DEPARTMENT OF ECONOMICS
AND BUSINESS ECONOMICS
AARHUS UNIVERSITY

EXAM ASSIGNMENT

Study Programme and level	**MSc Operations and Supply Chain Analytics + elective**						
Term	Winter 23-24r						
Course name and exam code(s)	Tools for Analytics						460202E016
Exam form and duration	Written exam, WOAI						3 hours
Date and time	19 February 2024						09.00-12.00
Supplementary material/aids	All	X	Specified		No		
Hand-in of extra material (appendix) in WISEflow allowed	Yes	X					
Hand-in of hand-written material allowed	Yes		No	X			
Anonymous exam	Yes	X	No		Please do **not** write your name or student ID number anywhere. Use your flow-id number (find this on the cover sheet in WISEflow).		
Other relevant information	**Avoid being suspected of exam cheating** Remember to state references and use quotation marks, if you copy text from other sources or re-use parts of a previously submitted exam paper (plagiarism and self-plagiarism). Students must answer the exam assignment **individually**. All submitted exam papers are checked for plagiarism, so cheating and collaboration between students will be detected. Please note that use of AI in exams at AU is not allowed.						
Number of pages (incl. front page)	6 pages						

Practical information

- This exam is with internet. You may use any on-line or book-based resource you would like, but you must include citations for any code that you use (directly or indirectly). You may not consult with anyone else during this exam. That is, you cannot ask direct questions on the internet, use answers given on the internet after the start of this exam, use large language models (such as ChatGPT), or consult with each other, not even for hypothetical questions.
- This assignment has an appendix available for download from WISEflow.
- Please note that the weights on each assignment are only guideline weights, and that they only provide information regarding the relative weight of the assignments. The final evaluation will be given based on the total material handed in.
- If you find that some information is missing in the assignments, you may make the necessary assumptions and clearly specify these.
- Handing in: You must hand in a single Excel macro enabled file (`.xlsm`) as "Appendix material". Due to the system, you must also upload an empty PDF document named `yourFlowId.pdf`.
- An Excel template file is given in the appendix that you should use as a starting point. Rename it to `<your flow id>.xlsm`.

VBA - Assignment 1 (60%)

Consider the Excel template file provided. The file contains different datasets in worksheets starting with *Data*. You may assume that all datasets have a header in the first row (starting in cell A1). Each column contains either doubles or strings. If the header title of a column starts with *str* then the column contains strings; otherwise the column contains doubles.

A dataset can be stored in two arrays where one store the column headers and one the data. For instance, the following code in VBA read the data in worksheet *Data1* into two arrays:

```
Sub ReadData1()
   Dim aryData As Variant
   Dim aryHeaders() As String
   Dim rng As Range

   ThisWorkbook.Worksheets("Data1").Activate
   Set rng = Range("A1:" & RngGetCurRegionLastCol(Range("A1"), asLetter:=True) & "1")
   Call AryRead(aryHeaders, rng)
   Set rng = RngGetCurRegionRange(Range("A1"), row:=2, col:=1)
   Call AryRead(aryData, rng)
End Sub
```

The array storing the header titles contains strings. However, since we do not know the datatype of a column in the dataset, the array `aryData` is of type `Variant`. That is, we don't write `Dim aryData() As Variant` (which we normally do when defining a dynamic array), but `Dim aryData As Variant`, since a variant in itself can be an array.

Write a set of VBA procedures that answer/complete the following questions/tasks. All procedures should be coded in module *EM_Transform* and MUST be documented using a skeleton similar to the examples given in the course notes (Section D1). This module also already contains a set of procedures (e.g. `getColumnIndex` and `isColumnString`) that you may use as is during the exam.

Question 1

Given the header titles, create sub `Rename` that rename a header title with the following features:

- The function takes an array as argument containing the header titles.
- The function takes a string containing the header title to rename as argument.
- The function takes a string containing the new name of the header title as argument.
- If the header title to rename is present in an entry in the array, it is renamed (the array modified `ByRef`); otherwise the array is not modified.

For instance, if the header titles are array `("Ship", "Profit", "Cost")`, then renaming `Profit` to `Income` would modify the array to `("Ship", "Income", "Cost")`.

Question 2

Create a procedure `Arrange` that arranges/sorts a column in a dataset stored using two arrays with the following features:

- The function takes the two arrays representing the dataset (header and data) as arguments.

- The function takes a string as argument containing the header title of the column to sort.
- The sorted dataset is output/pasted to the worksheet named *Arrange* (starting in cell A1), i.e. no new arrays are needed to store the modified dataset. This worksheet is cleared if it contains old data.
- The procedure should work for different datasets stored in the arrays (e.g. the dataset in worksheet *Data1*).
- The procedure does not have any intermediate steps which use ranges to sort the dataset.

Hint: The course procedure `AryQuickSort` stored in module `ModAry` may be useful. Note that this procedure takes a `Long` as second argument.

Test `Arrange` by writing a procedure `ArrangeTest` that:

- Loads the dataset in worksheet *Data2* into two arrays (a dataset of fright handling operations).
- Renames the *delivery_cost* column to *cost*.
- Arranges the dataset based on column *cost*.
- Uses a message box to write out the meaning and value of cell E2 in worksheet *Arrange*.

Add a button running the procedure to worksheet *Arrange*.

Question 3

Create a procedure `Distinct` that finds all distinct/unique values of a column in a dataset stored using two arrays with the following features:

- The function takes the two arrays representing the dataset (header and data) as arguments.
- The function takes a string as argument containing the header title from which to find distinct values.
- The distinct values are output/pasted to the worksheet named *Distinct* (starting in cell A1), i.e. no new arrays are needed to store the new dataset. This worksheet is cleared if it contains old data.
- The procedure should work for different datasets stored in the arrays.
- The procedure does not have any intermediate steps which use ranges to find distinct values.

Hint: One way to find distinct values of a column could be to sort first and then output each time values in the column change. If you need to compare using strings, you may need to cast to a string using the `CStr` function in VBA.

Test `Distinct` by writing a procedure `DistinctTest` that:

- Loads the dataset in worksheet *Data2* into two arrays.
- Finds the distinct values of column *str_type*
- Uses a message box to write the number of different values.

Add a button running the procedure to worksheet *Distinct*.

Question 4

Create a procedure `Mutate` that adds a new column to a dataset stored using two arrays with the following features:

- The function takes the two arrays representing the dataset (header and data) as arguments.
- The function takes an array with two strings as argument containing the header titles to mutate from.
- The function takes a string as argument containing the operator to use on the two columns. The valid operators are

  - `+` : The new column is the sum of the two columns (valid if both columns are doubles).
  - `-` : The new column is the first column minus the second column (valid if both columns are doubles).
  - `*` : The new column is the product of the two columns (valid if both columns are doubles).
  - `/` : The new column is the first column divided by the second column (valid if both columns are doubles).
  - `join` : The new column (strings) is the first column joined by the second column with a separator in between (valid for all column types).
- The function takes a string as argument containing the name of the new column.
- The function takes a string as argument containing the separator used for the `join` operator with default value `" "`.
- The mutated dataset is output to the worksheet named *Mutate* (starting in cell A1), i.e. no new arrays are needed to store the modified dataset. This worksheet is cleared if it contains old data.
- If the header titles or the compare operator is not valid then nothing should be output to the *Mutate* worksheet.
- The procedure should work for different datasets stored in the arrays.
- The procedure does not have any intermediate steps which use ranges to add columns.

For instance, calling the procedure with header titles `("income", "cost")`, operator `-` and new column `"profit"`, will output the dataset with a new column *profit* which is the difference between columns *income* and *cost*.

Test `Mutate` by writing a procedure `MutateTest` that:

- Loads the dataset in worksheet *Data1* in two arrays.
- Creates a new column *total* which is the product of columns *price* and *quantity*.
- Uses a message box to write out the meaning and value of cell I2 in worksheet *Mutate*.
- Given the original dataset, create a new column *ship_w_desc* which is obtained by joining columns *ship* and *str_item_desc* seperated by `" - "`.

Add a button running the procedure to worksheet *Mutate*.

VBA - Assignment 2 (40%)

Consider a simple game using an uneven dice with outcomes 1-8. The outcome D of the dice follows a custom discrete distribution (unknown to the player):

$$\Pr(D=x)=\begin{cases}0.1, & x=1,2,4,5,7\\ 0.3, & x=3\\ 0.15, & x=6\\ 0.05, & x=8\end{cases}$$

The player of the game bet on an outcome x, next he throws the dice 6 times and wins if the dice hit x at least one time.

Write a set of VBA procedures that answer/complete the following questions/tasks. All procedures should be coded in module *EM_Game* and MUST be documented using a skeleton similar to the examples given in the course notes (Section D1).

Question 1

Create a function `PlayAGame` that returns true if the game is won given argument x (the number to bet on); otherwise false is returned.

Question 2

Create a procedure `Play` with the following features:

- Use an input box to ask for the number x to bet on.
- Play the game 4 times and store the results of each play in a collection.
- Use message box(s) to output which games you won or that you did not win any games.

Add a button running the procedure to worksheet *Game*.

Question 3

If you bet on number x then it costs you $x/2$ and if you win then you receive x. That is, the profit is

$$\pi = \begin{cases} -x/2, & \text{if loose} \\ x - x/2, & \text{if win} \end{cases}$$

Make a sub `ProfitSim` that does a simulation with the following features:

- For each bet x calculate the average profit based on 1000 runs.
- Output the results in worksheet *Game* so one row is given for each bet x with the bet x and average profit.

Add a button running the procedure to worksheet *Game*. Which bet gives the best average profit (add this as a comment inside the procedure)?