Operations Guide 2024

Document version: January 2024

E-mail: info@avvoka.com | Website: www.avvoka.com | Telephone: 020 3865 5899
Registered in England & Wales with no.: 09729807 | Registered office: 2.05 12-18 Hoxton Street, London, England, N1 6NG | VAT number: 234 6111 39

Avvoka

Contents

Avvoka Operations Guide

1. Background

1.1. Introduction

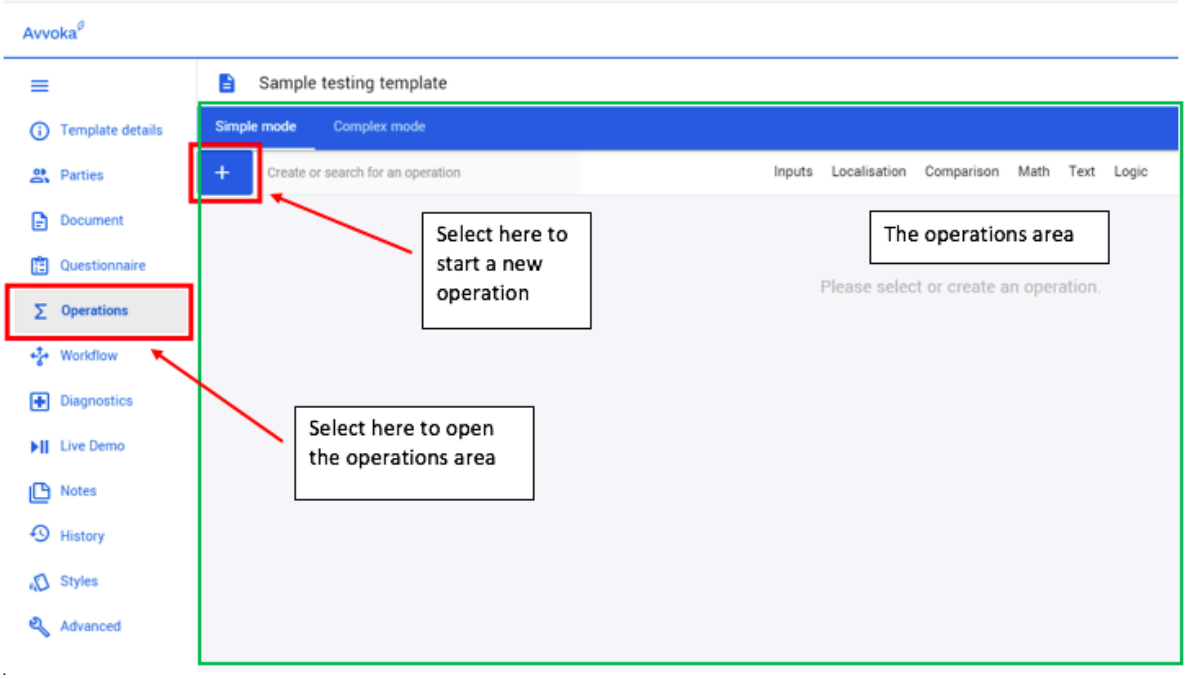
1.1.1. This operations guide will explain each operation and their function.

1.1.2. Operations reformat and repurpose information entered into the questionnaire. They present an opportunity to stretch the content of those answers into broader characterizations. By layering different rules for the information to flow through, operations have the potential to create highly customised outputs.

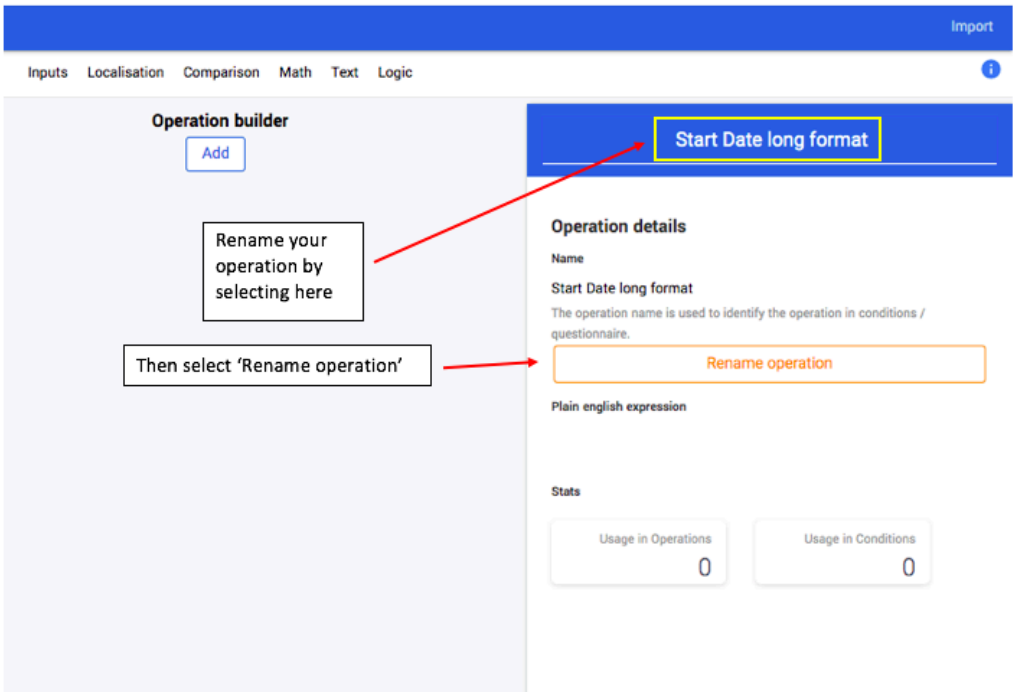
1.1.3. For more help on using Operations, contact help@avvoka.com.

1.2. Creating an Operation

1.2.1. To start off, navigate to the Operations tab in your template. Then, click the blue + button.

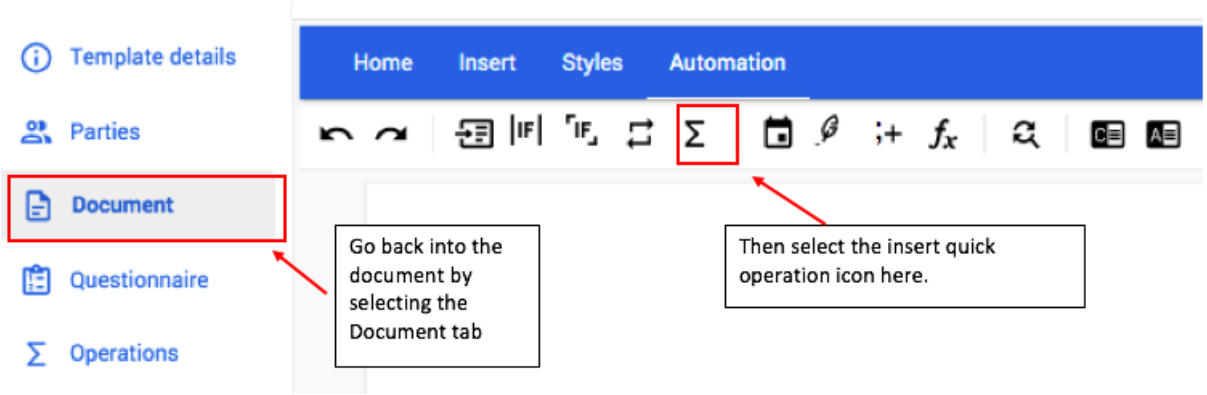


1.2.2. A pop up menu will appear on the right. Rename your operation to your desired name and click "Rename operation".

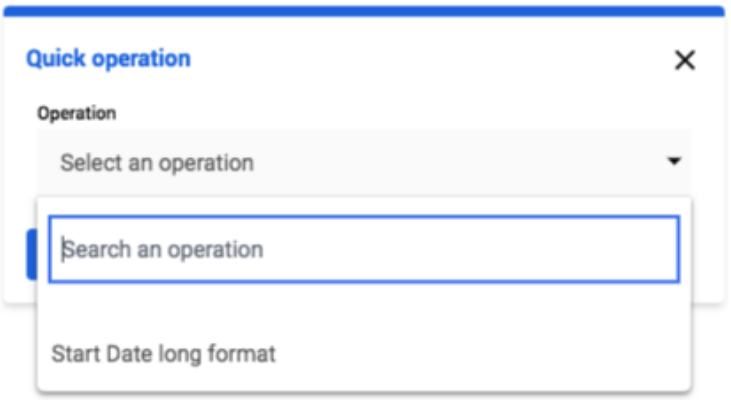


1.2.3. After you have done so, click the "Add" button to start creating a new operation.

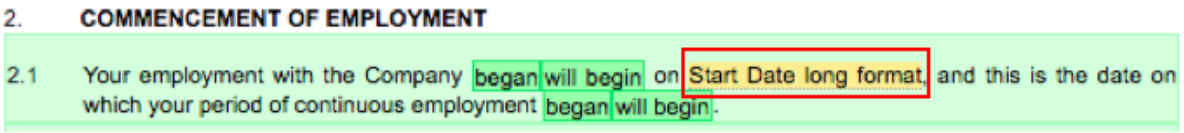
1.2.4. To add an operation to a document, navigate back to the "Document" tab and click on the document body where you would like to include the operation. Select the Automation tab in the top menu and click on **Σ**.



1.2.5. Selecting this icon will prompt a window, enabling you to select the operation in the drop down list. Then, select the operation you have created and click insert to add it to the document.



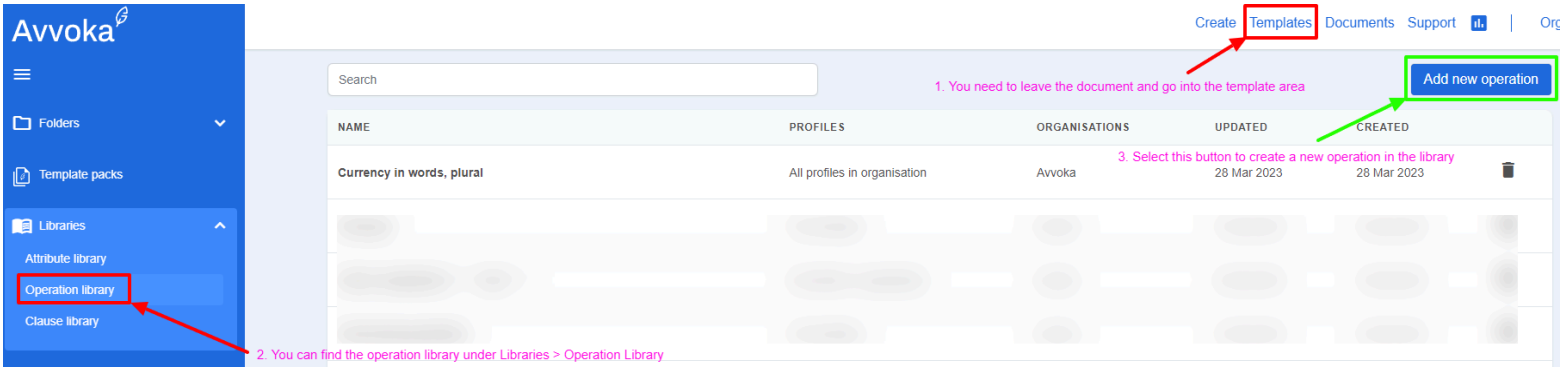
1.2.6. The operation will appear in the document as yellow highlighted text.



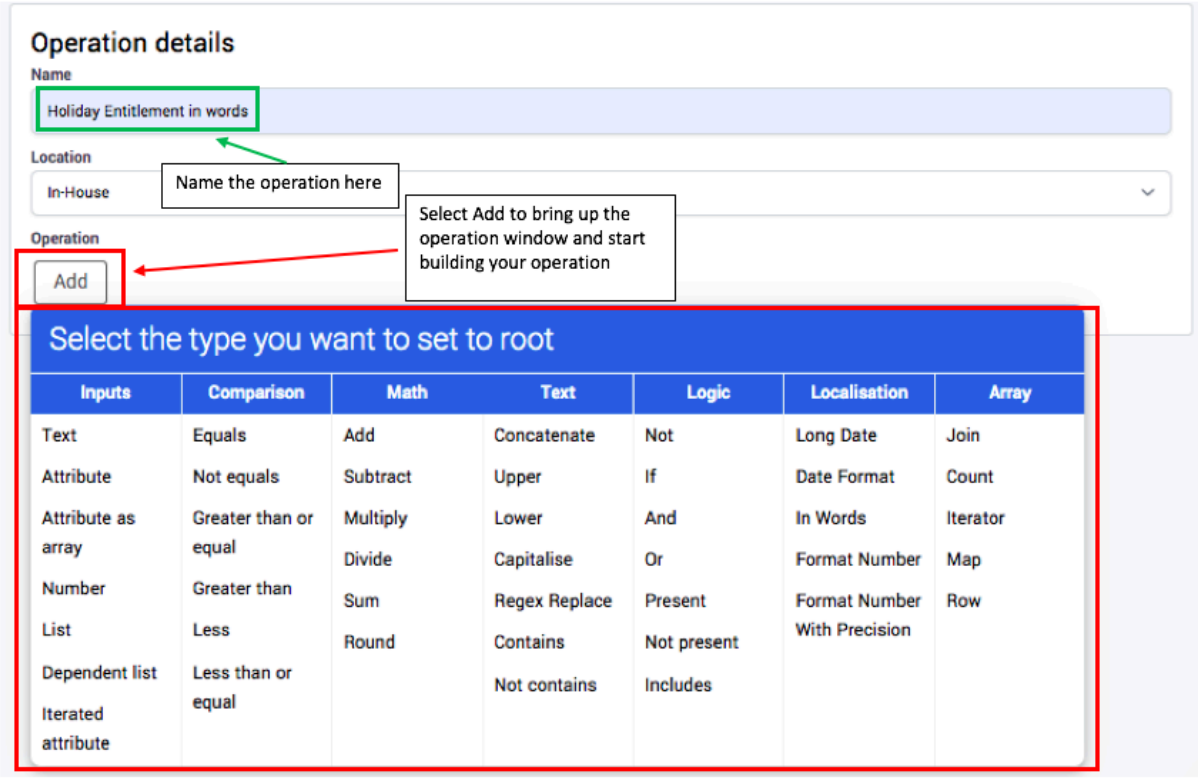
1.3. Operation library

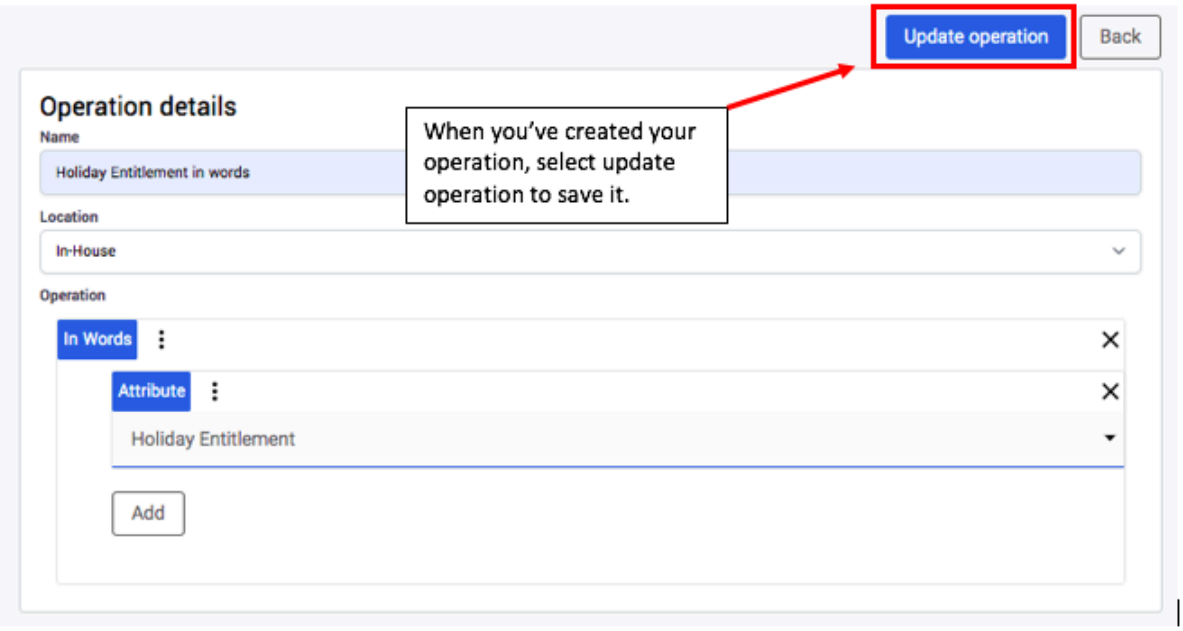
1.3.1. Users can create and save Operations in the Operation library.

1.3.2. You can access the Operation library by going into the templates area, and selecting Libraries > Operation library in the left panel. To create a new operation in the library, click ‘add new operation’.

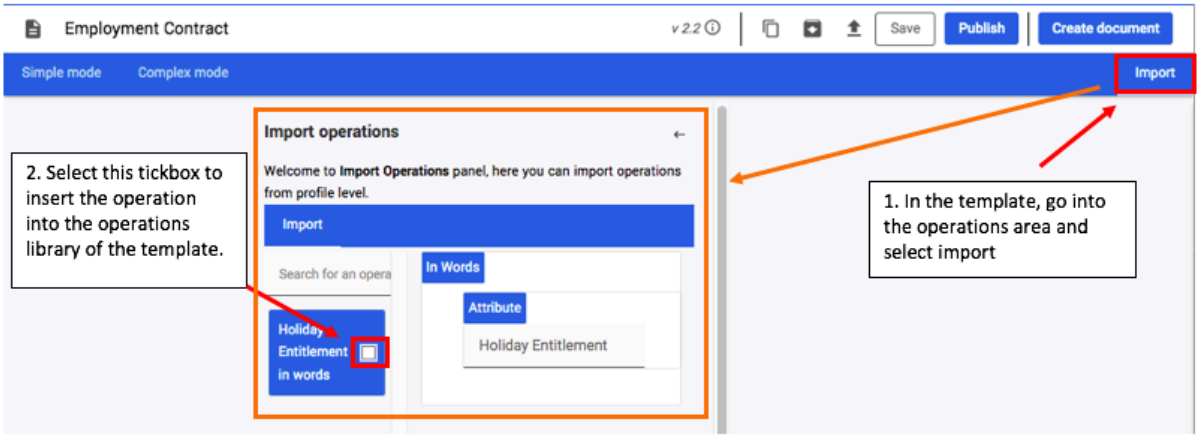


1.3.3. Press the “Add” button to start building your operation. Once you have completed building the Operation, click “Create Custom Operation”.





1.3.4. To apply the saved Operation, go into the relevant template and select the Operation tab. Click the "import" button at the top right of the screen and tick the checkbox next to the saved Operation you wish to import. The operation will now appear in your Operations tab.



1.4. Explanatory note

Connective Inputs

1.4.1. Connective inputs are the nuts and bolts of an operation. They serve as the essential mechanisms of the rules, or logic that construct the operation. As a result, connective inputs are present in almost every operation. Using the information entered into the questionnaire, connective inputs steer the transformation of that information.

1.4.2. Generally, connective inputs dictate the rules of an operation. 'Comparisons', for instance, are a clear example of how you can use connective inputs as the driving arguments for the operation. You would use these arguments (e.g. equals, not equals, less than etc.) to compare values against one another.

1.4.3. Other examples of connective inputs are "Inputs" and "Logic". Ultimately, they all collectively represent the method of controlling the output of an operation by bringing the relevant information together.

Functional Inputs

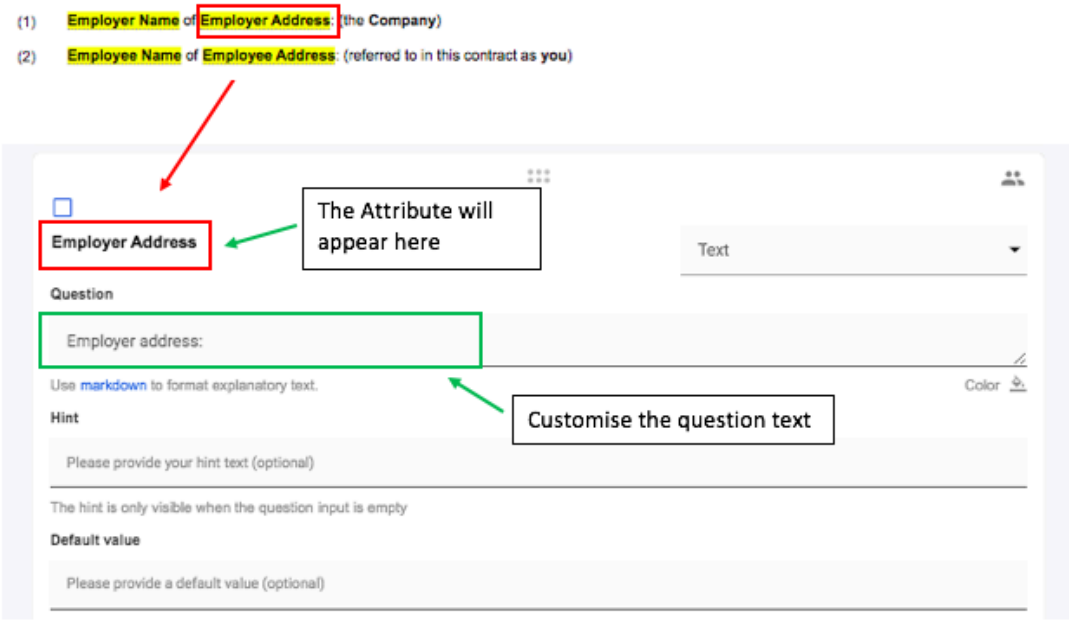
1.4.4. Functional inputs have a more defined use. While connective inputs identify the information or logic, functional inputs create that change.

1.4.5. Examples of functional inputs are localisation (long date, in words etc), text, and maths. These transform the information into defined formats, such as a number into a word or calculating a sum. In complex operations, it is possible to layer multiple functional operations together. An example could be creating a mathematical calculation and then transforming the numerical value into its word format (localisation).

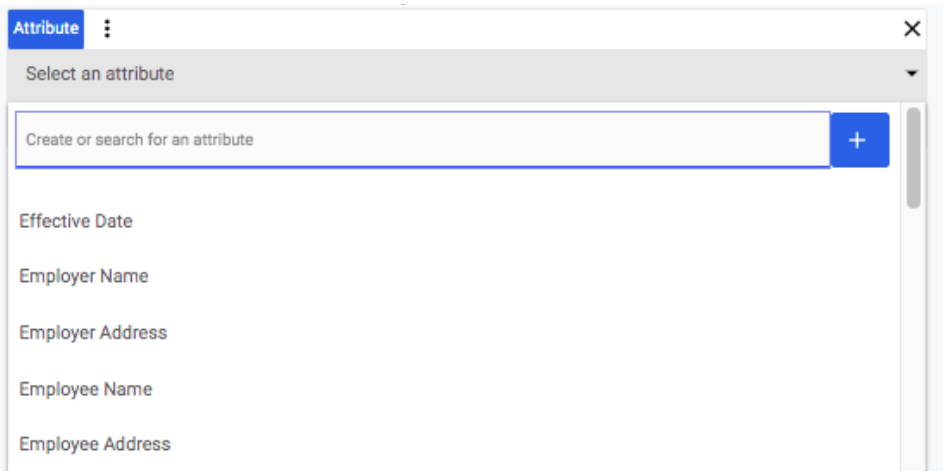
1.4.6. Using these two inputs in a single operation, connective inputs identify the relevant information, and the functional inputs create the transformation (e.g. identify the date attribute (connective) and transform it into a long date (functional)).

2. Connective Operations

2.1. Inputs

Text	Text allows a user to enter a custom, static text which can be used within the operation. The Text input serves different purposes in different operations. For instance, it could be used to search for a word, replace text or adopt language using a locale code. Therefore, it is perhaps best to understand the Text input in the context of those operations. 
Attribute	Each attribute is tied to a question in the questionnaire. The title of the attribute is visible in the questionnaire above the question text (the text highlighted in red below). These attributes can be created through placeholders or any conditional automation such as block conditions and inline conditions.  Selecting the Attribute button in the Operations interface will display a drop down list

of all the existing attributes in the document (as shown below).

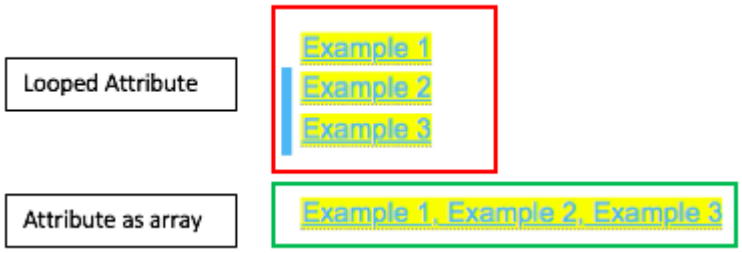


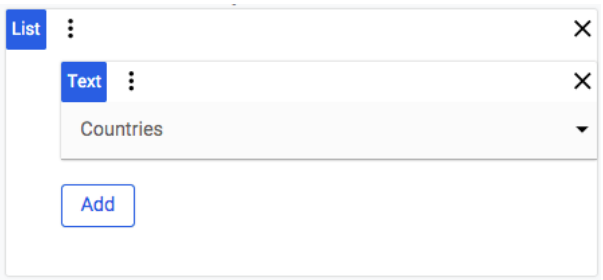
Since an attribute is intrinsically linked to a question in the questionnaire, attributes are used in operations to link the question-answer to the operation.

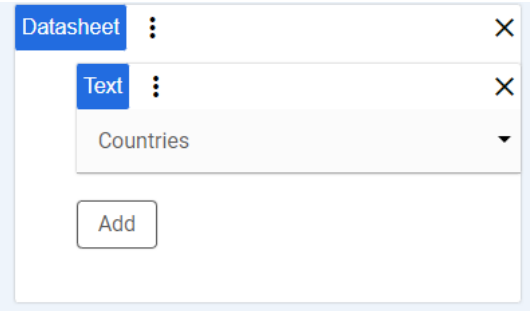
In the capitalise operation below, the attribute “Employee Name” is used to ensure the first letter of each word is capitalised. If I answer “JOE BLOGGS or joe bloggs” to the question ‘Employee Name’, the operation will take the question-answer and modify that to be “Joe Bloggs Or Joe Bloggs”.

Pro tip: You can also create new attributes using an operation. To do this, write the name of the attribute you would like to create in the Attribute box and press the ‘+’ button. As a result, a new question will be created in the questionnaire. This is helpful if you would like to create a question without adding a placeholder in the document.

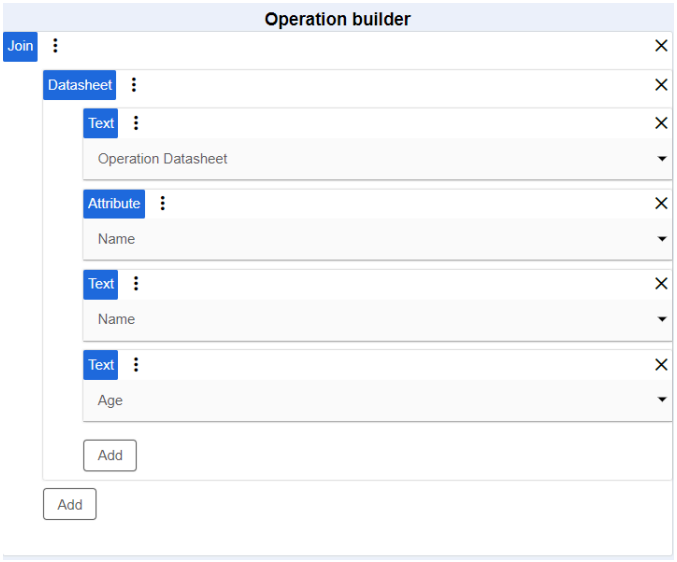
A key example of this would be when you want a date to appear as a long date, and do not want the short date format to show in the document. The solution is to create the “Date” attribute in the Long Date operation. This will create a new question in the questionnaire without needing to place a “Date” placeholder in the document. This is explained in more detail below.

Attribute as Array	**Specific to loops**: Attribute as Array is specific to placeholders which are placed within a loop. By default, looped attributes display stacked on top of each other. Using Attribute as Array, we can display these attributes so they display on a single line.  This is especially useful if there are multiple outputs being returned from an operation. Ultimately, the input enables the user to structure the data returned by the operation. See this in action below. Attribute as array is similar to the attribute function, in that existing attributes can be selected, or new attributes created using this operation.
Number	This allows a user to enter a custom, static number which can be used throughout an operation. The Number functionality is especially useful whenever using mathematical calculations.
List	Avvoka's List functionality allows users to upload lists. These are used to create a dropdown list of answer options within the Questionnaire. A list must be created before it is used in an operation. You can create a list by going to the template area and selecting in the left panel Data > List. Used in an operation, the list defines a set of values. 

	When referring to a list, select the List input and the Text input, to write the name of the list. In the example below 'Countries' is the name of the list (referring to the list shown above).  In a service contract, or a due diligence report, the List input can be used to compare an answer against values provided in a list. For example, you could have a list of high risk services, or countries. We can ask a question, 'what services do you provide?', and if the answer is included in the 'high-risk services list', return a value saying it is high risk. See our worked example here.
Datasheet	Avvoka's Datasheets functionality enables users to create a database at a profile or organisation level. Users can define custom column names and add a new row (or record) of data. Datasheets can then be used to create a dropdown list of answer options within the Questionnaire. Datasheets work similarly to lists and dependent lists. A Datasheet must be created before it is used in an operation. You can create a datasheet by going to the template area and selecting in the left panel Data > Datasheets. When referring to a Datasheet, select the Datasheet input and the Text input, to write the name of the Datasheet. In the example below 'Countries' is the name of the Datasheet (referring to the Datasheet shown above).

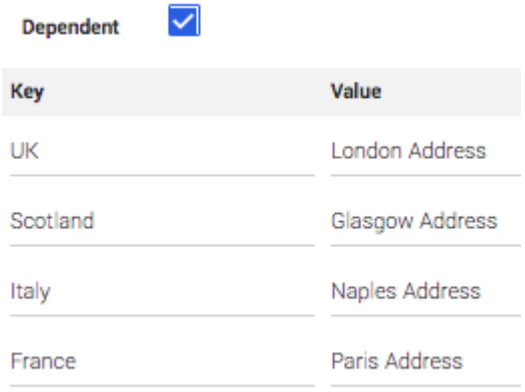
<table>
<tr>
<td></td>
<td>


You can build dependencies out of datasheets using this operation. If you have a datasheet with the columns ‘Name’ and ‘Age’, you can use this operation to output the Age data without needing an associated question:



The arguments above refer to:
Text - the name of the datasheet.
Attribute - the name of the dependent attribute.
Text - the dependent column header in the datasheet
Text - the name of the column in which the desired output sits

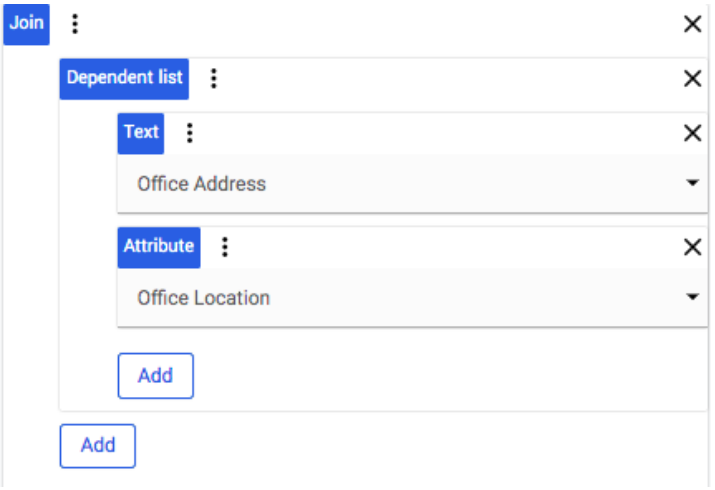
Read more about Datasheets in our User Guide.
</td>
</tr>
<tr>
<td>Dependent List</td>
<td>Dependent Lists use related data to narrow down a list of options based on a previous answer. Using a dependent list could mean that answering question A differently would vary the list of options for question B.</td>
</tr>
</table>



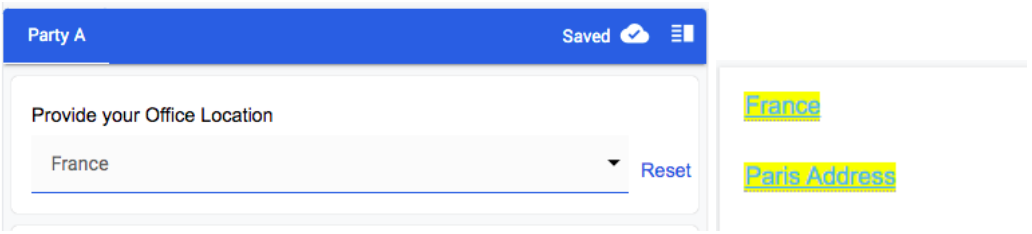
For example, if you are completing the questionnaire for an employment agreement, it would be useful to automatically populate the office address dependent on the selected country / city. E.g. if I selected the country to be the UK, it would show the UK office addresses.

A dependent list has two inputs, the first is the Key, (i.e. the country) and the second input is the value (i.e. office address).

Pro tip: If there is only one value to a key, if I select 'UK' for Office Location, 'London Address' will automatically populate the answer to Office Address. *(UK (key) and London Address (value)).*



This example uses the 'Join' array. Join is discussed here

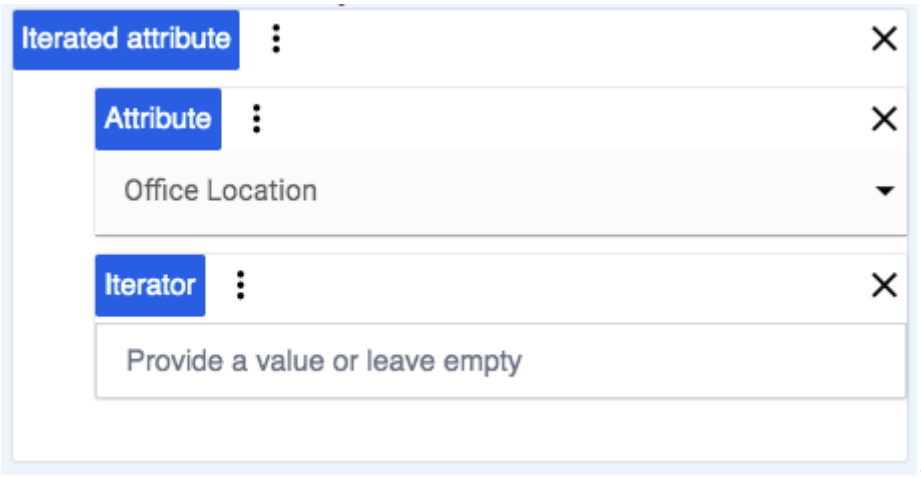


Watch this **video** on creating a dependent list.

Iterated attribute

Specific to loops: An iterated attribute refers to attributes contained within a loop. In a loop, an attribute can be duplicated to create multiple values from a single question. Each loop is referenced by an index number. For instance, if the attribute Office Location is looped 3 times, the index will be 3. The iterated attribute functionality enables you to extract the last looped value.

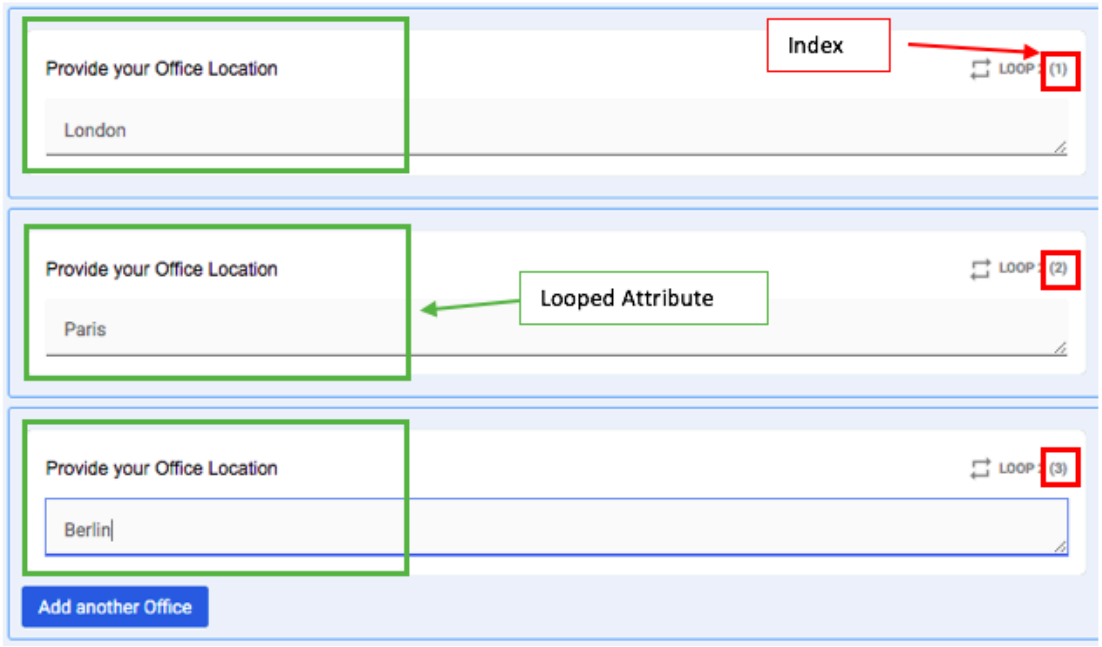
If you would like more information about loops, check out our article here or our User Guide.

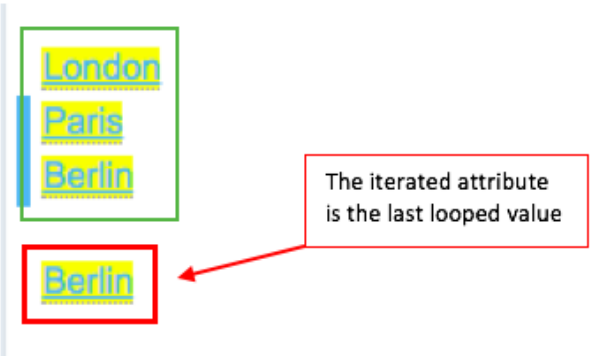


s

This function contains two arguments:

1. the Attribute being looped (note: it is Attribute, not Attribute as array);
2. the Iterator function (the index of the looped attribute).

<table>
<tr>
<td></td>
<td>




</td>
</tr>
<tr>
<td>Dictionary</td>
<td>A user is able to define concepts using the Dictionary operation. For instance, if you have an attribute named ‘seller’ then you are going to have a text: seller has but if the attribute is ‘sellers’ it will say: sellers have.

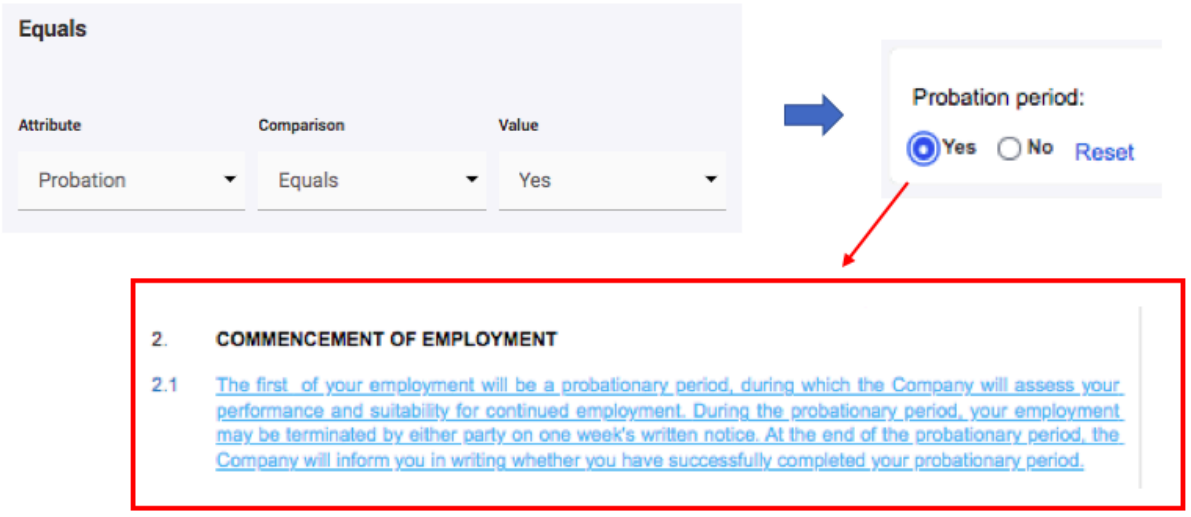
See our worked example here.</td>
</tr>
</table>

2.2. Comparison

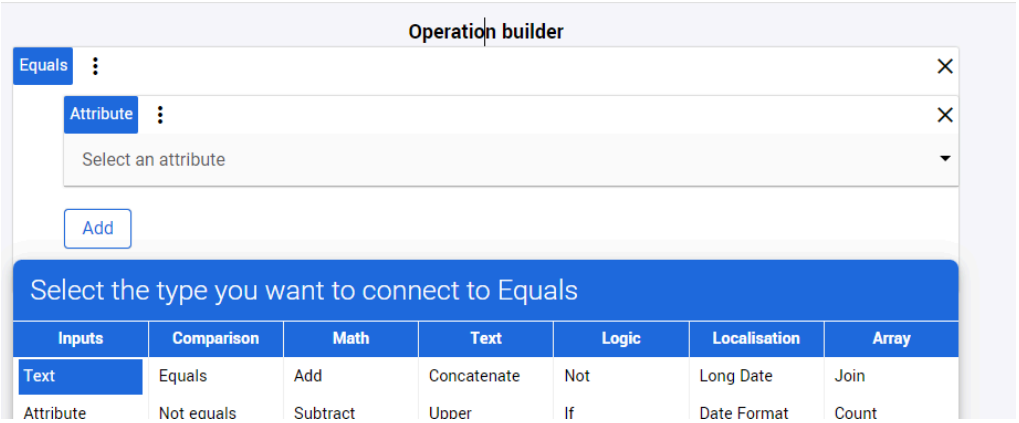
2.2.1. Comparison blocks will take two inputs, i.e. attributes or text, and evaluate if the comparison between these inputs is true. If the inputs match, the operation will output the value as "true". On the other hand, If the compared outputs do not match, the Operation will output the value as "false".

2.2.2. For example, in the below example, the comparison block will be evaluating whether the Attribute [Probation] equals the Value [Yes]. If Probation equals Yes, then the Operation will output the value "true". If Probation does not equal Yes, the Operation will output the value "false".

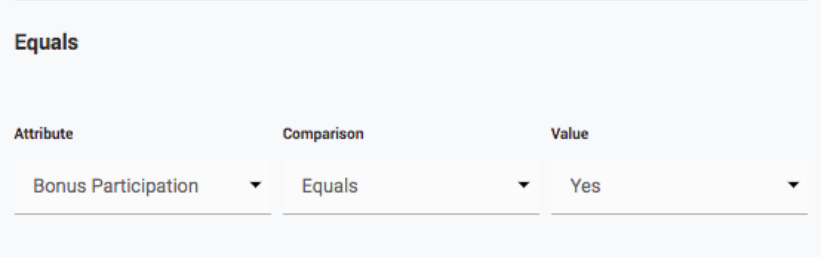
2.2.3. In an employment contract, this can be helpful as a rule triggering a condition (i.e a block condition for a probation clause). In this example, if the user answers yes to probation, the probation clause will drop into the contract.



Equals	If the compared inputs are the same, the Operation will output the value "true" and vice versa. For instance, as per the example below, if the Attribute and Value are the same, this Operation will output the value "true". Learn how to make use of these outputted "true"/"false" values below.



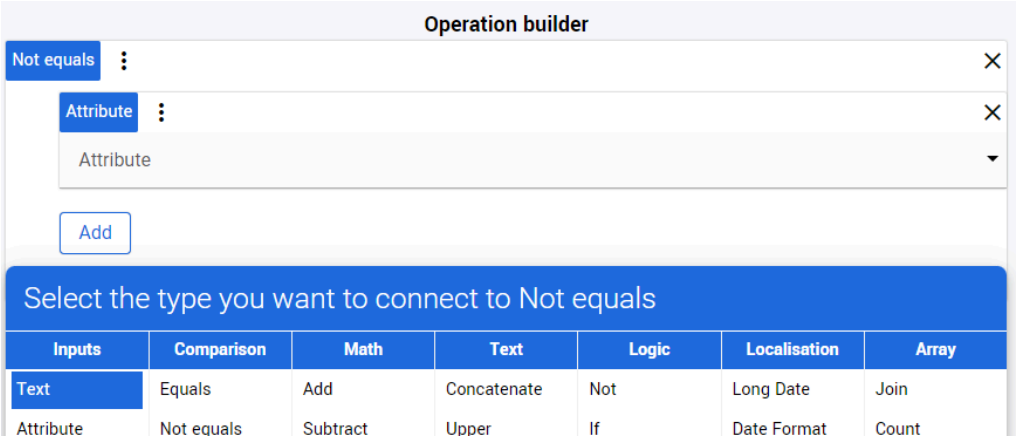
To create this operation, select the “Equals” function. Then, add an “Attribute” function and select the relevant attribute. Following that, add a Text Input. Your screen should then appear as below

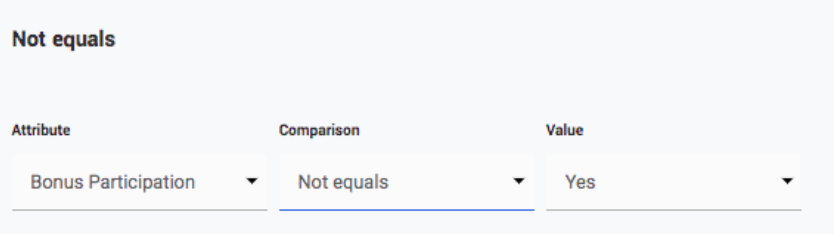
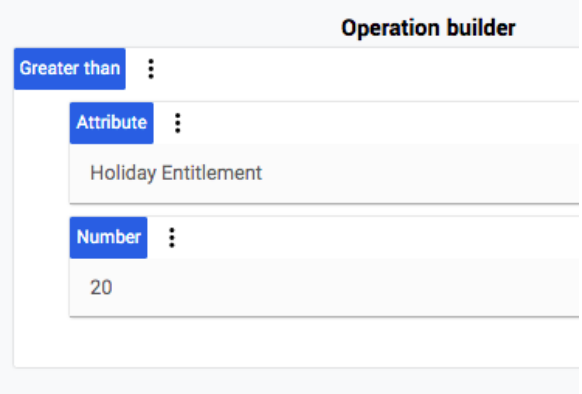


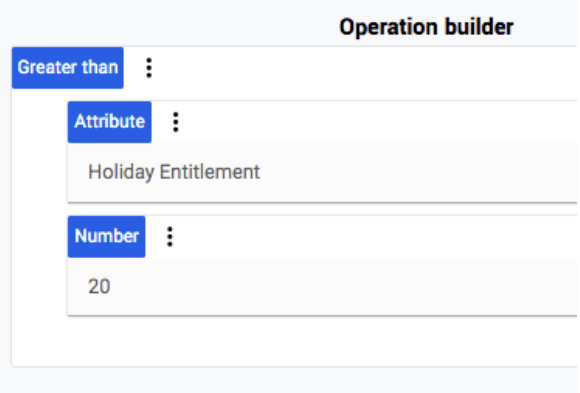
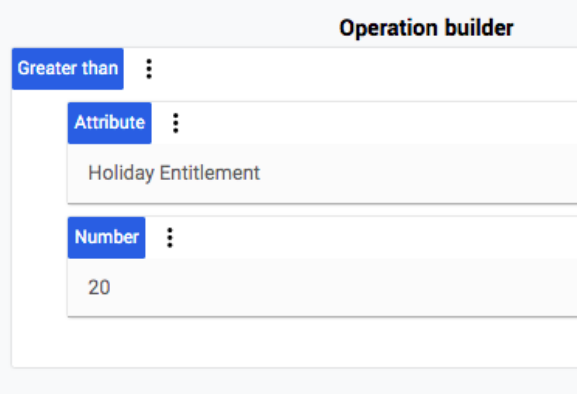
Not equals

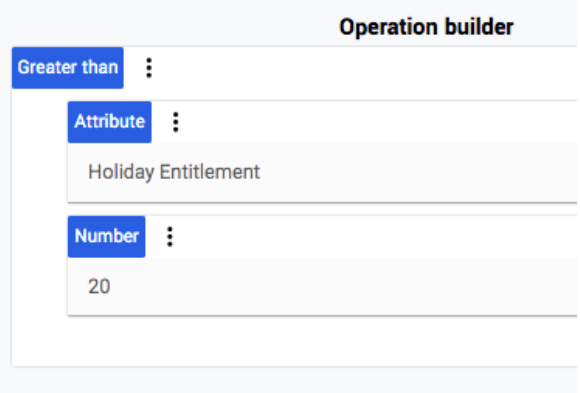
If the compared inputs **do not** match, the Operation will output the value “true” and vice versa. For instance, as per the example below, if the Attribute and Value do not match, this Operation will output the value “true”.

Learn how to make use of these outputted “true”/“false” values below.

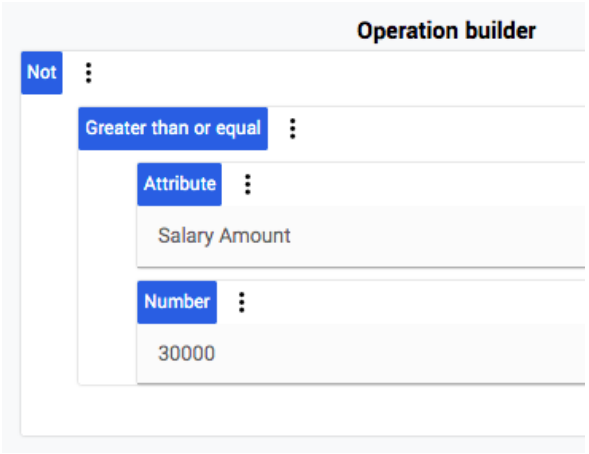


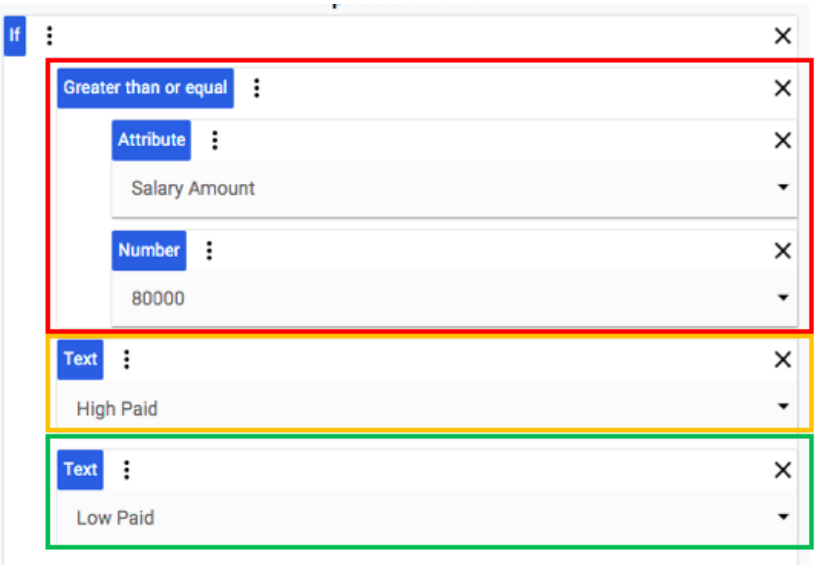
	To create this operation, select the “Not Equals” function. Then, add an “Attribute” function and select the relevant Attribute. Following that, add a Text Input. Your screen should then appear as below. 
Greater than or equals	If the value of the first input is greater than or equal to the value of the second input, the operation will output the value “true” and vice versa. For instance, as per the example below, if the employee is given 20 or more days of holiday, this operation will output the value “true”. Learn how to make use of these outputted “true”/“false” values below. 
Greater than	If the value of the first input (e.g an attribute, such as [Holiday Entitlement]) is greater than the value of the second input (e.g a number input [21]), the operation will output the value “true”, and vice versa. Using the example below, this would mean that if the employee is given 21 or more days of holiday, this operation will output the value “true”.

	 Learn how to make use of these outputted “true”/“false” values below.
Less	If the value of the first input is less than the value of the second input, the operation will output the value “true” and vice versa. For instance, as per the example below, if the employee is given 19 days or less of holiday, this operation will output the value “true”. Learn how to make use of these outputted “true”/“false” values below. 
Less than or equal	If the value of the first input is less than or equal to the value of the second input, the operation will output the value “true” and vice versa. For instance, as per the example below, if the employee is given 20 days or less of holiday, this operation will output the value “true”. Learn how to make use of these outputted “true”/“false” values below.

	 Operation builder Greater than Attribute Holiday Entitlement Number 20

2.3. Logic

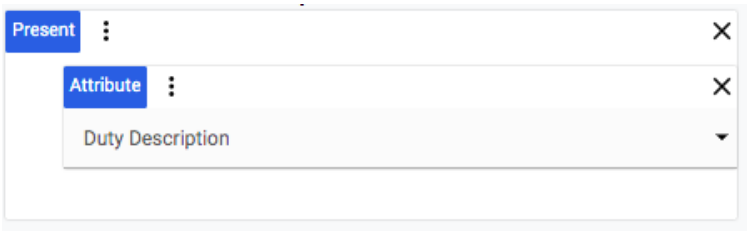
Not	The operation will output the value "true" if the value does **not** satisfy the logic, and vice versa. In the example below, the logic is asking whether the salary amount is equal to or above £30,000. Therefore, if the salary is £25,000, the operation would return a value of true. 
If	The if function takes three arguments: 1. a logical test (**the block in red**); 2. the value to be outputted if that test is true (**the block in orange**); 3. the value to be outputted if the test is false (**the block in green**).



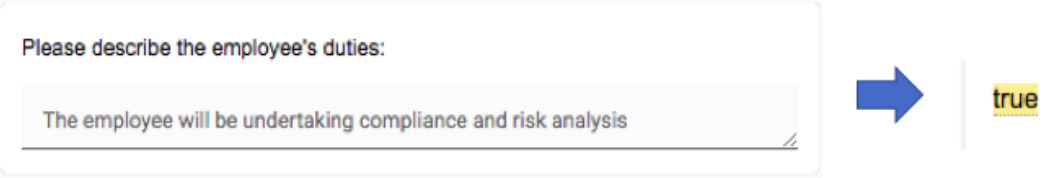
For example, the following operation will output “High Paid” if the salary amount is greater than or equal to £80,000 – otherwise it will output “Low Paid”.

Multiple ‘if blocks’ can be nested by adding a new ‘if’ block as the third input, ie. the value to be outputted if the first ‘if block’ is false. Avvoka will then evaluate the second ‘if block’ and work through nested ifs until there are no more, or one is true.

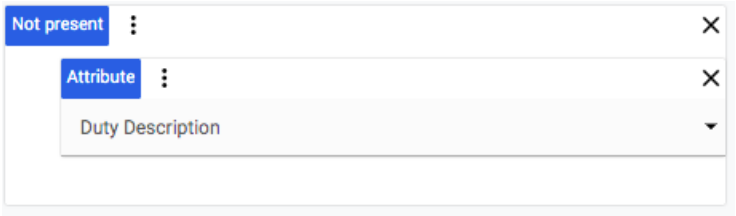
	As you can see in the example above, another “if block” has been added to the third input. The following operation will output Trainee if the salary is equal to or less than £45,000, and Low Paid if the salary is between £45,000 - £80,000.
And	You will be able to add multiple comparisons into this block. This block will output the value “true” if **all** of the comparisons are true. If all the comparisons are false, the block will output the value “false”.  In the above example, if the job title is Associate, and the employee will have access to a company car, this block will output the value “true”. But, if the job title is Partner, and the employee will have access to a company car, this block will output the value “false”. Learn how to make use of these outputted “true”/“false” values below.
Or	You will be able to add multiple comparisons into this block. This block will output the value “true” if **at least one** of the comparisons is true.

	 In the above example, if the job title is Partner, and the employee will have access to a company car, the block will output the value "true". However, if the job title is Partner, and the employee will not have access to a company car, this block will output the value "false". Learn how to make use of these outputted "true"/"false" values below.
Present	This operation checks whether the argument inserted contains any string of text. If there is any string of text present, this operation will output the value "true". If no text is present, this operation will output the value "false".  In the above example, if any string of text is given describing the employee's duties in the questionnaire, then the operation will output the value "true". If there is no text

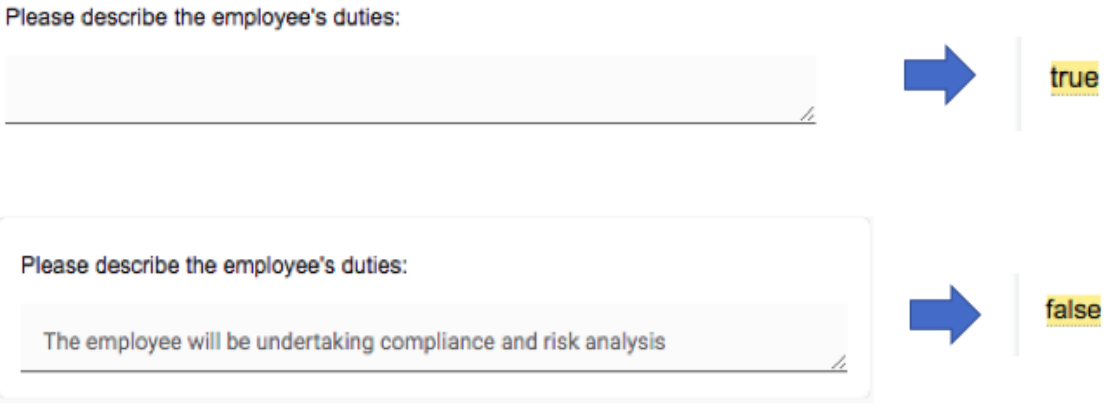
<table>
<tr><td></td><td>present at all, the operation will output the value “false”.

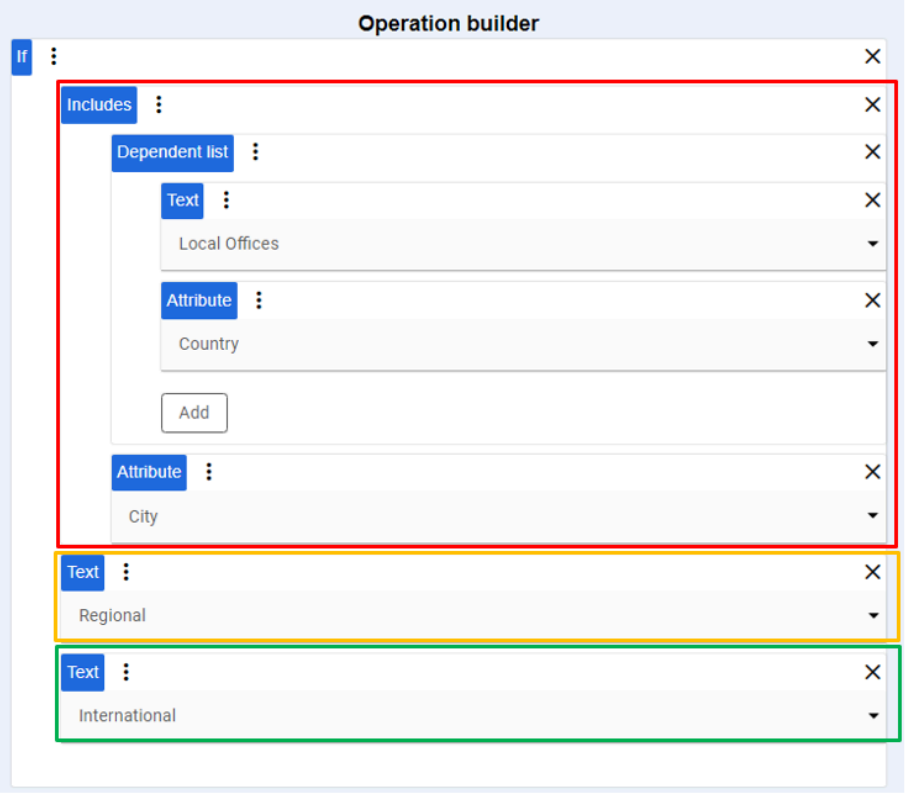
Learn how to make use of these outputted “true”/“false” values below.</td></tr>
<tr><td>Not Present</td><td>This operation checks whether the argument inserted contains any string of text. If the argument does not contain any string of text, this operation will output the value “true”. If text is present, this operation will output the value “false”.

In the above example, if there is nothing present for the attribute “Example 1” in the questionnaire, then the operation will output the value “true”. However, if there is any text present, the operation will output the value “false”.

Learn how to make use of these outputted “true”/“false” values below.</td></tr>
</table>

Includes	Includes looks for the presence of text within an answer, or list. It is useful in conjunction with the If operation, whereby two values can be extracted on the basis of that text, or phrase, being present. Below is an example of how this operation could be used to compare a value against a list. This function takes three arguments: 1. a logical test checking if a string exists in an array drawn from a **dependent list** (**the red block**); 2. the value to be outputted if that test is true (**the orange block**); 3. the value to be outputted if the test is false (**the green block**).  For example, if a global company provides the opportunity for employees to work abroad or within its regional offices, we could use the 'Includes' operation to compare a a number of lists:

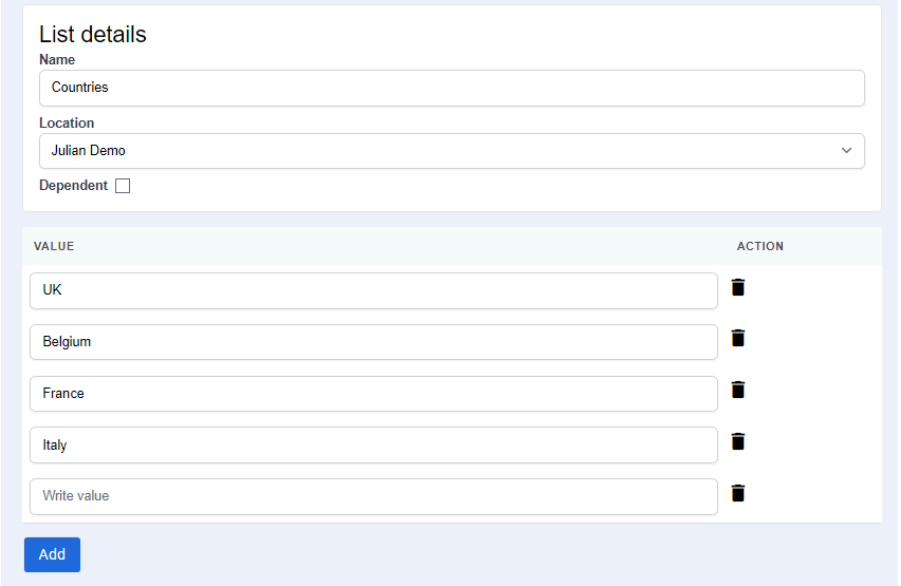
A) A list of select countries the company has an office in

B) A dependent list of all the office locations. This is dependent because it can automatically populate, or refine, the cities those offices are located dependent on the country provided.

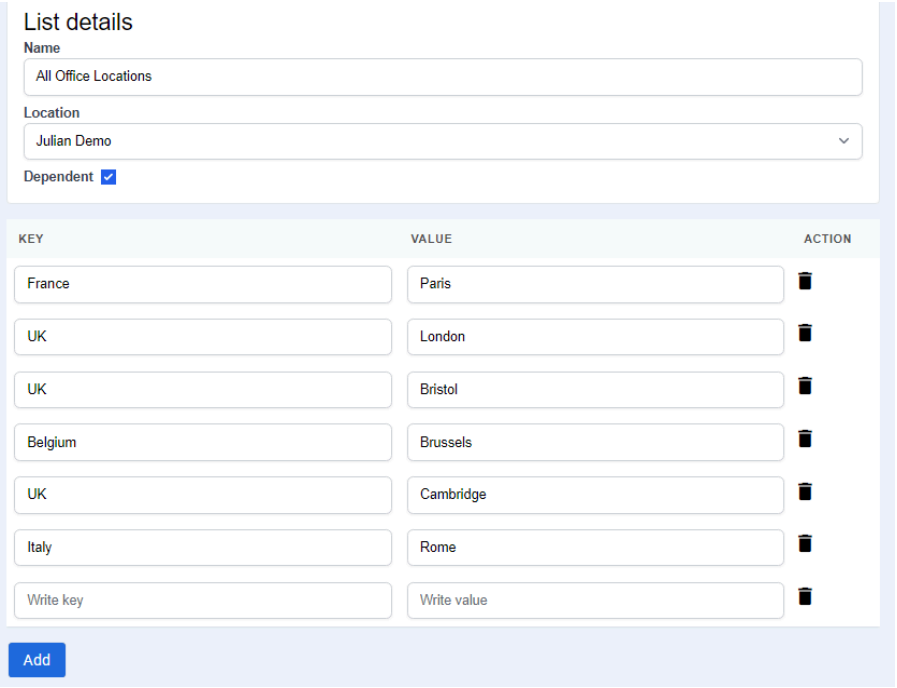
C) A dependent list of the company's local offices.

Below are screenshots of these lists:

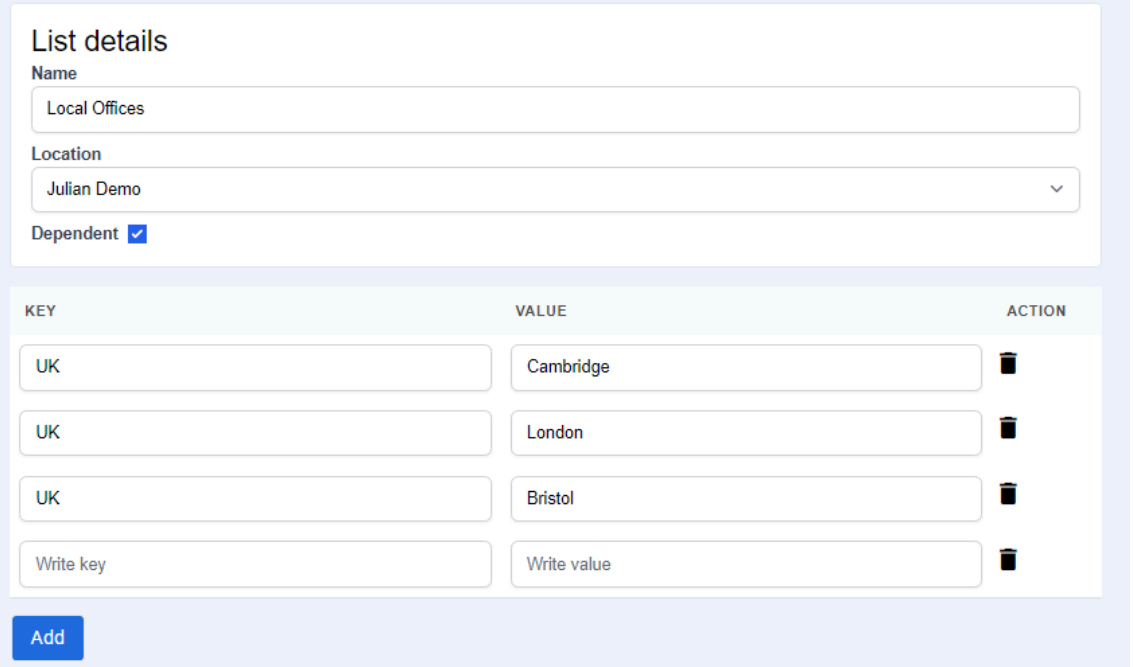
A



B



c



The list of countries and the dependent list of the company's office locations are 'full' lists - these display every office location for that company. The final dependent list, local offices, is used for the purposes of identifying the local offices.

First, the dependent list will produce an array of cities (the office locations) according to each country selected.

In the questionnaire, the user will first select a country and this answer will refine the list of options for the next question, asking for the city. The operation will look at the answers provided and compare them with the dependent list of local offices, seeing if the answer matches. If the logic is true (there is a match) the operation will return a value 'Local'. On the other hand, if there is not a match, the operation will return the value 'international'.

Country: UK

City: Cambridge

The company's ~~International~~Regional office

Country: Belgium

City: Brussels

The company's International office

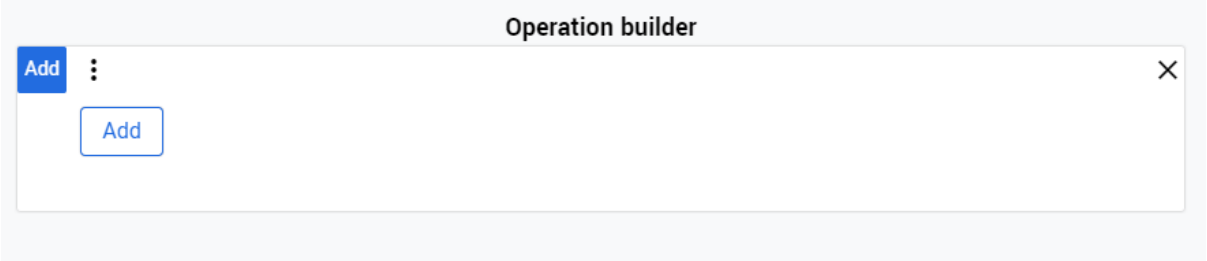
3. Functional Operations

3.1. Math

3.1.1. This operation will allow you to perform calculations on your template.

3.1.2. All inputs in this section must be numbers (either numbers in the calculation or attributes which have numbers entered into them).

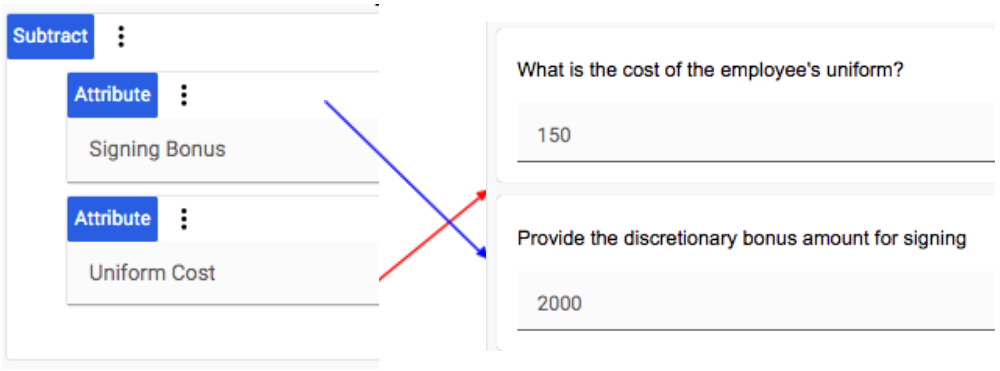
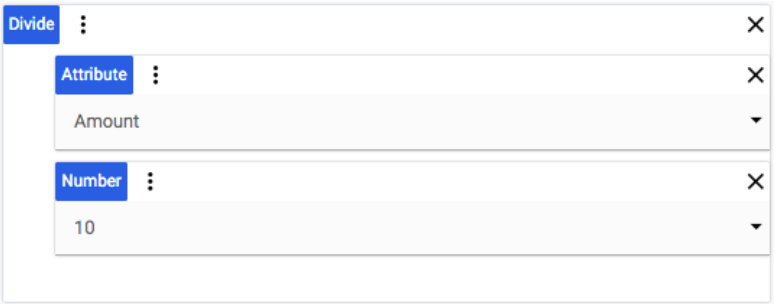
3.1.3. To get started on your calculation operation, select one of the options under “Math”. In this case we will select “Add”. You should now see this below.

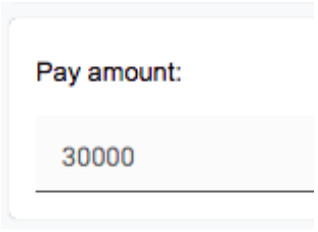
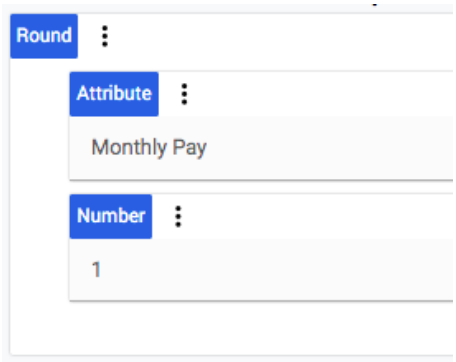
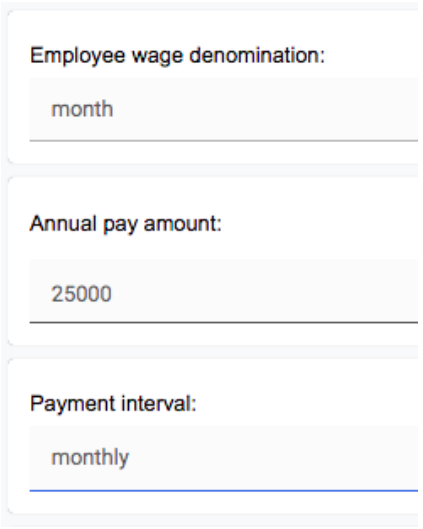


3.1.4. Click the ‘Add’ button and choose the numbers or attributes that you wish to use.

3.1.5. The two numbers within this container will be added together in your desired Operation.

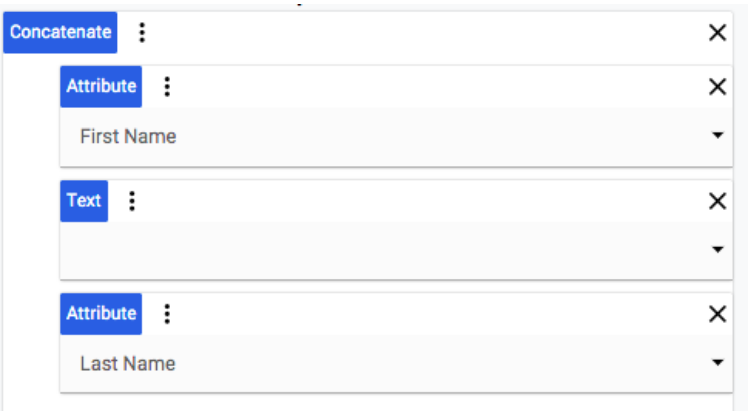
3.1.6. Please see below for a summary of the Math functions.

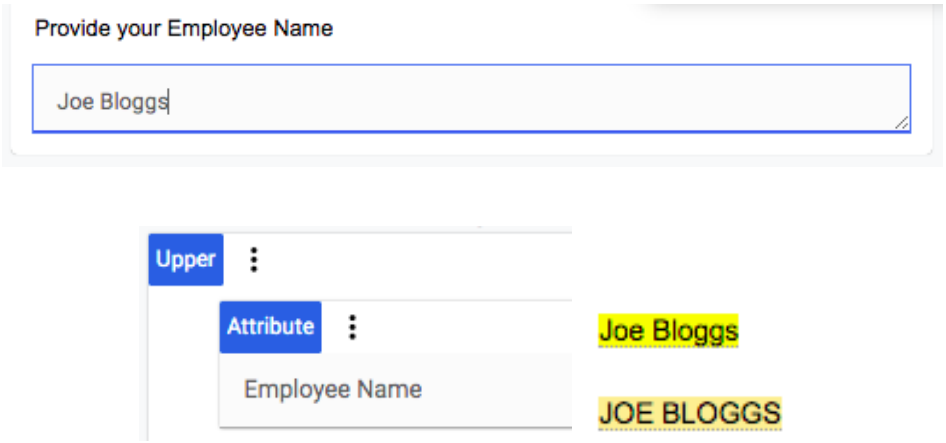
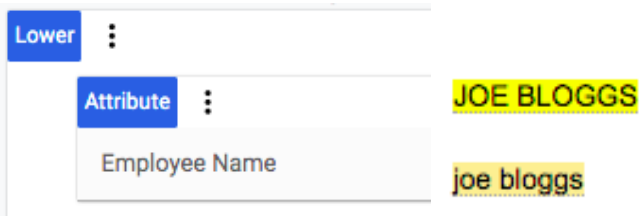
Add	This will add both values in the container - This only works for **TWO** values. To add more than two values together use “Sum”.
Subtract	This will subtract the second value in the container from the first, and output the result. For example, the cost of an employee’s uniform might be deducted from their signing bonus. The operation will return the aggregate amount the employee would receive.  7.5 You will be entitled to a discretionary signing bonus of £1850.
Multiply	This will multiply the two values in the container and output the result.
Divide	This will divide the first value by the second value and output the result. For example, the bonus might be a tenth of the employee’s salary. In the example below the salary is £30,000, so the bonus will be £3000. 

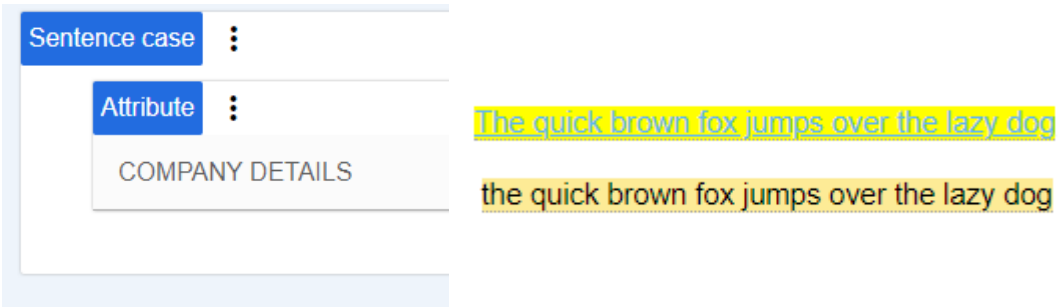
	 7.5 You will be entitled to an annual bonus of £3000.
Sum	Sum allows you to add up **multiple inputs** (unlike Add which will only allow you to add two values together). This operation will output the result of the equation, however you can have any number of arguments to sum together. View this example below on how to manage the "sum" function in a table.
Round	This will round off the first value to the number of decimal places as expressed in the second argument and output the result. In the example below, if the annual pay is £25,000, the employee will be paid £2083.33 a month.   *Without Round* 7.1 Your basic salary/wage is £2083.3333333333335 per month payable by monthly instalments in arrears on or around the last working day of each month directly into your bank account, less deductions for income tax and National Insurance contributions. *With Round* 7.1 Your basic salary/wage is £2083.3 per month payable by monthly instalments in arrears on or around the last working day of each month directly into your bank account, less deductions for income tax and National Insurance contributions.

	In this example, the operation will output 2083.3, rounding the number to one decimal place. If 2 is used in the second argument, the operation will output 2083.33. If 0 is used, the operation will output a whole number of 2083. If you wish to round a number off to their hundredths/thousandths place instead, simply use negative numbers.  In the above example, if the employee's monthly salary was £2083.33, this will output the number 2080.

3.2. Text

Concatenate	This will output a string made up of multiple inputs joined together (e.g combining the values of two attributes in a single line). This is helpful if you want to join the answers of two questions. A basic example might be where there are placeholders for first name and last name, and you want to join them together to make a full name. To create a separator between the words, such as a space, use 'text' and insert the separator - as shown below.  First Name: Joe Last Name: Bloggs Full Name: Joe Bloggs Alternatively, we might use this operation to change the date format to meet local styles, e.g. American format (month/day/year). See the worked examples section for an example of this.
Upper	This will output the inputted string in all upper-case. For example, we have created a question to provide the 'Employee Name' using the attribute input. The Upper Operation transforms the text into UPPER TEXT.

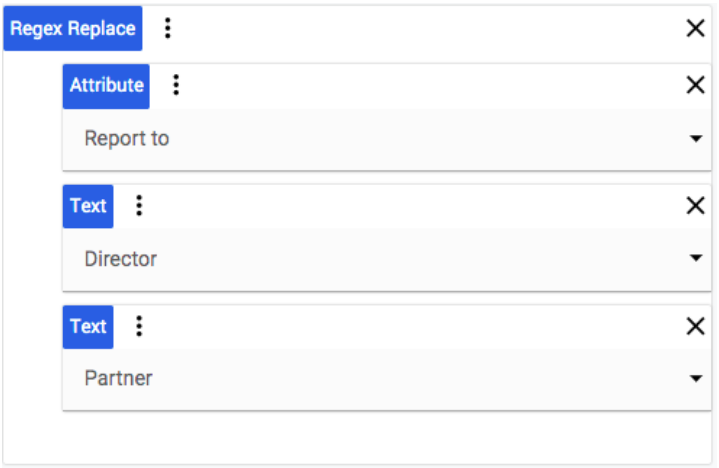
	
Lower	This will output the inputted string in all lower-case. For example “EMPLOYEE NAME” will become “employee name”. 
Capitalise	This will output the inputted string with the first letter of each word in uppercase type. For example “joe bloggs" will become “Joe Bloggs”. 
Sentence case	This will output the inputted string with the first letter of each sentence in uppercase type. For example “the quick brown fox jumps over the lazy dog” will become “**T**he quick brown fox jumps over the lazy dog”.

<table>
<tr>
<td></td>
<td> 
</td>
</tr>
<tr>
<td>Regex Replace</td>
<td>This allows for phrases (including numbers) to be modified through regex replace.

• Attribute: The first input is the string to be modified.

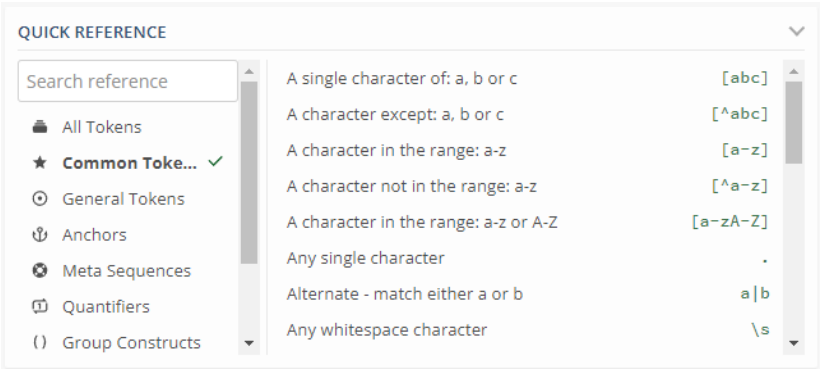
• Text: The second input is the regular expression that will be matched and replaced by the third input.

• Text: The third input is what should replace each of the matches listed in the second input.



In the above example, whenever “Director” is typed in the document questionnaire for the question on who the individual should report to, it will be replaced with “Partner”. This could be helpful when consolidating precedent banks, so they update according to the answers provided.

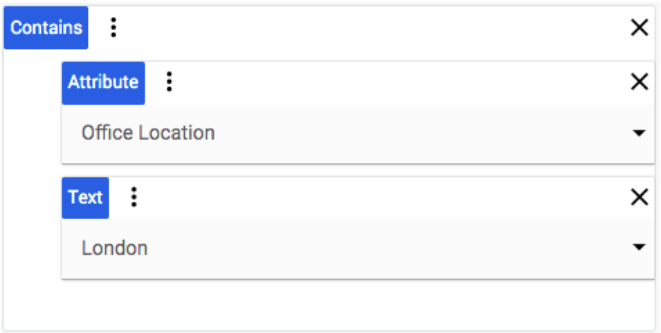
For more complex regex replace operations, it is possible to insert Regular Expressions into the second or third inputs. To see this in action, view the Numbers in Ordinal Form (Regex) example below.</td>
</tr>
</table>



Regular expressions should be considered advanced functionality and require a knowledge of regular expressions, training on which cannot be provided by Avvoka.

Contains

This operation checks whether the second string is contained within the first string, such that the following operation would output the value "**true**".

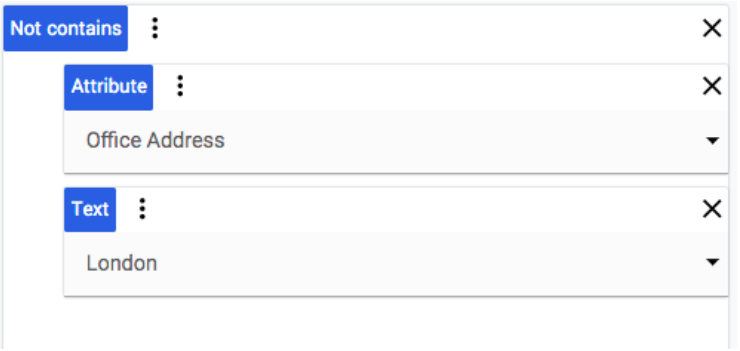


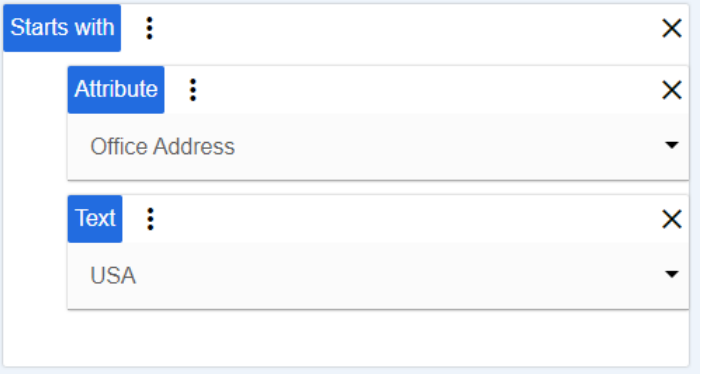
In this example below:

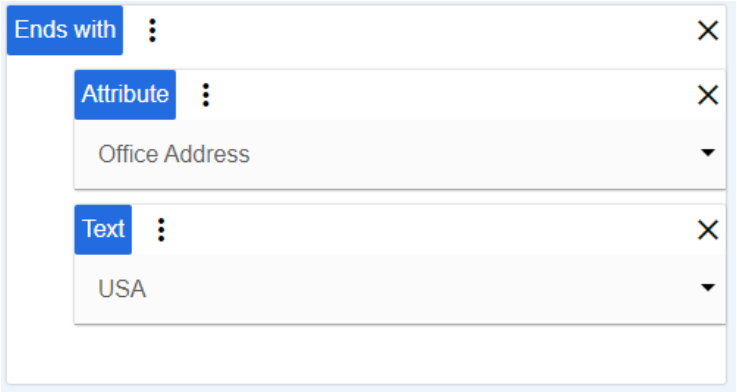
- If the user answered "Hoxton Street, London" to the question "Provide the full office address" in the questionnaire, the operation would output the value "**true**".

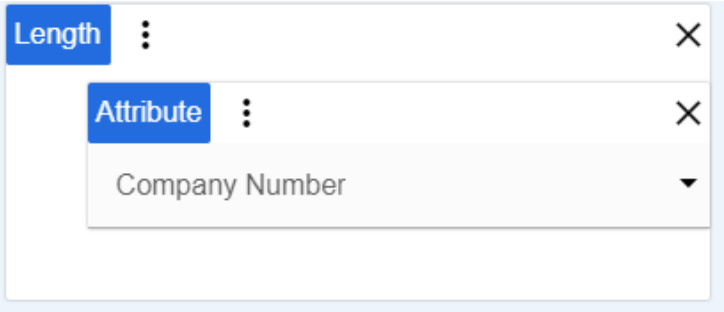
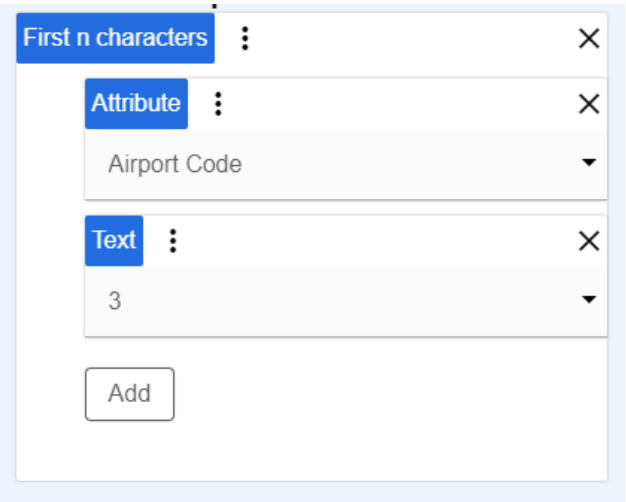
> Hoxton Street, London
> true

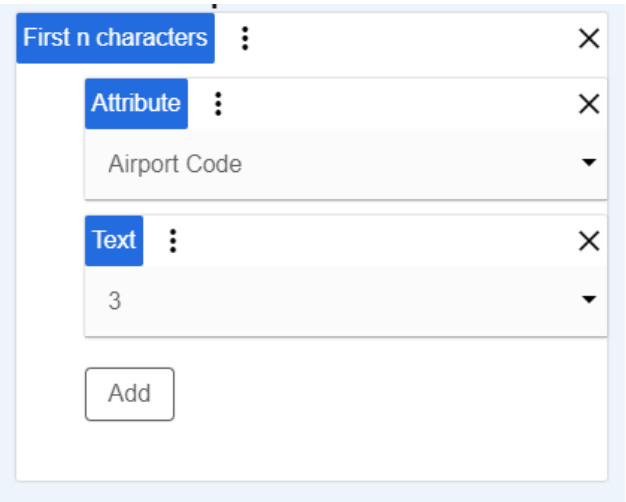
- If the user answered with "Hammersmith, London", the operation would also output the value "**true**".

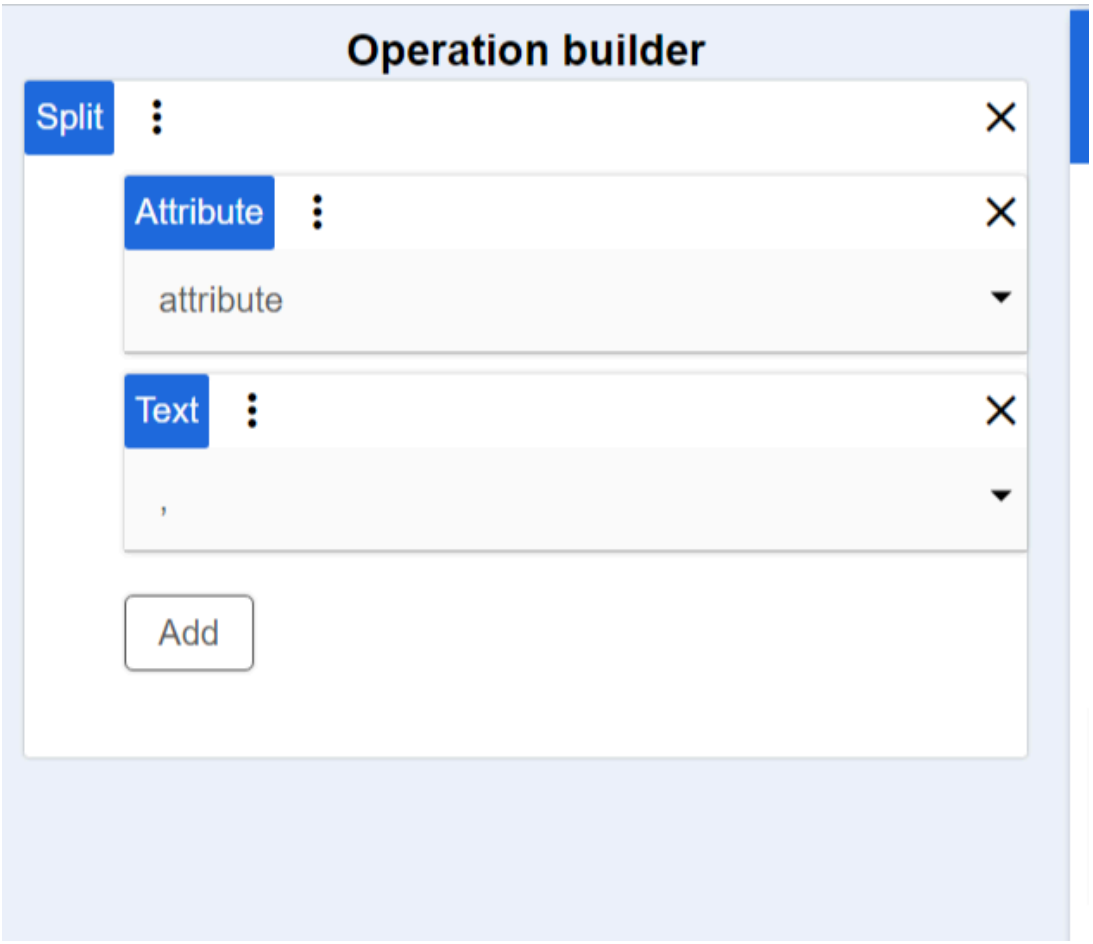
	Hammersmith, London true • However, if the user gave an answer that did not include London at all, for example “The Champs-Élysées, Paris” the operation would output a result of “false”. The Champs-Élysées, Paris Learn how to make use of these outputted “true”/“false” values below. Contains can also be used to compare a value against a list. View this worked example for more information.
Not contains	This operation checks whether the second string is contained within the first string. If the second string is not found within the first then the operation will output the value “**true**”. In this example below: 1. If the user answers “London” to the question “Provide the full office address” in the questionnaire, the operation would output the value “**false**”. 2. If the user answered “The Champs-Élysées, Paris” (or any other answer that does not contain the text London), the operation would output the value “**true**”. 

	The Champs-Élysées, Paris true Hammersmith, London ~~true~~ Learn how to make use of these outputted “true”/“false” values below.
Starts with	This operation checks whether the first string starts with the text contained within the second string. If the first string starts with the text contained within the second string then the operation will output the value “**true**”. In this example below: 1. If the user starts their answer with “USA” to the question “Provide the full office address” in the questionnaire, the operation would output the value “**true**”. 2. If the user answered “France” (or any other answer that does not start with the text USA), the operation would output the value “**false**”.  USA - Florida true ~~USA~~France - ~~Florida~~Paris ~~true~~ Learn how to make use of these outputted “true”/“false” values below.

Ends with	This operation checks whether the first string ends with the text contained within the second string. If the first string ends with the text contained within the second string then the operation will output the value "**true**". In this example below: 1. If the user ends their answer with "USA" to the question "Provide the full office address" in the questionnaire, the operation would output the value "**true**". 3. If the user answered "USA - Florida" (or any other answer that does not end with the text USA), the operation would output the value "**false**".  Florida - USA ~~Florida~~USA - ~~USA~~Florida true ~~true~~ Learn how to make use of these outputted "true"/"false" values below.
Length	This operation outputs the number of characters contained within the string. In this example below, the operation will output the number of characters in the answer to the Company Number question.

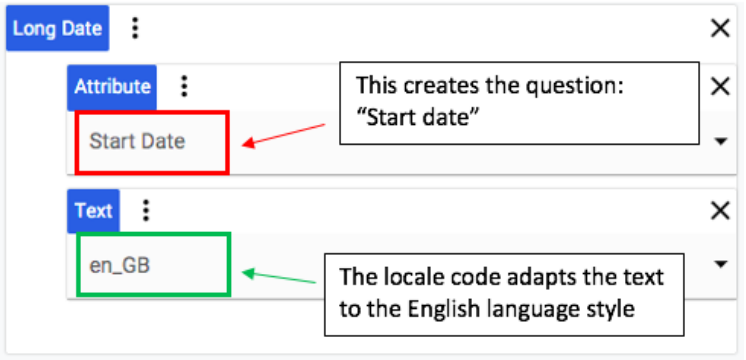
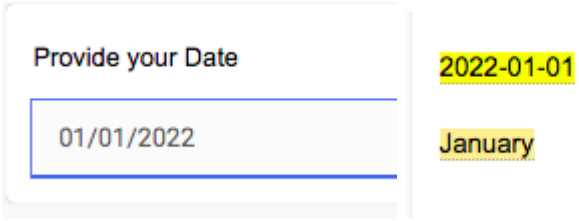
	 IE9769936F 10
First n characters	This operation outputs the first n characters contained in the first string. The “n” number is determined by the number inserted into the second string. For example, in the example below, the operation will return the first 3 characters contained in the first string.  NRT - Narita JFK - John F. Kennedy International NRT JFK
Last n	This operation outputs the last n characters contained in the first string. The “n”

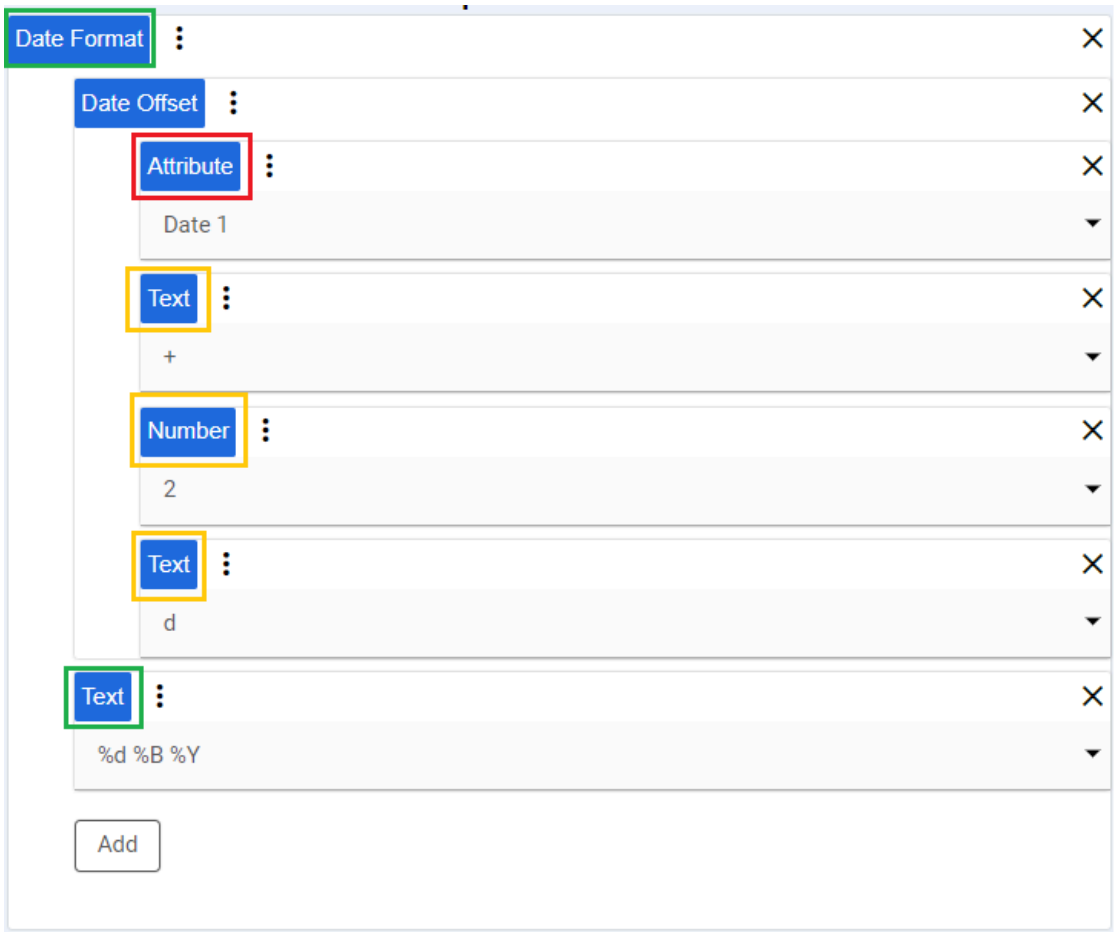
characters	number is determined by the number inserted into the second string.For example, in the example below, the operation will return the last 3 characters contained in the first string.  Narita - NRT John F. Kennedy International - JFK NRT JFK
Split	The Split operation takes a string as its input and generates an array based on a specified separator. It requires two parameters to function properly. The first parameter is a string of text that you want to split, while the second parameter is a separator text which you want to use as a separator to split the given strings in the first parameter. For instance, if you have a string [ABCD] that you want to split using the separator ",". You can do so by selecting the **'Split'** operation (found under the **Text** column), and then add an attribute which will be the first parameter as the input text that you want to split, and the second parameter as the separator text ("," in this case).


Operation builder
Split
Attribute
attribute
Text
,
Add

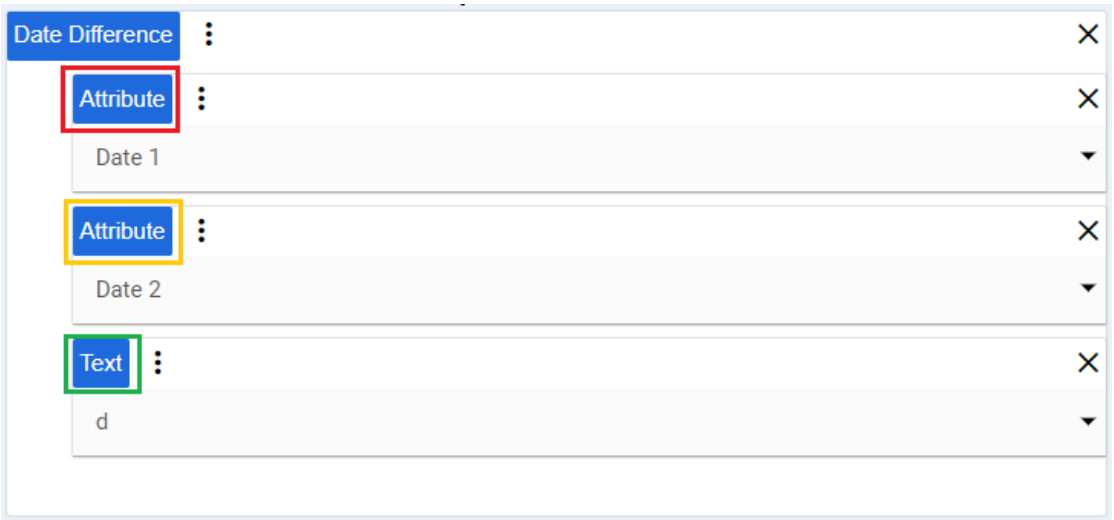
3.3. Localisation

3.3.1. This section will explain the functions of the localisation blocks. The setup of these operations will be explained in more detail in our “Frequently Used Operations” section.

Long Date	This operation takes an inputted short date format and outputs the date in a longer format, such as in the screenshot below. The date can be localised with locale codes.   This operation is explained in more depth below.
Date Format	This operation takes an inputted short date format and outputs a specific formatted element of the date, such as day, month or year. The date can be localised with locale codes. (see the codes here) 

	This operation is explained in more depth below.
Date Offset	The Date Offset operation allows you to add or subtract a certain number of days/weeks/months/years to/from an existing Date in the template.  How to set up a Date Offset operation: • Select the date you want to subtract from or add to (this will usually be an attribute set to a Date type in the questionnaire) (in red) • Select whether you want to subtract from (**-**) or add to (**+**) this date. Then choose the number of days/weeks/months/years you want to add or subtract. Enter '**d**' to signify days, '**w**' for weeks, '**m**' for months, and '**y**' for years. (in orange) • **The output** of this operation will come in the format **YYYY-MM-DD**, so if you would prefer a different format, make sure to wrap the Date Offset operation in a Date Format operation (in green)

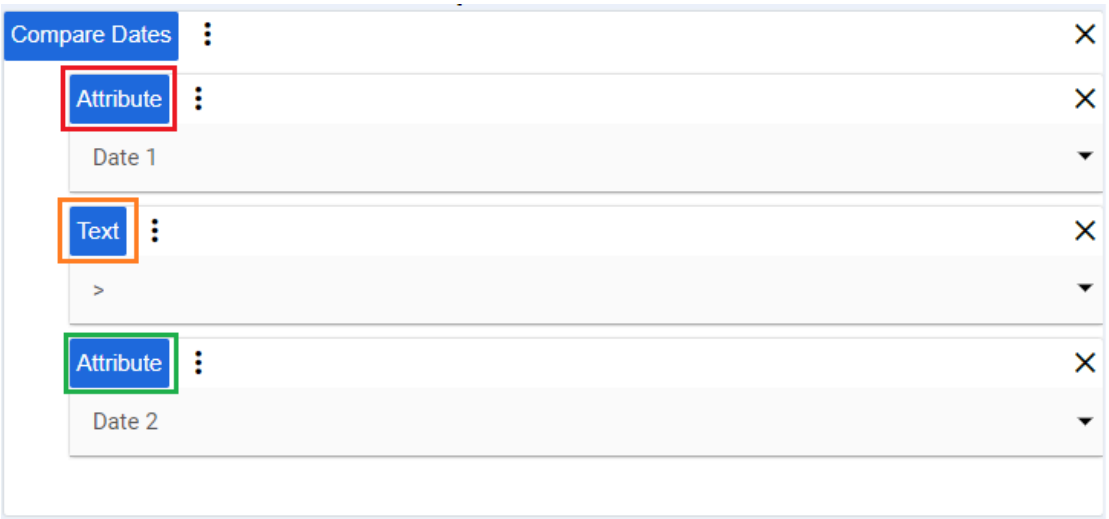
<table>
<tr>
<td>Date Difference</td>
<td>This operation allows you to find the number of days/weeks/months/years between two dates.

How to set up this operation:

• Enter the first date (in red)
• Enter the second date (in orange)
• Provide whether you want the result to be given in days (d), weeks (w), months (m) or years (y) (in green)

The output will be given as a number. Please note that the output will be negative if the second date is an earlier date than the first date.</td>
</tr>
<tr>
<td>Compare Dates</td>
<td>This operation allows you to evaluate the relationship between two dates (e.g. find out if Date 1 comes after Date 2).</td>
</tr>
</table>



To set up this operation:

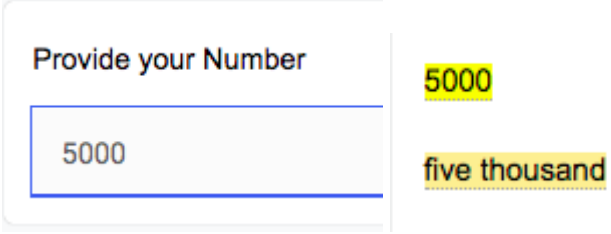
- Select the first date (in red)
- Enter (in orange);

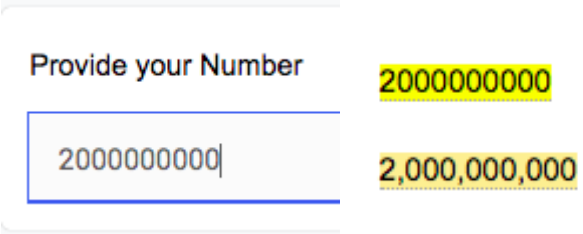
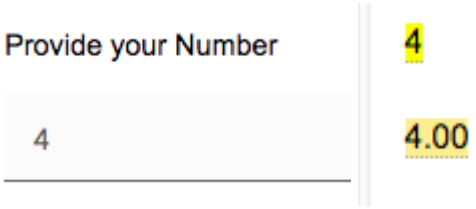
=	Equal to
!=	Not equal to
>	Greater than (comes after)
>=	Greater than or equal to
<	Less than
<=	Less than or equal to (comes before)

- Enter the second date (in green)
- **The output** of this operation will be either **true** or **false**

In Words

This operation will take an inputted number and output the number in words, such as in the screenshot below. The output can be localised with locale codes.



	Users may also wish to wrap this operation with ‘Capitalise’ in the first string, to turn the first letters of the words into capital letters; ‘Five Thousand’. This operation is explained in more depth below.
Format Number	This operation will take an inputted number and output the number with the appropriate punctuation, such as in the screenshot below. The output can be localised with locale codes.  This operation is explained in more depth below.
Format Number With Precision	This operation works in conjunction with the “round operation” to ensure that integers when rounded off, do so with a .00 as in the example below. The output can be localised with locale codes.  This operation is explained in more depth below.
Sysdate	The Sysdate operation enables users to retrieve the creation date of a document for display or use in other operations (i.e: "Date Offset" "Date Difference" or "Compare Dates"). It is important to specify a Date Format to ensure that the retrieved date is formatted according to specific and desired requirements. **Example:** By using the Sysdate operation with the appropriate Date Format, a user can extract the creation date of a document and display it in a desired format, such as

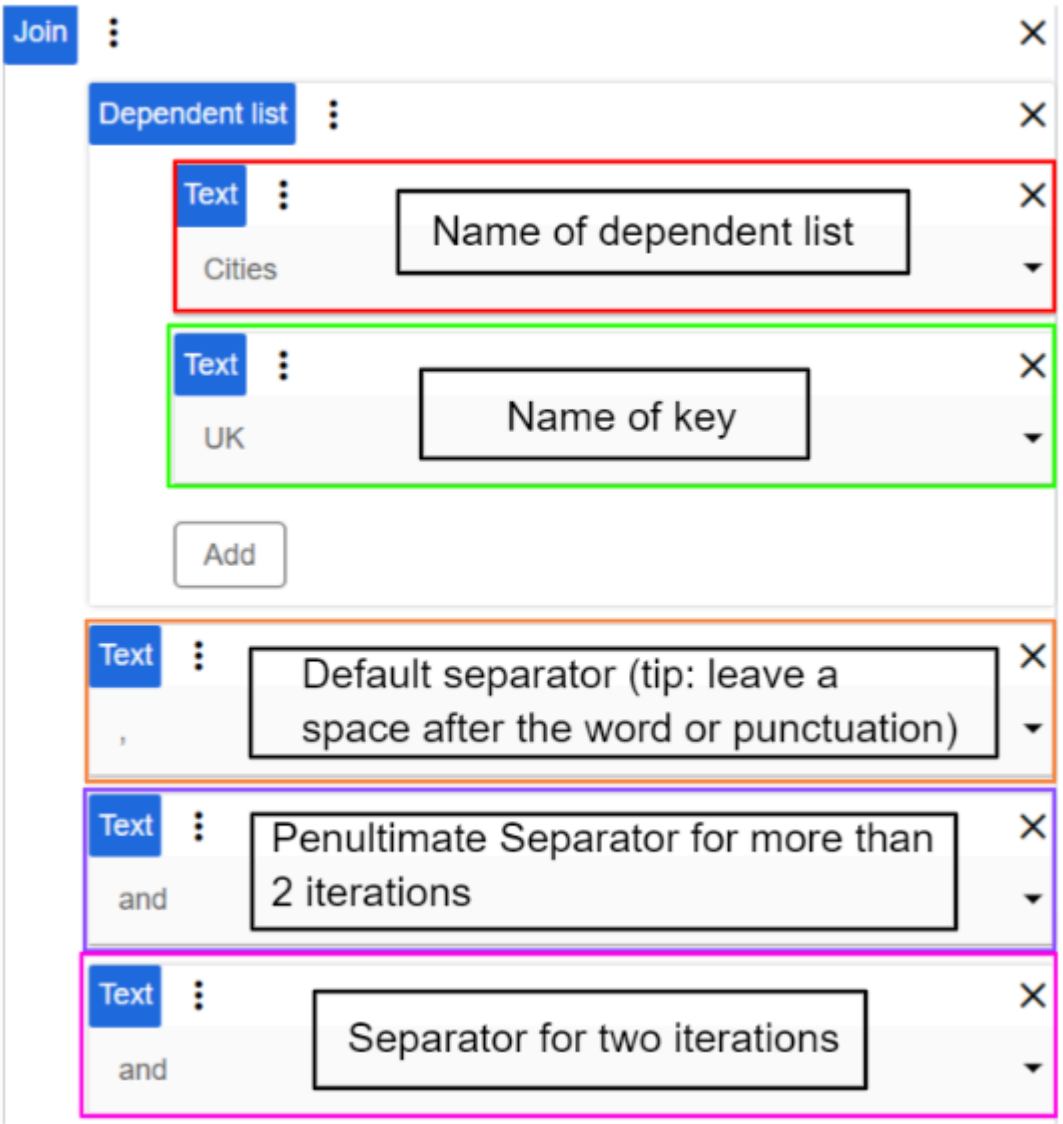
	"MM/DD/YYYY" or "YYYY-MM-DD". This ensures consistency and meets the specific formatting needs of the user or application.

Commonly-used locales A full list of locale codes can be found here.	
British English	en_GB
American English	en_US
Spanish	es
French	fr
German	de
European Portuguese	pt
Brazilian Portuguese	pt_BR
Chinese (Traditional)	zh_TW
Chinese (Simplified)	zh_CN
Japanese	ja
Korean	ko

Date format codes		
Year		
Full year	%Y	2000, 2001, 2019
Short year with zero padding	%y	01, 02, 19
Short year without zero padding	%-y	1, 2, 19
Month		
Full month name	%B	January, February, March
Abbreviated month name	%b	Jan, Feb, Mar
Month number with zero padding	%m	01, 02, 03
Month number without zero padding	%-m	1, 2, 3
Week		
Week of the year (Monday as start of the week)	%W	00, 01, 02, …, 99
Week of the year (Sunday as start of the week)	%U	00, 01, 02, …, 99
Day		
Day of the year with zero padding	%j	000, 001, 002, ..., 366
Day of the year without zero padding	%-j	0, 1, 2, …, 366
Day of the month with zero padding	%d	01, 02, …, 31
Day of the month without zero padding	%-d	1, 2, …, 31
Full weekday name	%A	Monday, Tuesday, Wednesday
Abbreviated weekday name	%a	Mon, Tue, Wed
Weekday as number	%w	1, 2, 3

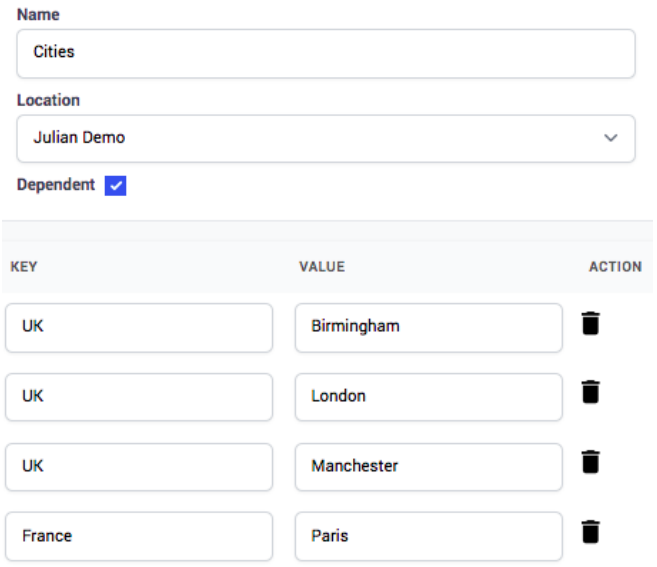
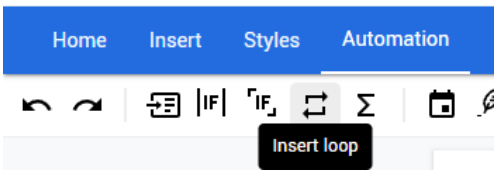
3.4. Array

Join	This strings multiple outputs together. It is composed of four parts (two mandatory): 1. The first argument which produces the outputs to be joined (in red). 2. The second argument that contains the default text separator for each output (in orange). 3. The third argument contains the penultimate text separator for two or more iterations (in purple). 4. The fourth argument contains a text separator when there are two iterations (in pink). *N.B. The last two arguments are optional.* This will serve to join the outputs into a single string and include the separator between each output. The outputs are an array (typically those returned by a Dependent List block or a looped attribute).

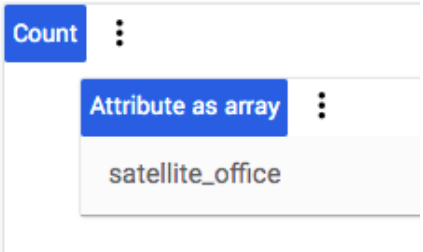


The arguments for the operation above are as such:

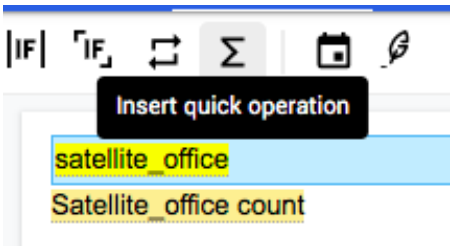
- **Dependent list**
    - **Text**: Title of the Dependent List (Cities)
    - **Text**: List key (UK)
- **Separator Text**: Delimiter (,) (*with a space after*)
- **Penultimate Separator Text for more than 2 iterations**: and (with a space after)

	• **Separator Text for two iterations:** and (with a space after)  In this example, the title of the Dependent List is “Cities”. We might use this operation within an existing operation to extract all the values attached to a key in an array, e.g using ‘contains’. Or alternatively, display all those values in an array in the document itself. This example will result in the below; Birmingham, London, Manchester As previously noted, the 'Join' function is compatible with any array. Therefore, if desired, a looped attribute can substitute for a Dependent list argument.
Count	This functions to count the number of times an attribute in a loop has been looped. To see this in action, we will first have to create a looped placeholder in the template (text highlighted in a light shade of blue)  satellite_office

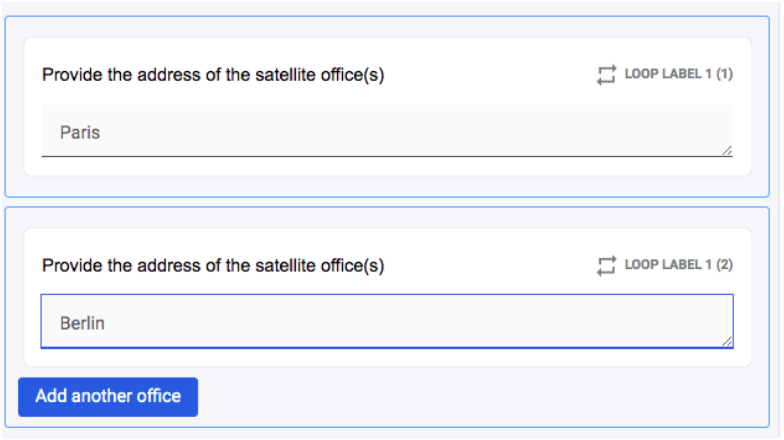
Then, navigate to the operations menu. Since the placeholder is looped, we will have to use the function “Attribute as array”.



Once you have created the operation, insert it into the template.

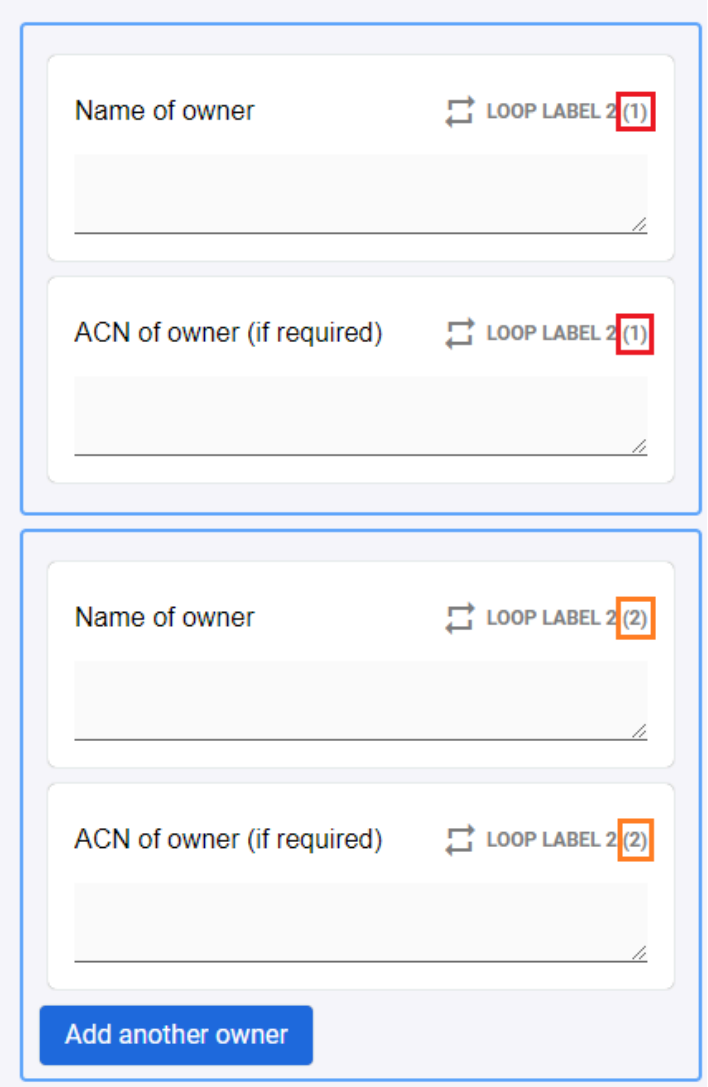


Once you have answered the relevant questions, the Operation should output the number of times the placeholder has been looped.



In this example, the employer may define other office locations the employee will be expected to work at. If the employee lists a number of additional offices, we can use the count operation in an inline condition with an ‘s’ to make office[s], if satellite office >= 2.

	 5.3 The employee may be required to work at the company's satellite offices, located at Paris, Berlin
At	This function allows you to extract a specific index item in an array similar to rows, except it is specific to non-looped arrays. For instance, the array could be ["A", "B", "C", "D]. Now you want to pull value 2 from the array you need to subtract 1 from this number as it takes the index of the array to start from 0 rather than 1. There would be two parameters for this: 1.) The array parameter that you want to pull the value from. 2.) The number parameter for the index of the array that you want to pull values from. You need to subtract 1 from this number as it takes the index of the array to start from 0 rather than 1. This is used in the following worked example.
Iterator	**Specific to loops**: This functions as a label for the attributes being looped and can be named however you want it. It is used with the Iterated attribute function. You can either choose to input a value into the Iterator section or leave it blank. If left blank, the attributes will be labelled according to their index. This index is highlighted in red and orange below.

	
Map	**Specific to loops**: This is the input which ties together all the needed components and outputs an array (the output). This is used in the following worked example.
Row	Row helps to extract a specific index item in an array. For instance, a looped attribute could be ["1", "2", "3", "4"]. So, the row operation in this case would then have 2 parameters: 1. A Text parameter for the index that you want to extract (e.g. "2"); & 2. An Attribute parameter that points to the looped attribute that you're extracting from. This is used in the following worked example.

Reference	Reference is always used when the Dictionary operation is used. A user is able to define concepts using these operations.

Worked Examples

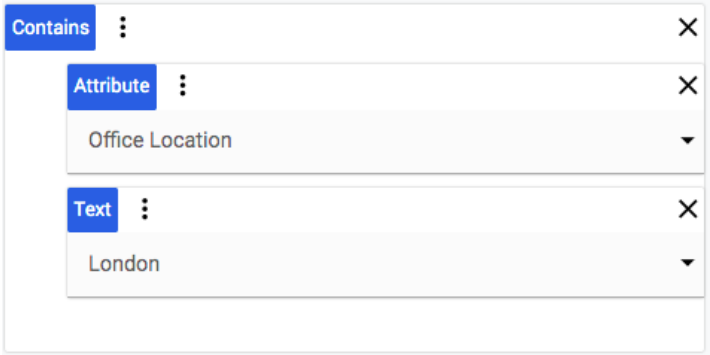
4. Making use of “true”/”false” values

4.1.1. Throughout this guide, you may have seen multiple references to Operations outputting the value “true”/”false”.

4.1.2. We can make use of these values “true” and “false” values outputted by the Operation to build conditions. For example, if the value outputted by the Operation is “true”, a clause or sentence could be dropped into the document.

4.1.3. To do this, when building a “block condition” or “in-line” condition, select the Operation from the attributes list (it will have an E next to it). Then, set the final value to “true” or “false” as so desired.

4.1.4. To illustrate this, we will use the below “Contains” block as an example.

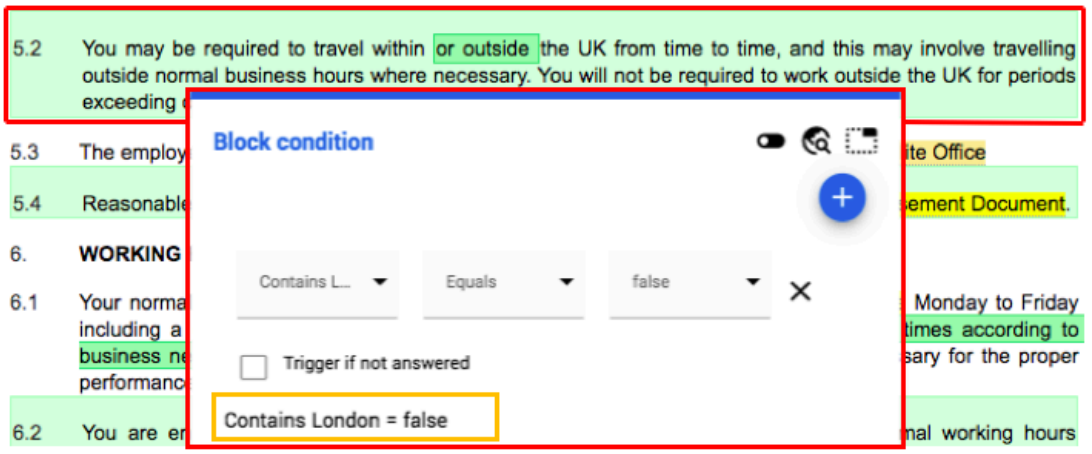


4.1.5. With reference to how a Contains block operates:

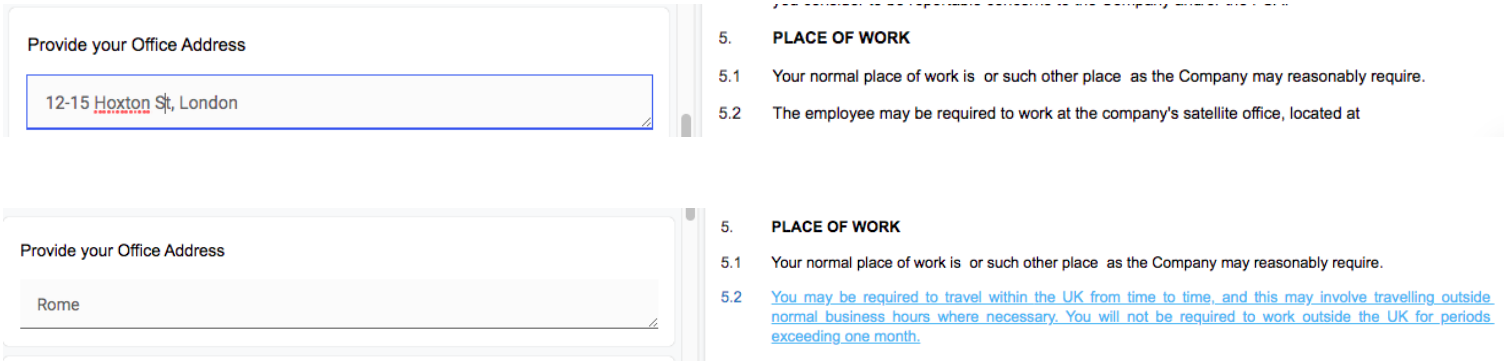
- If the user’s answer contained “London” to the question “Provide the office address” in the questionnaire, the operation would output the value “**true**”.
- However, if the user gave an answer that did not include London at all, the operation would output the value “**false**”.

4.1.6. If the office is in London, the employee may not need to travel at all. As such, the clause on travel in the employment contract will not be necessary. Therefore, the clause should drop in only if the operation returns a value of false.

4.1.7. Highlight the whole clause on travel and create an in-line/block condition over it. Set the in-line/block condition up so that the first input is the “Contains London” operation and the final input is “false” as shown below.



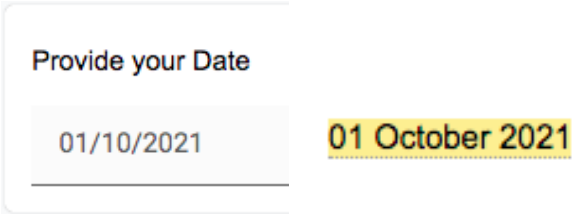
4.1.8. The operation should now work as intended!



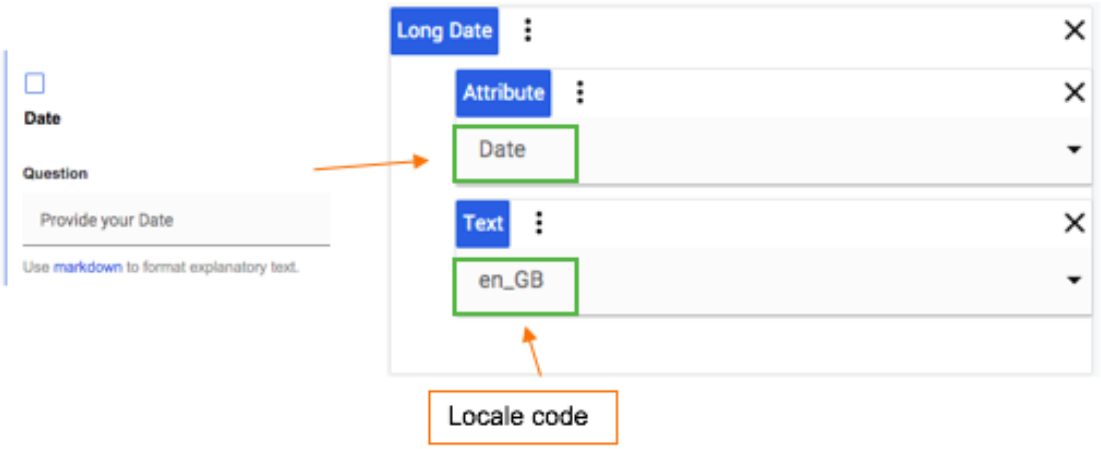
5. Worked Examples on Dates

5.1. Long Date

5.1.1. The Date format in the questionnaire is set by default to be short form (e.g. 01/10/2021). The long date operation takes this inputted information and transforms it into its long form (1st October 2021).



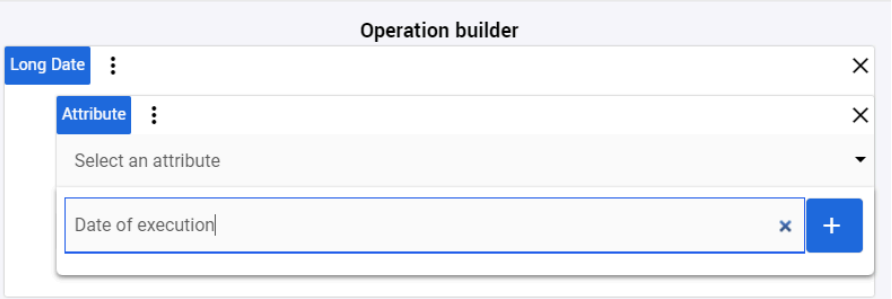
5.1.2. To create the operation, select Localisation under Long Date, so that long date is in the top band. Then choose the attribute that the operation will pull the information from (e.g. Date).



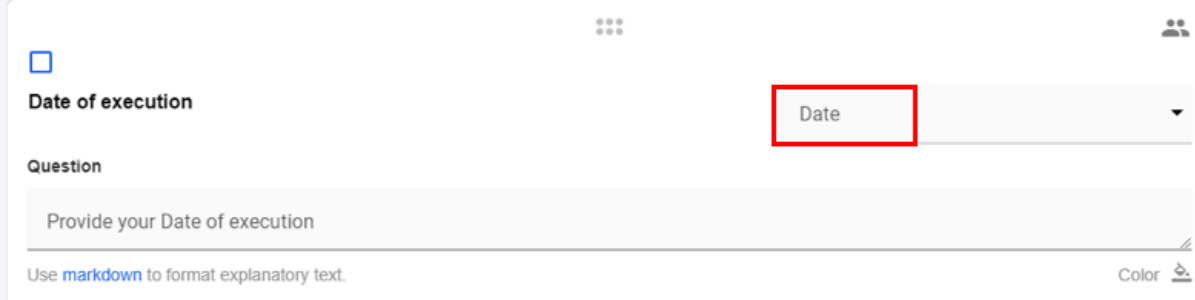
5.1.3. In the drop down list for the attribute, you can either select an existing Date attribute that you wish to convert to a long date. Or, you can create a wholly new attribute that

you wish to convert to a long date. For example, you may wish to create a new attribute for ‘Date of Execution’, as opposed to ‘Date’.

5.1.4. To create a new attribute, type in the desired name ‘Date of Execution’ and click +. This will create a new attribute, forming a new question in the questionnaire.



5.1.5. Then, go back into the questionnaire and navigate to the relevant attribute. Click the drop down list to the right of the attribute and change the question type to a Date question.



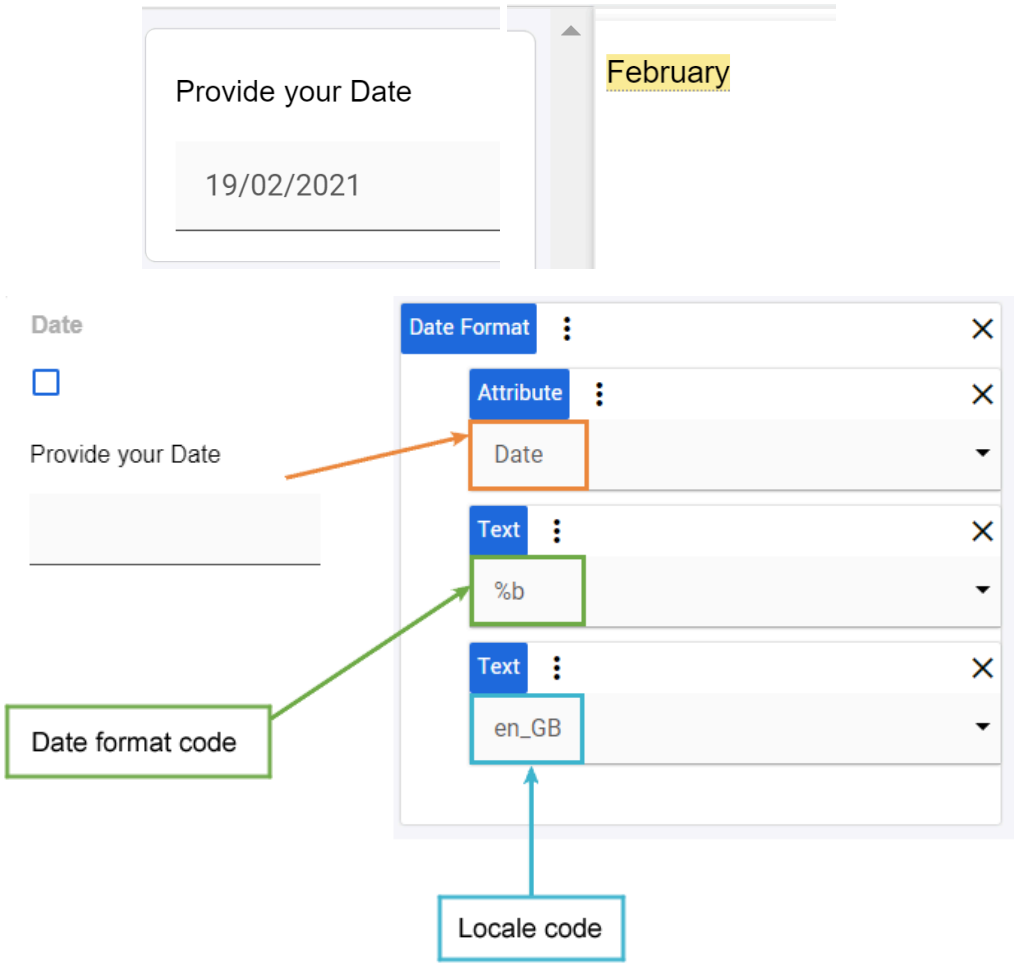
5.1.6. Finally, the date can be localised using locale codes, e.g. US all the way to Chinese. This is inputted using the Text input at the bottom of the operation.

5.1.7. Feel free to watch this video on how to create the Long Date operation.

5.2. Date Format

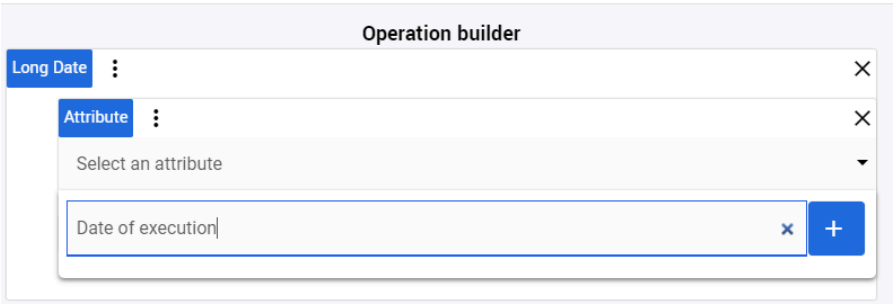
5.2.1. The Date format in the questionnaire is set by default to be short form (e.g. 01/10/2021). However on some occasions, you might want to extract a specific element of the date. For example, the day, month or year.

5.2.2. In the example below, the specific month of February has been extracted from the answered date.



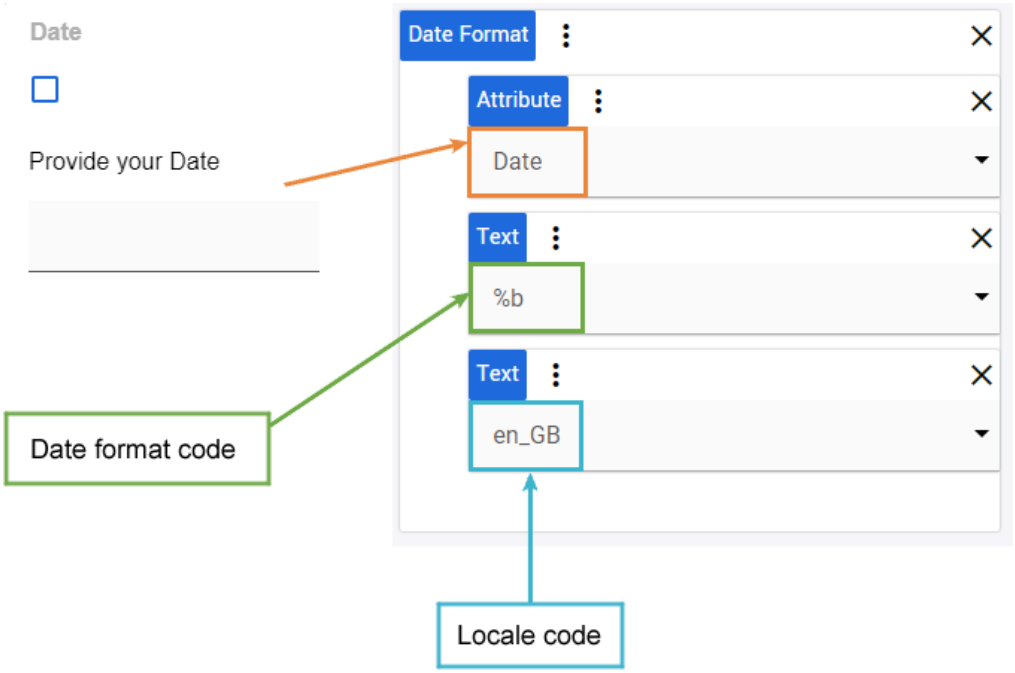
5.2.3. To create the operation, select Date Format under Localisation, so that Date Format is in the top band. Then choose the attribute that the operation will pull the information from (e.g. Date).

5.2.4. In the drop down list for the attribute, you can either select an existing Date attribute. Or, you can create a wholly new attribute.



5.2.5. Then, go back into the questionnaire and navigate to the relevant attribute. Click the drop down list to the right of the attribute and change the question type to Date.

5.2.6. Then, choose the element of the date that you wish to extract. This is inputted by selecting the text input and typing in the relevant date format code.



5.2.7. Feel free to watch this video on how to create the Date Format operation.

5.3. Bespoke Date Format (eg. US Date Format)

5.3.1. The date, even in its short form, can vary across different countries. For example in the US, the short date format is **(MM/DD/YYYY)**, as opposed to the English format of **(DD/MM/YYYY),** which is the default in Avvoka.

5.3.2. Users can use date format codes to customise their date formats. However, users can also create more bespoke date formats using the following operation if they so choose.

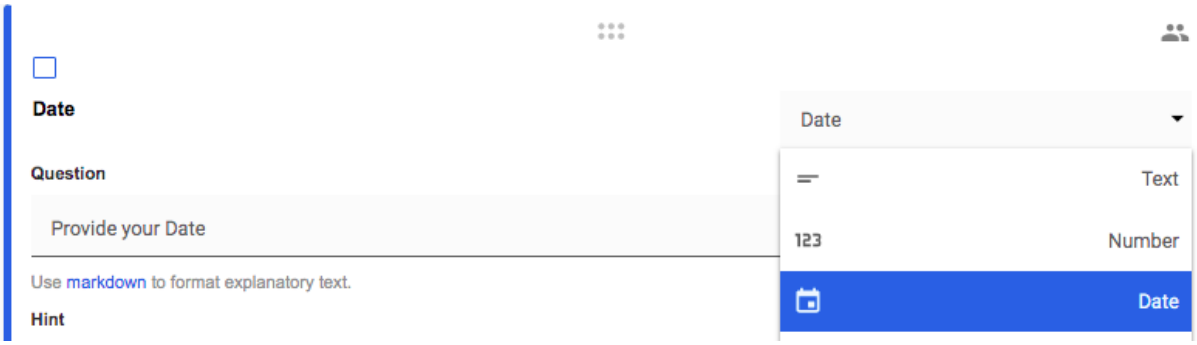
5.3.3. To create the operation, select 'Date Format', under localisation, and add an attribute, e.g. Date.

5.3.4. Add a second argument using Text:

- The code for month number is %m, the code for day number is %d and the code for the year number is %Y.
- Arrange these codes in your preferred order. Ensure that there is a "/" between each code.

5.3.5. The operation should look similar to the image below. The example below shows an operation for the US Date format **(MM/DD/YYYY)**.

5.3.6. Then go into the questionnaire and change the format of the question under the attribute Date to the Date question type.



5.3.7. Finally, add the operation into the document using the ‘insert quick operation’ button.

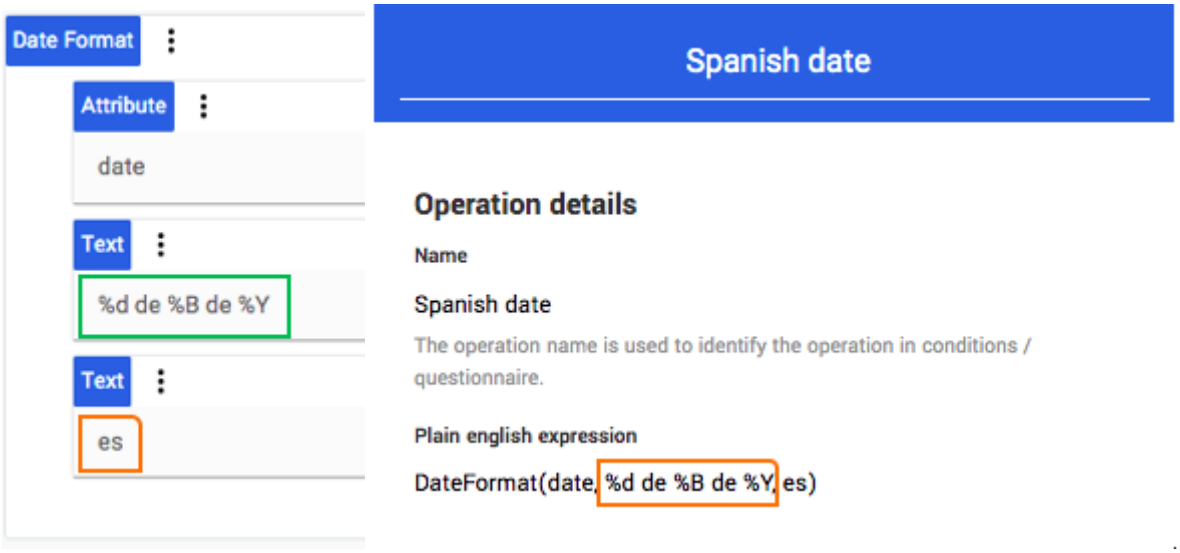
5.4. Bespoke Date Format (Date in Spanish)

5.4.1. Format your date in Spanish through the Long Date and Date Format operation.

5.4.2. For Long Date, carry out the same steps as seen here. Then apply the spanish locale code ‘es’ in the text box, as shown by the image below:

5.4.3. For Date Format, start the first string with the Date Format operation:

- For the first argument choose Attribute, and select (or create), the attribute you would like to use for this operation.
- For the second argument, you should use Text. Use the date format codes, separated with a space and ‘de’ between each code. You can view a preview of what the operation might look like, when the codes are replaced with words, by looking at the ‘plain english expression’ under operation details to the right.
- For the third argument, you should use Text. Apply the Spanish locale code ‘es’.


Date Format
Attribute
date
Text
%d de %B de %Y
Text
es
Spanish date
Operation details
Name
Spanish date
The operation name is used to identify the operation in conditions / questionnaire.
Plain english expression
DateFormat(date, %d de %B de %Y, es)

6. Worked Examples on Lists

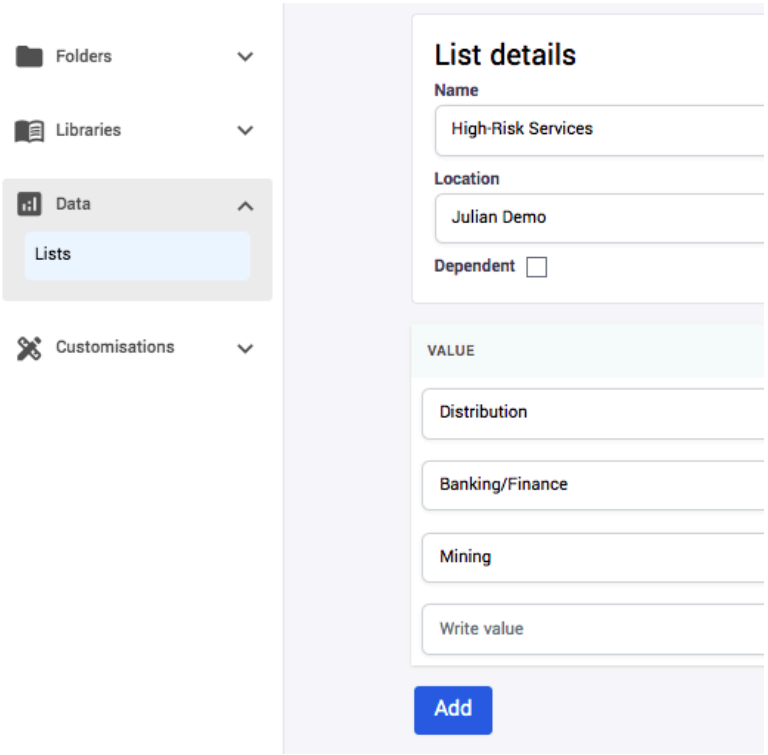
6.1. Comparing a value against a list

6.1.1. Watch this video to see how this operation is created, or read the instructions below.

6.1.2. This operation takes a value and compares it against a list. A basic example might be that a counterparty is asked what services they are providing. The given value can then be compared against a list of high-risk services, and if any of the services on the high-risk list are present, a 'high-risk' warning to the legal team will be triggered.

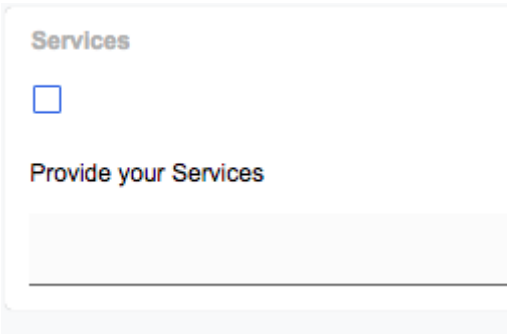
6.1.3. To create the operation, you firstly need to create a list. Lists are created by going into the template area, and selecting Data in the panel on the left.

6.1.4. Lists are stored on a template level meaning they can be updated, or added to over time.



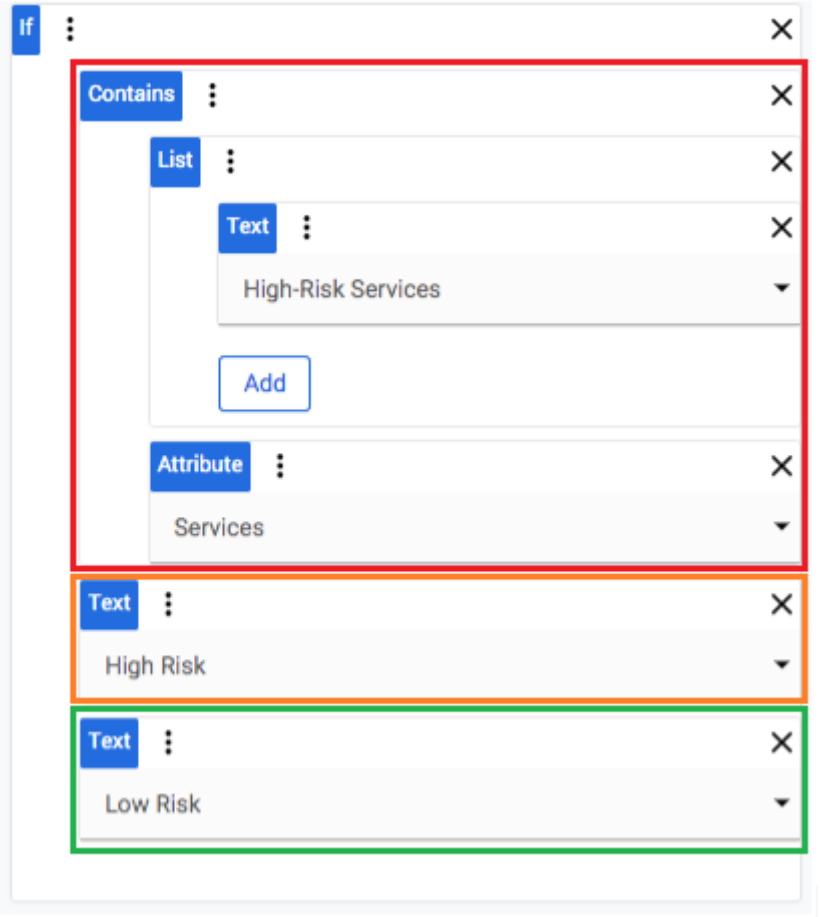
6.1.5. For this example, we are using a list of High-Risk Services. We want it so that if the value matches any in the above list, our operation will output the value "High Risk".

6.1.6. Go back into the template where you can either create a new placeholder, or condition, which will create the question. In this example, we will create a new placeholder called ‘Services’. Feel free to customise the question.



6.1.7. For the operation, we begin with ‘If’. “If” provides the means to control the output of the operation. To break it down, with reference to how an “if” block works:

- **Red block**: a logical test. The test here is essentially “Does the answer to the attribute ‘services’ contain a high-risk service from the high-risk list?”
- **Orange block**: the value to be outputted if that test is true - High Risk
- **Green block**: the value to be outputted if the test is false - Low Risk



6.1.8. Within the red block, we have an argument beginning with ‘contains’. This block is essentially checking to see if “Services” contains any of the services from the “High-Risk Services” list.

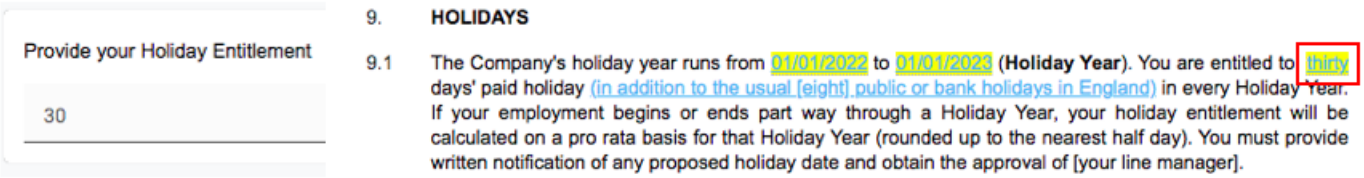
6.1.9. As such, the first list block is used to identify which list we want to use. Make sure you provide the exact name of the list as it appears in the “Lists” menu - The input is case sensitive.

6.1.10. The second block is ‘Attribute’, which links the operation to the answer provided to this question. It signifies which value we want to compare the first “List” block to.

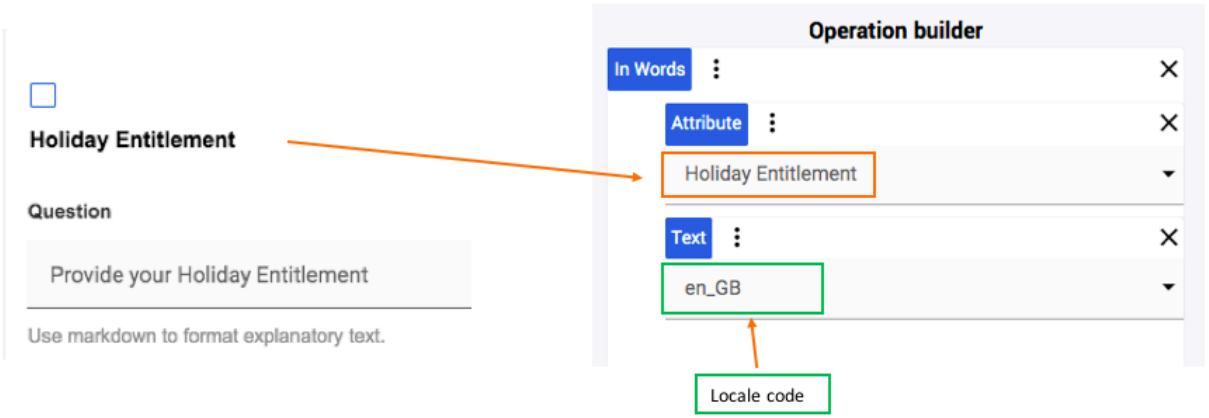
7. Worked Examples on Numbers

7.1. In Words

7.1.1. This operation will take an inputted number and output the number in words, such as in the screenshot below. For example, 30 will be outputted as 'thirty' as in the example below.



7.1.2. To create the operation, select In Words under Localisation, so that In Words is in the top band. Then choose the attribute where the operation will pull the information from (e.g. Holiday Entitlement). In the drop down list for the attribute, you can either select an existing attribute or you can create a wholly new attribute.

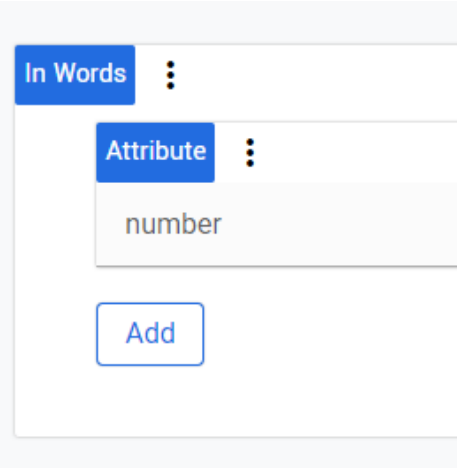


7.1.3. Finally, the number can be localised for different countries, e.g. from the US all the way to China. This is inputted by selecting the text input and typing in the relevant locale code.

7.1.4. Feel free to watch this video on how to create the In Words operation.

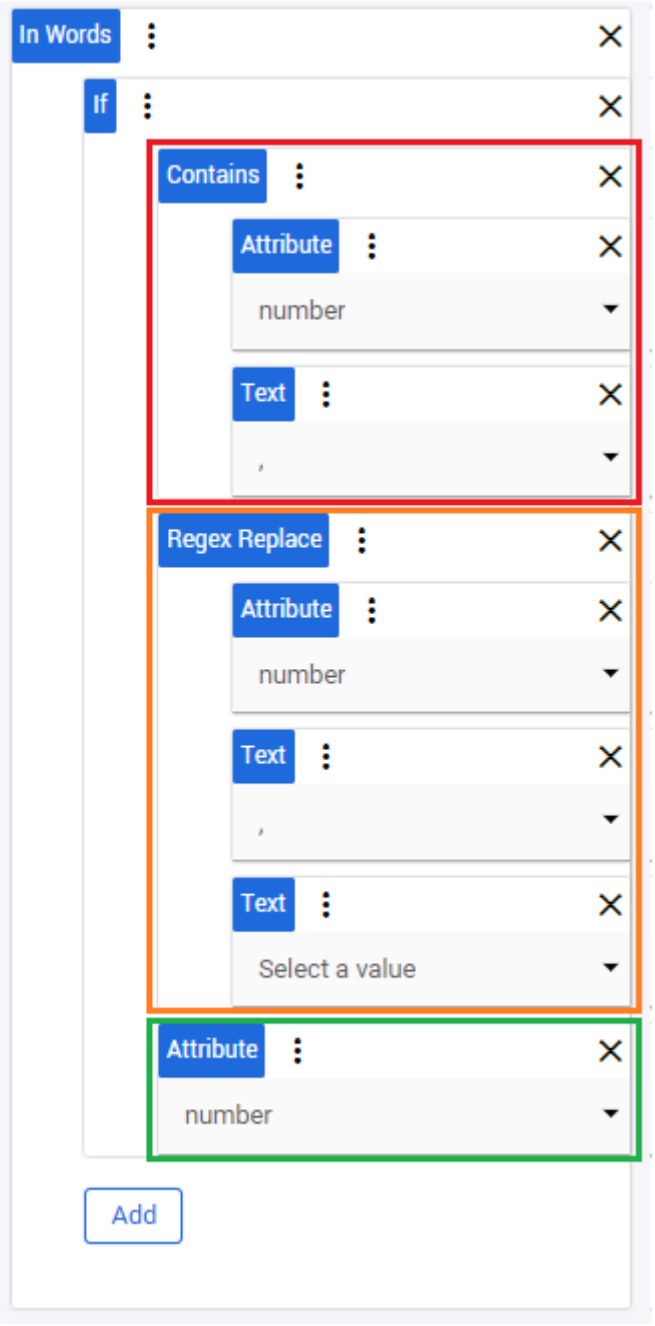
7.2. In Words: Numbers with Commas into Words

7.2.1. When utilising the simplified "In Words" function, numbers with commas may not be output in a desired manner. For example, an input of "2,000" will appear as an output of "two" rather than "two thousand".



7.2.2. To solve this problem, we can use the operation below. To break it down, with reference to how an "if" block works:

- **Red block**: a logical test. The test here is essentially "Does the answer to the attribute 'number' contain a comma?"
- **Orange block**: the value to be outputted if that test is true
- **Green block**: the value to be outputted if the test is false



Orange block: the value to be outputted if the answer to the attribute 'number' contains a comma

7.2.3. To break it down, with reference to how a "regex replace" block works:

- Attribute: The string to be modified.

- Text: The expression that will be replaced by the Text block below it. In this case, the expression in the first Text box is ",".
- Text: This is what should replace each of the matches listed in the above Text box. In this case, this text box is blank.

7.2.4. This regex replace block essentially replaces any "," in a number with a blank, effectively removing the "," from any number.

7.2.5. For example, "2,000" will be outputted as the value "2000".

7.2.6. This will allow the "In Words" function to read the number in its pure numeric form, so that it can accurately convert it to "two thousand".

Green block: the value to be outputted if the answer to the attribute 'number' does not contain a comma

7.2.7. If the attribute "number" does not contain a comma, it will simply be outputted as it was. For example, "2000" will be outputted as the value "2000". This is because no change is needed since the number already doesn't have a comma in it. The "In Words" function will be able to read the number just fine.

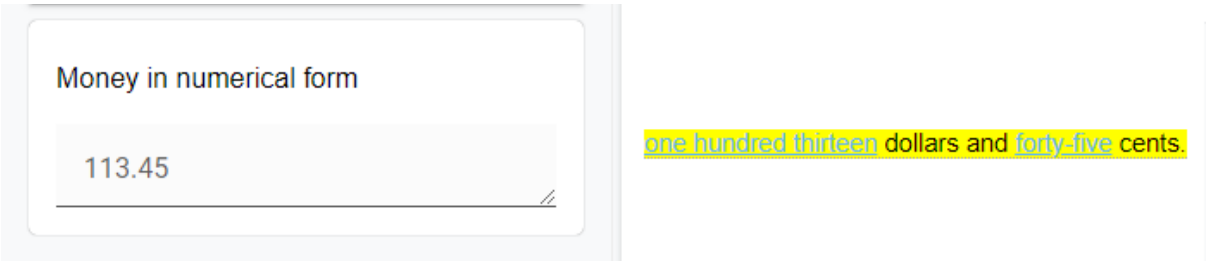
7.2.8. Watch the video **here** on how to create this operation

7.3. In Words: Money

7.3.1. This operation will take an inputted amount of money and output this amount of money in words, such as in the screenshot below. For example, “113.45” will be outputted as “one hundred thirteen dollars and forty-five cents” as in the example below.

7.3.2. Note: there is a currency format in the questionnaire, however, this only alters the currency symbols rather than transforming that number into its worded form (£113.45 or 113.45 AUD). This operation can look at the numbers on each side of the [.] and transform them into their worded format.

7.3.3. To create this function, we will require 3 different operations.



The general logic of the operation set

7.3.4. We will be splitting the inputted number into two sections, the numbers before the decimal place, “.” and the numbers after the decimal place, “.”

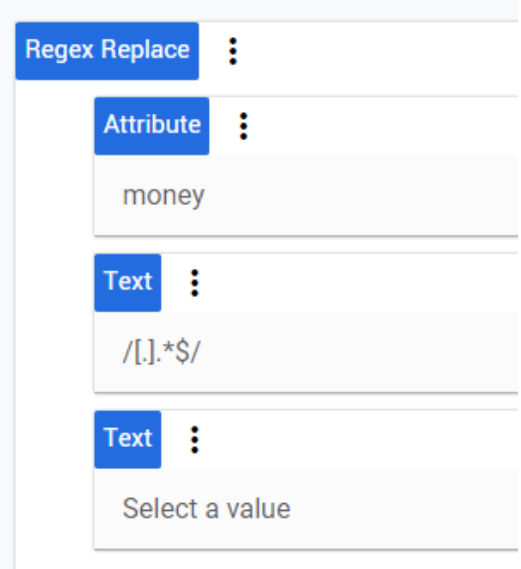
7.3.5. We will then convert these numbers into words and join them with the words “dollars and” and “cents” in the appropriate order so that they appear as they do above.

Setting up the number attribute

7.3.6. First, we will have to set up an attribute that will allow us to input a number. In this case, a placeholder titled “money” has been created.

First operation: Isolating the numbers before the decimal place

7.3.7. This first operation removes all digits from a number except for those before the decimal place. For example, “9876.54321” will be outputted as “9876”.



7.3.8. Above is the first operation. In this example, the operation has been given the name “regex before”. With reference to how a “Regex Replace” block works:

- **Attribute**: The string that will be modified. In this case, the attribute “money” has been inserted.
- **Text**: The expression that will be replaced by the Text block below it. In this case, the expression in the first Text box is “/[.].*$/”. This is a regular expression which selects every digit after the decimal place, including the decimal place.
- **Text**: This is what should replace each of the matches listed in the above Text box. In this case, this text box is blank.

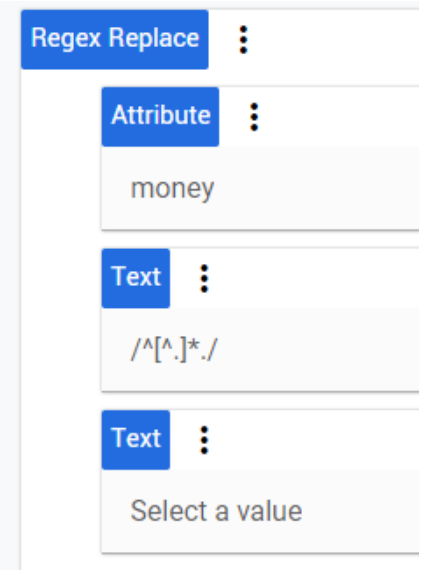
[Tip: This example is looking for the numbers on each side of the [.]. Altering the symbol within the squared brackets to a [-] means you can also separate a sequence if it was divided by a [-]. E.g a client matter number (1234567-0004321)**]**.

7.3.9. The operation replaces every single digit after the decimal place (including the decimal place) with a blank. In the number “9876.54321”, the selection “.54321” will be replaced with a blank, leaving the number “9876”.

7.3.10. As such, this operation titled “regex before” will only output the digits before a decimal place.

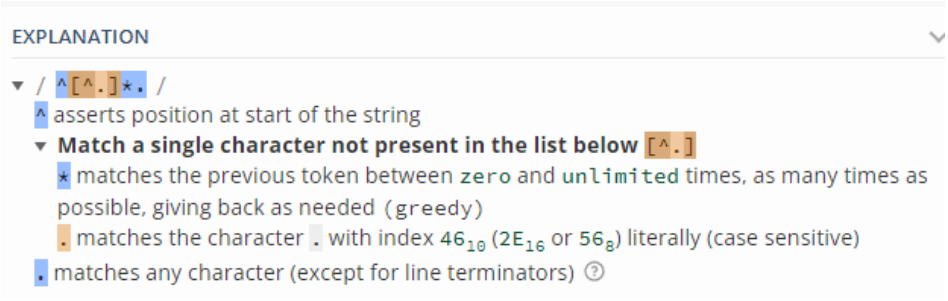
Second operation : Isolating the numbers after the decimal place

7.3.11. This second operation removes all digits from a number except for those after the decimal place. For example, “9876.54321” will be outputted as “54321”.



Above is the second operation. In this example, the operation has been given the name “regex after”. With reference to how a “Regex Replace” block works:

- **Attribute**: The string that will be modified. In this case, the attribute “money” has been inserted.
- **Text**: The expression that will be replaced by the Text block below it. In this case, the expression in the first Text box is “/^[^.]*./”. This is a regular expression which selects every digit before the decimal place, including the decimal place.



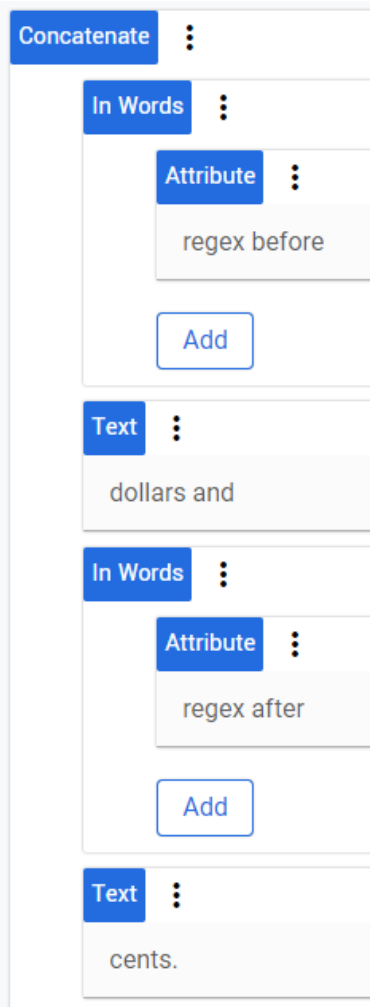
- **Text**: This is what should replace each of the matches listed in the above Text box. In this case, this text box is blank.

7.3.12. This operation is replacing every single digit before the decimal place (including the decimal place) with a blank. In the number "9876.54321", the selection "9876." will be replaced with a blank, leaving the number "54321".

7.3.13. As such, this operation titled "regex after" will only output the digits after a decimal place.

Third operation : Joining the outputs together

7.3.14. Now we will proceed to use "concatenate" to join the various strings of text together

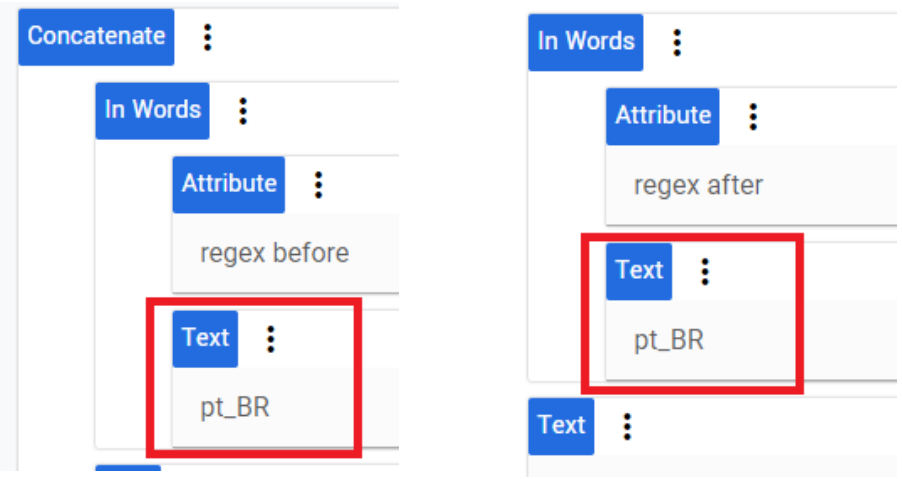


- **In Words**: This argument will return the output of the "regex before" operation in words. Essentially, the numbers before the decimal place will be output as words.
- **Text**: This is the text that will follow the outputted numbers above. In this case, we have typed " dollars and ". Ensure that the spaces are present in the inputted text so that they appear in the final output.
- **In Words**: This argument will return the output of the "regex after" operation in words. Essentially, the numbers after the decimal place will be output as words.
- **Text**: This is the text that will follow the outputted numbers above. In this case, we have typed " cents.". Ensure that

the spaces are present in the inputted text so that they appear in the final output.

7.3.15. In this example, this operation has been given the name “concatenate”

7.3.16. If you would like to change the language in which the number is outputted, simply add a “Text” input into the “In Words” block after “Attribute”. Then, type in your desired locale code. Also feel free to change the Text in between the “In Words” blocks to suit your desired language.

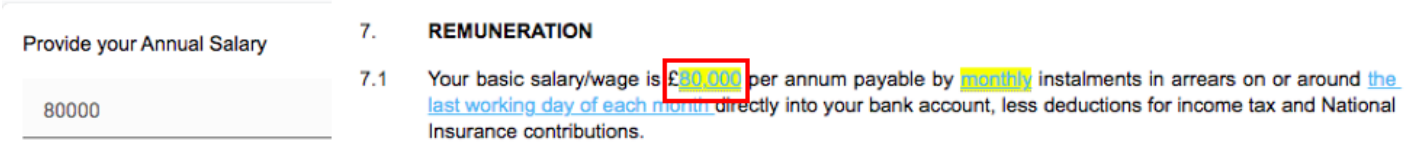


Final steps

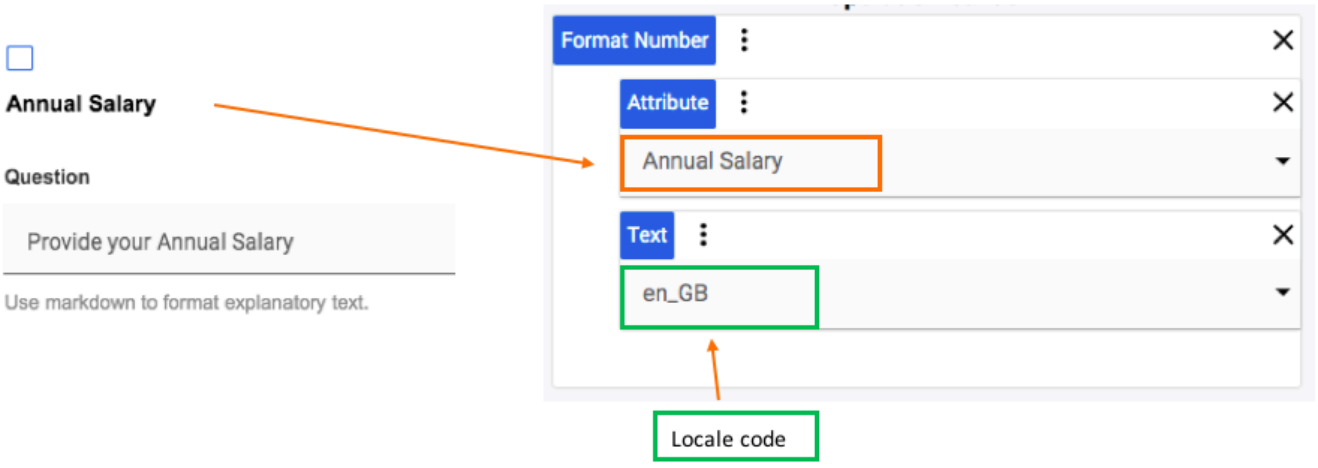
7.3.17. Now that we have our 4 operations, we will return to the “document area”. Insert the operation titled “concatenate” into the document. The operation will now work as intended!

7.4. Format Number

7.4.1. This operation will take an inputted number and output the number with the appropriate punctuation, such as in the screenshot below. The output can be localised with locale codes. For example, 80000 will be outputted as 80,000 as in the example below.



7.4.2. To create the operation, select Format Number under Localisation, so that Format Number is in the top band. Then choose the attribute where the operation will pull the information from (e.g. Annual Salary). In the drop down list for the attribute, you can either select an existing attribute or you can create a wholly new attribute.

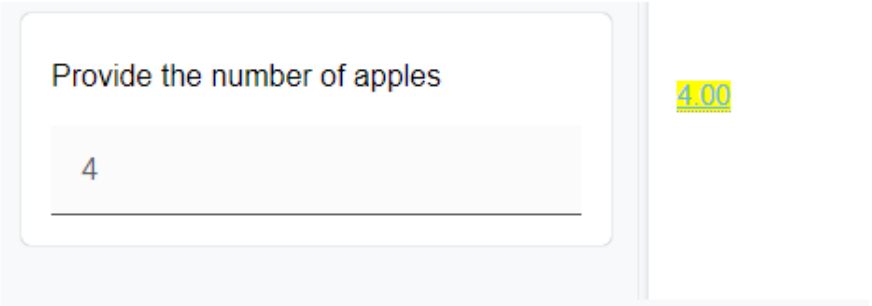


7.4.3. Finally, the number can be localised for different countries, e.g. from the US all the way to China. This is inputted by selecting the text input and typing in the relevant locale code.

7.4.4. Feel free to watch this video on how to create the Format Number operation.

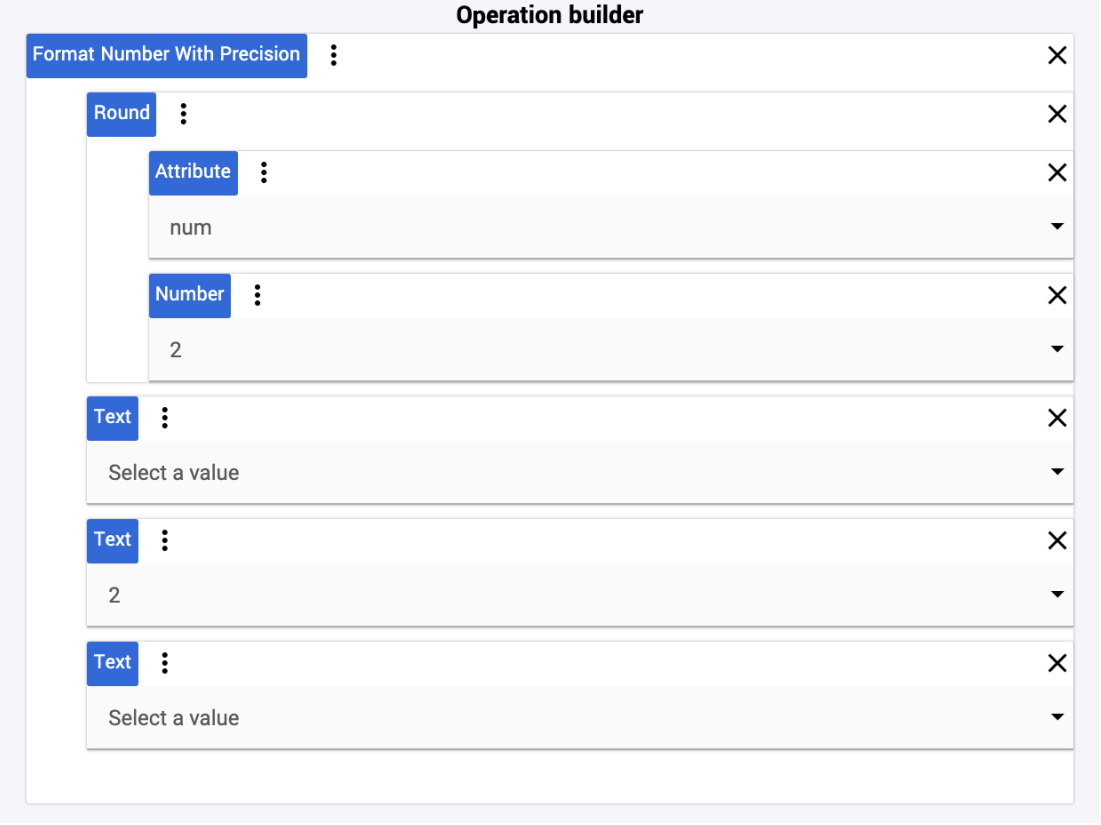
7.5. With Precision - Including 0's when rounding numbers

7.5.1. This operation works in conjunction with the "round operation" to ensure that integers when rounded off, do so with a .00 as in the example below.



7.5.2. Please note that using the round operation on its own will not include any trailing 0's when the number is rounded. For example, without the "With Precision" wrapper, the number 4.10 will appear as 4.1.

7.5.3. Now we will show you how to create this operation.



7.5.4. An example of this operation can be seen above. The arguments for this operation are:

7.5.5. **Round:**

- **Attribute:** The attribute that will contain the number you'd like to be rounded
- **Number:** The number of decimal places you'd like the number to be rounded to. For example, if you insert the number 2 here, an input of 4.235 will be rounded to 4.24

7.5.6. **Text:** This is the locale code, which can be used to customise the rounded number. You can either enter a locale code or leave the text box blank.

7.5.7. **Text:** The number of decimal places you'd like the result of the round operation to be outputted with. For example, if you insert the number 3 here, an input of 4 will be outputted as 4.000.

7.5.8. **Text:** You can either leave this argument blank or enter 'keep_zeros'. If you enter keep_zeros, whole numbers such as 4 will be formatted to display as 4.00. If you leave it blank, whole numbers will display as 4. However, numbers such as 4.60 will display as 4.6.

7.5.9. You can then use this operation directly in the template itself.

7.5.10. Watch this video on creating the operation.

7.6. Numbers in Ordinal Form

7.6.1. Below is a short list of Cardinal Numbers vs Ordinal Numbers.

Cardinal Numbers	Ordinal Numbers
1	1st
2	2nd
3	3rd
4	4th
5	5th

7.6.2. If you would prefer to just create the operation, simply copy the two operations below. However, if you would like to understand how the operation works, do read on.

The general logic of the operation set

7.6.3. The operation set that we are creating will ensure that all numbers inputted will be outputted as Ordinal Numbers. This would be simple if all the numbers ended in a "th". However, as can be seen above, "1st", "2nd" and "3rd" do not end with a "th".

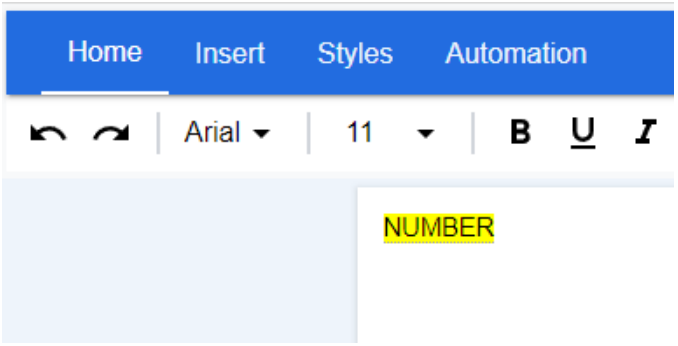
7.6.4. A solution to this would be to set up an argument that detects when the number "1", "2" or "3" appears at the end of a string of numbers. Once those numbers are detected, the appropriate ordinals will appear at the end of the string. (**Operation 1**)

7.6.5. However, a problem arises when we realise that "11", "12" and "13" also end with "1", "2" and "3". But, their appropriate ordinals are "th". If we were to only have the above argument, they would appear as "11st", "12nd" and "13rd".

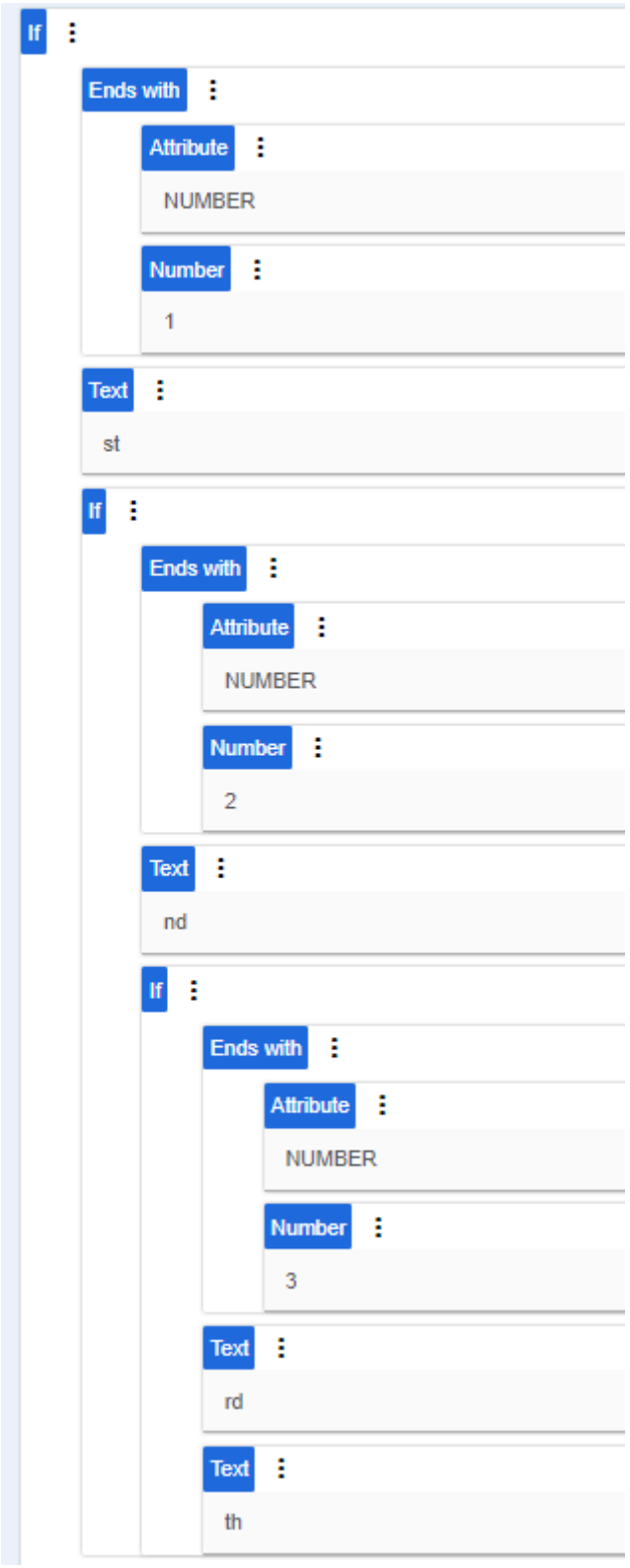
7.6.6. To solve this problem, we will have to add another argument that detects when the numbers "11", "12" or "13" appear at the end of a string of numbers. Once those numbers are detected, the appropriate ordinal "th" will appear at the end of the string. (**Operation 2**).

Setting up the number attribute

7.6.7. First, we will have to set up an attribute that will allow us to input a number. In this case, a placeholder titled “NUMBER” has been created.



Operation 1


If
Ends with
Attribute
NUMBER
Number
1
Text
st
If
Ends with
Attribute
NUMBER
Number
2
Text
nd
If
Ends with
Attribute
NUMBER
Number
3
Text
rd
Text
th

7.6.8. This first operation will be titled “st,nd,rd,th”. This operation “st,nd,rd,th” will detect the last digit of NUMBER and will output the appropriate ordinal.

7.6.9. Casting our mind back to how “if statements” work, the if function takes three arguments:

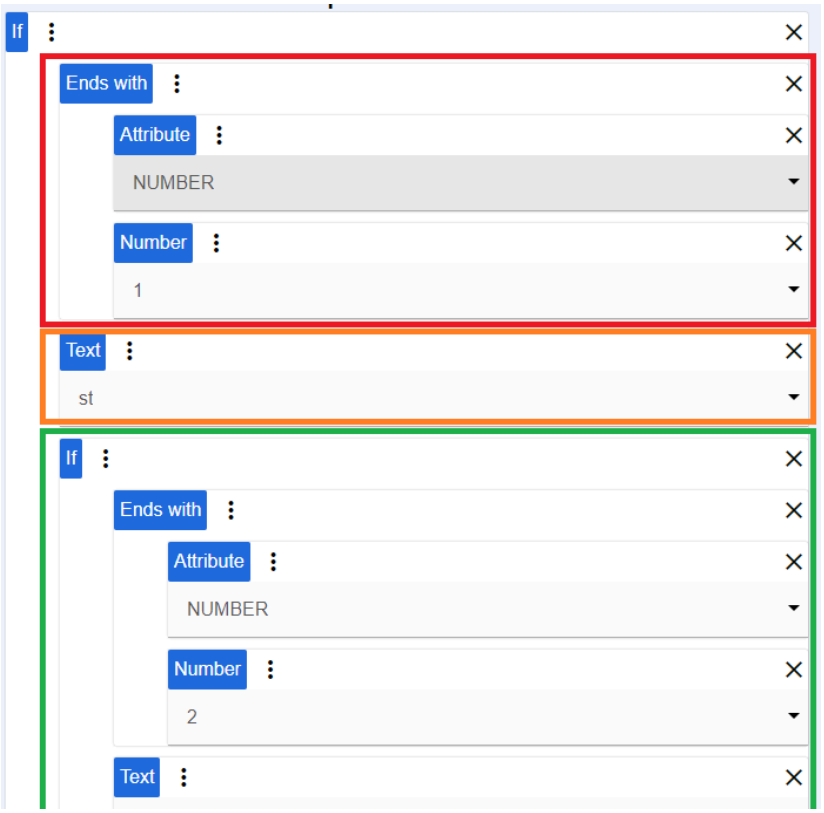
1. **The block in red**: a logical test. In this case, the test is “Is the last digit of NUMBER the number 1?”

2. **The block in orange**: the value to be returned if that test is true. In this case, if the last digit of NUMBER is the number 1, the test will be true and the value “st” will be returned.

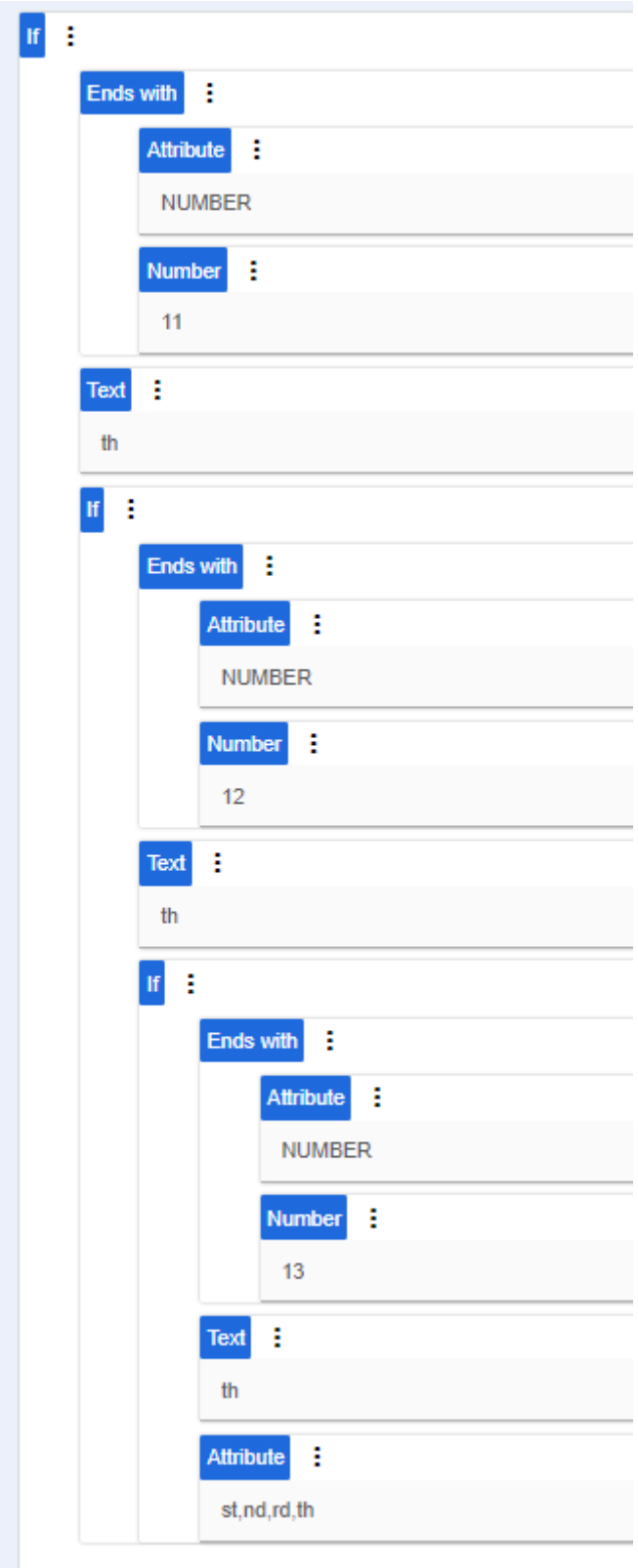
3. **The block in green**: the value to be returned if the test is false. In this case, if the last digit of NUMBER is NOT the number 1, the test will be false.

    a. If the test returns as false, the operation will move onto the next “if statement”, where the argument is “Is the last digit of NUMBER the number 2?” and vice versa.

    b. If the last digit of NUMBER is not “1”, “2” or “3”, the test at the end will return as false and the operation will output the value “th”. For example, “4” will cause the test to return as false and the operation will output the value “th”.



Operation Number 2


If
Ends with
Attribute
NUMBER
Number
11
Text
th
If
Ends with
Attribute
NUMBER
Number
12
Text
th
If
Ends with
Attribute
NUMBER
Number
13
Text
th
Attribute
st,nd,rd,th

7.6.10. This second operation will be titled “st,nd,rd,th last 2”. “st,nd,rd,th last 2” will detect the last two digits of NUMBER and will output the appropriate ordinal.

7.6.11. Casting our mind back to how “if statements” work, the if function takes three arguments:

1. **The block in red**: a logical test. In this case, the test is “Are the last two digits of NUMBER the number 11?”

2. **The block in orange**: the value to be returned if that test is true. In this case, if the last two digits of NUMBER are the number 11, the test will be true and the value “th” will be returned.

3. **The block in green**: the value to be returned if the test is false. In this case, if the last two digits of NUMBER are not the number 11, the test will be false.

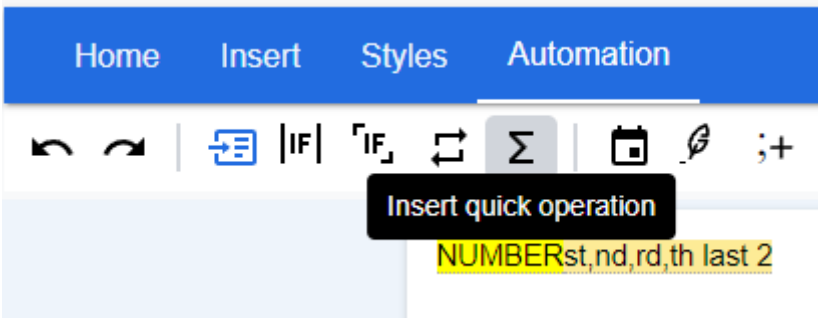
    c. If the test returns as false, the operation will move onto the next “if statement”, where the argument is “Are the last two digits of NUMBER the number 12?” and vice versa.

    d. If the last two digits of NUMBER are not the numbers “11”, “12” or “13”, the test at the end will return as false. The operation will then move onto the operation “st,nd,rd,th” that we discussed above.

        i. For example, “14” will cause the test to return as false and the operation will move on to the operation “st,nd,rd,th”. After the number “14” passes through that argument, it will be outputted as “th”.

Final steps

7.6.12. Now that we have our 2 operations, we will return to the “document area”. Insert the operation titled “st,nd,rd,th last 2” into the document behind the placeholder titled “number”. The operation will now work as intended!



7.7. Introduction to calculations in a table

7.7.1. This section will show you how to perform calculations in a table utilising functions from Math.

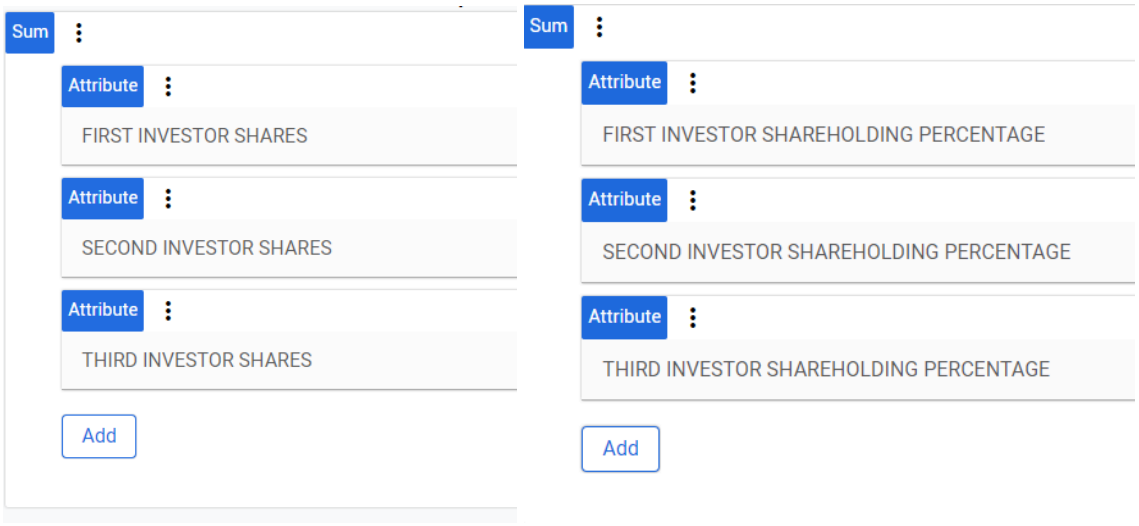
Investor	Number of Shares	Shareholding percentage
FIRST INVESTOR	FIRST INVESTOR SHARES	FIRST INVESTOR SHAREHOLDING PERCENTAGE
SECOND INVESTOR	SECOND INVESTOR SHARES	SECOND INVESTOR SHAREHOLDING PERCENTAGE
THIRD INVESTOR	THIRD INVESTOR SHARES	THIRD INVESTOR SHAREHOLDING PERCENTAGE
Total	TOTAL SHARES	TOTAL SHAREHOLDING PERCENTAGE

7.7.2. In this instance, we would like a sum of the number of shares at the bottom of its respective column. We would also like a sum of the shareholding percentage at the bottom of its respective column.

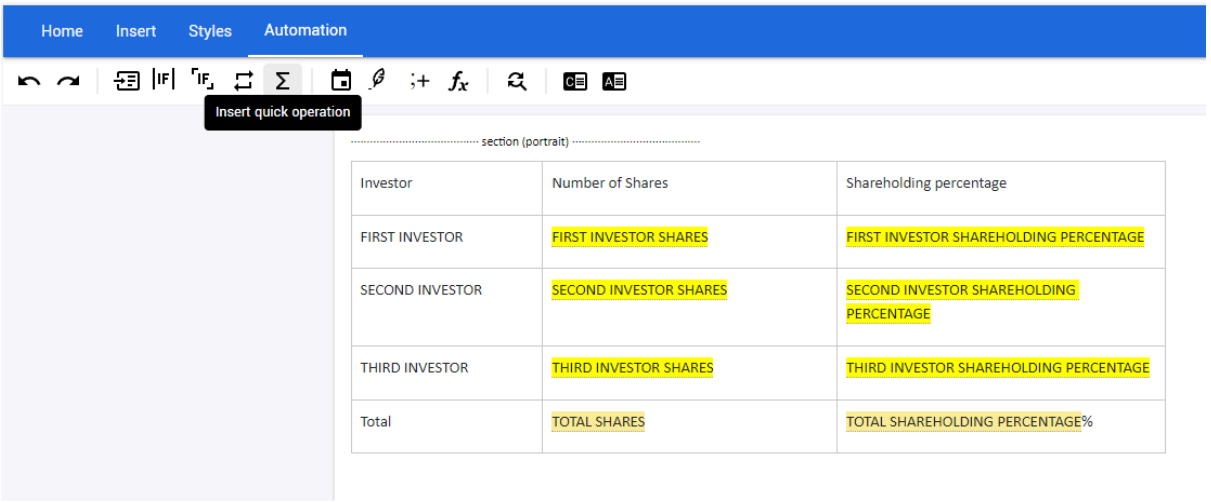
7.7.3. As such, we will create placeholders for the Investor Shares and Shareholding Percentage:

Investor	Number of Shares	Shareholding percentage
FIRST INVESTOR	FIRST INVESTOR SHARES	FIRST INVESTOR SHAREHOLDING PERCENTAGE
SECOND INVESTOR	SECOND INVESTOR SHARES	SECOND INVESTOR SHAREHOLDING PERCENTAGE
THIRD INVESTOR	THIRD INVESTOR SHARES	THIRD INVESTOR SHAREHOLDING PERCENTAGE
Total	TOTAL SHARES	TOTAL SHAREHOLDING PERCENTAGE

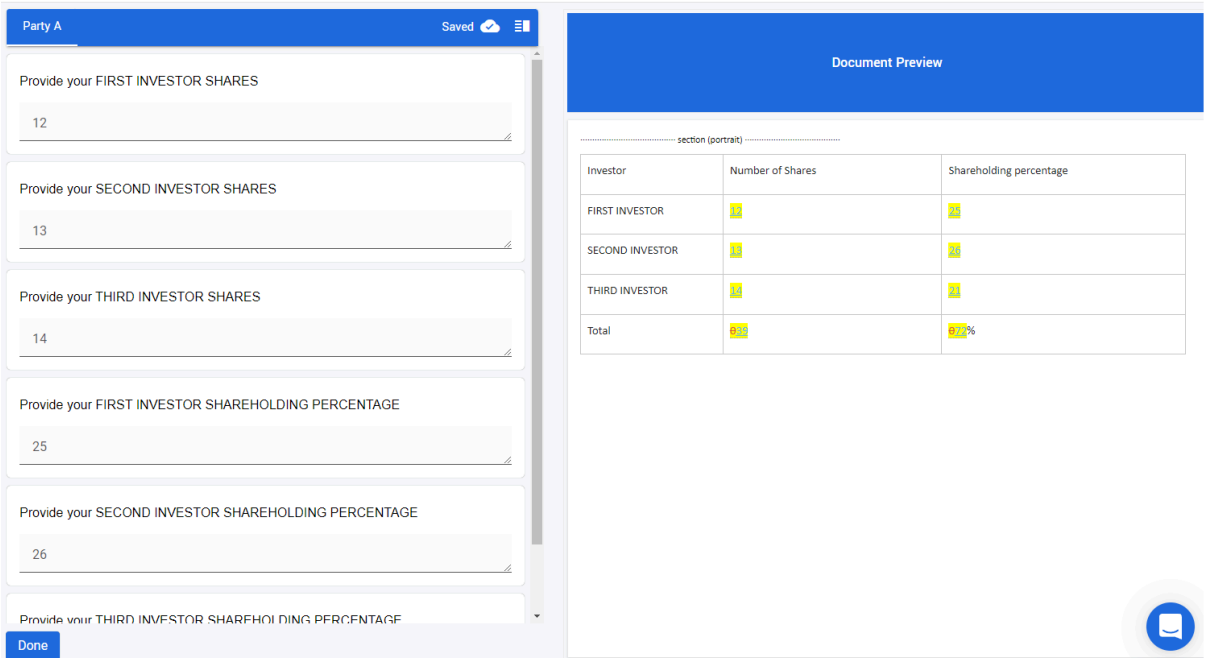
7.7.4. Now, we will navigate to the Operations tab to create the relevant operations.



7.7.5. When you have completed setting up your operations, return to the “document” section and add the operations to the table as such. Ensure that a “%” sign is added at the back of “TOTAL SHAREHOLDING PERCENTAGE”.



7.7.6. When creating the document, fill out the questions as directed. Your operation should function like this:



7.7.7. Other calculations can also be performed in a table like this. Do refer to the Math section on how this can be done.

7.8. Complex Calculations in a table

- Calculate the total sum in each column
- Transform these figures into their respective percentage values
- Round the figures to 1 decimal place

7.8.1. First, import or create a table in the editor. For the purpose of demonstration, we will create a table for the total shareholding of a company. There are four groups of shareholders: Main Shareholders, Directors, Employee Share Scheme and Other.

7.8.2. For the column to show the total number of shares held by each group, create a placeholder. E.g. 'main_shareholder_no'. Then go into the questionnaire and customise the question for this attribute (main_shareholder_no) to something such as: "provide the total number of shares held by the Directors". Update the question format to 'Number', so users can only enter a number into the input field.

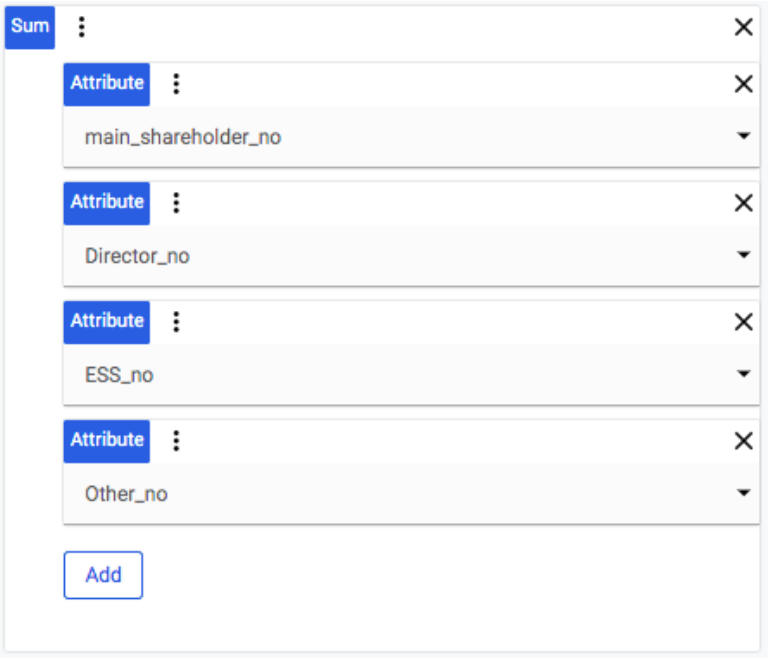
7.8.3. Do the same for each group, so that they all have a placeholder.

Name	No. Shares	Percentage of Company
Main Shareholders	main_shareholder_no	MS_percentage%
Directors	Director_no	Director_percentage%
Employee Share Scheme	ESS_no	ESS_percentage%
Other	Other_no	Other_percentage%
	total: Total_Share	**total:** Total Percentage%

7.8.4. To calculate the total number of shares in the column we need to create an operation using 'Sum' (there are more than two values).

7.8.5. Go into the operations area and create a new Operation - e.g. "Total_Share". Use the Sum attribute as the top string.

7.8.6. For the first argument, use attribute and select “main_shareholder_no”. Then, add another attribute, selecting ‘Director_no’ - and so on. This will add together each attribute.



7.8.7. Go back into the document, click on the ‘insert quick operation’ button and add the operation where the total number of shares should appear.

Turn each value into a percentage

7.8.8. In Avvoka, you are able to perform numerous calculations within the same operation, or draw out the output of previous operations into a new one. Here, we want to divide the number of shares for each group by the total amount: [(A / total) x 100].

7.8.9. To prevent each number from having multiple decimal places, we also want to round the number to 1 decimal place.

7.8.10. In the first string, use the ‘Round’ operation.

7.8.11. In the second string use ‘Multiply’. This will multiply the decimal place by 100 to give us a percentage value: [(A / total) x 100].

7.8.12. In the third string use ‘Divide’. The result of this calculation is the one I intend to multiply by 100: [(A / total) x 100]. The first argument is the attribute e.g. ‘main_shareholder_no’. The second argument is the number I am dividing by. Use ‘attribute’ again and select “Total_Share”.

7.8.13. The string below this is the number I am multiplying by. Make sure the format is ‘Number’ and choose the number you would like to use, e.g. 100.

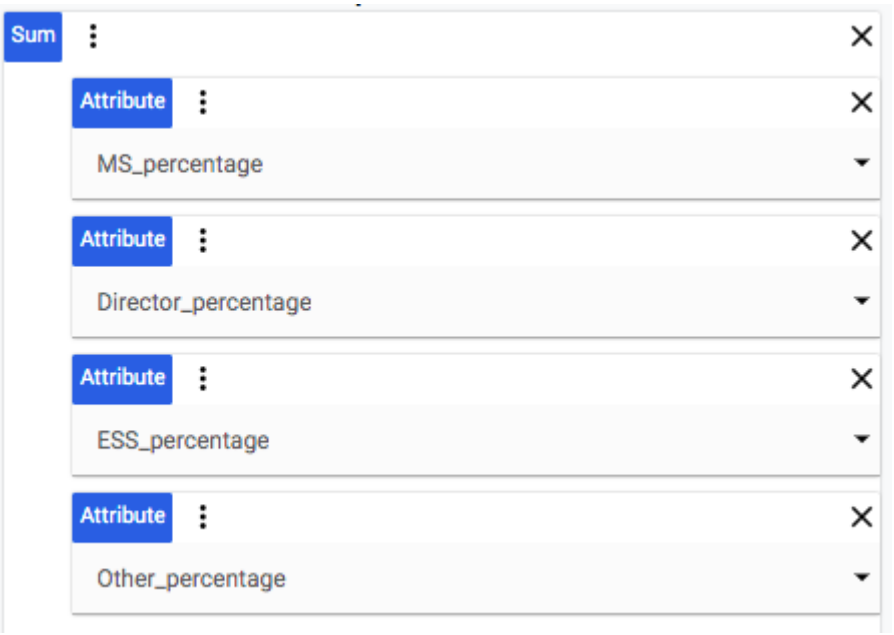
7.8.14. The final number is related to ‘Round’. We use the ‘Text’ format. Each number determines how many decimal places you would like to round the number to. In this example, we have chosen only 1.

7.8.15. Repeat this operation four times, changing where we used the attribute ‘main_shareholder_no’ to reflect the number of shares held by each group of shareholders.

Calculate total percentage

7.8.16. As demonstrated in the first part of this example, now use the operation ‘Sum’ to calculate the total percentage.

7.8.17. Where Sum is the first string, use Attribute and select the names of each operation used to calculate the percentage to add these together.

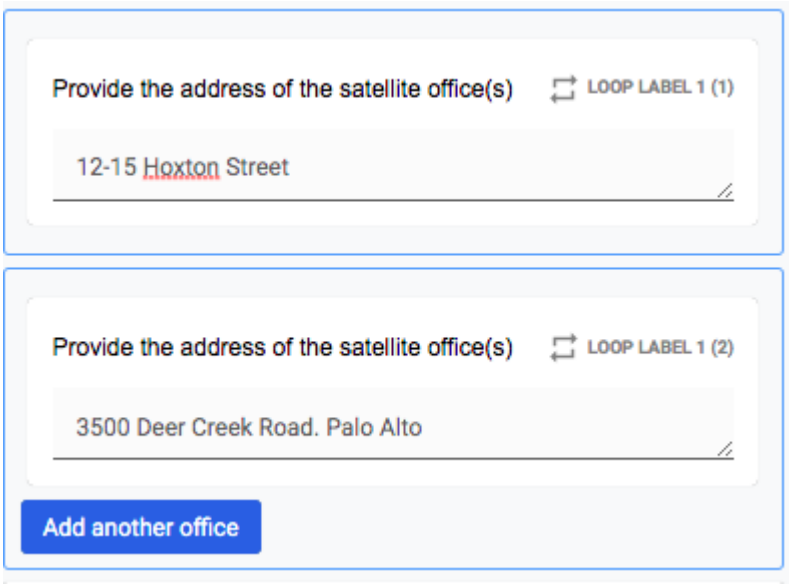


8. Worked Examples on Loops

8.1. Creating plural text when there is more than one looped attribute

8.1.1. In a template, you may have a situation where an attribute may be looped multiple times or not at all. If the attribute is looped multiple times, you may wish to have a word change to a plural form. If the attribute is not looped at all, it will remain singular.

8.1.2. For example, in the case where the looped attribute is “Office Location”, you may wish to have the word “offices” appear if there is more than one office provided. However, if only one office is provided, you would want “office” to remain singular.



5. **PLACE OF WORK**

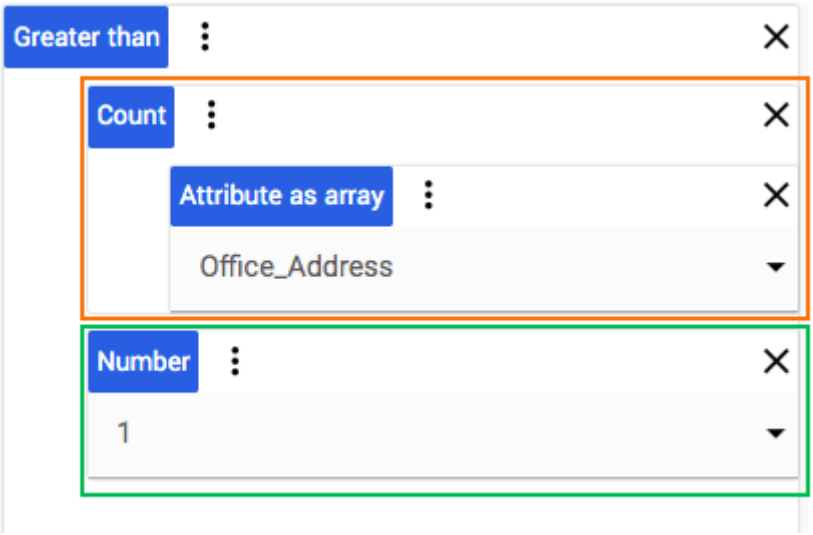
5.1 Your normal place of work is or such other place as the Company may reasonably require.

5.2 The employee may be required to work at the company's satellite offices, located at 12-15 Hoxton Street, 3500 Deer Creek Road. Palo Alto

8.1.3. First, we will have to create a looped attribute in the editor. To do so, create your desired placeholder. In this scenario, we have created the placeholder “Office_Address”.

8.1.4. Then, highlight the placeholder with your cursor and click on the button in your **Automation** tab in the top toolbar to create a loop over them. Your highlighted text will then be housed in a blue loop container.

8.1.5. Now, navigate to the Operations Tab and construct the following Operation.

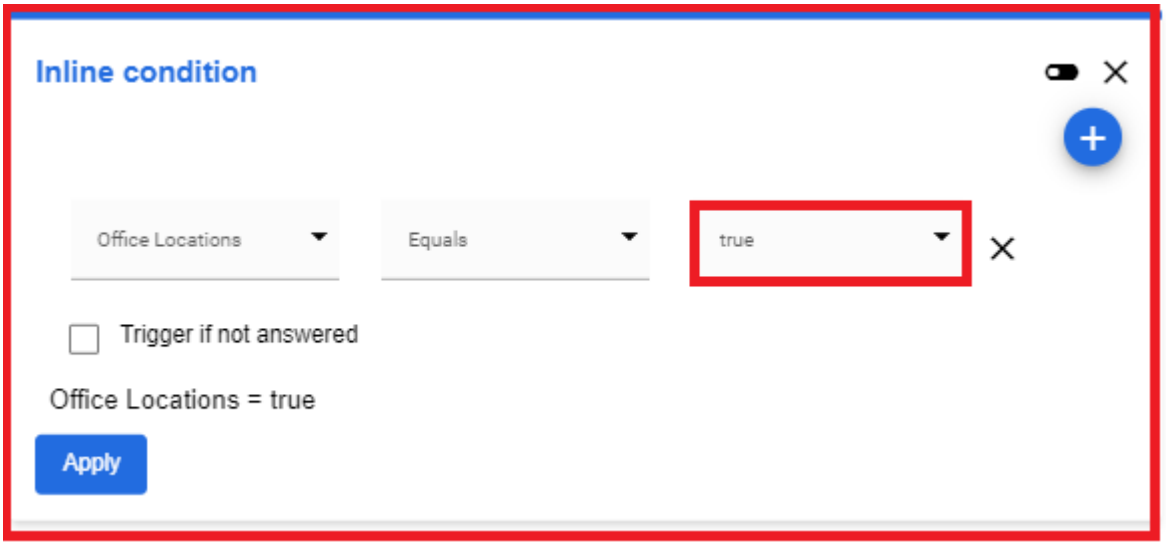


8.1.6. To break it down, with reference to how a “greater than” block works:

- **Greater than**: If the value of the **Orange block** is greater than the value of the **Green block**, the operation will evaluate the comparison to be “True”.
- **Orange block**: the value of the number of times the attribute “Office_Location” has been looped.
- **Green block**: the value “1”

8.1.7. Essentially: If the number of times “Office_Location” is looped is greater than 1, the Operation will output the value “**true**”.

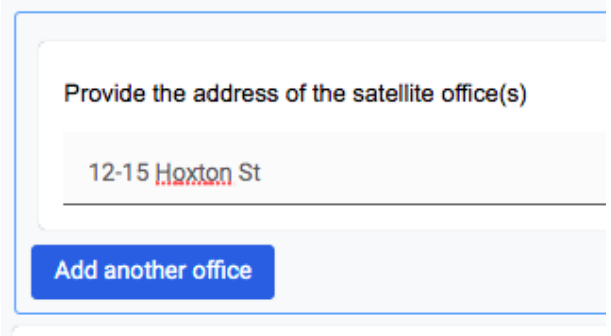
8.1.8. Now, we will return to the document. Type the word “Offices” and add an in-line condition to the “s” at the back of it. We will set it up such that the “s” will appear if the relevant Operation returns as **true**.



8.1.9. We have used an inline loop to display the office locations in an array. This uses the operation titled ‘Office Locations’, which has been added to the document shown in the image above.

8.1.10. To create this operation, wrap the looped attribute with a block condition, using a value that will not be met so that it doesn’t show in the document. See more on creating an inline loop here.

8.1.11. Your operation should now work as intended. If only one office location is provided, then the document should read as “office”.

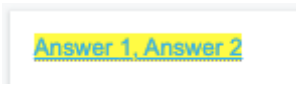


5.2 The employee may be required to work at the company's satellite office, located at 12-15 Hoxton St

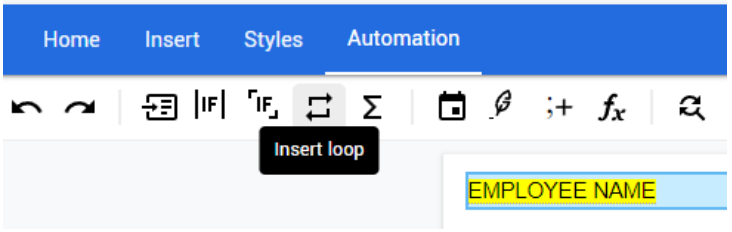
8.1.12. If multiple colours are inputted, then the document should read as “offices”, as shown at the beginning of this article.

8.2. In-line loops

8.2.1. This will show you how to display your looped responses in one single line, rather than multiple lines.

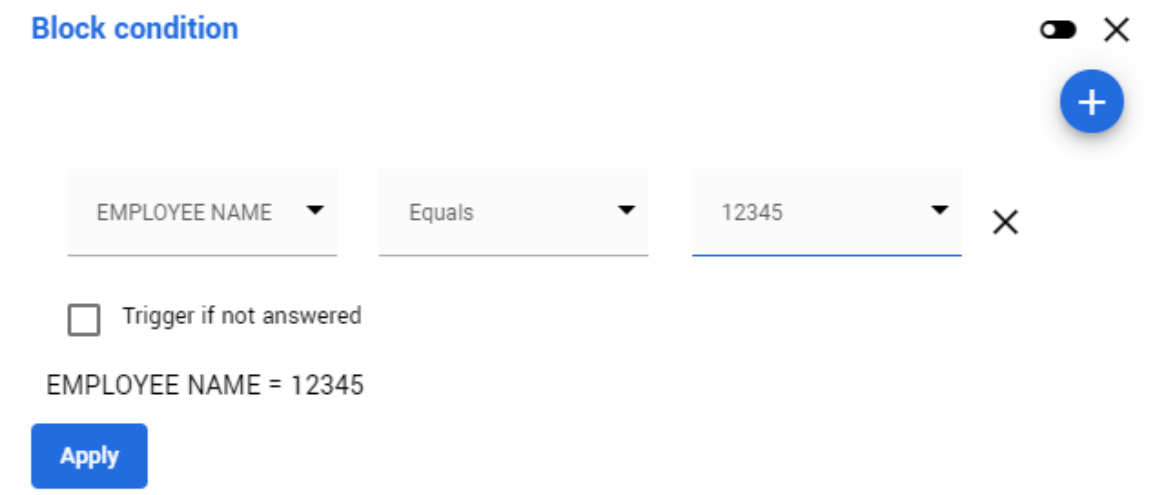


8.2.2. To start off, create a placeholder in your document. Then, highlight the placeholder with your cursor and click on the button in your **Automation** tab in the top toolbar to create a loop over them. Your highlighted text will then be housed in a blue loop container.



8.2.3. **NOTE**: If you do not want the original placeholder to display in the template body itself, you can hide the loop. To do so, highlight the entire placeholder and create a block condition over it.

- Set the attribute value to be any existing attribute.
- Set the final value to a value that will never be met by the attribute. (*like below*)



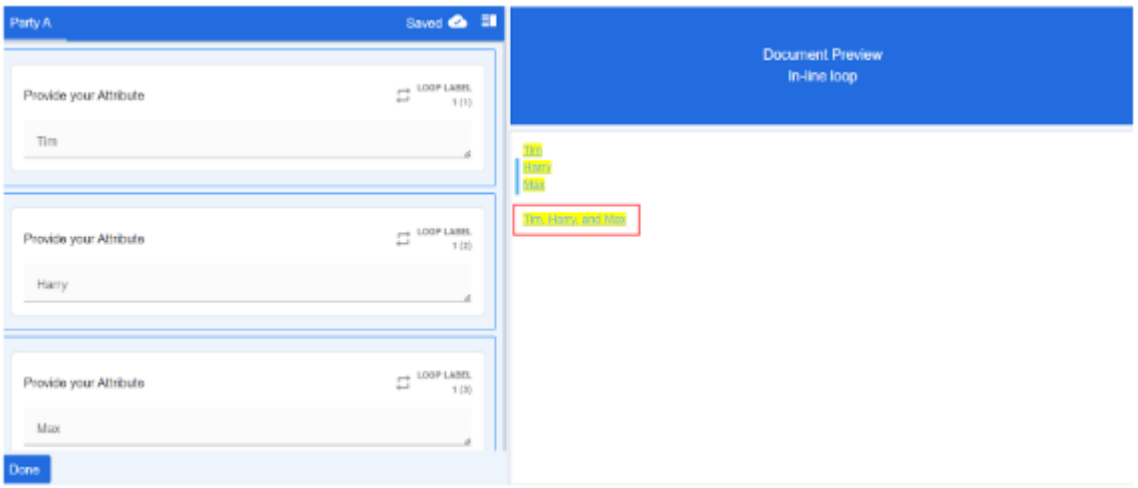
8.2.4. As such, this block condition will never be met. Hence, the loop and attribute will never appear in the final document. Your editor should now look like this.

EMPLOYEE NAME

8.2.5. Now that we have created our looped placeholder, create the following operation. This is a Join operation with up to four arguments:

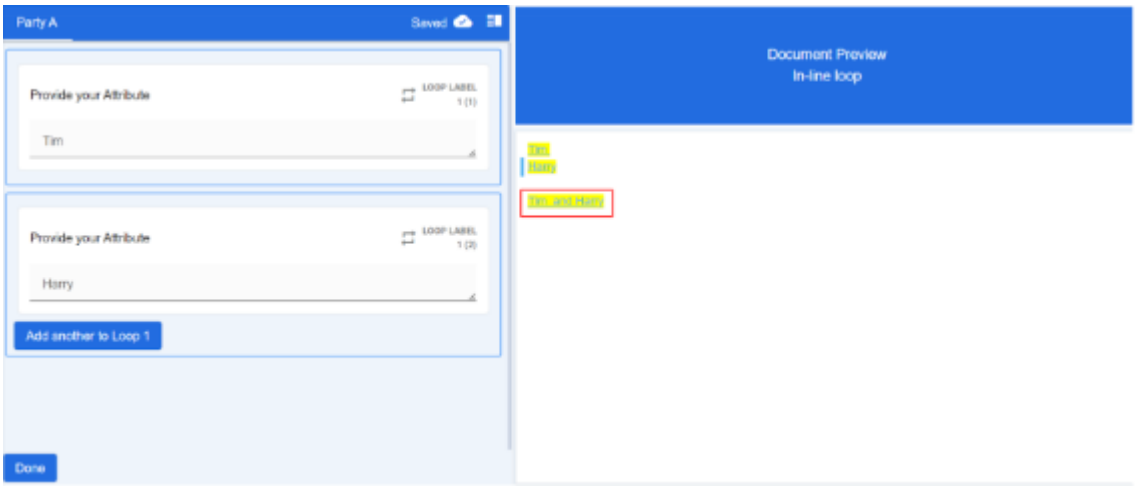
- **Looped Attribute (Attribute as array)**: The name of the attribute that corresponds to the looped placeholder. (*Required*)
- **Default Separator (Text):** The separators that you want between each looped response. So in this example if you want to display your answers as “Answer 1, Answer 2, Answer 3” you should use “, “ as the separator, including the space after the comma. (*Required*)
- **Penultimate Separator for more than 2 iterations (Text)**: The separator distinguishes between the penultimate and final iterations of your attribute. For instance, if you plan to display your answers as 'Answer 1, Answer 2, and Answer 3,' then specify ', and' as the penultimate separator.

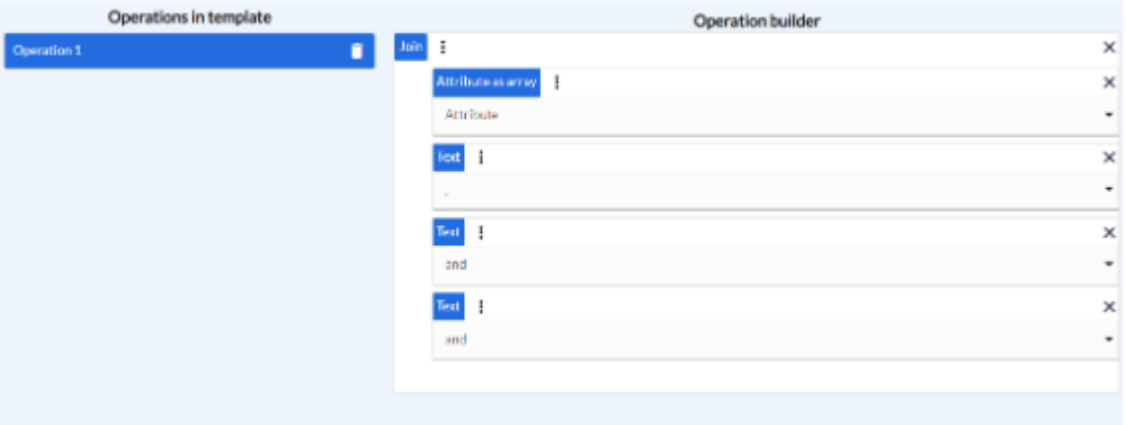
  **N.B. Ensure that you include the desired spacing in this argument.*



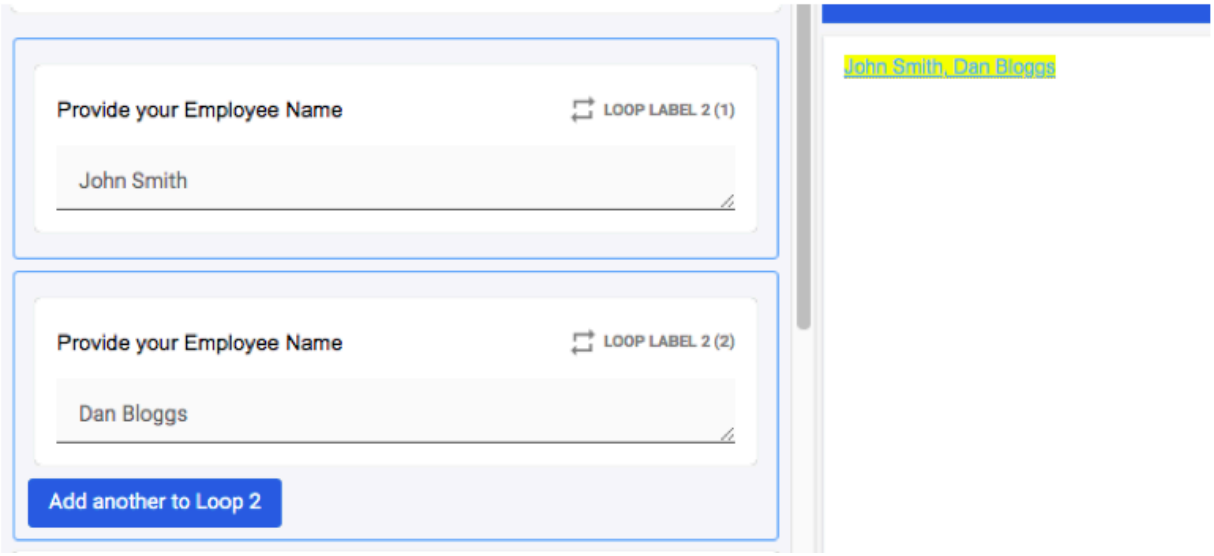
- **Separator for two iterations (Text):** An optional separator for specifying a separator for if there are only two iterations of the looped attribute. For example, if you want to showcase your answers as 'Answer 1 and Answer 2,' employ ' and ' as the separator.

  **N.B., make sure there is space both before and after 'and'.*





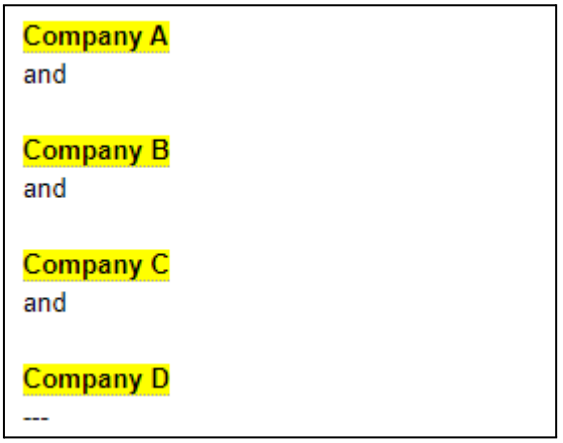
8.2.6. You can then insert the operation directly into the template itself using the “Quick Operation” button in the Automation tab.



8.2.7. Above is an example of how your content will appear when a document is created.

8.3. Automatic list separators across loops

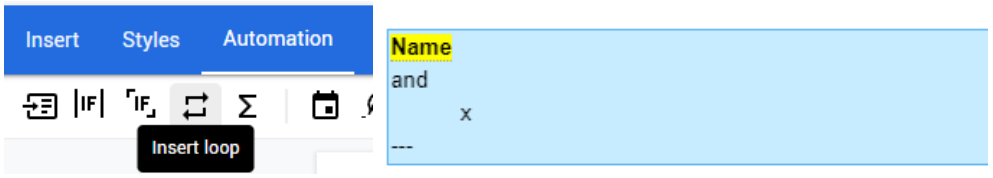
8.3.1. Using operations as conditions, you can set up automatic list separators to work across loops. For example, you may need to list multiple entities in a document's preamble under the same party as such:



8.3.2. First, set up the placeholders you would like to loop as well as the separators in the editor. Be sure to account for any line breaks in your separators as well. To ensure that line breaks are accounted for in the block condition, press on your **Tab** key to insert a tab space.

8.3.3. You may also want to insert any character after the tab space so it will be easier to highlight the line break later when you need to set up a block condition. This character can be removed after the block condition has been created. In this example, an x has been inserted after the tab space.

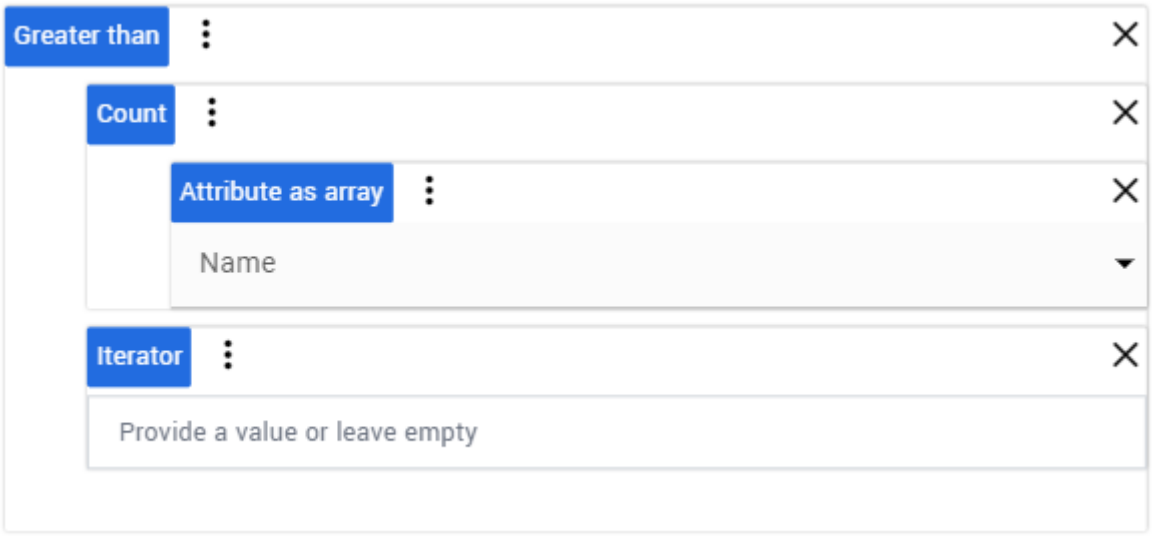
8.3.4. Second, highlight all placeholders and separator text with your cursor then click on the button in your **Automation** tab in the top toolbar to create a loop over them. Your highlighted text will then be housed in a blue loop container.



8.3.5. Next, set up block conditions over each separator. These block conditions will contain an operation which counts which compares the number of times the loop has been utilised to which iteration of the loop a certain value is. This will determine which separator is inserted as a block.

8.3.6. To do so, the operation will not be set up in the typical operations menu. Instead, it is set up in the template editor directly within the condition builder, but in **rendering mode** which allows you to construct operations in the condition builder directly. You can switch to **rendering mode** by clicking on the toggle in the top right corner of your condition builder.

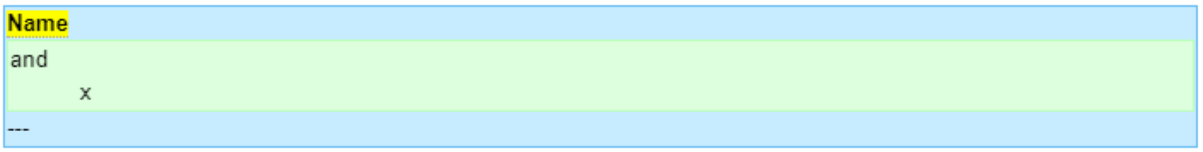
8.3.7. We will first set up the operation needed for the “and” separator along with the line break.



8.3.8. The operation built consists of two arguments nested in the **Greater than** function. The first argument returns a value reflecting the total number of times an attribute has been looped. First, add a **Count** function, then within it, add an **Attribute as array** function. In the dropdown menu, select any of the attributes which are in the relevant loop. This attribute will serve as the “count” for the argument.

8.3.9. The second argument numbers each instance of the loop according to which instance of the loop the value belongs to. For example, the 1st instance of the loop will return a value of 1, the 3rd instance a value of 3, etc. For this argument, insert the **Iterator** function. You can leave this function empty.

8.3.10. Thus, this block condition is triggered if the value drawn from the first argument (the total number of looped instances) is greater than the second argument (the Xth instance of the loop, where X is the value being examined). Click “apply” to apply the condition. This will create a green block container over “and” and the line break. You can now delete the **x** as it was there only to allow you to highlight the line break easily. However, leave the tab space there.

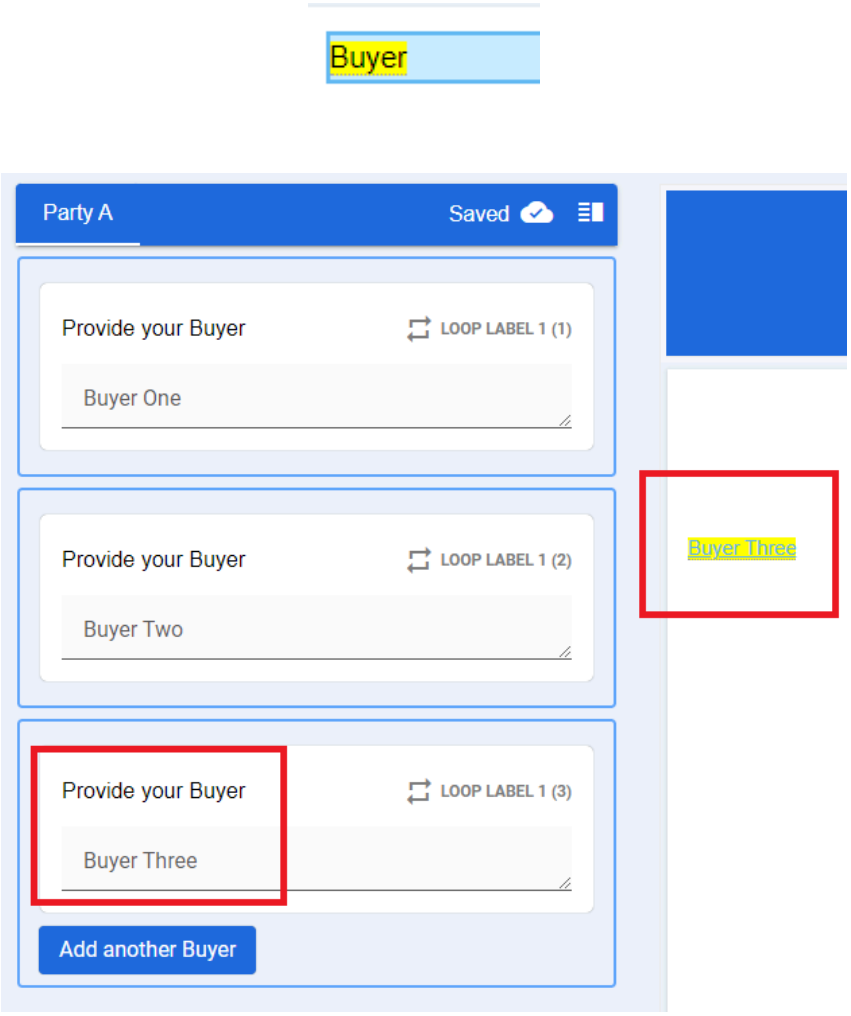


8.3.11. You can set this up similarly for the final separator. However, instead of using the **Greater than** function, you will use the **Equals** function. The two arguments within the **Equals** function remain the same.

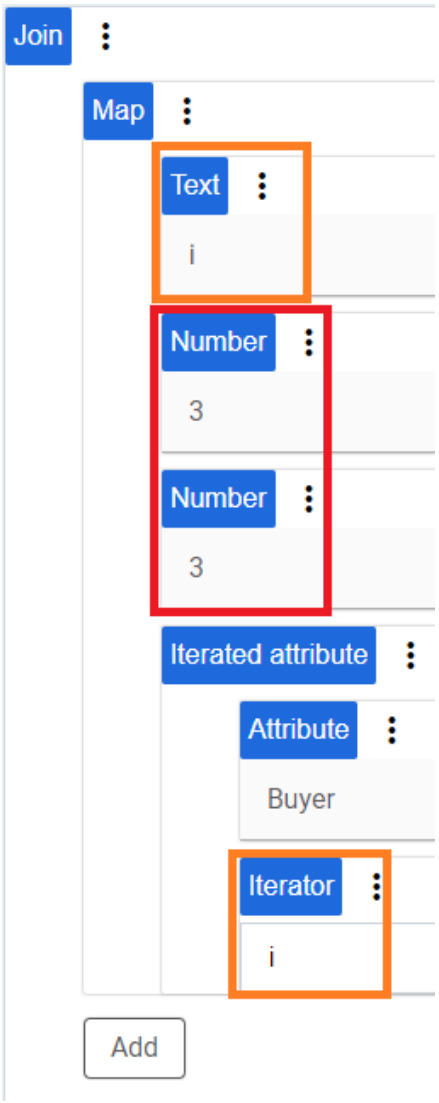
8.3.12. Thus, when the value drawn from the first argument (the total number of looped instances) is equal to the second argument (the Xth instance of the loop, where X is the value being examined), the block condition will be triggered.

8.4. Extracting one of the answers from a looped question

8.4.1. After you have set a loop such as the one below, you may wish to extract a particular value from the looped answers. Below, we have successfully extracted the third answer out of all the looped questions.



8.4.2. We will have to build the following Operation to extract the relevant looped input.



Join: Join will link up the outputs from the questionnaire

Map: Map is simply the function which is used to tie all of the nested components together

Text: The **text "i"** functions as a label to tie the looped attributes together

Note that the **text "i"** is reiterated in the "Iterator" section below

Number: These numbers signify the **range of values** to be extracted from the looped answers. The top number refers to the first number in the range while the bottom number refers to the last number in the range

In this case, since we are simply extracting the third output from a loop, we will set both numbers to "3"

Iterated Attribute: Attribute

- The looped Attribute

Iterator

- The Iterator inputs function as a label to tie the looped attributes together and this label is named earlier in the operation
- Since the label we used earlier in the operation was the **text "i"**, we will reiterate it here to tie the looped attributes together

8.4.3. You can then insert the operation directly into the template itself using the "Quick Operation" button in the Automation tab.

8.5. Display multiple placeholders within a loop in an inline list

8.5.1. When unmodified, loops display each iteration as a separate paragraph. However, you may want to have the looped values contained within a single line or even within a paragraph. If there are multiple placeholders within that loop, each set of values may need to be grouped together as well.

consolidated: chapter I, page 23; chapter II, page 57; chapter III, page 744

Chapter: I
Page: 23
Chapter: II
Page: 57
Chapter: III
Page: 744

8.5.2. For example, in this screenshot, a loop was set up over the “Chapter” and “Page” lines, hence the repeat. However, using an operation, we can consolidate the entries into a single line.

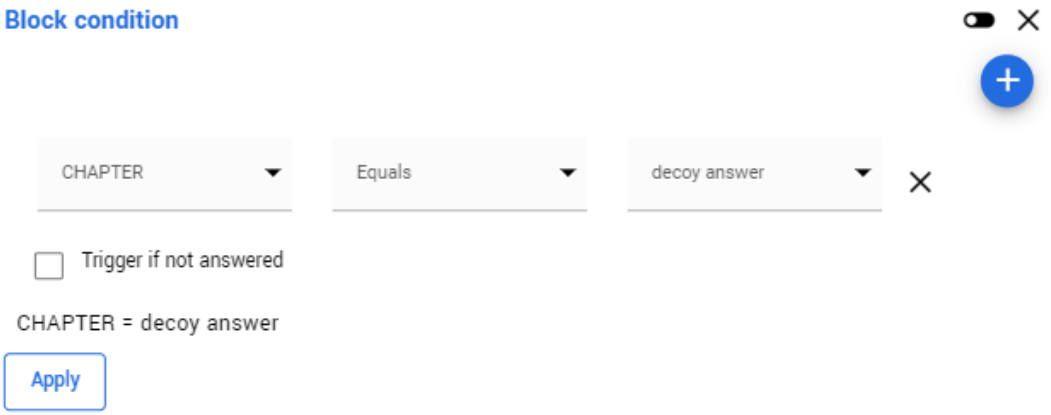
8.5.3. Click here for a video guide showing how to construct the operation. If you are interested in the explanation of how the operation works, please read on.

The set-up

8.5.4. First, insert the placeholders you want to loop into your editor. These can be placed anywhere in the document as they will be made “invisible” using an unfulfillable block condition. When the placeholders have been inserted, highlight them and click  under the Automation tab. A blue container will appear over the placeholders.

Chapter: CHAPTER
Page: PAGE

8.5.5. Next, highlight the same text and insert a block condition, making sure to set it up such that this block condition can never be fulfilled. For example, I have used CHAPTER = decoy answer.

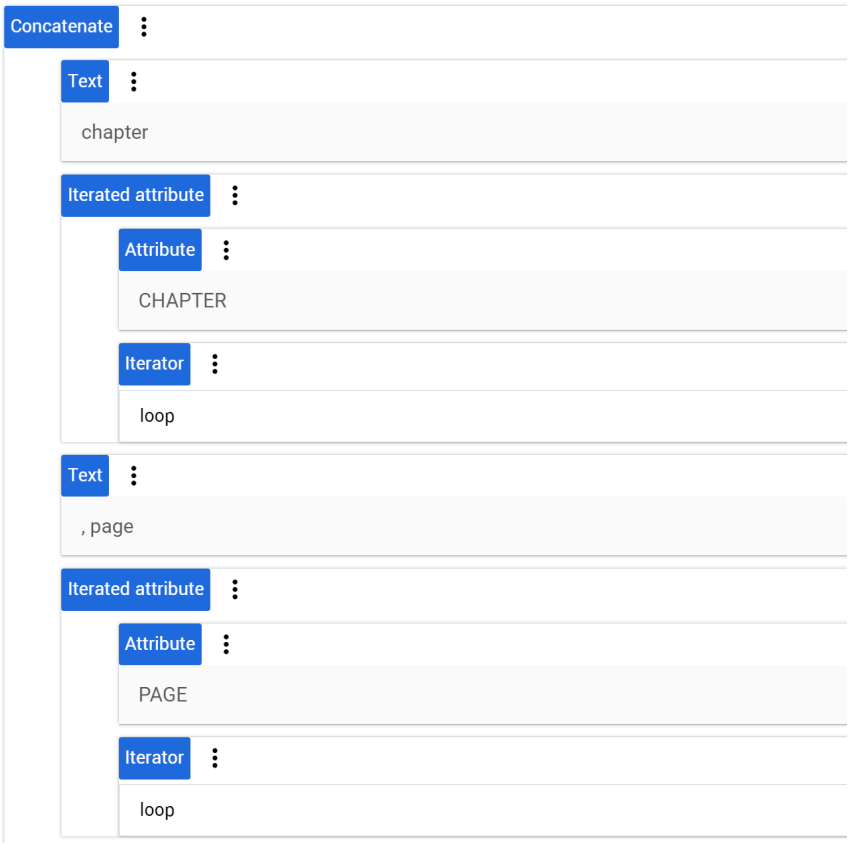


8.5.6. Apply the condition and you should now see the light green block condition container overlaying the loop blue container. This way, you can avoid having to show the repeated loops which will take up multiple paragraphs of space and instead have only the operation visible.

How to build the operation

8.5.7. To see how to build the operation, please refer to this video.

The "core" of the operation



8.5.8. *Concatenate* is used to join up the four arguments of Text, Iterated attribute, Text, and Iterated attribute respectively. We will look at each argument to build how this should look.

8.5.9. **First *Text* Argument:**

- This is a text input containing "chapter ". Note that a space (which is not visible here) is deliberately inserted after the word "chapter". This is to leave a space between "chapter" and the next argument, which would in this case return the Chapter number.
- At this point, if the operation was in effect, it will return "**chapter** ".

8.5.10. **First *Iterated attribute* argument**

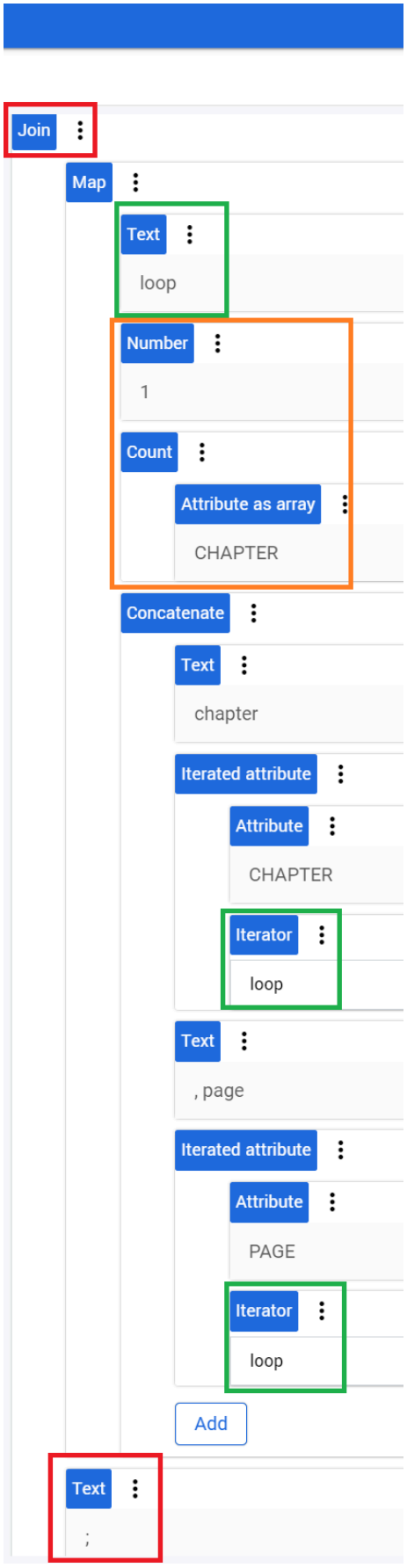
- *Iterated attribute* contains the attribute which you wish to loop. When you insert an *Iterated attribute* input, an empty *Attribute* input will automatically be nested within it. Select the attribute you wish to loop - in this case, it is CHAPTER.
- Under this *Attribute* input, insert an *Iterator* input. This input is the **label** of the loop which would be named in a *Text* input earlier in the operation (this will be shown below). The label can be anything but it is named a generic “loop” here. The naming of this *Iterator* input is crucial as it is used to link this *Iterated attribute* input with the next.
- At this point, if the operation was in effect, it will return “**chapter A**”.

8.5.11. Second *Text* argument

- This is a text input containing “, page ”. Again, there is a space after “page” to maintain a space between “page” and the page number drawn from the next input.
- At this point, if the operation was in effect, it will return “**chapter A, page** ”.

8.5.12. Second *Iterated attribute* argument

- Similar to above, the attribute you want to loop here is PAGE. Note that the *Iterator* input is named identically to that in the first *Iterated attribute* input.
- At this point, if the operation was in effect, it will return “**chapter A, page 1**”.



Map input

Map is simply the function which is used to tie all of the nested components together. It is the function used to kickstart this entire process of an inline loop operation. Ultimately, the *Map* input loops whatever is contained *below* the **orange** box.

Green boxes

As mentioned above, the *Iterator* inputs function as a label to tie the looped attributes together and this label is named earlier on in the operation.

This naming is done in the first instance of the green box using a *Text* input.

Orange box

The two arguments in the orange box serve as the range from which the Xth instance of answers to the looped questions in the questionnaire should be included. Here, *Number* is **1**, meaning that the loop will include all values **from** the first instance. Conversely, if *Number* was **2**, the answers to the first instance of the question will be ignored and only the subsequent questions will be considered. If there is a set upper limit, you can simply insert another *Number* input in the place of the *Count* operation.

Alternatively, if there is no upper limit, the *Count* operation can be used. The *Count* operation counts the number of times this attribute has been looped in the questionnaire and returns it as a number. For example, if I have looped the CHAPTER question four times, the *Count* operation will return **4.** This means the upper limit is 4. That said, since *Count* will always return the *total* number of times looped, it is able to function as an endless upper limit.

Red boxes

Finally, you use the *Join* operation to link up the multiple outputs you have. Here, *Join* wraps all other components such that they form the first argument and has a final 2nd argument of "**;** ". This serves as a separator for each return. For example:

chapter A, page 1; chapter B, page 3

8.6. Row: extract a specific index item in an array

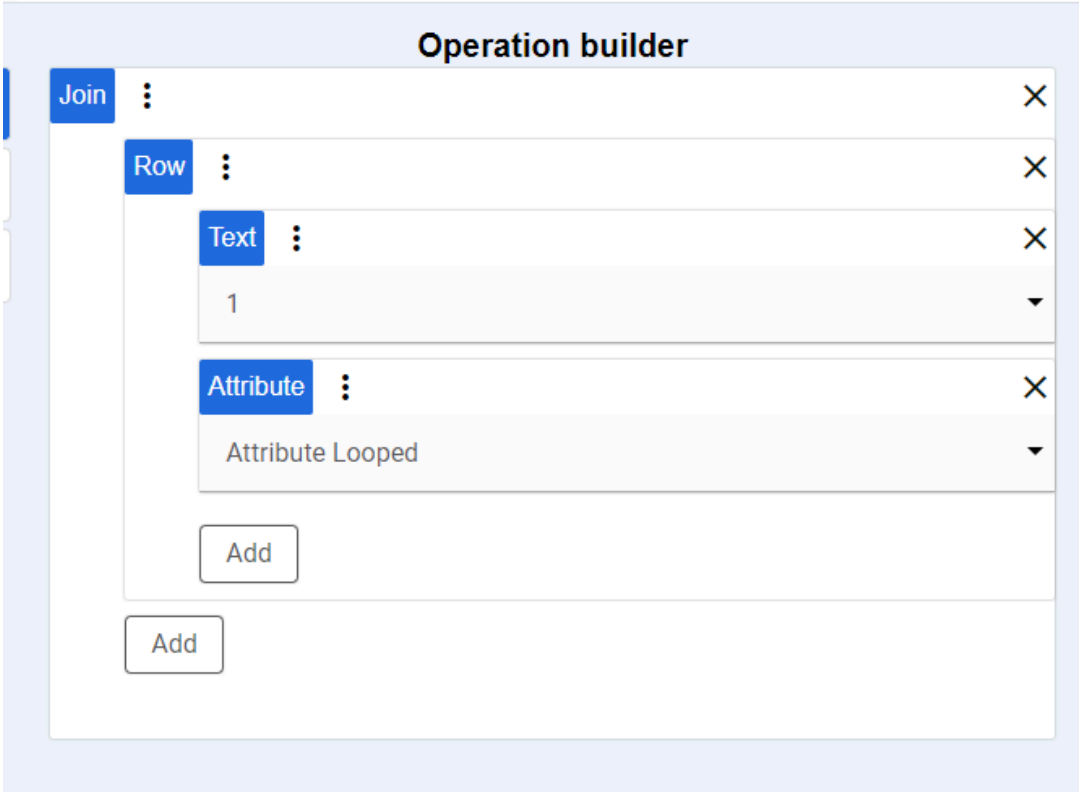
8.6.1. Row helps to extract a specific index item in an array. For instance, a looped attribute could be ["1", "2", "3", "4"]. So, the row operation in this case would then have 2 parameters:

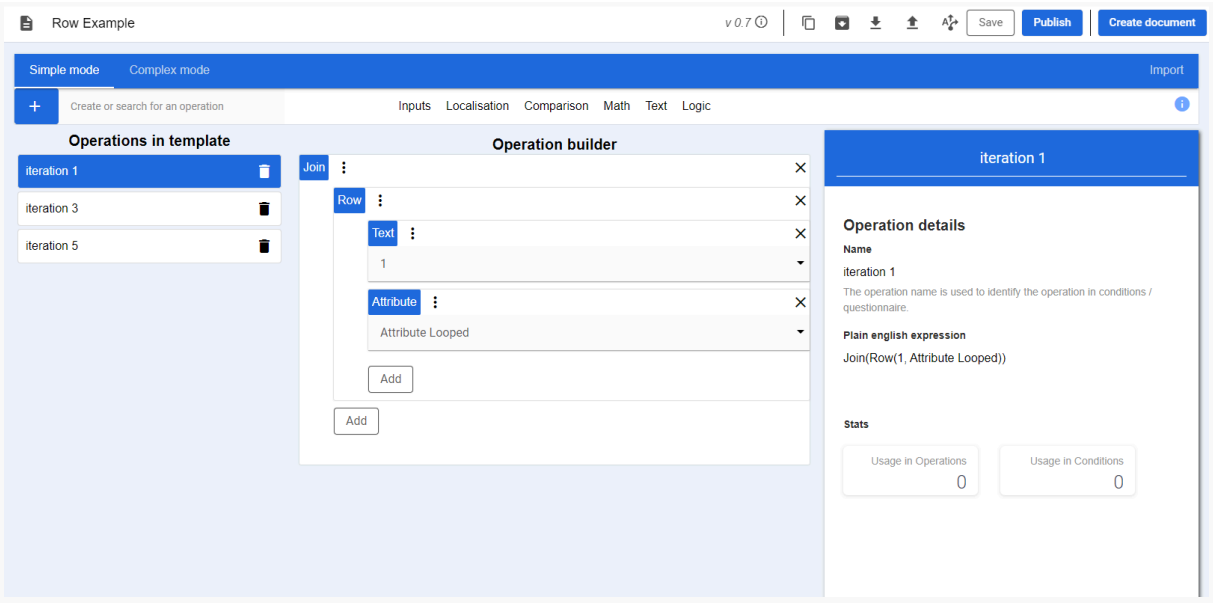
1. A Text parameter for the index that you want to extract (e.g. "2"); &
2. An Attribute parameter that points to the looped attribute that you're extracting from.

8.6.2. The row operation essentially allows you to choose which iteration you want to take from a loop. For instance, in a case where you want the third answer from the loop somewhere else, row would help you to find that one specifically.

8.6.3. A standard operation would constitute:

- **Row** containing two arguments, out of which the **text** argument refers to which number of the index you want to choose (i.e. 1st, 2nd, 3rd);
- The **attribute** is the looped attribute; and
- **Join** converts the resulting array into a string (removing brackets and quotation marks)





8.6.4. In the example below, the row operation has been used in the template to pick out answers 1,3 and 5 specifically. If 5 loops are created, the row operation would help pick up the three answers.

Attribute Looped

Specific iterations picked out:

iteration 1

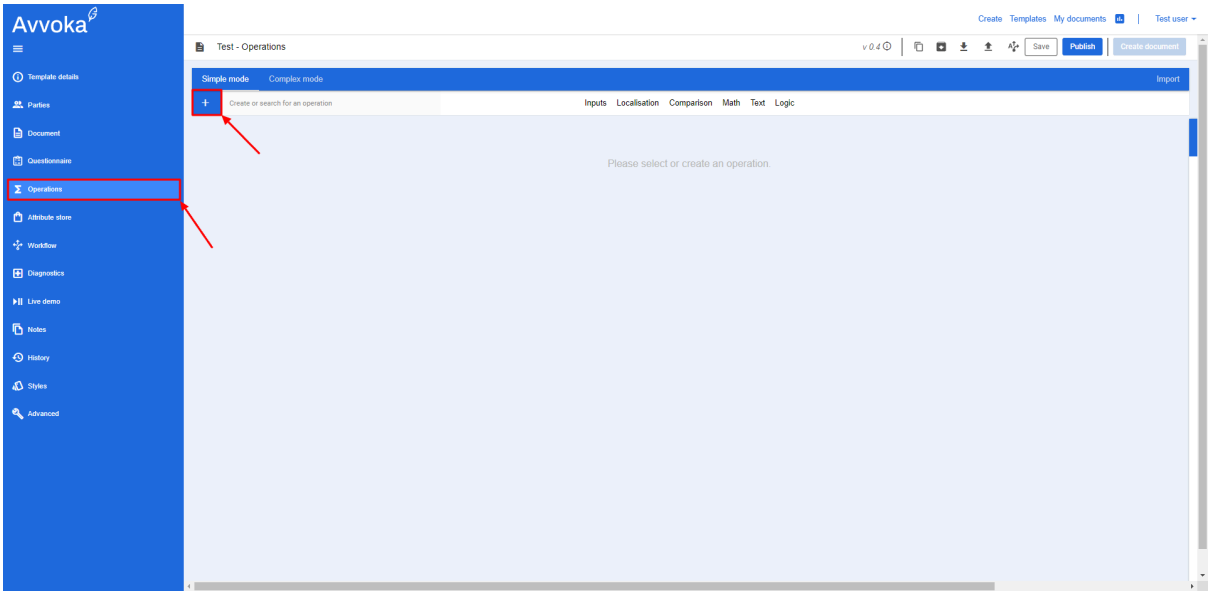
iteration 3

iteration 5

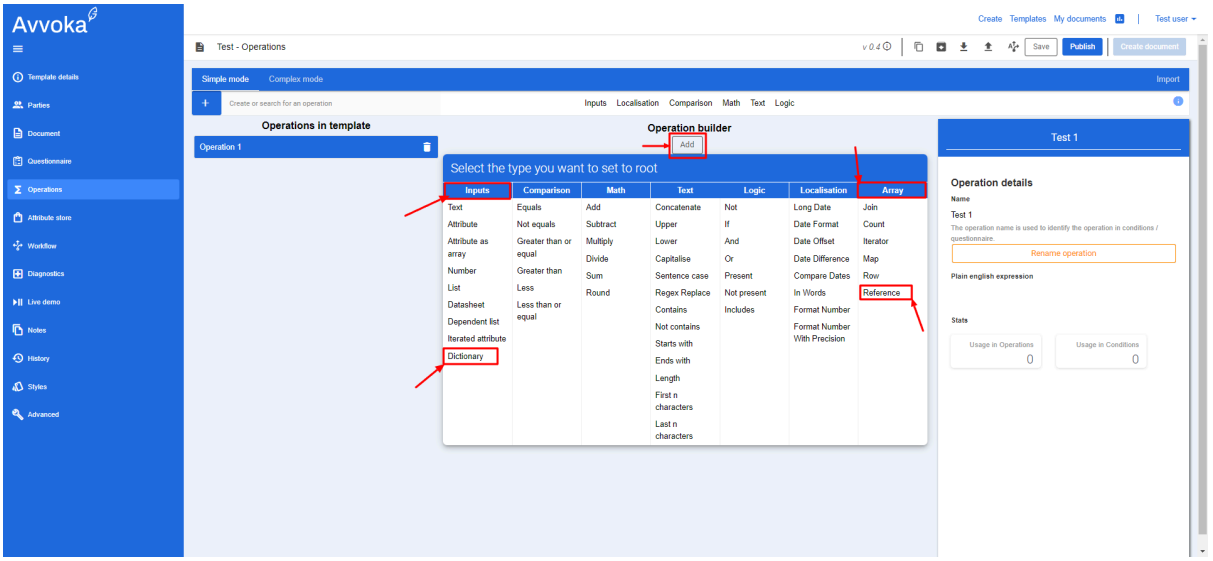
8.7. Worked Examples for Dictionary

8.7.1. **‘Dictionary’** and ‘**Reference’** are two operations amongst many present in Avvoka. For using the **‘Dictionary’** operation, a **‘Reference’** is used as well. While **‘Reference’** can be found under an **‘Array’**, **‘Dictionary’** is present under **‘Inputs’**. A user is able to define concepts using these operations. For instance, if you have an attribute named **‘seller’** then you are going to have a text: **seller has** but if the attribute is **‘sellers’** it will say: **sellers have**.

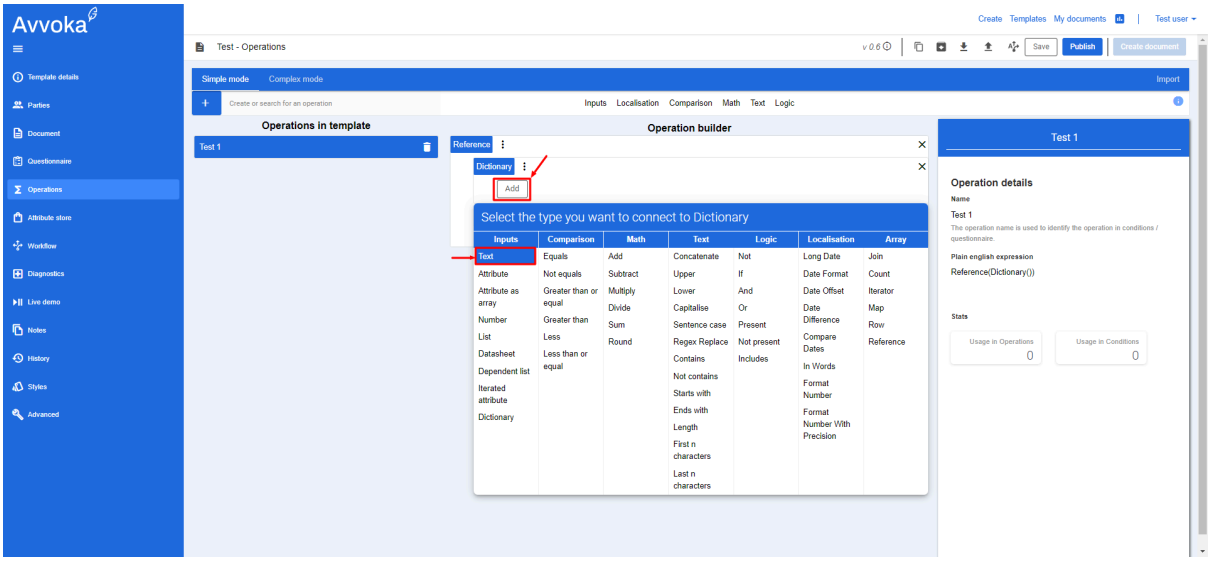
8.7.2. To add a reference and dictionary operation, access the **‘Operations’** tab in the left side pane and then click the blue + button as shown below.



8.7.3. Now, add an operation by clicking the relevant button and choose **‘Reference’**, then add **‘Dictionary’** under it.



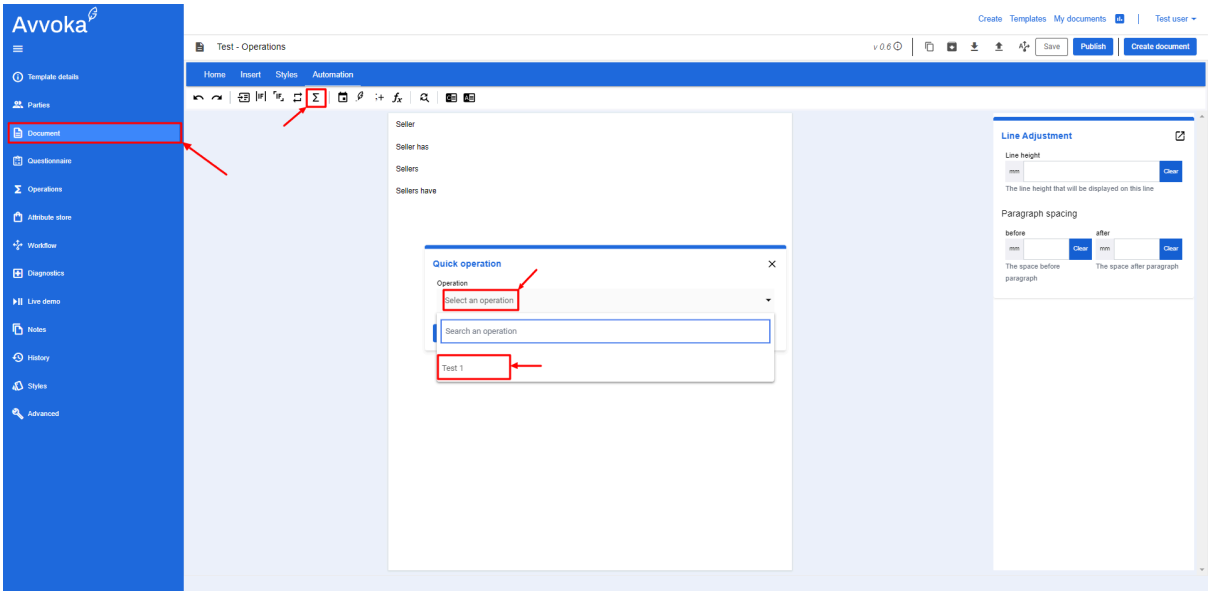
8.7.4. Next, choose **'Text'** which is the first option present under **'Inputs'**, and add a word.



8.7.5. For instance, in the example below, various texts have been added including: Seller, Seller has, Sellers, and Sellers have. Selecting **'Seller'** would populate **'Seller has'** in the document whereas choosing **'Sellers'** would populate **'Sellers have'**.

8.7.6. Then, navigate to the **'Document'** tab on the left side pane and under the **'Automation'** tab present in the blue bar, click the operations button Σ which when hovered over, says: **Insert quick operation**. This will prompt a window to open with

a drop-down list. Then, select the operation you have created to add into the document.



8.8. Worked Examples for At

8.8.1. This function allows you to extract a specific index item in an array similar to rows, except it is specific to non-looped arrays. For instance, the array could be ["A", "B", "C", "D]. Now, if you want to pull value 2 from the array, you will need to subtract 1 from this number as it takes the index of the array to start from 0 rather than 1.

8.8.2. There would be two parameters for this:

1.) The array parameter that you want to pull the value

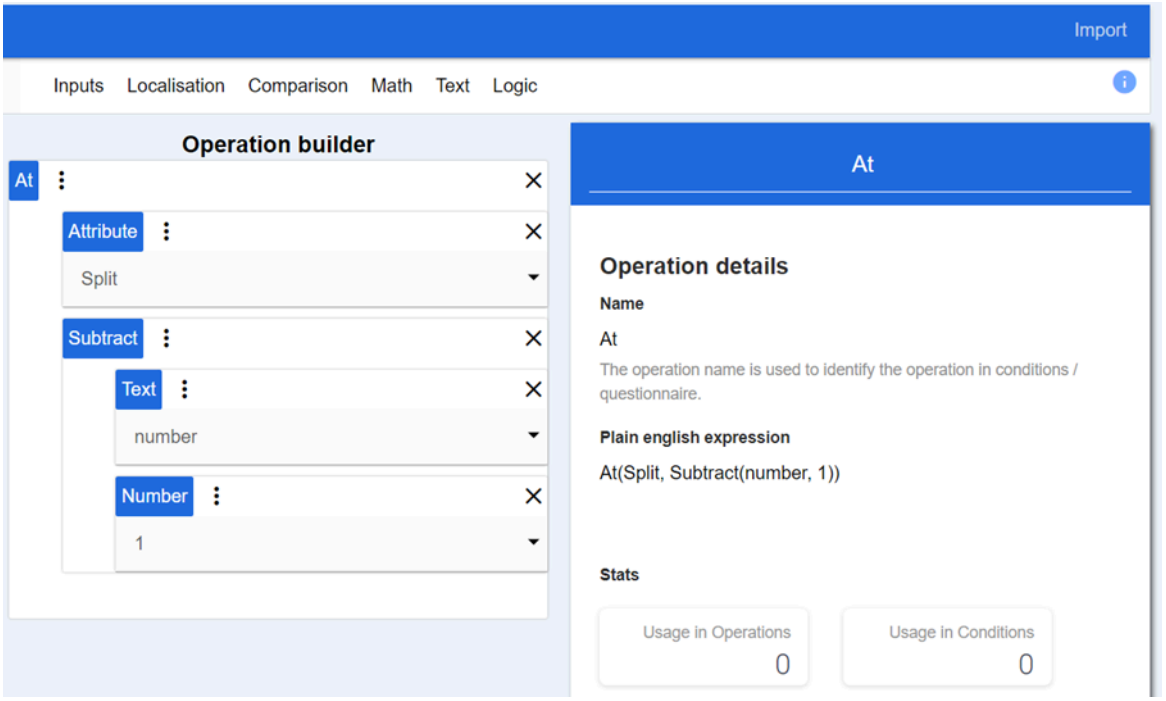
2.) The number parameter for the index of the array that you want to pull values from

Please note that you will need to subtract 1 from this number as it takes the index of the array to start from 0 rather than 1.

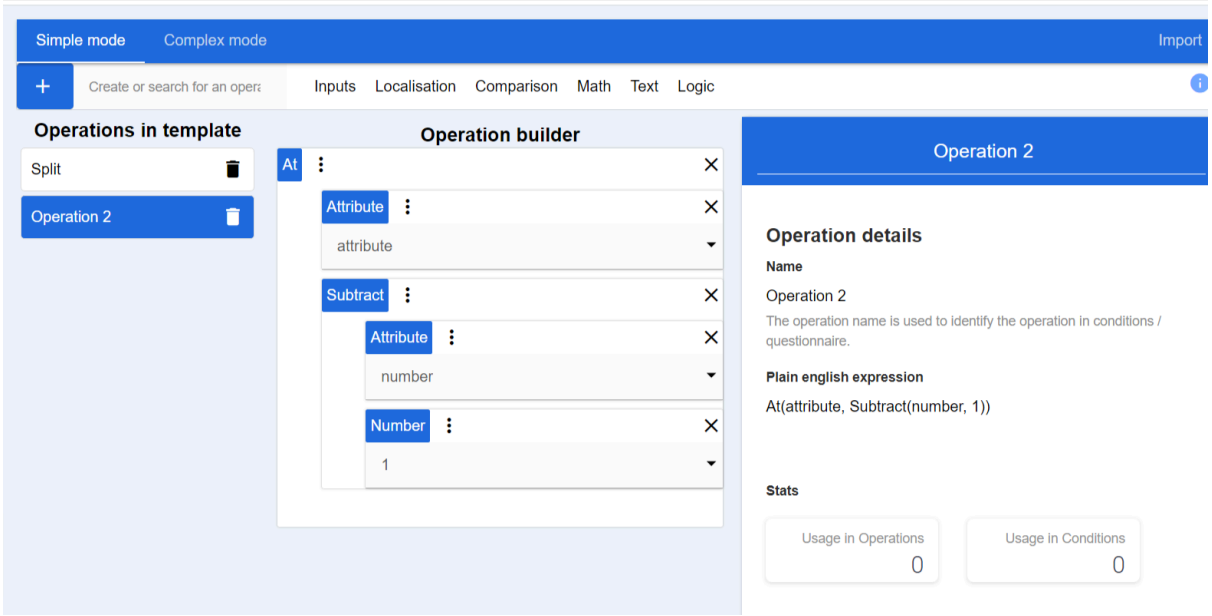
8.8.3. A standard **'At'** operation would constitute:

1.) The first argument would be the array through which you fetch the value from

2.) The second argument would be the number for the index of the array of the value you want



8.8.4. The following example utilises the **'At'** operation to extract a specific value from an array, particularly for arrays without loops. To create an **'At'** operation, select it from under the array column. Then, choose **'Attribute'** from the **Inputs** column and attribute an attribute like in the given example below.

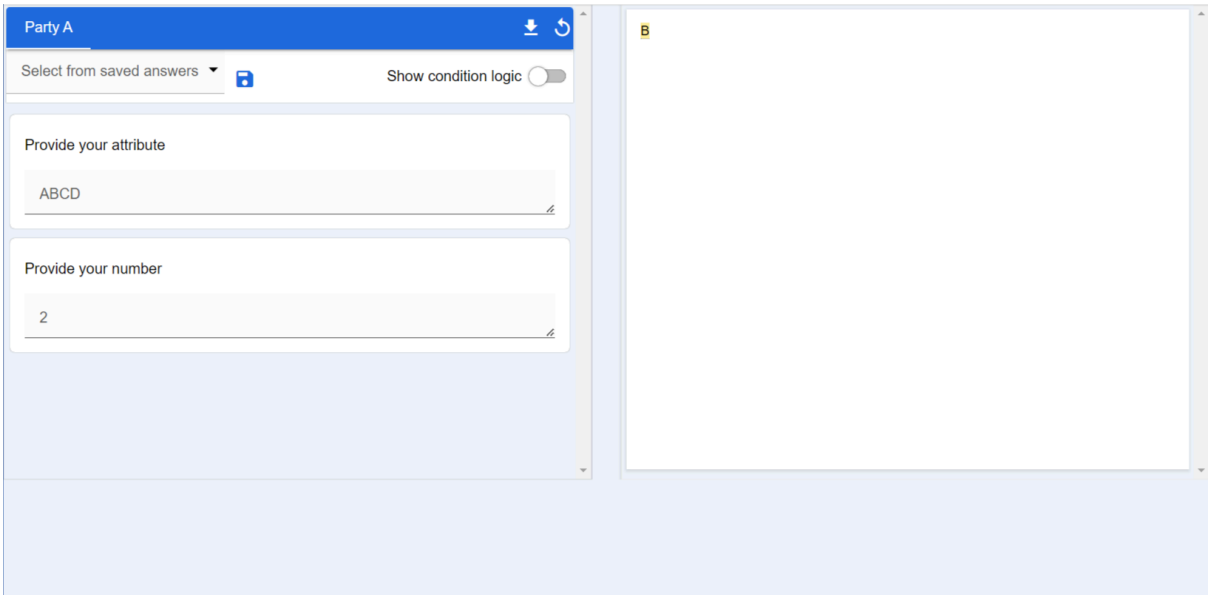


8.8.5. Then, choose **'Subtract'** under the **Math** column and add an attribute there. For instance, in the example below, the selected attribute is: **number**. Then, select

‘Number’ from the **Inputs** column and enter the value: 1. This way, the number the user enters will be subtracted by 1.

8.8.6. 

You can now see that the desired value B has been extracted in the example below using the **‘At’** operation when 2 was entered as input.

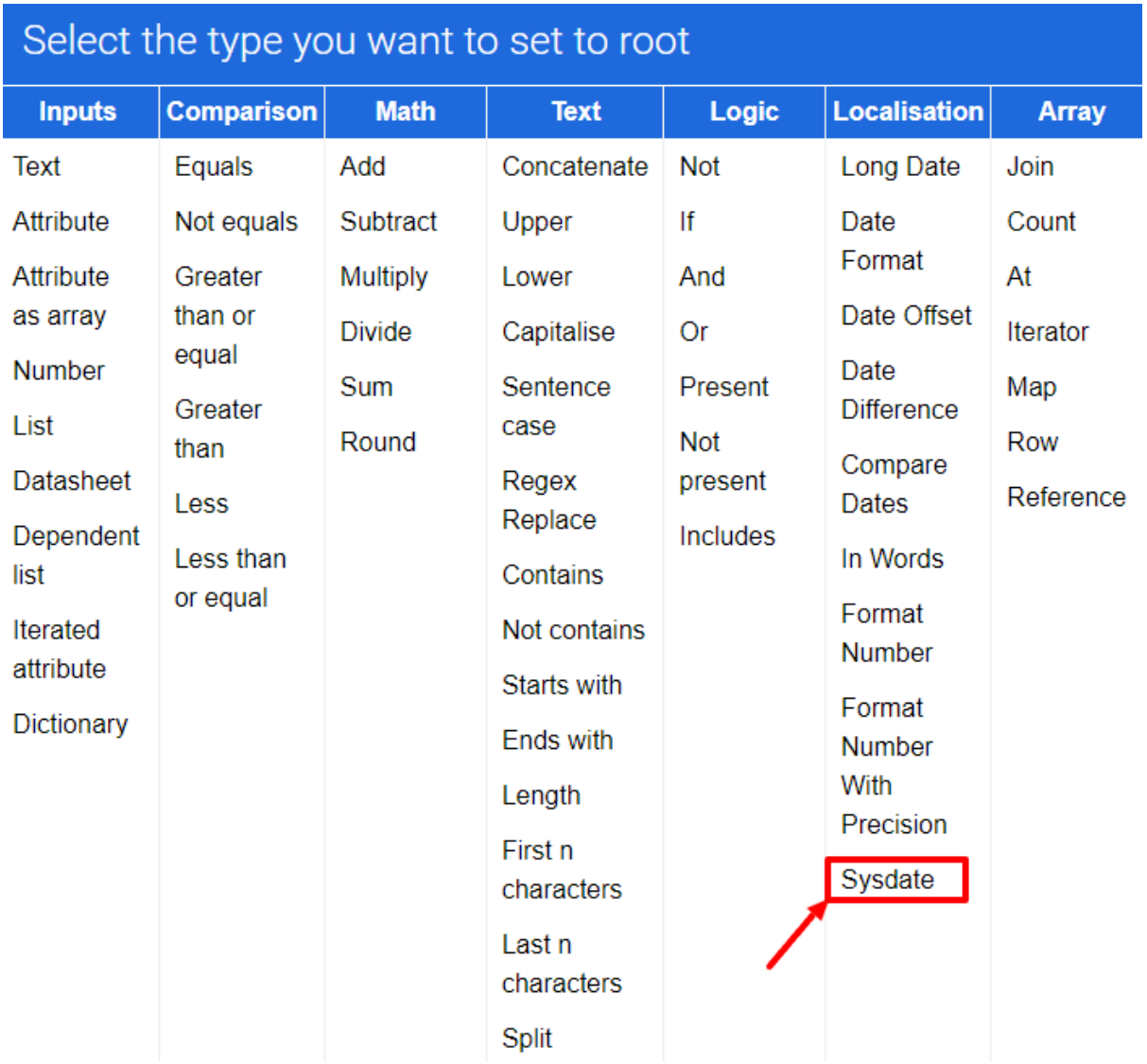


8.9. Worked Examples for Sysdate

8.9.1. This function enables users to retrieve the creation date of a document for display or use in other operations. **Note:** It is important to specify a Date Format to ensure that the retrieved date is formatted according to specific and desired requirements.

8.9.2. To use this operation, follow the steps below:

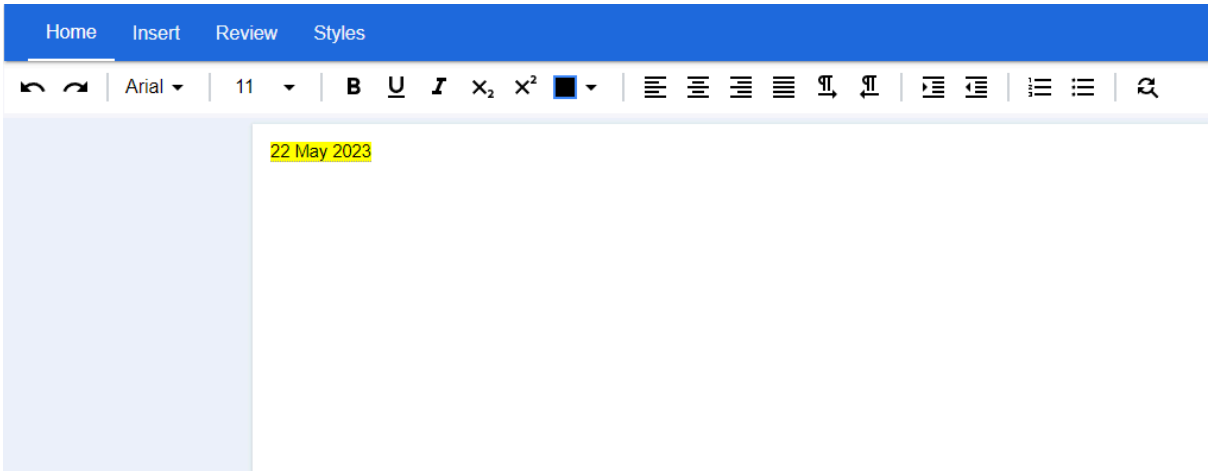
1. Select **'Sysdate'** from the **'Localisation'** column in Operations.



2. Then, select the 'Date document was created' option from the drop down menu to add in under Sysdate.

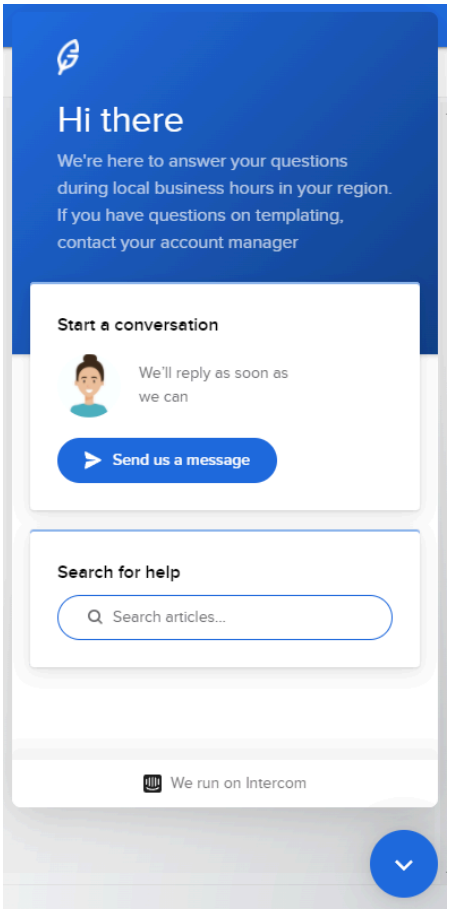
3. Now, insert the operation in the Document.

You can test it by creating the document where the date of the document creation (i.e.the current date) would automatically populate in the document.



9. Additional support

9.1.1. You can ask us a question by clicking on the support button ( ). A window will open and you can type your questions there. Our staff will be able to help you during business hours.



9.1.2. For additional support, contact help@avvoka.com or call +44(0)20 3519 2237.

10. Archived Worked Examples

Numbers in Ordinal Form (Regex)

10.1.1. Below is a short list of Cardinal Numbers vs Ordinal Numbers.

Cardinal Numbers	Ordinal Numbers
1	1st
2	2nd
3	3rd
4	4th
5	5th

10.1.2. If you would prefer to just create the operation, simply follow this video. However, if you would like to understand how the operation works, do read on.

The general logic of the operation set

10.1.3. The operation set that we are creating will ensure that all numbers inputted will be outputted as Ordinal Numbers. This would be simple if all the numbers ended in a "th". However, as can be seen above, "1st", "2nd" and "3rd" do not end with a "th".

10.1.4. A solution to this would be to set up an argument that detects when the number "1", "2" or "3" appears at the end of a string of numbers. Once those numbers are detected, the appropriate ordinals will appear at the end of the string. (**Operation Set Number 1**)

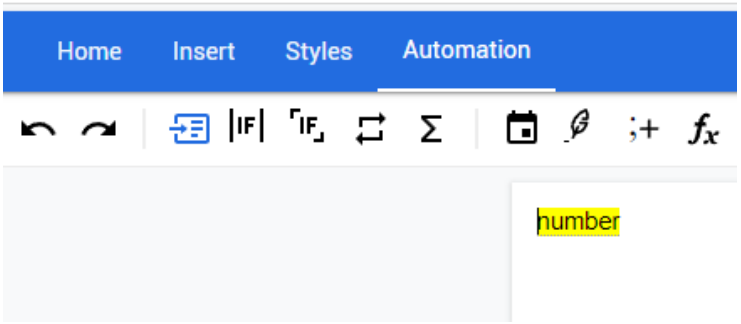
10.1.5. However, a problem arises when we realise that "11", "12" and "13" also end with "1", "2" and "3". But, their appropriate ordinals are "th". If we were to only have the above argument, they would appear as "11st", "12nd" and "13rd".

10.1.6. To solve this problem, we will have to add another argument that detects when the numbers "11", "12" or "13" appear at the end of a string of numbers. Once those

numbers are detected, the appropriate ordinal “th” will appear at the end of the string. (**Operation Set Number 2**).

Setting up the number attribute

10.1.7. First, we will have to set up an attribute that will allow us to input a number. In this case, a placeholder titled “number” has been created.

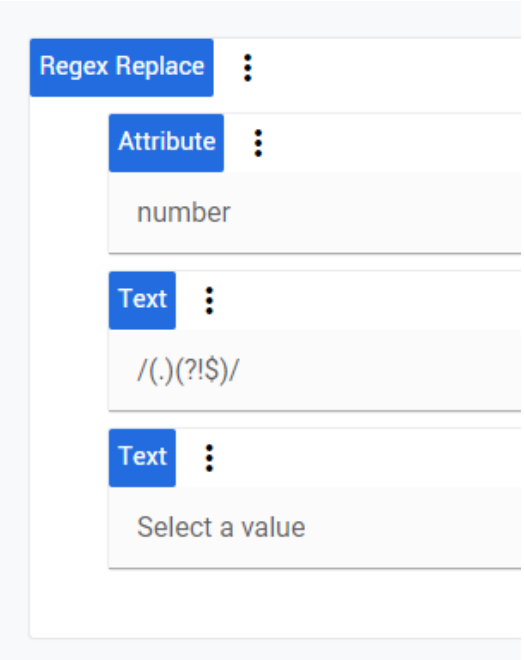


Operation Set Number 1

10.1.8. Operation Set Number 1 is made up of two operations:

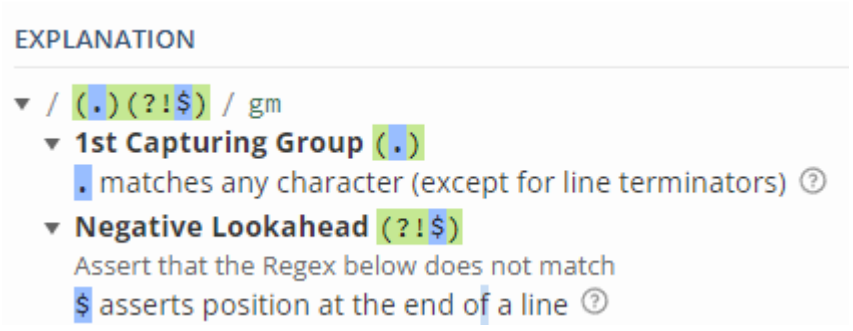
1. The first operation removes all digits from a number except for the last number in the string. For example, “987654321” will be outputted as “1”.

2. The second operation detects the output from the first operation and outputs the appropriate ordinal. For example, the operation will detect the output “1” from the previous operation and will output the ordinal “st”.



10.1.9. Above is the first operation. In this example, the operation has been given the name "regex". With reference to how a "Regex Replace" block works:

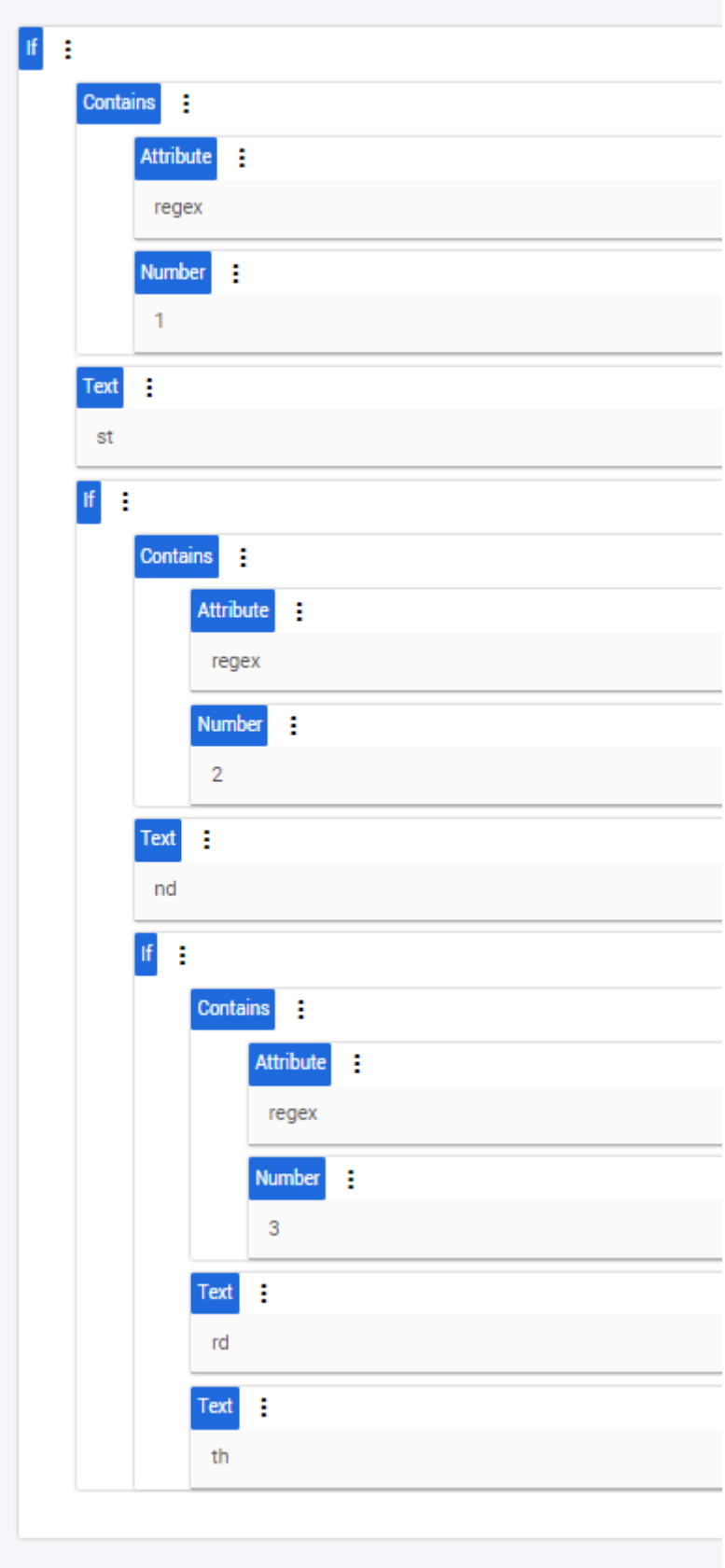
- **Attribute**: The string that will be modified. In this case, the attribute "number" has been inserted.
- **Text**: The expression that will be replaced by the Text block below it. In this case, the expression in the first Text box is "/(.)(?!$)/". This is a regular expression which selects every digit except for the last digit in a string of numbers.



- **Text**: This is what should replace each of the matches listed in the above Text box. In this case, this text box is blank.

10.1.10. This operation is replacing every single digit before the final digit with a blank. In the number “987654321”, the digits “98765432” will be replaced with a blank, leaving the number “1”.

10.1.11. As such, this operation titled “regex” will only output the final digit in a string of numbers.


If
Contains
Attribute
regex
Number
1
Text
st
If
Contains
Attribute
regex
Number
2
Text
nd
If
Contains
Attribute
regex
Number
3
Text
rd
Text
th

10.1.12. Now that we have our output from the operation titled “regex”, we will create a second operation. This operation will be titled “st,nd,rd,th”. This second operation “st,nd,rd,th” will detect the output from the first operation (the final digit of the number) and will output the appropriate ordinal.

10.1.13. Casting our mind back to how “if statements” work, the if function takes three arguments:

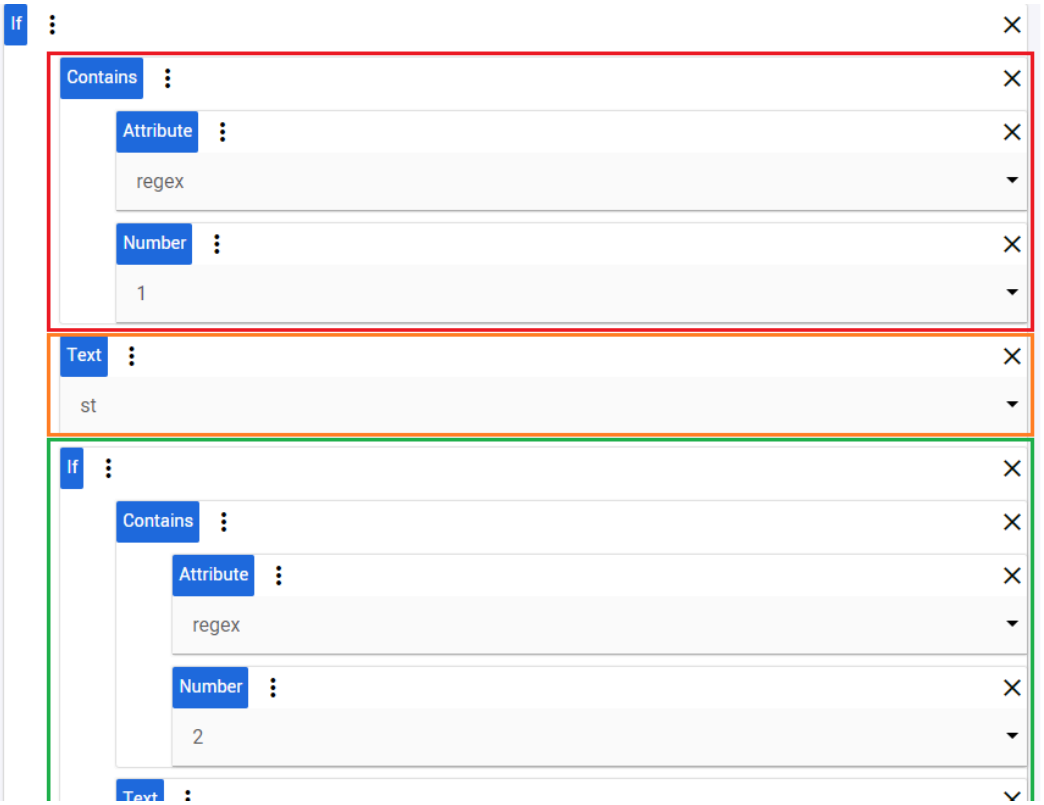
4. **The block in red**: a logical test. In this case, the test is “Does the output from regex, the final digit of the number, contain the number 1?”

5. **The block in orange**: the value to be returned if that test is true. In this case, if the output from regex contains the number 1, the test will be true and the value “st” will be returned.

6. **The block in green**: the value to be returned if the test is false. In this case, if the output from regex does not contain the number 1, the test will be false.

    a. If the test returns as false, the operation will move onto the next “if statement”, where the argument is “Does the output from regex, the final digit of the number, contain the number 2?” and vice versa.

    b. If the output from regex does not contain “1”, “2” or “3”, the test at the end will return as false and the operation will output the value “th”. For example, “4” will cause the test to return as false and the operation will output the value “th”.

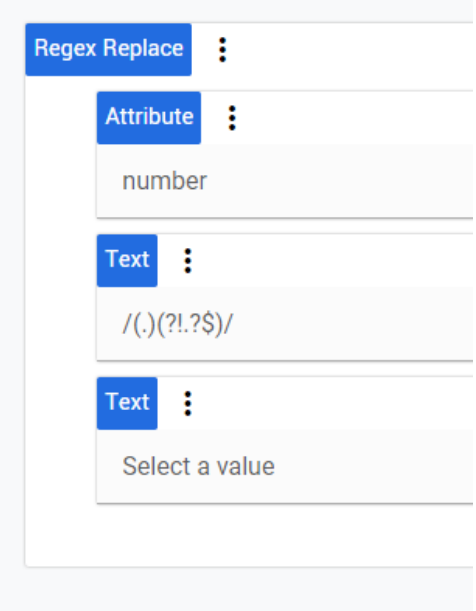

If
Contains
Attribute
regex
Number
1
Text
st
If
Contains
Attribute
regex
Number
2
Text

Operation Set Number 2

10.1.14. Operation Set Number 2 is made up of two operations:

1. The first operation removes all digits from a number except for the last two numbers in the string. For example, “9876543211” will be outputted as “11”.

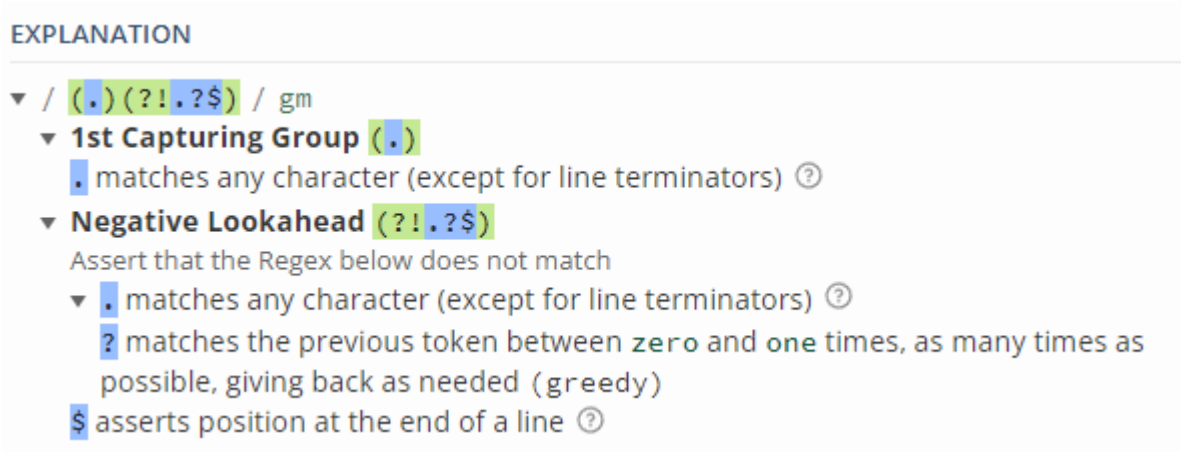
2. The second operation detects the output from the first operation and outputs the appropriate ordinal. For example, the operation will detect the output “11” from the previous operation and will output the ordinal “th”.



10.1.15. Above is the first operation. In this example, the operation has been given the name “regex last 2”. With reference to how a “Regex Replace” block works:

- **Attribute**: The string that will be modified. In this case, the attribute “number” has been inserted.

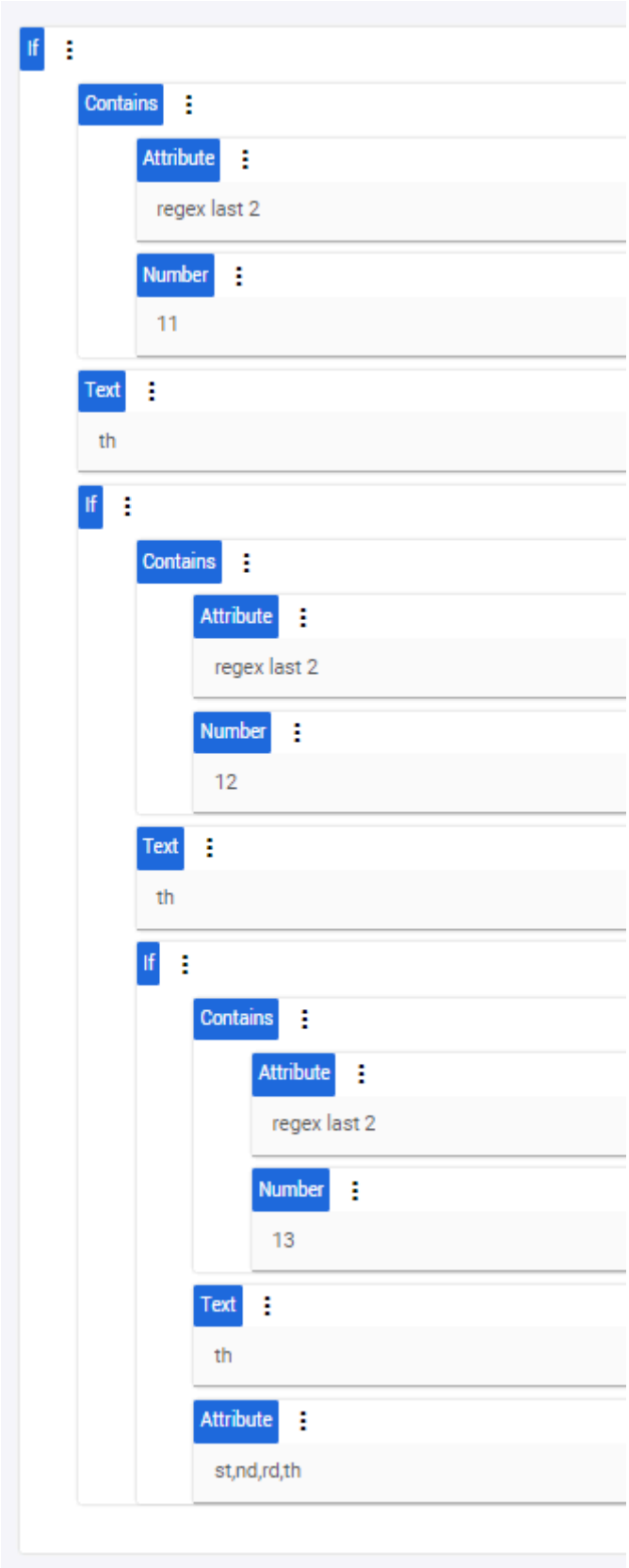
- **Text**: The expression that will be replaced by the Text block below it. In this case, the expression in the first Text box is “/(.)(?!.?$)/”. This is a regular expression which selects every digit except for the last two digits in a string of numbers.



- **Text**: This is what should replace each of the matches listed in the above Text box. In this case, this text box is blank.

10.1.16. This operation is replacing every single digit before the final digit with a blank. In the number "9876543211", the digits "98765432" will be replaced with a blank, leaving the number "11".

10.1.17. As such, this operation titled "regex last 2" will only output the final two digits in a string of numbers.


If
Contains
Attribute
regex last 2
Number
11
Text
th
If
Contains
Attribute
regex last 2
Number
12
Text
th
If
Contains
Attribute
regex last 2
Number
13
Text
th
Attribute
st,nd,rd,th

10.1.18. Now that we have our output from the operation titled "regex last 2", we will create a second operation. This operation will be titled "st,nd,rd,th last 2". This second operation "st,nd,rd,th last 2" will detect the output from the first operation (the final two digits of the number) and will output the appropriate ordinal.

10.1.19. Casting our mind back to how "if statements" work, the if function takes three arguments:

4. **The block in red**: a logical test. In this case, the test is "Does the output from regex last 2, the final two digits of the number, contain the number 11?"

5. **The block in orange**: the value to be returned if that test is true. In this case, if the output from regex last 2 contains the number 11, the test will be true and the value "th" will be returned.

6. **The block in green**: the value to be returned if the test is false. In this case, if the output from regex last 2 does not contain the number 11, the test will be false.

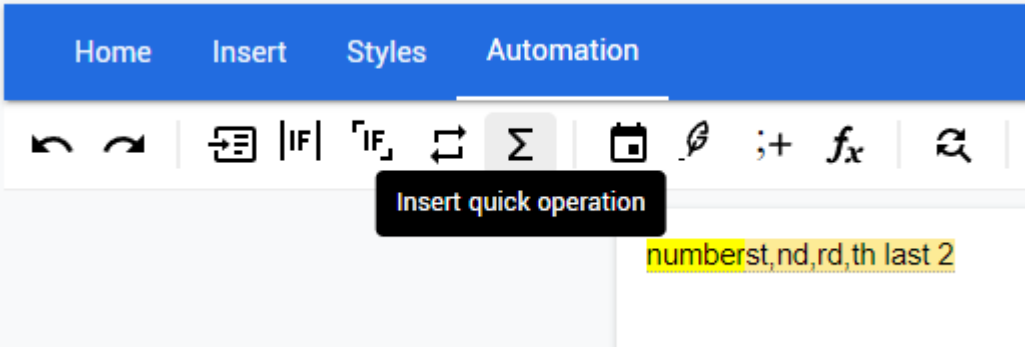
    c. If the test returns as false, the operation will move onto the next "if statement", where the argument is "Does the output from regex last 2, the final two digits of the number, contain the number 12?" and vice versa.

    d. If the output from regex last 2 does not contain "11", "12" or "13", the test at the end will return as false. The operation will then move onto the operation "st,nd,rd,th" that we discussed above.

        i. For example, "14" will cause the test to return as false and the operation will move on to the operation "st,nd,rd,th". After the number "14" passes through that argument, it will be outputted as "th".

Final steps

10.1.20. Now that we have our 4 operations, we will return to the “document area”. Insert the operation titled “st,nd,rd,th last 2” into the document behind the placeholder titled “number”. The operation will now work as intended!



11. Changelog

Set out below are the major changes to this operations guide:

Date	Description	Document Version
23/03/2021	New Operations Guide Format	1.0
1/06/2022	Added Date Operations	1.1
10/02/2023	Updated Ordinal Number Operation Added new text Operations	1.2