PSEUDOCODE

IMPORT PERSON CLASS MODULE

IMPORT CUSTOMER CLASS MODULE

IMPORT SHOPPINGCART CLAS MODULE

IMPORT CUSTOM MODULE(UTILITIES) MODULE

Beginning of Program

MAIN FUNCTION START

Declare dictionary items

Declare list of sentinel values

Declare string cont for user's choice

Instantiate a shopping cart object : cart with ShoppingCart() constructor

Prompt user and welcome them to the store

DISPLAY "Please Choose From The Menu"

WHILE LOOP STARTS : while true

CALL utl.show_menu function

Declare menu = INPUT: Enter your choice

Match case start – equivalent to switch case in other languages

case 1 is for adding/purchasing to cart

Update so that cont = "y"

WHILE LOOP STARTS: while cont is in the sentinel value list

CALL utl.show_items(passing items as argument) function

CALL cart.display_cart() function to display what's in the cart while adding to it

Declare choice = INPUT: Please enter the number of the item you'd like to order:

START OF TRY/EXCEPT BLOCK

try:

if the user's choice is valid, get the item, price, and quantity of that item

Declare item = items[choice][0]

Declare price = items[choice][1]

Declare quantity = INPUT : Quantity of item

if quantity they want is a valid digit

Update quantity and cast it to an int type

else:

Update quantity = INPUT: Re-enter quantity

CALL cart.add_item(pass in item, price, and quantity as arguments) function

else:

raise ValueError exception if the user chooses something not on the list

Exception ValueError:

OUTPUT: Error! Invalid Choice!

END OF TRY/EXCEPT , END OF WHILE LOOP

Update cont = INPUT : ask if they want to continue shopping

case 2 is to remove items from the cart on the menu

CALL cart.remove_item() function # shoppingcart class object has a remove_item() function

case 3 is for displaying what is in the cart currently. cart object has display_cart() method

OUTPUT: Displaying Cart

CALL cart.display_cart() function from the cart object

case 4 is for checking out and scheduling a delivery date & time for the items

make sure the cart is not empty before checking out

if (call is_empty method from cart class to check if cart is empty)

OUTPUT: Cannot Check Out: Cart Empty!

continue

START OF WHILE LOOP : while True and cart is not empty

OUTPUT: Now Placing Order and Delivery Date

Try/except block for input validation

try: do input validation for all the required variables needed to construct a customer object:

full name, complete address and contact info for all variables make sure input is a string, not empty, and appropriate length otherwise raise ValueError exception

Declare first_name = INPUT : Enter first-name:

if not a string or input is empty or length is > 20

raise ValueError : First name must be a non-empty string

PSEUDOCODE

Declare last_name = INPUT: Enter last-name:

if not a string or input is empty or length is > 20 :

raise ValueError : Last name must be a non-empty string

Declare address = INPUT : Enter address

if not a string or input is empty or length is > 20 :

raise ValueError: Address must be a non-empty string

Declare city = INPUT: Enter city

if not a string or input is empty or length is > 25 :

raise ValueError : Address must be a non-empty string

Declare state = INPUT: Enter state

if not a string or input is empty or length is > 30:

raise ValueError: Address must be a non-empty string

Declare zip_code = INPUT: Enter zip-code

if not a string or input is empty or length does not equal 5:

raise ValueError: Zip-code must be a non-empty string with 5 digits

Declare phone_number = INPUT: Enter phone number

if not a string or input is empty or length is <7 or length > 25 :

raise ValueError: Phone number must be a non-empty string and correct length

Declare email = INPUT: Enter email:

if not a string or input is empty or email does not contain @ or length is > 30 :

raise ValueError: Email must be a non-empty string and must contain @

OUTPUT: prompt for user to verify the info they input

Prompt: VERIFY INFORMATION:

OUTPUT: first name last name, address, city, state, zip code, phone number, and email

break

Exception ValueError:

OUTPUT: Invalid input!

End of try/except block

customer_1 = cust.Customer(first_name, last_name, address, city,state, zip_code,phone_number, email)

PSEUDOCODE

Declare verify = INPUT: Is Your Information Correct? (Y/N)

Update verify = verify.lower()

If verify == 'y'or verify =='yes'

Call cart.calculate_totals() function to get subtotal, tax, shipping, grand total so the customer object has correct amounts

Declare now = CALL utl.datetime.now() function to get the current date and time at this moment using import datetime in utilities module

Declare delivery_time, delivery_date = Call utl.get_selected_info() function so that the chosen delivery time and date variables can be passed to the receipt function to be printed

Declare current_datetime = Call utl.datetime.now() function to get the current date and time

Declare current_date = Call current_datetime.date() function to get current date

Declare current_time = Call current_datetime.time() function to get current time

Format the date and time

Call utl.create_receipt(cart, customer_1, delivery_date, delivery_time, date_of_order, time_of_order) function : the utilities function has the print_receipt function which takes in the following parameters: cart object, customer object, the desired delivery date and time, and the time and date the order took place

Print Receipt with order details and all necessary information is printed for the customer in receipt.txt file

OUTPUT: Your Receipt Has Been Printed!

OUTPUT: Thanks For Shopping at Bob's Music Emporium!

OUTPUT: Check receipt.txt

break

case 5 is for exiting the program

OUTPUT: Exiting!

break

case : default case when invalid input is entered

OUTPUT: Invalid choice. Please try again.")

END OF WHILE LOOP

END of main() function

END OF PROGRAM

PSEUDOCODE

Person class - base class for Customer class

This class 8 member variables and 3 member functions

Declare Person class

Constructor (Parameters:

self, first_name, last_name, address,city,state,zip ,phone_number, email)

Declare self.first_name and initialize to first_name

Declare self.last_name and initialize to last_name

Declare self.address and initialize to address

Declare self.city and initialize to city

Declare self.state and initialize to state

Declare self.zip and initialize to zip

Declare self.phone_number and initialize to phone_number

Declare self.email and initialize to email

Start of get_full_name member function definition for getting the person's full name

Function get_full_name(Parameter: self)

Declare full_name : initialize and format person's first and last name so it can be returned

return full_name

End of function definition

Start of get_full_address member function definition for getting the complete address of person object

Function get_full_addrss (Parameter: self)

Declare full_address : initialize to formatted string that contains full address which is the person's

Street number, street, city, state, and zip code

return full_address

End of function definition

Start of get_contact member function definition for retrieving a person object's contact info

Function get_contact(Parameter: self)

Declare contact : Initialize to formatted string containing person's phone number and email address

return contact

End of function definition

PSEUDOCODE

Customer class inherits from Person class - it is a sub-class of the Person class. It inherits all the parent's member variables and methods. It also has 2 additional member variables and 2 more member methods apart from what it inherits.

FROM PERSON MODULE IMPORT PERSON CLASS

IMPORT RANDOM MODULE

Declare class Customer that inherits from Person class:

Constructor (Parameters:

self,first_name, last_name, address,city,state,zip_code, phone_number, email):

Call Parent classes' constructor to inherit those member variables and methods: super( ) function (Parameters: first_name, last_name, address,city,state,zip_code ,phone_number, email)

Declare self.cust_num : initialize to the first letter of the person's first name and last name and append a random number

Declare self.invoice_num : initialize to a random number, simulating a unique invoice number for the customer's order

Start of get_cust_num member function definition for getting the customer number member variable

Function get_cust_num(Parameter: self)

Declare num : Initialize to formatted string containing self.cust_num

return num

End of function definition

Start of get_invoice_num member function definition for getting the invoice number member variable

Function get_invoice_num(Parameter: self)

Declare inv_num : Initialize to formatted string containing self.invoice_num

return inv_num

End of function definition

PSEUDOCODE

ShoppingCart class has a dictionary cart_items, the tax rate and ship rate as member variables

Items are added to the shoppingcart object in main which the class holds in a dictionary

Declare ShoppingCart class

Constructor()

Declare cart_items dictionary to hold items of the class objects

Declare self.tax_rate and set equal to 0.0775

Declare self.ship_rate and set equal to 0.09

Start of isEmpty member function definition

Function isEmpty(Parameters: self)

if there are no items in the cart

return True

else

return False

End of function definition

Start of add_item member function definition

Function add_item(Parameters : self, item, price, quantity)

if the item is not in the cart already

add the item and set the price and quantity accordingly

else

only update the quantity

End of function definition

Start of remove_item function definition

Function remove_item(Parameters: self)

If the cart is empty do nothing

OUTPUT: Cart is empty.

return

Prompt: Items in Cart:

for i, (item_name, item_info) in the cart of items :

OUTPUT: the items' name and quantity

PSEUDOCODE

Declare choice = INPUT Enter the number of the item to remove:

try/except block so user can only remove items that are in the cart and for valid quantities, meaning don't let them remove more items than are in the cart

try:

declare choice and cast as integer

if the number to remove of the item is in the list of items

Declare item_to_remove = Declare list(store items they want to remove)

Declare quantity_to_remove =INPUT: Enter the quantity of the item to remove

if the quantity they want to remove is less than 0 or more than what is in the cart:

OUTPUT: Invalid quantity!

Else if the quantity the user wants to remove is the same as the quantity in the cart:

DELETE the key associated with the item in the dictionary

OUTPUT: echo back which item has been removed

else

reduce the quantity of the item by the amount specified

OUTPUT: the quantity and item which was removed out on the console

else

OUTPUT: Invalid choice. Please enter a valid number

except ValueError:

print("Invalid input. Please enter a number.")

End of try/except block

End of function definition

Start of display_cart member function definition

Function display_cart(parameter: self)

if there are items in the cart

Declare sub_total and set to 0

OUTPUT: Shopping Cart Status

for item_name, item_info in classes items dictionary

Update sub_total += (the price * the quantity) of all the items in the class member variable dictionary

OUTPUT: the item name , price, and quantity of the items in the cart dictionary

OUTPUT: sub_total

else

OUTPUT: Cart is Empty

End of function definition

Start of calculate_totals member function definition

Function calculate_totals(parameter: self)

Declare sub_total and initialize to 0

Declare shipping and initialize to 0

Declare taxes and initialize to 0

Declare grand_total and initialize to 0

get subtotal which is based on price and quantity of each item

for item_info in items of the class variable cart_items

Calculate sub_total += item *quantity

Calculate taxes = sub_total*self.tax_rate

Calculate shippping = sub_total*self.ship_rate

Calculate grand_total = sub_total+taxes+shippping

return sub_total,taxes,shippping,grand_total

End of function definition

PSEUDOCODE

CUSTOM UTILITIES MODULE

IMPORT TKINTER

IMPORT TKCALENDAR

IMPORT DATETIME AND TIMEDELTA

Start of show_menu function definition

Function show_menu():

OUTPUT: Menu

OUTPUT: 1. Shop For Items (Add Items To Cart)

OUTPUT: 2. Remove Items From Cart

OUTPUT: 3. Display Cart Items

OUTPUT: 4. Place Order And Delivery

OUTPUT: 5. Exit

End of function definition

Start of show_items function definition

Function show_items(parameter: items)

Prompt: Instruments For Sale

for key, (item, price) in the dictionary called items:

OUTPUT: the key, items, and price of the nested dictionary

End of function definition

Start of show_selected_info function definition used for GUI Widget calendar

Function show_selected_info(Parameters:

calendar, time_spinbox, date_label, time_label, message_label)

get the date the user selected for the delivery so it can be used as needed

Declare selected_date =Call calendar.get_date() built in function from tkcalendar

Declare selected_time = Call time_spinbox.get() built in function

OUTPUT: Date Selected: selected_date

OUTPUT: Time Selected: , selected_time

Prompt : Exit Widget or Select Other Date and Time For Delivery

Declare date_label.config : text=Selected Delivery Date: OUTPUT: selected_date

PSEUDOCODE

Declare time_label.config : text= Selected Delivery Time: OUTPUT: selected_time

let user know they can exit the widget window to continue with program

Declare message_label.config: text=Exit to Continue.

End of function definition

Start of get_selected_info function definition used for GUI Widget

Function get_selected_info()

Declare root = Call Tk() constructor

Declare root.title: ("Delivery Calendar")

Prompt : Choose Delivery Date & Time On Pop-up Widget.

Prompt: After Selecting Date & Time, Exit the Window to Continue.

Declare now = Call datetime.now() built in function

Declare one_year initialize to 1 year from now

Declare calendar object = Calendar(Parameters:

root, selectmode="day", year=now.year, month=now.month, day=now.day, showToday=True, mindate=now,maxdate=one_year)

constructor

set calendar orientation: Call calendar.pack() function

Declare date_label = Label(Parameters: root, text="Selected Date: ") constructor

Set label orientation: Call date_label.pack() function

Declare time_label = Label(Parameters: root, text="Select Delivery Time:") constructor

Set orientation: Call time_label.pack() function

Declare time_spinbox = Spinbox(Parameters:

root, values=("9:00 AM", "10:00 AM", "11:00 AM", "12:00 PM", "1:00 PM", "2:00 PM", "3:00 PM", "4:00 PM", "5:00 PM"))

constructor

Set orientation: Call time_spinbox.pack() function

Declare select_button = Button(Parameters:

root, text="Press To Select Date & Time", command=lambda: show_selected_info(calendar, time_spinbox, date_label,time_label, message_label))

PSEUDOCODE

constructor

Set orientation: Call select_button.pack() function

Declare message_label = Label(Paremetes:

root, text="")

constructor

Set orientation: Call message_label.pack() function

Declare delivery_time = Call time_spinbox.get() function

Declare delivery_date = Call calendar.get_date() function

Call root.mainloop() for event loop

return delivery_time, delivery_date

End of function definition

Start of create_receipt function definition for creating receipt for order

Function create_receipt (Parameters:

cart, customer, delivery_date, delivery_time, date_of_order, time_of_order)

Call cart.display_cart() function

Try/finally block for exception handling

try:

Open the file in write mode

Declare with open("receipt.txt", 'w') as file:

Write to file: Receipt # customer.get_invoice_num()

Write order details

Write to file: Customer customer.get_cust_num() Information

Write to file: customer.get_full_name()

Write to file: customer.get_full_addrss()

Write to file: customer.get_contact()

Write to file: Date of Order: date_of_order

Write to file: Time of Order: time_of_order

Write to file: Scheduled Delivery Date: delivery_date

Write to file: Scheduled Delivery Time: delivery_time

PSEUDOCODE

Write to file: ITEM PRICE QTY

for item_name, item_info in the dictionary holding cart items:

Write to file: item_name: - price - quantity

Declare sub_total,taxes,shippping,grand_total = Call cart.calculate_totals() class function

Write to file: Sub-total : $sub_total

Write to file: Taxes(7.75%) : $taxes

Write to file: Shipping(9.0%) : $shippping

Write to file: Grand Total : $grand_total

Write to file: Thank you!

finally:

close file

Call file.close() function

END OF TRY/FINALLY

End of function definition