Software App Case Study:

Background:

We were asked to make an application that would keep an inventory of things; it could be for restaurants, your own house, or personal hobbies. This application will be coded in Python and will be able to store data in a file and read data from a file as well. So all the data that we put in is printable, and all the data is ready for visualization.
This application will be run through a menu of options where you can add data, delete data, and visualize data.

Issues/Problems:

I think the biggest challenge for this app was how to collect the right data from a file. Different types of data exist, and it was tricky to collect the right data. The other challenge for this app is deleting a specific piece of data without altering the rest of the data in the file. We can say that getting input/output from users is the easy part, but storing and deleting data is definitely the biggest challenge.

Process/Actions/Decisions:

We had a good leader who gave us all the tools and knowledge that we needed to get that application done in a much more approachable way. We had the right documentation that helped us develop this application in the best way possible.

Work:

Here, I will explain in detail all the steps necessary to get that application running.

First, we will divide this document into classes.

The class Product:

In this part, we needed to store data about a product, the properties of this product would be the name of the product and the price of the product. First, we will construct a function that will have a product_name variable and a product_price variable.

```
@staticmethod
def __init__(self, product_name: str, product_price: float):
# Use a property to set the attribute
self.product_name = str(product_name)
self.product_price = float(product_price)
```

Listing 1

After this is done, we need to create the getters and setters for these two variables. Both will have an exception message corresponding to the task that they will perform. In the case of the product name, if the user tries to input a number, it will show an error message. The opposite will occur if the user tries to input a word into a price that only accepts numbers.

```
@property # You don't use for the getter's directive!
def product_name(self): # (getter or accessor)
return str(self.__product_name).title() # Format attribute as Title case

@product_name.setter # The @NAME.setter must match the getter's name!
def product_name(self, value): # (setter or mutator)
if str(value).isnumeric() == False:
self.__product_name = value
else:
raise Exception("Names cannot be numbers")

@property # You don't use for the getter's directive!
def product_price(self): # (getter or accessor)
return float(self.__product_price) # Format attribute as Title case

@product_price.setter # The @NAME.setter must match the getter's name!
def product_price(self, value): # (setter or mutator)
if float(value).isnumeric():
self.__product_price = value
else:

raise Exception("Price can only be numbers")
```
Listing 2

Finally, the two implicit and explicit functions that will convert the code in strings and format our code

```
def to_string(self):
return str(self.product_name) + ',' + str(self.product_price)

def __str__(self):
return str(self.product_name) + ',' + str(self.product_price)
```
Listing 3

Class FileProcesor:

First, we created a function that will read data from a file, to do that we need to upload the data into the memory and then append the data in a list of rows.

```
def read_data_from_file(strFileName, list_of_rows):


list_of_rows.clear() # clear current data
file = open(strFileName, "r")
for line in file:
product, price = line.split(",")

row = product.strip(), price.strip()
list_of_rows.append(row)
file.close()
```

```
return list_of_rows
```
Listing 4

After this is done, we will create a function that adds data to a list. The variable row will store the string product and the float price, and the function will append the variable row to a list of rows.

```
def add_data_to_list(product, price, list_of_rows):
list_of_rows = []
row = str(product).strip(), float(price)
list_of_rows.append(row)
return list_of_rows
```
Listing 5

Finally, in this class, we will make a function that will write data to a file

```
@staticmethod
def write_data_to_file(file_name, list_of_rows):

success_status = False
file = open(file_name, "w")
for row in lstOfProductObjects:
file.write(row.__str__() + '\n')
file.close()
success_status = True
return success_status
```
Listing 6

Class IO:

In this class, we will start by creating a function that will print a menu of options, followed by another function that will ask the user which option they would like to choose

```
@staticmethod
def print_menu_items():
print('''
Menu of Options
1) Show current data
2) Add a new item.
3) Save Data to File
4) Exit Program
''')
print() # Add an extra line for looks in the terminal window

@staticmethod
def input_menu_choice():
choice = str(input("Which option would you like to perform? [1 to 4] -
")).strip()
print() # Add an extra line for looks
return choice
```

Listing 7
The next function will show the current product list by loading the data in memory and iterating through the row

```
@staticmethod
def current_products_in_list(list_of_rows):
print("******* The current Products are: *******")
for row in list_of_rows:
print(row.__str__() + '\n')
print("*****************************************")
print() # Add an extra line for looks
```

Listing 8
Finally, in this part, we will create a function that will get input from user and ask for a product and its price

```
@staticmethod
def input_new_product_and_price():
product = str(input("What is the Product Name?: ")).strip()
price = float(input("What is the product Price?: "))
return product, price
```

Listing 9

The Main Body of the script:

In this part, we put all the code together with conditional variables depending on the options that the user will choose.

```
FileProcessor.read_data_from_file(strFileName=strFileNames,
list_of_rows=lstOfProductObjects) # read file data

while (True):

IO.print_menu_items() # Shows menu
choice_str = IO.input_menu_choice() # Get menu option

if choice_str.strip() == '1':
IO.current_products_in_list(lstOfProductObjects)
continue

elif choice_str.strip() == '2':
lstOfProductObjects.append(IO.input_new_product_and_price())
continue
```

```
elif choice_str.strip() == '3':
FileProcessor.write_data_to_file(strFileNames, lstOfProductObjects)
continue

elif choice_str.strip() == '4':
break
```
Listing 10

Result:

The most positive aspect of developing this app was having the tools and knowledge to get it done, without the guidance of somebody with more experience, it probably would have taken much longer. But we were happy with the result. After testing the code many times, we adjusted what was not great until we got the results that made us happy.
The journey was at times difficult, but in the end, it was awesome.