|  |  |  |  |
| --- | --- | --- | --- |
| Req # | Test Case | Expected Result | Actual Result |
| 1 | The program accepts float entries for the amount | Input considered an integer or float will be accepted |  |
| 2 | The program does not accept strings for the amount | Input considered a string for amount will result in a handled Exception and prompt for other input |  |
| 3 | Use the decimal module to make sure the program doesn’t yield incorrect total sales due to floating-point errors. | The decimal module is imported, and variables used to calculate total sales are converted to decimal before calculating. |  |
| 4 | Store the functions for getting sales data from a user in a module named sales. | Functions for retrieving sales data are stored in sales.py and imported using a sales module. |  |
| 5 | Use docstrings to document the sales module. | docstrings appear right after the definition of every function, method, class, or module. |  |
| 6 | Use type hints to document the sales module. | A type hint will be present in the sales module referencing the data type being returned from the function. |  |
| 7 | Create a menu that allows the user to view existing sales data, add new sales data, and import sales data. | A menu is generated at startup with others for users to view sales data, add sales data, or import sales data. |  |
| 8 | Calculate the quarter based on the month value for each entity in the sales data. | When sales data is viewed, the Quarter is presented according to the month in the sales data. Sales months 1,2, and 3 are Quarter 1. Sales months 4, 5, and 6 are Quarter 2. Sales months 7, 8, and 9 are Quarter 3. Sales months 10, 11, and 12 are Quarter 4. |  |
| 9 | Allow users to import sales data from a CSV file that contains sales amount, year, month, and day. | When accessing the import option from the menu, check if the user’s file name ends in .csv. Check if the comma separated values are in the correct format and type to be added to existing data. |  |
| 10 | Use a text file to keep track of imported files so a user can’t import a file more than once. | Confirm that a text file is being generated and updated containing the names of successfully imported csv files only. |  |
| 11 | Store the functions for writing and reading the files in a separate module named db.py. | Confirm db.py is being used and contains functions for writing and reading txt files as well as functions for writing and reading csv files. Confirm these functions are being utilized by the program for these tasks. |  |
| 12 | When the program starts, it should read the sales data from a file named all\_sales.csv. | Load data from file all\_sales.csv when program starts to populate current sales data. This file should be created if it does not already exist. |  |
| 13 | When the program ends, it should write the sales data, including any imported data and any data entered by the user, to all\_sales.csv. | When user enters the exit command from the menu, save all added and imported sales data to the all\_sales.csv file before closing the program. |  |
| 14 | Don’t store the quarter value. Instead, calculate it from the month as needed. | When sales data is presented to the User, the quarter is calculated based on the sales month at that time and the quarter value is never stored in the all\_sales.csv list. |  |
| 15 | Handle the exception that occurs if the program can’t find the all\_sales.csv file, the file the user is trying to import, or the text file that tracks imported files. | If all\_sales.csv is not found when the program loads, the User is informed that the file was not found and then all\_sales.csv is created for the application to use. No sales would exist at this time. |  |
| 16 | Handle the exceptions that occur if the user enters a string or float where an integer is expected. | The user is presented with a string to explain why their input was rejected and asked to enter an integer instead. |  |
| 17 | Handle the exceptions that occur if the user enters a string where a float is expected. | The user is presented with a string to explain why their input was rejected and asked to enter float instead. |  |
| 18 | Handle the exceptions that occur if the data in the imported file can’t be converted to a float or an integer. Use asterisks and question marks to notify the user of bad imported data. | When a User attempts to import sales data, check if each piece of data is a string and present the data to the user with asterisks next to any strings if strings are found. The user is informed that strings cannot be imported into sales data. |  |
| 19 | The sales amount should be greater than zero. | The sales amounts for imported data and added sales data is verified to be greater than zero or the user is informed that the sales amount must be greater than zero before and the data is rejected. |  |
| 20 | The sales month should be between 1 and 12. | The sales month for imported data and added sales data is verified to between 1 and 12 or the user is informed that the sales month must be between 1 and 12 and the data is rejected. |  |
| 21 | The sales day must be between 1 and 31 if the sales month is 1, 3, 5, 10, or 12. | If the sales month is 1, 3, 5, 10, or 12, the sales day entered will encounter a handled exception if the day is not an integer between 1 and 31 and ask for another input. |  |
| 22 | The sales day must be between 1 and 30 if the sales month is 4, 6, 9, or 11. | If the sales month is 4, 6, 9, or 11, the sales day entered will encounter a handled exception if the day is not an integer between 1 and 30 and ask for another input. |  |
| 23 | The sales day must be between 1 and 28 if the sales month is 2. | If the sales month is 2, the sales day entered will encounter a handled exception if the day is not an integer between 1 and 28 and ask for another input. |  |
| 24 | The sales year must be greater than or equal to 2000 and less than or equal to 9999. | If the sales year is in integer less than 2000 and greater than 9999 will encounter a handled exception and ask for another input. |  |
| 25 | Use the locale module to display the sales amount using formatting that’s appropriate for your current locale. | The locale module will be imported and referenced when formatting the sales amount based on the specified locale. |  |
| 26 | Use f-strings to format the widths and alignment of the columns in the display table. | All data tables displayed related to sales data will be formatted using f-strings. |  |
| 27 | Use string operations to ensure the name of an imported file follows this format: sales\_qn\_yyyy\_r.csv, where n represents the sales quarter, yyyy represents the sales year, and r represents the sales region. | If the sales\_qn\_yyyy\_r.csv filename format is not followed when attempting to import sales data, the program will encounter a handled exception and ask the user for a valid filename and explain the required format. |  |
| 28 | Valid regions are w for West, m for Mountain, c for Central, and e for East. | The only regions considered valid are w,c,or e. |  |
| 29 | The program should get the sales region code from the filename and display and store it along with the other sales data. | The region code will be extracted from the filename and stored for the sales data imported from the file. |  |
| 30 | The program should also get a valid sales region from the user when they add sales data with the ‘add’ command. | The sales region is requested from the user when the user attempts to manually add sales data. |  |
| 31 | The imported CSV files should have one value for the sales date in YYYY-MM-DD format, rather than individual year, month, and day values. The user should be given the opportunity to correct the date. | Any csv file that does not store the sales day, month, and year as a single entry in YYYY-MM-DD format, inform the user of the error. Request the user to enter the date in the valid format of YYYY-MM-DD. |  |
| 32 | The year, month, and day should be entered by the user as a single entry in YYYY-MM-DD format, rather than as individual year, month, and day values. | The user will be prompted to enter the sales date when manually adding sales data as a single entry in YYY-MM-DD format or inform the user of the error and ask the user to modify the entry. Then try importing the data again. |  |
| 33 | Store the sales data as a list of dictionaries rather than a list of lists. To save the sales data to a CSV file, you’ll need to convert it back to a list of lists. | Data loaded from the sales data csv file will be converted to a list of dictionaries. Before saving sales data to csv, convert the list of dictionaries to a list of lists. |  |
| 34 | Store the valid regions in a dictionary with the region code (w, m, c, or e) as the key and the region name (West, Mountain, Central, or East) as the value. | Valid regions will be stored in a dictionary with the region letter as the key and the value as the capitalized full region name. |  |
| 35 | Use the regions dictionary to display the region name rather than the region code. | Regions are always displayed in tables as the region name and not the region code. |  |

**2. Outline**

main

load csv

display menu

view sales

add sales

import sales

display menu

exit program

save csv

**3. Pseudocode**

**IF** all\_sales.csv existed

load sales data into memory for current sales

Handle exception for file not found and create the all\_sales.csv file

Display the menu

**WHILE** user option does not equal exit

Get menu option from user

**IF** user input == view (view sales function)

Display table headings

**FOR** sales in sales

Display an incrementing number, date, quarter, region name, and sale amount formatted using the locale module

Display total sales

**ELSE IF** user input == add (add sales function)

Get amount, date as a string formatted as yyyy-mm-dd, and region letter

Handle exceptions for value error and type error by informing user of invalid entry

**IF** date not in yyyy-mm-dd format

Display message to user that date must be in yyyy-dd-mm- format

Get new date from user and check if it is the correct format

**IF** region does not equal w, m, c or e

Display message to user that region code must be w, m, c or e

**ELSE**

Create a dictionary of the amount, date, and region to add to the current sales list.

Display message to the user the sale with the sales data input was added

**RETURN** sales list

**ELSE IF** user input == import (import sales function)

Get import file name as string in the format “sales\_qn\_yyyy\_r.csv”

**IF** date not in yyyy-mm-dd format

Display message to user that date must be in yyyy-dd-mm- format

Get new date from user and check if it is the correct format

**IF** region in file name does not equal w, m, c or e

Display message to user that region code must be w, m, c or e

**FOR** entry in file

Check that the sales amount is a float and is greater than zero.

**IF** any entries in the import file are invalid

Print all data entries with a question mark at the beginning of each line with bad code and an asterisk next to each invalid entry

Print a string informing the user that there I bad data in the file and to correct the data before importing the data again

**ELSE** add name of file to text file of previously imported data files and the sales data from the imported file to the current sales list

**ELSE IF** user input == menu (menu function)

Display the menu

**ELSE IF** user input == exit (exit function)

Convery sales list of dictionaries to a list of lists

Save sales list of lists to all\_sales.csv

**ELSE**

Handle exceptions for value errors or type errors and inform the user that the input was invalid

**CONTINUE**