

EZY Property solutions

Buying and selling of properties (Property Management)



June 21, 2024

George goodey

IT PAT – Phase 1

Contents

[Task 1A – Task Definition 2](#_Toc174873488)

[Task 1B – User Stories & Acceptance Tests 2](#_Toc174873489)

[Everyday User (Client) 2](#_Toc174873490)

[Everyday Agent 2](#_Toc174873491)

[Admin User 3](#_Toc174873492)

[Task 2 – Database Design 3](#_Toc174873493)

[Clients Table 3](#_Toc174873494)

[Agents Table 4](#_Toc174873495)

[Properties Table 4](#_Toc174873496)

[Relationships 4](#_Toc174873497)

[Task 3A – Class Diagram 6](#_Toc174873498)

[Task 3B – Additional Data Structures 6](#_Toc174873499)

[Text Files 6](#_Toc174873500)

[Parallel Arrays 7](#_Toc174873501)

[Task 4A – Navigation & Screen Flow 8](#_Toc174873502)

[Task 4B – GUI-Design 8](#_Toc174873503)

[Task 5 – Input with validation, processing, output 24](#_Toc174873504)

[Task 5A – Input 24](#_Toc174873505)

[Task 5B - Processing 25](#_Toc174873506)

[Task 5C - Output 27](#_Toc174873507)

# Task 1A – Task Definition

The property market is a vast, ever-expanding, and major contributor to the growth and stability of our economy. There are many major property agencies in South Africa such as: Chas Everitt, Louw & Coetzee, and Pam Golding. However, there are many lesser-known property agencies who operate on a much smaller scale but cannot keep track of their properties in a fast and efficient manner.

I intend to provide a solution for those property agencies who struggle to keep track of the buying and selling of their properties. I intend to code a fast and efficient program for them in which they can catalogue, keep track of, and manage the various properties, agents, and clients in the business. In addition, the property agencies will also be able to perform calculations to generate statistics based on the properties themselves, the revenue generated, the buying prices, and the selling prices to make calculated decisions when considering future expansions or further investment into the property market.

# Task 1B – User Stories & Acceptance Tests

## Everyday User (Client)

**As a** client,

**I want** to view all listed properties and their respective information,

**So that** I can decide on whether I want to buy the property.

**Acceptance Test:**

1. Ability to view all listed properties.
2. Ability to buy the property.
3. Ability to get information on the agent who listed the property.

## Everyday Agent

**As an** agent,

**I want** to view all listed properties and their respective information,

**So that** I can list new properties or remove old listings.

**Acceptance Test:**

1. Ability to view all properties.
2. Ability to list properties and remove listings.
3. Ability to get information on the client connected to a specific property.

## Admin User

**As an** admin,

**I want** to view all clients, agents, and properties,

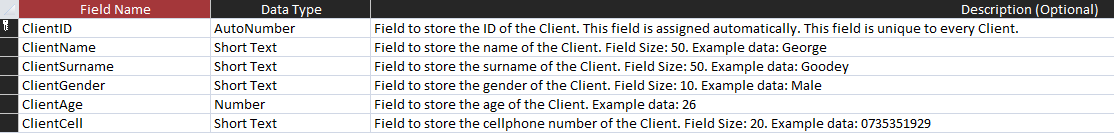
**So that** I can calculate relevant statistics, add new records, update existing records, and delete specific records.

**Acceptance Test:**

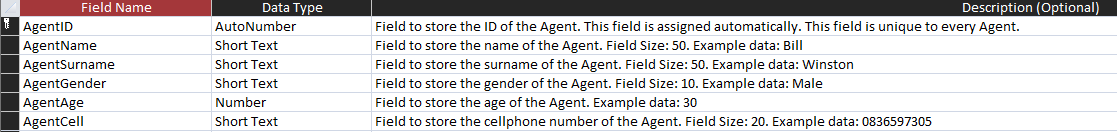
1. Ability to generate relevant statistics.
2. Ability to add, remove or update records.
3. Ability to view all clients, agents, and properties.

# Task 2 – Database Design

## Clients Table

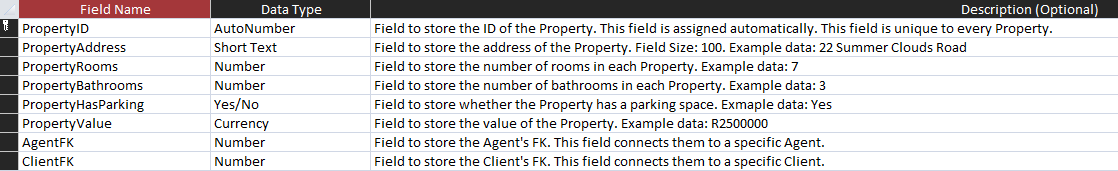
The goal of this table is to store all data on each client. This table has six distinct fields which describe each client, namely: ClientID, ClientName, ClientSurname, ClientGender, ClientAge, and ClientCell. The ClientID field uniquely identifies each client in the database.

## Agents Table

The goal of this table is to store all data on each agent. This table has six distinct fields which describe each agent, namely: AgentID, AgentName, AgentSurname, AgentGender, AgentAge, and AgentCell. The AgentID field uniquely identifies each agent in the database.

## Properties Table

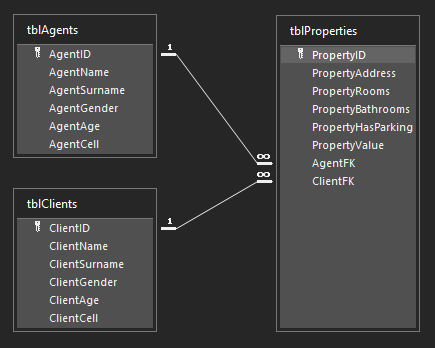
The goal of this table is to store all data on each property. This table has eight distinct fields which describe each property, namely: PropertyID, PropertyAddress, PropertyRooms, PropertyBathrooms, PropertyHasParking, PropertyValue, AgentFK, and ClientFK. The PropertyID field uniquely identifies each property in the database. The AgentFK field links the property to a specific Agent. The ClientFK field links the property to a specific Client.



## Relationships

This database consists of two relationships between the tables:

The first relationship exists between the clients table and the properties table. The ClientID field in the clients table is being linked to the ClientFK field in the properties table. One client might own more than one property, thus the relationship between the two tables is one-to-many. One client can own multiple properties, but one property cannot be linked to multiple clients.

The second relationship exists between the agents table and the properties table. The AgentID field in the agents table is being linked to the AgentFK field in the properties table. One agent might be involved in multiple properties, thus the relationship between the two tables is one-to-many. One agent can be involved with multiple properties, but One property cannot have multiple agents.

# Task 3A – Class Diagram

|  |
| --- |
| <<entity>>  **Property** |
| -fPropertyAddress : String  -fPropertyRooms : Integer  -fPropertyBathrooms: Integer  -fPropertyHasParking : Boolean  -fPropertyValue : Real  -fAgentFK : Integer  -fClientFK : Integer |
| +create(pPropertyAddress : String, pPropertyRooms : Integer,  pPropertyBathrooms : Integer, pPropertyHasParking : Boolean,  pPropertyValue : Real, pAgentFK : Integer, pClientFK : Integer)  +getAddress() : String  +getRooms() : Integer  +getBathrooms() : Integer  +getParkingStatus() : Boolean  +getValue() : Real  +estimateMarketValue() : Real |

# Task 3B – Additional Data Structures

## Text Files

I intend to use text files for storing user credentials in the program. Each set of credentials corresponds to a specific user in the database. When you attempt to log into the program, it will first decrypt the encrypted text file and then it will check whether the username and password combination was valid according to the decrypted data from the text file. The user will also have the option to register new credentials. These credentials will be encrypted and added to the text file.

## Parallel Arrays

I intend to use parallel arrays when loading the decrypted data from the text files. When the user tries to log into the program it will: decrypt the text file, store the username and password pairs in the parallel arrays, and compare the user’s inputs to the arrays. If the user inputs the correct information, the user will be allowed to enter the program. If the user did not enter the correct information, the user will receive an error message.

# Task 4A – Navigation & Screen Flow



# Task 4B – GUI-Design

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to allow the user to either log into an existing agent or client or register a new agent or client. |
| **Input components: what it inputs.** | **LOGIN:**  Username: Here the user can input their username to log into the program.  Password: Here the user can input their password to log into the program.  **REGISTER:**  Username: Here the user can input their username to be registered.  Password: Here the user can input their password to be registered.  User Type: Here the user can choose one of two options to determine the type of user. They can be either a client or an agent.  Name: Here the user can input the name of the client or agent.  Surname: Here the user can input the surname of the client or agent.  Gender: Here the user can choose one of two options to determine the gender of the client or agent. They can be either Male or Female.  Age: Here the user can determine the age of the client or agent.  Cell: Here the user can input the cellphone number of the client or agent. |
| **Output components (what is displayed)** | Errors: Error messages with regards to validity of the information entered by the user will be displayed dynamically.  Help: A help dialog will be displayed dynamically in the event the user needs help. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Login Button: The program will check if the username and password combination is valid. If so, the user will be logged into the corresponding account.  Register Button: The program will validate the information entered by the user. If everything is valid, the program will create a new account and log the user into that account.  Help Button: Displays a help dialog dynamically.  Close Button: Closes the program. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to display all relevant information on every agent. The admin user will be able to create, update, or delete a record from the table. |
| **Input components: what it inputs.** | Sorting Direction: Specifies how the records will be sorted. Either ascending or descending according to some metric.  Sorting Fields: Specifies by which metric the records will be sorted.  Search Bar: Gives the admin user the ability to search for a specific record in the table. The records will be filtered by surname.  **DYNAMIC (Create):**  AgentName: A dynamically created input field will allow the admin user to input an appropriate name for the agent yet to be created.  AgentSurname: A dynamically created input field will allow the admin user to input an appropriate surname for the agent yet to be created.  AgentGender: A dynamically created input field will allow the admin user to specify a gender for the agent yet to be created.  AgentAge: A dynamically created input field will allow the admin user to specify an age for the agent yet to be created.  AgentCell: A dynamically created input field will allow the admin user to input an appropriate cellphone number for the agent yet to be created.  **DYNAMIC (Update):**  AgentName: A dynamically created input field will allow the admin user to input an appropriate name for the agent yet to be updated.  AgentSurname: A dynamically created input field will allow the admin user to input an appropriate surname for the agent yet to be updated.  AgentGender: A dynamically created input field will allow the admin user to specify a gender for the agent yet to be updated.  AgentAge: A dynamically created input field will allow the admin user to specify an age for the agent yet to be updated.  AgentCell: A dynamically created input field will allow the admin user to input an appropriate cellphone number for the agent yet to be updated.  **DYNAMIC (Delete):**  AgentID: A dynamically created input field will specify which agent to remove from the table. |
| **Output components (what is displayed)** | DBGrid: All filtered or unfiltered records will be displayed in a grid format here.  Errors: Error messages with regards to validity of the information entered by the user will be displayed dynamically.  Help: A help dialog will be displayed dynamically in the event the user needs help. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Show All Button: Removes all filters applied to the table thus displaying all records in the table.  Create a record: Creates a dynamic panel on the form to allow the admin user to create a new agent.  Update a record: Creates a dynamic panel on the form to allow the admin user to update an existing agent from the table.  Delete a record: Creates a dynamic panel on the form to allow the admin user to delete a specific agent from the table.  Help Button: Displays a help dialog dynamically.  Close Button: Closes the form. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to display all relevant information on every client. The admin user will be able to create, update, or delete a record from the table. |
| **Input components: what it inputs.** | Sorting Direction: Specifies how the records will be sorted. Either ascending or descending according to some metric.  Sorting Fields: Specifies by which metric the records will be sorted.  Search Bar: Gives the admin user the ability to search for a specific record in the table. The records will be filtered by surname.  **DYNAMIC (Create):**  ClientName: A dynamically created input field will allow the admin user to input an appropriate name for the client yet to be created.  ClientSurname: A dynamically created input field will allow the admin user to input an appropriate surname for the client yet to be created.  ClientGender: A dynamically created input field will allow the admin user to specify a gender for the client yet to be created.  ClientAge: A dynamically created input field will allow the admin user to specify an age for the client yet to be created.  ClientCell: A dynamically created input field will allow the admin user to input an appropriate cellphone number for the client yet to be created.  **DYNAMIC (Update):**  ClientName: A dynamically created input field will allow the admin user to input an appropriate name for the client yet to be updated.  ClientSurname: A dynamically created input field will allow the admin user to input an appropriate surname for the client yet to be updated.  ClientGender: A dynamically created input field will allow the admin user to specify a gender for the client yet to be updated.  ClientAge: A dynamically created input field will allow the admin user to specify an age for the client yet to be updated.  ClientCell: A dynamically created input field will allow the admin user to input an appropriate cellphone number for the client yet to be updated.  **DYNAMIC (Delete):**  ClientID: A dynamically created input field to specify which client to remove from the table. |
| **Output components (what is displayed)** | DBGrid: All filtered or unfiltered records will be displayed in a grid format here.  Errors: Error messages with regards to validity of the information entered by the user will be displayed dynamically.  Help: A help dialog will be displayed dynamically in the event the user needs help. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Show All Button: Removes all filters applied to the table thus displaying all records in the table.  Create a record: Creates a dynamic panel on the form to allow the admin user to create a new client.  Update a record: Creates a dynamic panel on the form to allow the admin user to update an existing client from the table.  Delete a record: Creates a dynamic panel on the form to allow the admin user to delete a specific client from the table.  Help Button: Displays a help dialog dynamically.  Close Button: Closes the form. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to display all relevant information on every property. The admin user will be able to create, update, or delete a record from the table. |
| **Input components: what it inputs.** | Sorting Direction: Specifies how the records will be sorted. Either ascending or descending according to some metric.  Sorting Fields: Specifies by which metric the records will be sorted.  Search Bar: Gives the admin user the ability to search for a specific record in the table. The records will be filtered by address.  **DYNAMIC (Create):**  PropertyAddress: A dynamically created input field will allow the admin user to input an appropriate address for the property yet to be created.  PropertyRooms: A dynamically created input field will allow the admin user to specify the number of rooms for the property yet to be created.  PropertyBathrooms: A dynamically created input field will allow the admin user to specify the number of bathrooms for the property yet to be created.  PorpertyHasParking: A dynamically created input field will allow the admin user to specify whether the yet to be created property has parking or not.  PropertyValue: A dynamically created input field will allow the admin user to input an approximate value for the property yet to be created.  AgentFK: A dynamically created input field will specify to which agent the property is connected.  ClientFK: A dynamically created input field will specify to which client the property is connected.  **DYNAMIC (Update):**  PropertyAddress: A dynamically created input field will allow the admin user to input an appropriate address for the property yet to be updated.  PropertyRooms: A dynamically created input field will allow the admin user to specify the number of rooms for the property yet to be updated.  PropertyBathrooms: A dynamically created input field will allow the admin user to specify the number of bathrooms for the property yet to be updated.  PorpertyHasParking: A dynamically created input field will allow the admin user to specify whether the yet to be updated property has parking or not.  PropertyValue: A dynamically created input field will allow the admin user to input an approximate value for the property yet to be updated.  AgentFK: A dynamically created input field will specify to which agent the property is connected.  ClientFK: A dynamically created input field will specify to which client the property is connected.  **DYNAMIC (Delete):**  PorpertyID: A dynamically created input field will specify which property to remove from the table. |
| **Output components (what is displayed)** | DBGrid: All filtered or unfiltered records will be displayed in a grid format here.  Errors: Error messages with regards to validity of the information entered by the user will be displayed dynamically.  Help: A help dialog will be displayed dynamically in the event the user needs help. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Show All Button: Removes all filters applied to the table thus displaying all records in the table.  Create a record: Creates a dynamic panel on the form to allow the admin user to create a new property.  Update a record: Creates a dynamic panel on the form to allow the admin user to update an existing property from the table.  Delete a record: Creates a dynamic panel on the form to allow the admin user to delete a specific property from the table.  Help Button: Displays a help dialog dynamically.  Close Button: Closes the form. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to display all relevant information on every property and agent. The client user needs to be able to see the relevant agent information tied to the property. The client user will be able to purchase empty properties or sell their existing properties. |
| **Input components: what it inputs.** | Sorting Direction: Specifies how the records will be sorted. Either ascending or descending according to some metric.  Sorting Fields: Specifies by which metric the records will be sorted.  Search Bar: Gives the admin user the ability to search for a specific record in the table. The records will be filtered by address.  Property to purchase: If the client user wants to purchase an empty property, they will be able to select an empty property ID from the combo box.  Property to sell: IF the client user wants to sell one of their properties, they will be able to select one of their property IDs from the combo box. |
| **Output components (what is displayed)** | DBGrid: All filtered or unfiltered records will be displayed in a grid format here.  Errors: Error messages with regards to validity of the information entered by the user will be displayed dynamically.  Help: A help dialog will be displayed dynamically in the event the user needs help. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Show All Button: Removes all filters applied to the table thus displaying all records in the table.  Show Mine Button: Adds a filter to the table to only show the client their properties.  Help Button: Displays a help dialog dynamically.  Log out Button: Logs out of the program. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to display all relevant information on every property and client. The agent user needs to be able to see the relevant client information tied to the property. The agent user will be able to list properties on the market or remove them from the market. |
| **Input components: what it inputs.** | Sorting Direction: Specifies how the records will be sorted. Either ascending or descending according to some metric.  Sorting Fields: Specifies by which metric the records will be sorted.  Search Bar: Gives the admin user the ability to search for a specific record in the table. The records will be filtered by address.  Property to list: If the agent user wants to list a specific property that is not already listed, they will be able to select a property ID from the combo box.  Property to unlist: If the agent user wants to remove a listing for a specific property that they have listed, they will be able to select a property ID from the combo box. |
| **Output components (what is displayed)** | DBGrid: All filtered or unfiltered records will be displayed in a grid format here.  Errors: Error messages with regards to validity of the information entered by the user will be displayed dynamically.  Help: A help dialog will be displayed dynamically in the event the user needs help. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Show All Button: Removes all filters applied to the table thus displaying all records in the table.  Show Mine Button: Adds a filter to the table to only show the agent the properties they have listed.  Help Button: Displays a help dialog dynamically.  Log out Button: Logs out of the program. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to allow the admin user to move between the different forms. |
| **Input components: what it inputs.** | No input elements. |
| **Output components (what is displayed)** | No output elements. |
| **Components with event handlers attached (what happens – not detail or how it happens).** | Agents Button: Takes the user to the agent form.  Clients Button: Takes the user to the client form.  Properties Button: Takes the user to the property form  Stats Button: Takes the user to the stats form.  Help Button: Displays a help dialog dynamically.  Log out Button: Logs out of the program. |

|  |  |
| --- | --- |
|  | |
| **Purpose** | The purpose of this form is to allow the admin user to generate relevant statistics on the agents, clients or properties. |
| **Input components: what it inputs.** | No input elements. |
| **Output components (what is displayed)** | Rich Memo: Generated statistics will be displayed here. |

|  |  |
| --- | --- |
| **Components with event handlers attached (what happens – not detail or how it happens).** | **CLIENTS:**  Avg Age: Determines the average age of all clients.  Sum of Ages: Determines the sum of all ages of clients.  Min Age: Determines the minimum age of all clients.  Max Age: Determines the maximum age of all clients.  # of Records: Determines the number of records for clients.  **AGENTS:**  Avg Age: Determines the average age of all agents.  Sum of Ages: Determines the sum of all ages of agents.  Min Age: Determines the minimum age of all agents.  Max Age: Determines the maximum age of all agents.  # of Records: Determines the number of records for agents.  **PROPERTIES:**  Avg Value: Determines the average value of all properties.  Sum of Values: Determines the sum of all values of properties.  Min Value: Determines the minimum value of all properties.  Max Value: Determines the maximum value of all properties.  # of properties with parking: Determines the number of properties with parking.  # of properties without parking: Determines the number of properties without parking.  Avg Bathrooms: Determines the average number of bathrooms per property.  Sum of Bathrooms: Determines the sum of all bathrooms of properties.  Min Bathrooms: Determines the minimum number of bathrooms of all properties.  Max Bathrooms: Determines the maximum number of bathrooms of all properties.  Avg Rooms: Determines the average number of rooms per property.  Sum of Rooms: Determines the sum of all rooms of properties.  Min Rooms: Determines the minimum number of rooms of all properties.  Max Rooms: Determines the maximum number of rooms of all properties.  # of Records: Determines the number of records for clients. |

# Task 5 – Input with validation, processing, output

## Task 5A – Input

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input – NAME OF FORM** | | | | | |
| **General** | | | **Validation** | | |
| **Source (GUI Component)** | **Data Type** | **Format – what it looks like** | **What** | **Method** | **Feedback** |
| edtClientName | String | Plain Text | Checks if the yet to be created client’s name is valid | Checks if the string is empty or null. Loop through every character and check if the string contains any digits. | If the check failed, it displays an appropriate error message (e.g. Information entered was invalid. Please retry.) |
| edtClientAge | Integer | Integer | Automatically validates the input | The SpinEdit makes use of a MinValue and MaxValue. This ensures the input will always be correct. |  |
| Text File | String | Plain Text |  |  |  |
| cmbPropertyHasParking | Boolean | Plain Text | Automatically validates the input | The combobox only has two option. The user is forced to pick either, thus the input will always be correct. |  |

## Task 5B - Processing

|  |  |  |  |
| --- | --- | --- | --- |
| **Processing** | | | |
| **Nr** | **Name of Process** | **Description** | **Algorithm** |
| 1 | populateClientCombo | Adds all Client ID’s to a combobox on the Stats form. | Loops through very record in the Clients table. For every record it finds, it adds the corresponding ID to the ComboBox’s items. Since all ID’s are unique, there is no need to check if it is already added. |
| 2 | createDynamicPanel | Initializes every dynamic component on the form. It also creates the base panel all dynamic components appear on. |  |
| 3 | estimateMarketValue | Calculates the estimate value of the property according to fixed prices. | The estimate has a base value of R1 million. First it retrieves the number of rooms and bathrooms for the property. For every room it adds R150 000 to the estimate. For every bathroom it adds R100 000 to the estimate. If the property has indoor parking it adds R200 000 to the estimate. Finally, it returns the estimate. |
| **4** | populateDynamicArray | Populates the dynamic array with property objects. (array of objects) | Loops through every property in the properties table. For every property it creates a new property object with its values. Change the length of the dynamic array to fit all the objects. Append the object to the dynamic array. |
| 5 | getSHA256 | Returns the SHA256 hash of a given string. This is used when checking login information. |  |
| 6 | readFile | Reads the credentials text file. | Attempts to read the credentials.txt file. Upon reading, it splits the lines up and adds everything to a specific array. One of these arrays are for the username + password hash combos. The other is to check if the user account is an admin or not. |
| 7 | writeFile | Writes to the credentials text file. | Attempts to write to the credentials.txt file. It simply appends whatever is passed as parameter to the file. In this case it would be the account information. |
| 8 | valueSum | Gets the collective value of all properties owned by a specific client. | Loop through every object in the array of property objects. If the Client ID selected in the ComboBox is the same as the Client FK of the property (this property is owned by this client), add the value of the property to a sum. Finally, print the collective sum. |

|  |  |  |
| --- | --- | --- |
| **Output** | | |
|
| **GUI component** | **Format – what does it look like** | **Data - What** |
| Rich Edit  (Stats Form) | Plain Text | Display generated statistics from the stats form. |
|
| Dynamic Panel  (All applicable forms) | Plain Text | When the user inputs invalid information, an appropriate error message will be displayed. |
| DBGrid  (Agents, Clients +  Properties  Form) | Grid/Table | When the user creates, updates, deletes or sorts the table |
| Property Value Edit  (Properties Form) | Currency | Displays the value of the property as a currency. Converts from float to currency string. |

## Task 5C - Output