

**Askia Training**

Askiascript 2.0

Introductory training

Course 150



Participant's Coursebook

# Contents

<b>Introduction</b>	<b>5</b>
Format	5
Module topics	5
<b>Session 150-1 Askiascript 2.0 Basics: Objects and Operators</b>	<b>7</b>
Outline	7
Material covered	8
What is a script?	8
Scripts defined in the routing	8
Questions as Objects	10
Operators	12
Recap	15
Practical exercises	15
Exercise 1: Using script to set a variable	15
Exercise 2: Editing your script	15
<b>Session 150-2 Logical structures</b>	<b>17</b>
Outline	17
Material covered	18
<i>If ...Then ... Else</i> structures	18
<i>Return</i> statement	18
Worked examples of <i>If ... Then ... Else</i>	18
Comments	19
Using <i>Has</i> , <i>HasAll</i> and <i>HasNone</i> in <i>If</i> blocks	20
Expressions	20
Inline script	21
Recap	22
Practical exercise	22
Writing control structures	22
<b>Session 150-3 Variables and keywords</b>	<b>25</b>
Outline	25
Material covered	26
Virtual variables	26
Working with sets and arrays	26
Recap	29
Practical exercises	30
Exercise 1: Creating question loops	30
Exercise 2: Using <i>Dim</i>	31
<b>Session 150-4 Advanced Loops</b>	<b>33</b>
Outline	33

Material covered.....	34
Questions residing in standard loops .....	34
Loops within Loops .....	34
Object methods for loops .....	35
<i>AllValues</i> property.....	35
<i>For/Next</i> Loops .....	36
<i>Break</i> command.....	37
Recap .....	38
Practical exercise .....	38
Writing script loops.....	38
Example questionnaire .....	41



# Introduction

## Format

This course comprises five flexible modular sessions, which permit different learning pathways through the training course. The course has been designed for several alternative delivery methods, to offer maximum flexibility to learners:

- As a taught course lasting a little under one day
- For online delivery as a series of webinar tutorials
- For one-to-one delivery (e.g. peer-to-peer training)

In each case, the course is delivered as a series of separate modules or sessions. Each session is intended to last no more than an hour, though it may take longer, if additional time is required to complete the practical work.

Each session follows the same format:

1. Introduction (by tutor)            2-3 minutes
2. Tutorial and demonstration    15-20 minutes
3. Summary (by tutor)                2 minutes
4. Practical exercises                variable
5. Recap, feedback and questions

## Audience and course pre-requisites

This course is primarily intended for experienced Askia scriptwriters and programmers who should already be familiar with the Askia software and have a working knowledge of all the material covered in the Askia Design course (Course 100).

## Module topics

Session 150-1	Askiascript 2.0 Basics: Objects and Operators
Session 150-2	Logical structures
Session 150-3	Variables and keywords
Session 150-4	Advanced loops

## Related course

In addition, the single session short course Data Editing (Course 410), combines learning from both this course and also the Introduction to Askia Tools (Course 400). For audiences already familiar with the content of the Tools course, it may be useful to continue immediately on to Course 410 after completing session 150-4 in this course.

## Recommended learning pathways

All the modules of this course are considered to be 'core modules' and should be followed in sequence.

### *One day plan*

#### **Morning**

Module 1  
Module 2

#### **Afternoon**

Module 3  
Module 4

### *Two day plan*

#### **Day 1** (morning or afternoon)

Module 1  
Module 2

#### **Day 2** (morning or afternoon)

Module 3  
Module 4

Online delivery

## Session 150-1 **Askiascript 2.0 Basics: Objects and Operators**

### Outline

#### Topics presented

In this session, we will introduce you to:

- The concept of an askiascript
- Objects and Properties
- Useful operators

#### Learning outcomes

At the end of this session you will understand:

- How askiascript interacts with the different Askia modules
- How and where to add askiascript definitions
- How to use questions as objects in your scripts
- Which properties to use when referring to question objects
- How to define and use operators within scripts

## Material covered

### What is a script?

Askiascript is an extension to askiadesign that allows you to program Askia at the time it is executing a questionnaire, so that you can enhance how questionnaires are delivered or how they collect data, and you can use them to automate tasks which may otherwise require manual intervention to achieve.

Askiascript consists of an easy-to-learn set of syntax commands and keywords which interact with the normal variables or questions that you create in askiadesign. They allow you to perform tasks before or after questions are displayed, or at the beginning or the end of an interview. There are many different applications for scripts, and many different occasions when they can be useful.

You can define scripts in the askiadesign module, in the questionnaire routing. Your scripts can perform tasks or instructions during the interview, either before, during or after a question. You can also create scripts “inline” inside the text displayed in question or answer captions.

### Scripts defined in the routing

You can create a script at two places in a routing instruction:

- To make a routing position; these scripts must end up returning a *true* or a *false* result, to control whether the routing is followed or skipped.
- To create a value; this is often to set the contents of a variable. It must return the right kind of value (e.g. numeric or string) to match the variable whose contents it is setting.

**Note:** The above script types are executed during the interview. A special **edit** version of routing scripts (covered in 150-5) lets you edit data after the interview is complete, to correct the data.

To create a script in the routing:

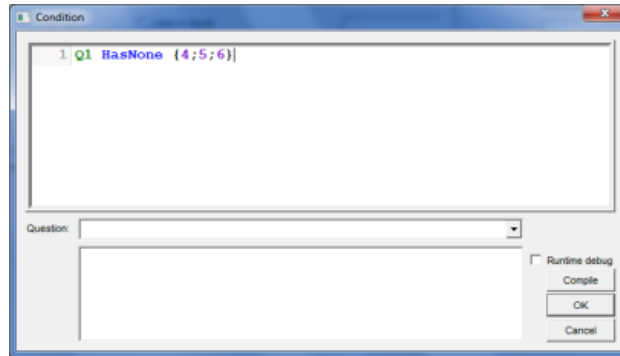


1. In the toolbar, click **routing mode**, if you are not already in this mode.
2. Select **condition**:

3. Click the **condition...** to open the script editor.



4. In the script editor, enter *your script*.



5. Click **OK**.

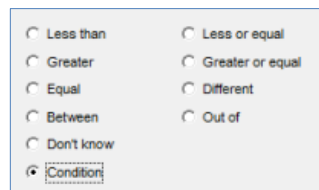
## Unconditional scripts

Routing scripts that you want to execute on every occasion should be given the condition *true*, so that no logical test is applied.

To create an unconditional script:

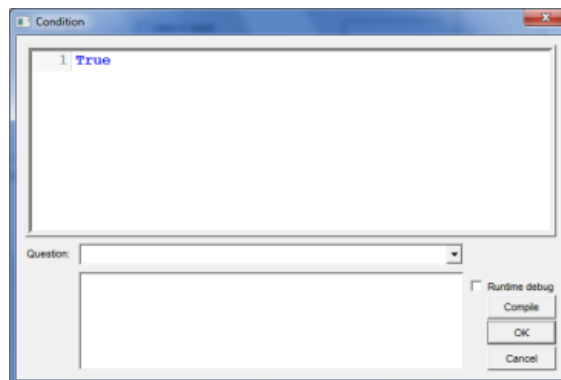


1. In the toolbar, click **routing mode**, if you are not already in this mode.
2. Select **condition**:



Condition...

3. Click the **condition...** to open the script editor.
4. In the script editor, enter *True*:



5. Click **OK**.
6. In **action**, select **set value**.
7. In **target question**, select the question whose value you want to set in your script.
8. Next to **value if true**, click ... to open the script editor.
9. Enter your script.
10. Click **OK**.

## Questions as Objects

In askiascript, any question can be handled as an object. You can always reference a question with its shortcut, for example:

Q1

Shortcuts containing spaces should be surrounded by carets (^). For example:

^Q2. Age^.

Objects have properties and methods:

- Object **properties** return information about different aspects of an object (e.g. the question type);
- Object **methods** allow you to modify the result when interrogating the property of an object.

Properties and methods are referenced by using the question name, then a dot and the property or method name. For example:

Q1.Value

## Working with Values

**Value** is the property you will use most often when writing scripts. In some cases, value is implicit. At other times, you will need to write .value (e.g. when adding one or more methods to the value property).

For example:

Gender.Value

will return 1 if the question contains the value 1 in the current interview.

- **LongCaption** and **ShortCaption** are properties that return the question's long caption and short caption respectively. They return information from the question definition, as set up in askiadesign, not the respondent's answer. For example:
  - `Q3.LongCaption()`
  - `Q3.Shortcaption()`
- `LastName.Value.ToUpperCase()` – here you must specify Value, as it cannot be implicit if you add a method. The method always comes after the property. For example:
  - `Q3.Value.ToLowerCase() = "bosch"`
  - `Q3.Value.ToUpperCase() = "BOSCH"`

There are several different kinds response values in Askia:

- **Numeric:** *numbers*
- **String:** *open text – letters, numbers, punctuation, spaces*
- **Sets:** *coded answers to single and multiple questions*
- **Date, Time, and Date and Time**
- **Boolean:** logical values *true* or *false*

You must always match the right type of value with the operation you are performing, the variable you are referring to, or to which you are assigning a value. If you do not, the script will not execute, and a warning message will appear.

## Sets and arrays

The responses to single and multiple questions are different from numeric or text questions in that they consist of a series of answer possibilities. In the case of multiple questions, they can also contain more than one answer after the question has been asked.

Response data for multiple questions can be referred to in scripts as **sets**. This consists of the response numbers, separated by semi-colons, within curly brackets. For example, if a respondent selected responses 1, 3 and 5, this would be referred to as:

```
{1; 3; 5}
```

Askiascript provides functions for comparing two or more sets, and for manipulating the values in a set. For details, please see 3. Comparing sets of values on page 14 and Working with sets and arrays on page 26.

## Response properties

Responses, and their methods and properties.

Like questions, responses are objects, and have various properties and methods.

The following special properties allow you to access much more information about single and multiple questions:

- **Responses** (returns the entire list of responses for a closed question);
- **AvailableResponses** (returns the list of available responses in the order of display);
- **Answers** (returns the list of responses selected by the respondent, in the order selected).

For example, if the responses 6 and 4 (“Lawn edger” and “Chain saw”) are selected at Q1, these script snippets will give the following results:

Script	Result	Notes
<code>Q1.Responses.Index</code>	1;2;3;4;5;6;7;8;9	This is all of the responses defined in the question, no matter what the respondent selected.
<code>Q1.AvailableResponses.Index</code>	3;1;4;2;8;6;9;7;5	Assuming the response order is randomised, and this was the order they were presented to the current respondent.
<code>Q1.Answers.Value</code>	6;4	The responses selected by the respondent.
<code>Q1.Answers.EntryCode</code>	21; 4	If the entry code for response 6 is 21.
<code>Q1.Answers.Caption</code>	Lawn edger; Chain saw	
<code>Q1.Answers[2].Index</code>	4	The second answer selected by the respondent.

Script	Result	Notes
		By using square brackets, you can refer to a specific item in an array. This can be used for Responses, AvailableResponses, Answers, etc.
Q1.Value	6;4	
Q1	6;4	This is equivalent to Q1.Value. The Value property is implicit.

## Operators

### 1. Comparing numeric values

The following operators allow you to compare numeric values:

**Equal to (=)**

True if the first value is equal to the second value.  
For example:

Age = 24

is true if the value in Age is 24.

**Not equal to (<>)**

True if the first value is not equal to the second value.

Age <> 24

is true if the value in Age is *not* 24.

**Greater than (>)**

True if the first value is greater than the second value.

Age > 24

returns is true if the value in Age is greater than 24.

**Greater than or equal to (>=)**

True if the first value is greater than or equal to the second value.

Age >= 24

is true if the value in Age is greater than or equal to 24.

**Less than (<)**

True if the first value is less than the second value.

Age < 24

returns true if the value in Age is less than 24.

**Less than or equal to (<=)**

True if the first value is less than or equal to the second value.

Age <= 24

is true if the value in Age is less than or equal to 24.

## 2. Comparing string values

```
If (Q1.Value.Trim().ToLowerCase() = "string trimmer")
Then
    Return True
Else
    Return False
EndIf
```

## Useful object methods for strings

An **Object Method** provides something that you can perform on an object, such as to provide a value or perform a test. Object methods are always specific to the type of object being referenced. Here we describe several methods that are useful when working with string questions.

The following methods are useful when working with string questions:

<b>Equal to (=)</b>	<p>True if the first string is equal to the second string. For example:</p> <pre>Q1.Value.Trim().ToLowerCase() = "string trimmer"</pre> <p>is true if the string in Q6 is "bosch" and false if the string in Q6 is "black &amp; decker".</p>
<b>Not equal to</b>	<p>&lt;&gt; is true if the first string is not equal to the second string.</p> <pre>Q1.Value.Trim().ToLowerCase() &lt;&gt; "string trimmer"</pre> <p>is true if the string in Q6 is not "bosch".</p>
<b>Left()</b>	<p>Returns the specified number of characters from the left of the string. For example:</p> <pre>Q6.Value.Left(3)</pre> <p>Returns "bos" if the string in Q6 is "bosch"</p>
<b>Right()</b>	<p>Returns the specified number of characters from the left of the string. For example:</p> <pre>Q6.Value.Right(2)</pre> <p>Returns "ch" if the string in Q6 is "bosch"</p>
<b>IsNumber()</b>	<p>True if the string contains a pure number; false if it does not.</p> <pre>Q6.Value.IsNumber()</pre> <p>Is true if the string in Q6 is "5".</p>

## Other useful object methods for strings

If you are familiar with regular expressions, you can use them in your scripts to perform complex comparisons. *Note that regular expressions are not part of the scope of this course.*

### Two examples

1. To check the format of an email address:

```
Dim email = "askia@askia.com"
email.IsMatch("\w+@\w+\.[a-z]+")
```

**Note:** there is built-in method for this so `email.IsEmail()` will produce the same result

2. This script will check if a string is a proper UK number:

```
Dim rgUkPhone = "(\+|00)\s*\d{2}(\s?\d{3}){2}\s?\d{4}"
("0044 207 689 5492").IsMatch(rgUkPhone) ' => true
```

### 3. Comparing sets of values

Some operators allow you to compare specific groups of items in multi-coded questions. A simple equals comparison (e.g. `Q1 = 1`) will not work with a multi, because more than one response might be selected.

To refer to a set of values we use curly brackets `{}` and separate the values inside with semi colons. For example:

```
{1;2;5;8}
```

We can use the “To” keyword to specify ranges of numbers:

```
{1 to 5;7;9 to 12;15}
```

is equivalent to

```
{1;2;3;4;5;7;9;10;11;12;15}
```

The operators are as follows:

<b>Has</b>	Checks that <b>at least one</b> of the referenced response items was selected. If this is the case, the function returns 1; otherwise, it returns 0.  <code>Q1 Has {4;5;6}</code> will return 1 if either the fourth, fifth or sixth were selected.
<b>HasNone</b>	The opposite of “has”. It returns 1 if <b>none</b> of the specified response items were selected by the respondent.  <code>Q1 HasNone {4;5;6}</code> will return 1 if none of the fourth, fifth and sixth responses were selected.
<b>HasAll</b>	Checks that <b>all</b> of the referenced response items were selected. Note that other responses may be selected too, and the function will still return 1. For example:  <code>Q1 HasAll {4;5;6}</code> will return 1 if the respondent answered 4, 5 and 6.
<b>HasAndNoOther</b>	Checks that <b>at least one</b> of the specified responses was selected by the respondent. However, if any responses not in the list were selected, then the function returns 0. For example:  <code>Q1 HasAndNoOther {4;5;6}</code> returns 1 if the respondent selected response 4. If she selected responses 1 and 5, the function will return 0 (because 1 is not in the list).
<b>HasAllAndNoOther or =</b>	Similar to “HasAll”, but it returns 0 if the respondent selected any responses other than those in the list. In other words, the selected response must match the list exactly. So:  <code>Q1 HasAllAndNoOther {4;5;6}</code> will return 1 if the respondent answered 4, 5 and 6, but 0 if she answered 1,4,5 and 6 (because 1 is not in the list of qualified values).

Note that the above example is equivalent to `Q1 = {4;5;6}`.

## Recap

In this session, we have:

- Looked at the question and response objects.
- Seen some of the operator comparisons that can be made in your scripts.
- Looked at some of the useful functions for examining string variables
- Seen how regular expressions can be combined with askiascript expressions.

## Practical exercises

### Exercise 1: Using script to set a variable

*Follow these steps:*

1. Open askiadesign. A new, blank QEX will be created.
2. Create a **Gender** question (closed) with the responses Male and Female.
3. Create an **Age** question (numeric).
4. Create an **AgeGroup** question (closed) with the responses *Younger than 25*, *25 to 34* and *35 or older* as the answers.
5. Now define routing instructions to set the appropriate category in the **Age Group** question, according to the value in **Age**. Do this by creating three separate routing instructions, each one checking for a specific age range in **Age**, and setting the value in **AgeGroup** accordingly. So, your first routing will set **AgeGroup** to 1, where appropriate, the second will set the value to 2, and so on.

**Note:** Later in the course, you will learn how to do this with only one routing condition, but for now create three separate routing instructions.

### Exercise 2: Editing your script

*Follow these steps:*

1. Add a **RespName** question (type: open), and position it at the start of your QEX.

*Note that we should not call a question "Name", as this is a reserved word in Askia, so we should use an alternative (in this case, "RespName").*

2. Add a second question **RespNameTrimmed** (type: open) and place it after *RespName*. Ensure that the question's **visible during data entry** property is *not* selected, so that it is hidden during interviewing.
3. Add a routing as follows, to remove any spaces from the start and end of the name given at *RespName*, and store that result in *RespNameTrimmed*.

When defining your routing, click **condition....** Then, simply enter **true**. This ensures that the condition is always executed. Click **OK**.

Set up the other settings of the condition as follows:

**Start question:** *RespName*

**Action:** set value

**Target question:** *RespNameTrimmed*

**Value if true:** *write an appropriate expression here*

4. Later in the QEX, add a routing to display the contents of *RespNameTrimmed*:

**Condition:** true (ensures the routing is always run)

**Action:** Show message

**Message text:** *Name is ??RespNameTrimmed??.*

5. Test your questionnaire. You should see the contents of the *RespNameTrimmed* variable displayed by your second routing.
6. Now go back into your first routing, and change it to convert the answer to capital letters.
7. Test the questionnaire, to see the results of your change.
8. If time permits, try to repeat steps 6 and 7 two more times, using different string handling operations each time, as suggested by the routing editor.



## Session 150-2 **Logical structures**

### Outline

#### Topics presented

In this session, we will introduce you to:

- **If/Then/Else** structures
- The **Return** statement
- Comments in script

#### Learning outcomes

At the end of this session you will be able to:

- Create complex logical structures using If/Then/Else
- Write more script that is more efficient to write and easier to understand

## Material covered

### *If ... Then ... Else structures*

If/Then/Else structures allow you to execute blocks (sections) of script according to specific conditions. For example, you might want to execute a block of code only if a specific combination of answers has been given by the respondent, and otherwise execute another set of code.

By using If/Then/Else structures, you can write more complex logical statements, and your logical statements will be easier to understand when you look at the script.

#### Syntax

The syntax is as follows:

```
If logical_expression Then
    'block of script to execute if logical_expression
    is True
EndIf

Or
If logical_expression Then
    'block of script to execute if logical_expression
    is True
Else
    'block of script to execute if logical_expression
    is False
EndIf
```

**Note:** *Else* is always optional; you do not have to include it in an If structure.

### *Return statement*

When a Return expression is reached, it immediately breaks the script flow (all following lines are ignored), and returns the expression associated. For examples:

```
Return
Return True
Return 1
```

## Worked examples of *If ... Then ... Else*

### *If ... Endif*

### If with else

```

If Q1 Has {4;5;6} Then
    '4, 5 or 6 was selected at Q1
    Return True
Else
    'none of these responses were selected at Q1
    Return False
EndIf

```

### If with elseif

ElseIf can be used in an If/Then/Else block in order to create logical expressions with multiple outcomes. For example:

```

If (Age >= 18 And Age <= 25) Then
    Return 1
ElseIf (Age >= 26 And Age <= 35) Then
    Return 2
ElseIf (Age >= 36 And Age <= 45) Then
    Return 3
ElseIf (Age >= 46 And Age <= 55) Then
    Return 4
ElseIf (Age >= 56) Then
    Return 5
EndIf

```

### Nesting If within If

If structures can be nested inside each other to create complex logical expressions. For example:

```

If (Q1 Has {1}) Then
    '1 is selected at Q1

    If (Age >=56) Then
        '1 is selected at Q1 and Age>=56
        Return True
    EndIf

ElseIf (Q1 Has {2}) Then
    '2 (but not 1) is selected at Q1

    If (Age >=56) Then
        '2 (but not 1) is selected at Q1 and Age>=56
        Return True
    EndIf

EndIf

```

## Comments

```

If (Q1 Has {1}) Then
    'Push Lawn mower owners

    If (Age >=56) Then
        'Push Lawn mower owners aged 56 and over
        Return True
    EndIf

ElseIf (Q1 Has {2}) Then

```

```

    'Riding Lawn mower owners

    If (Age >=56) Then
        'Riding Lawn mower owners aged 56 and over
        Return True
    EndIf

EndIf

```

You can include comments in your scripts to make them easier for you or others to understand. A single quotation mark or apostrophe (') introduces a comment. The rest of the line is not executed. You should use comments widely in order to make the meaning of the script to anyone who has to review or maintain the script in future. For example:

```
'this script returns True if 4, 5 or 6 is selected at Q1
```

Throughout the syntax explanations and code examples in this booklet, you will see comments that help explain the code being described.

## Using *Has*, *HasAll* and *HasNone* in If blocks

Conditions that refer to responses to a single or multiple questions, are best used with these operators because they all deal with sets of answers.

Examples:

```

If (Q1 Has {1 to 5}) Then
    'User selected at least one of the first 5
    responses
    Return 1
EndIf

If (Q1 Has {1 to 95}) Then
    'User selected at least one of the first 95
    responses
    Return 2
EndIf

If (Q1 HasAll {3;4;5;6}) Then
    'User selected responses 3, 4, 5 and 6
    Return True
EndIf

If (Q1 HasNone {8;9}) Then
    'User didn't select responses 8 or 9
    Return 1
EndIf

```

## Expressions

### Brackets in expressions

Curly brackets { } are used to enclose response values, for example ranges.

Examples:

```
{3;4;5;6}
```

```
{1 to 5}
{1 to 6;8;9}
```

Note that normal brackets are also used as part of Askia Script syntax, for example `Q1.Value.Trim()`.

Normal brackets can also be used to improve the clarity of logical expressions (for example to surround IF conditions), but this is not mandatory. For example:

```
((Gender Has {1}) And (Q1 Has {9})) Or ((Gender Has {2})
And (Q1 Has {9}))
```

## Logical expressions: using AND, OR and NOT

AND, OR and NOT determine how multiple expressions are combined.

### AND

The condition is *true* if **both** expressions are true. For example:

```
If (Age >= 18 And Age <= 25) Then
```

This is true if the value in Age is equal to or greater than 18 **AND** it is less than or equal to 25.

### OR

The condition is *true* if **either** expression is true. For example:

```
Q1 has {1} OR Q1 has {2}
```

This is true if the respondent selected either response 1 or response 2 at Q1.

### NOT

Allows you to reverse an expression, so the condition will evaluate true if the expression is false, and vice versa. For example:

```
NOT (Q1 has {1} OR Q1 has {2})
```

The expression is reversed so that it is true if neither response 1 nor 2 was selected at Q1.

**Note:** more advanced expressions are covered in the subsequent sessions of this course.

## Inline script

Inline script allows you to include askiascript code snippets in other contexts, such as question captions, response captions or online interview screen definitions.

You will already know this syntax for text substitution in a question or answer caption:

```
??Q1??
```

This will display the answers given to Q1

You can also write script to create the value to display directly within a question or answer caption:

```
{% If condition Then %}
  'Text alternative 1 to display
{% Else %}
  'Text alternative 2 to display
{% EndIf %}
```

You can also use the askiascript engine using `{% askiascript %}` to place askiascript code into a question or answer caption.

You can use `{% askiascript %}` to execute a script without output and `{%= askiascript %}` to execute a script and write the result.

For example:

```
{% If (Q1 HasAllAndNoOther {1;2}) Then %}
    Has both types of mower, but no other tools
{% ElseIf (Q1 HasAndNoOther {1;2}) Then %}
    Has only a mower (or either type, or both)
{% ElseIf (Q1 HasAll {1;2}) Then %}
    Has both types of mower and may have other tools
{% ElseIf (Q1 Has {1;2}) Then %}
    'Has a mower (or either type, or both) and may have other
tools
{%ElseIf (Q1 HasNone {1;2}) Then %}
    Does not own either type of mower
{% EndIf %}
```

The contents of the caption will change according to the response given to Q1.

## Recap

In this session, we have:

- Created control structures using **If/Then/Else**
- Looked at the syntax required to define conditions
- Introduced the **Return** statement
- Added comments to script
- Inserted askiascript snippets into text as “inline script”

## Practical exercise

### Writing control structures

*Follow these steps:*

*Follow these steps:*

1. Delete the routings you created in the first exercise.
2. Create just one routing to set the value of the **Age Group** variable, according to the value in **Age**. You will need to use an **If/Then/Else** structure.
3. Now put another control structure around the **If/Then/Else**, to test that **Age** contains a valid response between 16 and 99, and only recode the variable when it is.

4. Create a **Return** which will be True if **Age** was coded into the new variable (i.e. it was in the range 16..99) and False if it was not. Think about the best way to create that control structure.
5. Add the following question to your survey:

**Shortcut:** Q1

**Long caption:** Which of the following financial products do you have?

**Type:** closed multi

**Responses:** Investment plan, Life insurance, Mortgage, Pension plan

6. Add the following question:

**Shortcut:** Q2

**Long caption:** Which of the following financial products are you considering buying in the future?

**Type:** closed multi

**Responses:** Life insurance, Pension plan, None of these

7. Add a new routing containing a script that means that Q2 is asked only if either *life insurance* or *pension plan* were selected at Q1, but not both. It should also be asked if neither response was selected at Q1.
8. It is especially important with scripting to carry out thorough testing, to ensure that your scripts have the effect you want. Test your script to ensure that is working, and amend it if necessary.
9. Add an inline script into Q2's long caption, so that the caption varies according to the number of responses selected at Q1 from *Life insurance* and *Pension plan*:
  - **If one response is being presented:** Are you considering buying the following financial product in the future?
  - **If both responses are being presented:** Are you considering buying either of the following financial products in the future?
10. Now go through the script you have created and add comments, to make it more understandable to anyone else who has to read it.
11. *Optional:* If time permits, try to experiment by creating another version of this routing using different logical options, e.g. reverse the logic using *Not*, or using *And* and *Or*. See if you can improve on your first version.





## Session 150-3 **Variables and keywords**

### Outline

#### Topics presented

In this session, we will introduce you to:

- Virtual variables
- **Intersection** and **Union** set functions
- **Count** property
- The **SelectRandom** function

#### Learning outcomes

At the end of this session you will know:

- How to create new array variables in your script
- How to perform logical tests and combinations on arrays
- How to detect the number of elements in an array
- How to use scripts to make random selections from questions

# Material covered

## Virtual variables

Virtual variables (also known as “declarative” variables in some programming languages) allow you to create additional variables for use inside the script which only exist within the script:

### Creating a virtual variable

### Assigning values and performing comparisons

### Defining the virtual variable type

- The type of a virtual variable is determined when you first assign a value to it.
- Subsequently, you can only assign values of the same type to it; if you try to assign a value of a different type, you will get an error message.

Type	Example
Numeric	<code>Dim my_var = 0</code>
String	<code>Dim my_var = ""</code>
Array	<code>Dim my_var = {}</code>
Date	<code>Dim my_var = #25/03/2011#</code>
Time	<code>Dim my_var = #16:32:00</code>
Date and time	<code>Dim my_var = #25/03/2011 16:32:00#</code>
Boolean (logical)	<code>Dim my_var = true</code>

Note that Boolean variables accept one of only two values: true or false. True is given the internal value 1 and false is given the internal value 0. This means it is possible to treat a Boolean virtual variable like a numeric variable – but it will only ever have the value 1 or 0.

## Working with sets and arrays

### Union or ‘+’

`Union` merges the values contained in two different sets, such as the responses to two questions. You can use **union** or **+** interchangeably.

For the value to be present in the union of two sets, it must be present in either of the sets being combined, which includes any values which are present in both of the sets.

Assuming Q1 contains 6 and 4, and Q1\_S contains 6, 1 and 8:

```
Q1 Union Q1_S
```

The result will be {6; 4; 1; 8}

## Intersection

`Intersection` retrieves the common values of two sets. It compares the selected responses given to two questions, and discovers which responses were selected in both.

For the value to be present in the intersection of two sets, it must be present in both of the sets being combined.

Assuming Q1 contains 6 and 4, and Q1\_S contains 6, 1 and 8:

```
Q1 Intersection Q1_S
```

The result will be {6}

Note that the order of the returned items is determined by the leftmost set (the one before the intersection keyword).

## Operations with sets

You can intersect a variable with a constant set. Examples:

### Example

```
{1 to 6} intersection Q1
```

### Result

If Q1 contains the set {6; 3} then the result will be {3; 6} because the order is determined by the set on the left

```
Q1 intersection {1 to 6}
```

If Q1 contains the set {6; 3} then the result will be {6; 3} because the order is determined by the set on the left

You can start any expression with the empty set { } to ensure that the expression is evaluated as a set and returns a set, e.g.

```
{ } + Q3 + Q4 HasAll {5; 6; 7}
```

Returns true if all of the responses 5, 6 and 7 were selected at Q3 or Q4 (e.g. it would be true if 5 and 7 were selected at Q3 and 6 and 7 were selected at Q4).

## Using subtraction with sets

You can use minus (“-”) to remove items. It means *intersection with not these items*.

## Size() function

The `size` function is particularly useful when working with sets and combinations of sets as it tells you how many values the set contains – i.e. the number of answers given.

For example, the following script returns the number of responses in Q1 and Q1\_S combined (as a union of the two):

```
Size(Q1.Value + Q1_S.Value)
```

Note that when we use the `Size` keyword, we should also use `Value`, to ensure we retrieve the correct value.

The following script returns the number of responses selected by the respondent from the first five codes to Q1:

```
Size(Q1.Value Intersection {1 to 5})
```

The following script returns the number of responses selected by the respondent from codes 1, 3 and 4:

```
Size(Q1.Value Intersection {1;3;4})
```

**Note:** If you combine two questions in this way with a union, then duplicate responses will only be counted once.

## **.SelectRandom(n) method**

`SelectRandom` allows you to select a specific number of values at random from a set. For example, this could be the responses given to a multi-coded question. The syntax is as follows:

```
.SelectRandom(n)
```

### **Example**

```
{1 to 10}.SelectRandom()
```

```
Dim i = {1 to 10}  
i.SelectRandom(2)
```

```
{5 to 10}.SelectRandom(3)
```

```
Q4.Value.SelectRandom(2)
```

```
(Q1 + Q4).SelectRandom()
```

### **Result**

Returns one value randomly from 1 to 10 (such as 3).

Returns two values randomly from 1 to 10 (such as 3 and 7).

Returns three values randomly from 5 to 10 (such as 5, 6 and 9).

Returns two values randomly from the ones selected in Q17 (such as 2 and 4).

Returns one value randomly from the ones selected in Q1 and Q4 (such as 3).

The sequence of returned numbers reflects the order of the values in the array. So `{1 to 10}.SelectRandom(3)` will return 3 values but the lower will always be the first one, the next highest will be always the second value, and so on, because the array is ordered from 1 to 10. If you want to shuffle these selected responses then you can use the `Shuffle()` method.

## **.Shuffle() method**

Use `.Shuffle` on an array to return the array in a randomised order

```
Q1.Value.Shuffle()
```

This returns all the values in a random order

`.Shuffle` can be used with `.SelectRandom` for a random selection in a random order

```
Q1.Value.SelectRandom(3).Shuffle()
```

This returns three randomly selected values in a random order.

So, if Q1\_S contains {1; 2; 5; 7}, the `.SelectRandom(3)` method will select three items, such as {1; 5; 7} and shuffle will place these in a random order, e.g. {5; 7; 1}

```
{1 to 10}.SelectRandom(3).Shuffle()
```

It will therefore return three values randomly and shuffled (such as 7; 2; 5).

## Count property

Use *Count* to return the number of items in an array. It can be used as an alternative to the *Size* function to obtain the number of answers to a question.

For example, if Q1 has four responses, then

```
Q1.Value.Count
```

returns 4.

There is an alternative way to achieve the same effect. The following script returns *true* if the q1 has at least one response selected between 1 and 5:

```
Dim my_array = Q1 Intersection {1 to 5}

If my_array.Count > 0 Then
    Return true
Else
    Return False
EndIf
```

## Recap

In this session, we have explored:

- Using `Dim` (dimension) to declare new array variables
- Combining and performing logic on sets or array variables using `Union` and `Intersection`
- Subtracting items from a set with “-”
- Using the `Count` property to detect the number of items in an array
- Using `SelectRandom` to randomly pick items from an array
- Using `Shuffle` to randomise the order of a set

## Practical exercises

### Exercise 1: Creating question loops

*Follow these steps:*

1. Create a Brand Spontaneous awareness question with 10 brands and None.
  - Call it “*TopOfMind*”.
  - For the list of brands, come up with own list. You might use Coca Cola, Dr Pepper, Evian, Minute Maid, Orangina, Perrier, Pepsi, Schweppes, Sprite, and/or one or more brands from your country.
  - Ensure you include a *None* category.
2. Create a Brand Spontaneous awareness variable as follows:
  - Call it “*Others*”.
  - Give it the same responses as *Top of Mind*, including the *None* category.
  - If *None* was selected in the *Top of Mind*, then do not show ‘*Others*’ to the respondent, and set its value to *None*.
  - If one brand was selected in *Top of Mind*, then show the list of brands without the one selected in *Top of Mind*.
3. Create an “*Assisted*” Brand awareness variable with the same list of brands.
  - Call this question “*Assisted*”.
  - Do not show the brands selected in *Top of Mind* and *Others*.
  - If the respondent selected every brand in the combination of *Top of Mind* and *Others*, then do not ask *Assisted*.
4. Create a “*Favourite*” Brand variable with the same list of brands.
  - Call this question “*Favourite*”.
  - Only show the brands selected in *Top of Mind* + *Others* + *Assisted*.
  - Only ask this question if at least 2 brands were selected in *Top of Mind* + *Others* + *Assisted*.
  - If only one brand was selected in *Top of Mind* + *Others* + *Assisted*, then do not show the *Favourite* variable and code it as the selected brand.
5. Create a “*Random Selection*” variable where you will select 3 brands at random from the ones known (*Top of Mind* + *Others* + *Assisted*).
6. Create a Loop to ask about the 3 brands selected.
  - Ask if the respondents will buy this product in the next month.
  - To do this, create a yes/no question which will be asked for each of the 3 brands.
  - The loop could be named “*Loop Buy*” and the question “*Buy*”.
  - Do not ask the loop if none of the brands were selected.

## Exercise 2: Using Dim

You will now be creating a script to check that the responses to three questions add up to 100.

*Follow these steps:*

1. Add the following chapter to your survey:

Question type: *Chapter*

Long caption: *When you are considering your financial future, how important are the short term (the next five years), medium term (five to ten years) or long term (more than ten years)? Please allocate 100 points between these three time periods, to indicate your priorities.*

2. Add the following three numeric questions:

Question names: *Short\_term, Medium\_term, Long\_term*

Question type: *Numeric*

Minimal value: *0*

Maximal value: *100*

3. Add a further question:

Question name: *Priority\_total*

Question type: *Numeric*

Make the question invisible during data entry.

4. Add one or more routing conditions to add up the totals, using declared variable(s) to keep track of the running total. Store the running total in *Priority\_total* (using the *Set value* routing action).
5. Add a new chapter, and have its long caption display *Priority\_total*. Add a message to indicate how many further points they need to allocate. Add routing logic to ensure that the chapter is only displayed if *Priority\_total* is less than 100.





## Session 150-4 **Advanced Loops**

### Outline

#### Topics presented

In this session, we will introduce you to:

- The loop properties of questions defined within a loop
- The loop built-in test values **IsLastIteration** and **CurrentIteration**
- The **For / Next** loop
- The **Break** directive

#### Learning outcomes

At the end of this session you will understand:

- How to perform tests and computations of the different elements of array variables within loops
- How to know, when executing a script, how many times a loop has executed
- How to take particular actions when you are on the last iteration of a loop
- How to create loops that will repeat a defined number of times – even if this number is calculated in real time
- How to create loops that will repeat until a particular defined state is reached

## Material covered

*In this session, the tutor will be demonstrating the nested loop in the example QEX (PowerTools\_with\_hh\_loop\_WITH\_routings.qex). Please pay close attention to the narrative you will find below when leading this session.*

## Questions residing in standard loops

Any question defined within a standard loop (such as a loop to create a question grid) has additional object properties pertaining to the loop.

When the **start question** of a routing is set on a question which is inside a loop, the routing is executed on each of the iterations.

For example: we have a question within a loop called Q2 which asks how each type of power tool owned is powered. If the start question of the routing is on the loop, or on the question, then it is the same as having three routings: one on each type of power source (gas, corded or cordless).

In a routing condition where the start question is in a loop, the *value* property of the question always references the question in the current iteration. If you want to access values in other loop items, you can use the **Iteration()** method.

### Example:

If the first response ("Gas") is displayed on the first loop iteration or Q2.

```
Q2.iteration(1).Value = 1
```

will return "Gas".

Other examples:

#### Example

```
Q2.Iteration(1).Value
```

```
Q2.Iteration(1).Answers.Index
```

```
Q2.Iteration(3).Answers.Caption
```

#### Result

Returns the value for the first item.

Returns the value for the first item.

Returns the caption for the third item.

## Loops within Loops

The principle is the same with a **loop of loops**. If we have a loop of loops with 27 responses (3 fuel types for each of 9 power tools), then if a routing's **start question** is the loop statement, or the question inside the second loop, it is the same as having 27 routings. If the **start question** is the first loop or on a question inside the first loop but outside the second loop, then it is equivalent to 9 routings.

In loops of loops, *Iteration* lets you indicate the loop shortcut.:

```
Q2_S.Iteration(Q2Loop:3,ResidenceLoop:2)
```

This is equivalent to

```
Q2_S.Iteration(ResidenceLoop:2,Q2Loop:3)
```

**Note:** If the start question of the routing is outside the loop, the iteration method is mandatory.

## Object methods for loops

There are two object methods relevant to objects created inside loops which can be used in comparisons. These methods will only return a value when used on an object within a loop.

### *IsLastIteration and CurrentIteration*

There are two object methods relevant to objects created inside loops which can be used in comparisons. These methods will only return a value when used on an object within a loop.

### *IsLastIteration and CurrentIteration*

You can test at what point in the execution of the loop your script has reached. `IsLastIteration` is true if the script has reached the final iteration of the loop.

```
Q2loop.IsLastIteration
```

If you need to refer to a specific iteration, then you can use the `CurrentIteration` property. This returns, as a numeric value, the current iteration number of the loop.

```
Q2Loop.CurrentIteration = 9
```

Is true if this is the ninth iteration of the loop.

An alternative way to achieve this is:

```
Q2loop.CurrentIteration.EntryCode = "09"
```

## AllValues property

To obtain an array of all the answers to a question in a loop, we use the keyword `AllValues`.

Examples:

- If Q10 is a single question inside a loop three iterations, `Q10.AllValues` might return  
`{2; 1; 8}.`
- If Q11 is a multi-coded question, `Q11.AllValues` might return  
`{2; 5; 8; 1; 4; 8}`

## For/Next Loops

A FOR/NEXT loop is an entirely different construct to a question loop. Used in a script, it allows you to create repeating, or “iterative” logic at any point within your script. FOR/NEXT loops do not create repeated questions in the way ordinary loops do.

Syntax:

```
FOR control_variable = start TO end
    [section of code to repeat]
NEXT control_variable
```

**control\_variable** - will be used as the loop counter

**start** - is the initial value of variable

**end** - is the finish value of variable

- The control variable (loop counter) has a differing value each time through the loop.
- The 'start' and 'end' values specify what the control *variable's* values will be and how many times the loop is executed.
- Every time the 'next' line is reached, the value of *variable* is incremented by 1 (+1).
- When the value in the control variable is outside the start to end values, the looping stops and program flow continues from the line after the next command.
- FOR...NEXT loops can be nested (remember to use a different variable for each loop).

### Example

The code used is as follows:

```
Dim Arr_cordless = {}

Dim i
Dim j

    'Parent loop iteration
i = Residenceloop.CurrentIteration
For j=1 to Q2loop.Responses.Count
    If Q2.Iteration(Residenceloop : i, Q2loop : j)=3 Then
        Arr_cordless = Arr_cordless.Insert(j)
    EndIf
Next j
```

```
' return array of cordless tools
return Arr_cordless
```

Note that it is not good practice to manipulate the number of times that a loop might iterate by using a variable for the *end* value and then altering the value of that variable during the execution of the loop.

For example, if our loop begins `For LoopCount=1 to x`, then we should not change the value of `x` inside the loop. If we do:

- This can give rise to an endless loop during an interview, and unpredictable results.
- You should always use either a system-defined value, a variable that will not change, or a constant, so that the outcome is predictable (e.g. `For LoopCount = 1 to Q2loop.Responses.Count`).
- Inside the loop, if you want to check the value in the control variable, you should put the expression in parentheses to avoid changing the value. For example:

```
If (LoopCount = 1) then 'check whether this is the
first iteration of the loop
```

This ensures that you checking the value of the control variable (in this case, *LoopCount*), rather than setting it.

## Break command

You can use **break** to end a FOR/NEXT loop before the control variable reaches the end value.

Normally, a loop will repeat a set number of times, defined by the difference between the *start* and *end* values. However, there are occasions where you might want the loop to end before the end value has been reached, and for this you can use the `Break` keyword.

For example, suppose we want to check whether the respondent picked the first product category before the second one at Q1.

```

dim i

dim picked_a = False

for i = 1 to Q1.answers.count
    if Q1.answers[i] = 1 then
        picked_a = true
        break
    elseif Q1.answers[i] = 2 then
        break
    endif
next i

return picked_a

```

## Recap

In this session, we have looked at:

- Using loops to test and update elements of array variables
- Checking to see which iteration of a loop is currently being performed with the two built-in loop values `CurrentIteration` and `IsLastIteration`
- Writing Loops that repeat a defined number of times with `For ... Next`
- Loops within loops
- Using `AllValues` to aggregate the values from loop questions
- Using the `Break` directive to exit a loop

## Practical exercise

### Writing script loops

*Follow these steps:*

1. Ask the respondents about their hotel stays in the last year. Find out what proportion of their stays was for business, personal or other purposes. For each type of stay, ask them to allocate a percentage (0%-100%).

You can achieve this by having a numeric question inside a loop.

The sum of the percentages should equal 100. If it does not equal 100%, then show a warning message.

**Hint:** you can use the property `AllValues.Sum` to check the value of the numeric question.

**Note:** To show the current statement in the long caption, you can use `??ShortCutOfTheLoop??`. Drag the loop from the left-hand panel into the Long Caption of the question to insert it automatically.

2. Create another loop with 5 brands and 2 questions attached.

Would you consider buying this brand: [brand name]?

Responses: Yes, No

Why would you not buy [brand]?

*Only ask if no was selected in the first question (one screen per brand).*

3. Outside the loop, create a summary multiple question including all the brands with a yes selected. Ask the respondent to select his or her first choice.





# Example questionnaire

## POWER TOOLS SURVEY (North America)

### SCREENER SECTION

**S1 (Screeners). Do you have any power that you use tools at home? By power tools we mean tools like a lawn mowers, hedge trimmers, chain saws, pressure washers or power hand tools.**

- Yes – *Continue*
- No - *Close*

**S2 (Screeners) Are these tools that you own personally, or are they tools used for business or trade?**

- Only tools used for business or trade - *Close*
- Only tools owned personally - *Continue*
- A mixture of both - *Continue*

**S3 (Screeners) Do you also own a second residence for your own use, such as a vacation home?**

- Yes - *Continue*
- No - *Continue*

### MAIN QUESTIONNAIRE

**Q1. Which of these power tools do you own?**

*IF yes at S3, use this text instead: Which of these power tools do you own, and normally keep at your main home? For the moment, we will just focus on those those at your main home. ]*

- Push Lawn mower (1)
- Riding Lawn mower (2)
- Hedge trimmer (3)
- Chain saw (4)
- String trimmer (5)
- Lawn edger (21)
- Pressure Washer (6)
- Hammer drill or power drill (7)
- Power sander/orbital sander (8)

**Q2. What/And what/And finally, what kind of [power tool at Q1] is it?**

Use the text “**What**” the first time the question is presented, “**And what**” for each subsequent time, and use “**And finally**” the last time Q2 is presented, provided if it has been presented more than twice.

- Gas
- Electric (corded)
- Cordless /rechargeable

*Note: the following power options are allowed at Q2*

	Gas	Electric	Cordless/rechargeable
Push Lawn mower	Y	Y	Y
Riding Lawn mower	Y	n	Y
Hedge trimmer	Y	Y	Y
Chain saw	Y	Y	n
String trimmer	Y	Y	Y
Lawn edger	Y	Y	Y
Pressure Washer	Y	Y	n
Hammer drill or power drill	n	Y	Y
Power sander/orbital sander	n	Y	Y

*If 2 or more cordless/rechargeable tools.*

**Q3. Are you aware that some power tool manufacturers offer interchangeable rechargeable batteries or power packs that you share between your power tools?**

- Yes
- No
- Not sure

*If Q3.yes*

**Q4. Do you share batteries between any of these tools that you have?**

*Only present those selected which are battery-powered:*

- Hedge trimmer
- String trimmer
- Lawn edger
- Hammer drill or power drill
- Power sander/orbital sander

**Q1\_S. Now we would like to focus on any power tools you may normally keep at your second home. Which of these power tools do you own, and keep at your second home?**

*IF yes at S3, add: For the moment, we are just interested in those at your main home. ]*

- *same answer list as at Q1*

**Q2\_S. What kind of [power tool at Q1\_S] is it?**

Use the text “**What**” the first time the question is presented, “**And what**” for each subsequent time, and use “**And finally**” the last time Q2\_S is presented, provided if it has been presented more than twice.

- *same answer list as at Q2*

*If 2 or more cordless/rechargeable tools. AND Q3 was not asked:*

**Q3\_S. Are you aware that some power tool manufacturers offer interchangeable rechargeable batteries or power packs that you share between your power tools?**

- *same answer list as at Q2*

*If Q3.yes or Q3\_S.yes*

**Q4\_S. Do you share batteries between any of these tools that you have?**

*Only present those selected which are battery-powered:*

- *same answer list as at Q2*

**Q5. Which of these tools are you likely to replace in the next 12 months?**

*(Only present those selected at Q1 or Q1\_S)*

- Push Lawn mower
- Riding Lawn mower
- Hedge trimmer
- Chain saw
- String trimmer
- Lawn edger
- Pressure Washer
- Hammer drill or power drill
- Power sander/orbital sander

*For each selected, if power source is currently not cordless, ask:*

**Q6. Would you consider purchasing a rechargeable/cordless [tool at Q5]?**

*(If more than one item selected “And similarly, ...”)*

- Yes
- No
- Not sure

*If Q6.no ask:*

**Q7. What are your reasons for not choosing a cordless or rechargeable [tool at Q5]**

- *Open text response*

*From the combined answers to Q1 and Q1\_S, randomly select no more than 3 answers, and repeat Q8 for each answer selected.*

**Q8. What feature do you like the most about the [selected tool at Q1/Q1\_S] you currently own?**

- *Open text response*