**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. | 1Background.ps1 |
| Learning Objective Overview | **SAY:**  This course is designed to equip you with the required programming and PowerShell knowledge for you to independently work with PowerShell and perform further self-study; by the end you should be navigating the various scripting tools with ease, and be able to use PowerShell effectively for virtually any task.  A lot of the training out there assumes some basic programming or scripting knowledge, and tends to be highly specialised; it’ll train you on how to perform x or y for a certain program. However, this course approaches PowerShell as a programming language… meaning you should be able to use it as an effective tool for whatever you want after this course. | 1Background.ps1 |
| Fun examples | **SAY:**  First, just to give you an idea of how versatile the language is, we have some fun examples to go through… Don’t worry about the code for now, just sit back and see some of the different things this language can do.  **DO:**   * Explain that the first example relates to querying a RESTAPI about cat facts, and using the computer’s inbuilt speech synthesizer to communicate one of these… then run the CatFactsGalore function in 1Background.ps1 * Explain that the second example is downloading something over the internet and then displaying it to us. Run the ASCI rickroll line, note that this is quite loud so it’s recommended for the facilitator to turn down the volume on their device. | 1Background.ps1 |
| .NET | **SAY:**  With the fun examples out of the way, who here has any programming experience or knowledge?  **DO::**   * Explain that PowerShell is based on .NET. This means that .NET acts as our translator; it turns what we write in PowerShell into machine code that can actual run. Machine code is 1s and 0. As long as we have the right combination of .NET and PowerShell, our code can run anywhere! * The element of .NET that does the translation is called the common runtime library, or CRL. * The translation is done just-in-time as and when the code is run. * This is opposed to a compiled language, such as C#, whereby the code is translated first and then the translated version is run. * JIT is more portable; we can run it anywhere if we have interpreter and language installed. Compiled is faster, but requires compiling for each system it needs to run on.   Note: If anyone has experience with programming or .NET lean on them for the above explanation.   * This means that you can direct reference .NET elements. For example, a common use of C# is creating windows forms. But you can do that in PowerShell too by simply referencing the .NET object. Don’t worry about the syntax for now but see this example   + Run “create a gui”” example. Make sure to see $INSTALL\_DIRECTORY first. * And you can also referencing other items that use .NET, such as Excel   + Run “Invoke Excel” example | 1Background.ps1 |
| OOP | **SAY:**  One last thing before we move on to PowerShell specifics… does anyone know what I mean by object-orientated-programming, or OOP?   * Everything is an object, but what’s an object? * An object may be considered as a blueprint, and an instance something made from that blueprint. * The properties and methods of an object are defined, then any new instances based off the object can use these.   It may seem a little daunting but hopefully these activity sheets should help. If you get stuck or have questions please let me know.   * Hand out Background and Primitive Datatypes activity 1 sheet and give attendees 10 minutes. The facilitator sheet contains answers. |  |
| GUIs | **SAY:**  Finally, before we move on to the syntax of PowerShell, primitive datatypes and some activities… Can anyone tell me what program I’m using to run this code?   * If attendees do not guess right, answer with “Integrated Scripting Environment”, ISE for short. * The ISE is what it says on the tin; it’s a program that is intended to help you create scripts with PowerShell, it is heavily recommended to use this where you can instead of the standalone PowerShell console. * Walk through: scripts run in a linear, top-down, fashion. You can click on a row and press f8 to only execute that code. Control + C should stop running code. | 1Background.ps1 |
|  |  |  |
| Execution Policy | **SAY:**  In order to run any scripts whatsoever, we need to change our execution policy. This is done by the following cmdlet:  Set-ExecutionPolicy  By default, PowerShell will only run script files – which have an extension of .ps1 – if these are signed by a certificate which Microsoft trusts. Therefore in order to run your own scripts in a test environment you’ll need to change this to unrestricted, though this is not recommended on production systems. | 1Background.ps1 |
| Help | **SAY:**  Also, before we cover just what a cmdlet is and the different pieces, PowerShell comes built with various commands you can what to get some help about an item.  If you want to see what input a cmdlet takes, or some examples, you can do so with the “Get-Help” cmdlet. –Examples will show you examples, whilst –online will take you to a corresponding technet article.  You can also use the –whatif cmdlet to see what a cmdlet will do. | 1Background.ps1 |
| Verbs and nouns | **SAY:**  Now we have all that out of the way, lets look at cmdlets. Note that whilst this will be accurate for 99% of PowerShell, it may not apply to Exchange. This is as the Exchange Console was being developed in parallel to PowerShell… and once all of the standards for PowerShell were ratified, these were not retrospectively applied to the Exchange console. So when using that there may be some differences.  **DO:**   * Draw the word cmdlet somewhere in the room. * Use the example of “Get-Service” * Ask attendees to break down the cmdlet into two distinct sections * The key is that “Get” is a verb whilst “Service” is a noun. * Break down what a verb and a noun are:   + Verb: describe an action   + Noun: describe an object | 1Background.ps1 |
| Verbs | **SAY:**  In PowerShell, there are 5 basic verbs which will see you through 80% of your PowerShell needs. These are:   * Get * New * Add * Set * Remove   **DO:**   * Go through the examples under “START OF NAMING STRUCTURE” * Before this ensure that you declare $ItemPath * Ensure you go through these 1 by 1, giving attendees a chance to guess what will happen. The new, add, set and remove cmdlets are nicely shown by creating, adding to, overwriting and deleting a text file. * Get = retrieves an instance of an object * New = creates a new instance of an object * Add = adds something to an instance * Set = changes something already existing on an instance * Remove – removes an instance of part of an instance   Also use this opportunity to explain:   * The pipeline passes objects through from one cmdlet to another * Format-list lets you format the output | 1Background.ps1 |
| Nouns | **SAY:**  Now can anyone guess what the noun relates to?  The correct answer is that it relates to an object. The verb defines what you’re doing, and the noun defines what you’re doing this to.  **DO:**  Say that in PowerShell we can use the “Get-Member” cmdlet to find out what type of object something is.  Use “Get-Member” on “Get-Service” and explain the results to attendees. Ensure there are no-more questions and then offer a 5 minute break. | 1Background.ps1 |
| Break | **DO:**  Take a 5 minute break if required |  |
| Primitive Datatypes | **SAY:**  Now, lets take a look at the primitive datatypes. Once we’re done you’ll then be given a task sheet to complete during the remainder of the workshop.  There’s 3 datatypes we’ll cover, but there are literally millions. These are:   * Boolean * String * Int   A Boolean is true or false.  A string is a piece of text, denoted as something being surrounded by “quotation marks”  An INT is any whole number. By default it can use up to 32 bits to store the number.  **DO:**  Go through some examples with individuals, using get-member to demonstrate the data type. | 2Primitive data types.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.  Go through parking space and ensure all relevant questions are answered before moving on. | 2Primitive data types.ps1 |
| End | **DO:** Ask attendees to complete the assessment.  Also remind attendees of when the next workshop is. | Slide 4 |