 0):

self.item\_name = item\_name

self.item\_price = item\_price

self.item\_quantity = item\_quantity

#function to print item cost using all attributes

def print\_item\_cost(self):

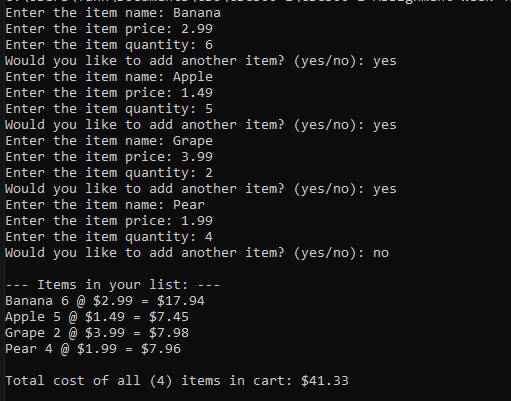
total\_cost = self.item\_price \* self.item\_quantity

print(f"{self.item\_name} {self.item\_quantity} @ ${self.item\_price:.2f} = ${total\_cost:.2f}")

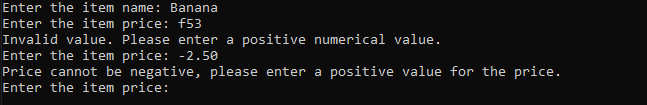
return total\_cost

**Test Cases (Screenshots)**

Case 1: Success



Case 2: Invalid price (negative value or not a numeral)



Case 3: Invalid quantity (negative value or not an integer)

