**Tell:** Use the text provided explain to the group.

**Do:** Follow the instructions for the activity.

|  |  |  |
| --- | --- | --- |
| **Session/Title/Topic** | **Content / Description** | **Assets/Media** |
| Introduction | **SAY:**  Hello and welcome to this course on PowerShell!  **DO:**  Facilitator to spend a brief 30 seconds introduction themselves. If they have relevant experience of PowerShell it would be useful to mention it here, but only in brief.  Create a “parking space” flipchart and stick it to the wall. Explain that any questions which the facilitator asks to park, due to them not being relevant at the time, attendees should write on a post-it note and stick to this flipchart.  Draw a “P” in blue whiteboard marker with a box round it, then a big box around the whole flipchart. | 3Variables.ps1 |
| Learning Objective Overview | **SAY:**  Welcome to the second workshop on this course! This workshop will cover variables and comparison operators. But before we go on, lets do a quick recap of what we covered in the 1st workshop.  **DO:**  Do a quick recap | 3Variables.ps1 |
| Variables | **SAY:**  Now that we’ve recapped over the first workshop, lets crack on with variables and comparison operators.  **SAY:**  A variable is just a way for us to store data for use later on in a script. In PowerShell in may be declared one of two ways:  $Name = Value  OR  New-Variable – Name “Name” –Value “Value”  The value may be static or the result of a cmdlet. For example:  $Message = “Hello World”  AND  $Date = Get-Date  Both result in a variable stored in memory. Note that variables only exist within a session, and once that session is ended – be it a script or a PowerShell ISE window – the variables are unloaded from memory. | 3Variables.ps1 |
| Typecasting | **SAY:**  PowerShell will implicitly “typecast” a variable into the appropriate data type. So:  $Name = “Ben” will be a string  $Age = 24 will be an int  $Married = false will be a Boolean  However, if required you can explicitly typecast a variable. So:  [string]$Age = 24 will be a string rather than an int  So  $Age = 24  $Age \* 3 will equal 72  Whereas  [string]$Age = 24  $Age \* 3 will equal 242424  Remember methods and properties? We can access these on the classes themselves by simply using ::. So, for example:   * Go through a few examples, such as string and int32 properties and methods. | 3Variables.ps1 |
| Activity 1 | **DO:**  Go through the examples in 3variables.ps1 before moving on.  Now you’ll be given a task sheet with 8 code snippets. Each snippet is jumped, not in the correct order. Rearrange these pieces of code to something that has the correct syntax to declare a variable. If you have any questions let me know.   * Hand out “Variables and Comparison Operators Activity 1” sheet and give attendees 10 minutes to complete | 3Variables.ps1/ Variables and Comparison Operators Activity 1 |
| Activity 1 recap | **DO:**  Go through the activity sheet together, answer any questions the group may have by encourage answers to come from within the group rather than the facilitator. Once questions have been cleared then move on to the second part of the workshop. | 3Variables.ps1/ Variables and Comparison Operators Activity 1 - Facilitator |
| Arrays and Activity 2 | **SAY:**  Now that we’ve discovered variables, we’ve got one more datatype to talk about before looking at comparison operators. An array.  An array in PowerShell is simply a way for us to store multiple types of the same object in one variable. The syntax to make an array is just the same as a variable, except you use commas to dictate different objects:  $StringArray = “Hello”, “World  ArrayIndexes start at 0, and you can access the array index by using [].  So $StringArray[0] will be the first object in the array, and so on and so forth. Each array will also have an inbuilt property called “count” which is equal to the number of items in the array.  **DO:**  State that arrays are vitally important for our next workshop, and essentially to manipulating large amounts of data. Hand out “Variables and Comparison Operators Activity 2” sheet and give attendees 10 minutes to finish. | 3Variables.ps1/ Variables and Comparison Operators Activity 2 |
| Activity 2 recap | Go through the activity sheet together, answer any questions the group may have by encourage answers to come from within the group rather than the facilitator. Once questions have been cleared then move on to the second part of the workshop. | 3Variables.ps1/ Variables and Comparison Operators Activity 2 - Facilitator |
| Comparison Operators | **DO:**  Say the below and write on a whiteboard where appropriate:  Before letting you lose on the final activity sheet of this workshop, we’ve one more area to cover: comparison operators. The syntax of these is quite simple:  Input Operator Value  And we’re going to cover four operators:  Greater Than (-gt)  Less Than (-lt)  Like (-like)  Equal To (-eq)  Now if the input – which is on the left of the operator – is a single instance of an object then the result is a Boolean: true or false.  If the input is an array, then the result is any true values. So if there are no true values then nothing is returned. | 4Comparison Operators.ps1 |
| Comparison Operators | **SAY:**  This all might be a bit confusing, but we’ve got some examples to go through.  **DO:**  Go through the 4Comparison Operators.ps1 example script. | 4Comparison Operators.ps1 |
| Task Sheet | **SAY:**  Now, if there are no-more questions, you have an activity sheet to keep you occupied until the ned of the workshop.  **DO:**  Hand out the learner sheet for this workshop and give attendees till the end of the workshop to complete. | Task Sheet |
| End | **DO:** Wrap up workshop |  |