**3-1– Level Up your Apps & Macros** 

In the beginner course, we have discussed the different interface tools that can assist you while creating an app or a macro. We also tackled how to create a basic or standard macro from an alteryx workflow as well as how to convert the input and output tool to macro input and output. For this lesson, we are going to discuss how to take your macros to the next level by discussing different ways of manipulating user inputs and integrating them into the macro/app process. In addition, this lesson will also guide you on creating your own iterative macro.

**An Iterative Macro will run through every record and then loop the records back through the workflow, repeating the entire process as many times as is specified, or until a condition is met. Iterative macros are useful for any process that needs to be repeated a certain number of times but does not necessarily match an input list. This is great for data sources which does not have a predetermined amount and workflows that requires user input to determine the number of looping.**

In simple terms, an iterative macro can be likened to a person who handles inventory stocks. The inventory personnel do not know the exact number of customers that will visit and how much products they will order so they will have to go back to the store room and check how many items are available every time a customer orders from the store. The personnel will need to do this over and over again until the stock runs out. If you convert this job to a macro, each customer order would be a single **loop**, and the **condition** to stop the loop is if product stock = 0.

Let’s try to create an iterative macro that will calculate the total payout for a one-time payment investment package. We will set 3 packages, Package A will have 2% interest for 15 years, Package B will have 4% interest for 10 years and finally, Package C will have 5% interest for 5 years. The user will be one to indicate the annual payment amount and package they wish to invest in. then the process will calculate its annual interest then add it to its value and loop until we reach the maximum term year.

To start off, drag a text input into the canvas. Type a new column called “Payment”. For its value, type “5000.00”. Next, convert this text input to a macro input by clicking right on the tool and selecting “convert to macro input”. This will serve as the sample data for our macro. On the configurations of the input macro, lets retain the name called “Input1” then check the box for “Show Field Map” so we can map it later. Next, connect a select tool after the macro input. Our Payment field should have the data type “double” since we are going to reuse this field to show the payout later. Next, add a formula tool for the interest calculation and the allocation of years. We will create 3 new fields on this tool as well as a new calculation for the Payment field. First, create a new column called “Package”. Since we are creating a macro, we need to add dummy data to this new field and then replace them later using an interface tool. Add the dummy data by typing in “A” enclosed in single quotes. Set the data type to V\_WString with size 10. Before we add the other calculations, lets add an interface tool so our user can choose their desired package. Drag a Drop-Down tool from the interface palette and place it above the formula tool. Connect the Q anchor of the dropdown tool to the lightning bolt icon of the formula tool. This should automatically create an action tool between them. Next, we need to display the available packages to the user. On the configuration window of the dropdown tool, the first textbox sets the text or question that will be displayed. Maximize this option by adding clear instructions and notes to your users. Since we only need them to select a package, type “Choose a Package:”. Next, we need to specify the name and values that will be shown inside the dropdown. You can add these values using different means, such as importing from an external source, data from the macro input and other types of alteryx supported sources. We will use the option to “Manually Set Values” so we can type in the package options directly in the multi-line textbox. First, type in the name (or the label for each package), then a colon symbol, and its respective value. Having a separate name and value allows us to create an informative label for each package option and assign a simple text value so we can easily look it up on our calculations later. In the multiline textbox, type “Package A (5% 15 Yrs):A” then press enter to go to the next line then type the second option “Package B (10% 10Yrs):B”. Once done, press enter to go to the next line and type our last option “Package C (15% 5Yrs):C”. As we can observe, our package labels are pretty long but we assigned a single letter for its value so we can easily type it on our calculations later. Below the list values is the option to set a Default Selection Name. This will be the default option for our dropdown. You can type in a label from one of the options we’ve listed or simply type a partial name; for example, type in “Package A”. This will still direct the default to our first option even though we only type the partial label. You can also leave this blank if you don’t need any default values.

In order to incorporate the value of the dropdown to our macro, we need to configure the action tool to replace the dummy value in the formula tool. There are 3 action types you can choose from. Update Raw XML with Formula, Update Value, and Update Value with Formula. Since we simply need to replace the dummy value, choose “Update Value”. Since the action tool is connected to the formula tool, its name is shown with dropdown to the elements that it contains. Open the formula field to see which values we can replace. Here we can replace the expression, field, size and type. We only need to replace the dummy data, so choose the line for “@expression – value = “A”. Next, indicate which part of the expression should be replaced. Check the replace a specific string function to enable the textbox. Finally type “A” on the textbox. This will remove the dummy A data with that of the package value that the user will specify using the dropdown.

Now that we have the interface tool in place, go back to the formula tool and add a new column called “Interest”. Type the following expression ”Switch([Package],0,nextline 'A',0.02, nextline 'B',0.04,nextline 'C',0.05)” This function will simply compare the Package value then assign its interest rate. Set its data type as Double. We will do the same for the term year. Add a new column and name it as “Years”. For its expression type “Switch([Package],0,nextline 'A',15,nextline 'B',10,nextline 'C',5)”. The year value will also depend on the package that will be selected by the user. Set its data type to INT16. We wanted to calculate the total payout by getting the total investment with its accumulated interest, so this field will be the basis for the number of looping or the number of times that the macro will run. Finally, update the value for field Payment. Set it to “[Payment]+([Payment]\*[Interest])” Notice that we added the “Payment” value again on the first part of the calculation. This is done in order to capture the payment value from the last run iteration and add it to that of the interest. If we did not add the “[Payment]+” calculation on this formula, the iterative macro will simply output the first payment value plus its first year interest, then disregard the rest of the years which makes the loop useless.

Next, configure the workflow to change its type to an iterative macro. Click a blank space on the canvas then navigate to the workflow configuration tab. Under Type, select macro, then select “Iterative Macro” on the dropdown. This enables us to use the **Iteration Number** constant which would be the total number of runs. Going back to our canvas, we need to specify the condition to stop the loop. To do so, add a filter tool from the Preparation Palette and connect it after the formula tool. Add a custom filter on its configuration. Type “([Engine.IterationNumber]+1) = [Years]” We added plus 1 to the engine iteration number since the iteration’s starting value is 0. You can also add other constants to other expression by clicking the variables button and choosing a constant listed on the window. So, if it is TRUE that the Iteration number plus 1 matches the field “Years” or the maximum term year of the investment it should trigger to end the loop. But if FALSE, the values will loop back to the start or the first tool. Connect a macro input to the False anchor of the filter tool. We need to specify an Output Name since this will be considered as the iteration data stream and its data will flow back to the starting tool. Plus, we need to track its name for mapping later. Input the Output Name “Iteration” then set its anchor to “I”. In order to visualize how the loop works, place this macro output near the macro input tool.

Now we can finally set the output schema for this macro. On the TRUE anchor, connect a select tool and rename the field “Payment” to “Payout”. Since we are on the data stream that satisfies the condition that we’ve reached the loop for the max year, the Payout value should contain the Payment plus all of the accumulated interest throughout the years. As the last tool on this macro, drag a Macro Output and connect it to that of the select tool. Set the Output Name to “Data Output” and indicate its anchor abbreviation as “O”.

Before saving this macro, navigate to the Interface designer tool and click the tree view button. Push the Macro Output (9) above the Macro Output (8) so that the O anchor is on top of the I anchor or iteration anchor. Then go to the Interface designer properties tab to map the appropriate fields. Under Iteration Input, map it to our macro input named “Input1”. For the Iteration Output, map it to “Iteration” which is the macro output for the data stream that did not satisfy the condition. You can also set the maximum number of loop or iteration and determine what happens if the max value is hit. Since we have 15 Years as our maximum, type 15. Then select “Error” if it reached the maximum number. For the output mode, select “Auto Configure by Name (Wait until all iterations run)” Records are unioned on output and this will set the behavior for outputting these records as some may have a different table schema. Finally, lets make our macro unique by adding a custom icon. Select the radio button for custom Icon and browse to the file location of the image and click “Open” to set it as the macro icon. Now that we have all of the things ready, save the macro to your macro folder. We will save this as “3-1-iterative-investmentinterest”. This will be saved with the file extension “.yxmc” to signify that it is an alteryx macro.

Finally, lets use the macro on the workflow. On a new workflow, add a text input with a column called “Payment” and value of “10000.00”. Add a select tool then make sure that its data type is double. Next, insert the iterative macro we’ve created. Click right on the canvas and select insert > Macro. Our macro should be visible on the list but if not, click browse to navigate to the location where you’ve saved it. Configure the macro by mapping the Payment field and selecting a Package. Let’s choose Package B with 4% interest for 10 Yrs. Then run the workflow. The total payout for package B would be “14802.442849” since we invested 10k for 10 Yrs. Meaning, the iterative macro also looped for 10 times to get the accumulated amount.