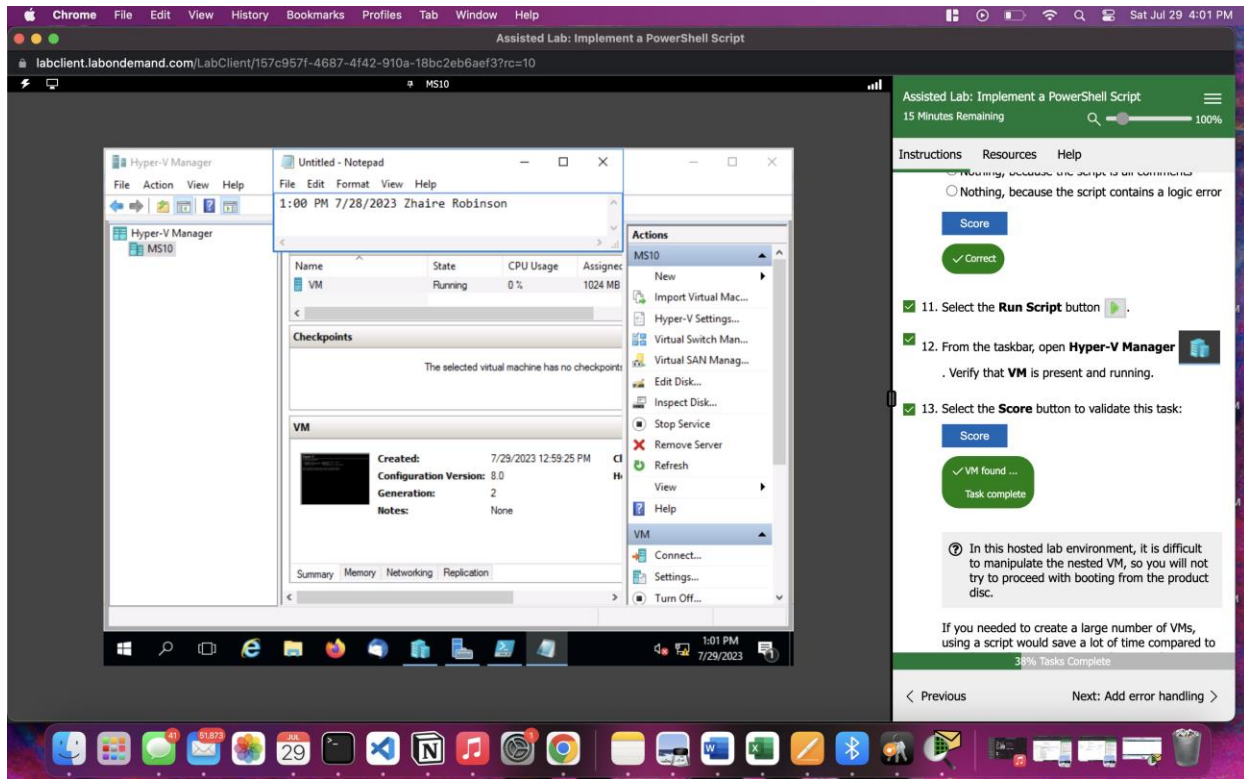


Lab: Implement a PowerShell Script (4)

Lab 1: Develop script (1)



Lab 2: Add error handling (1)

The screenshot displays a web-based lab environment titled "Assisted Lab: Implement a PowerShell Script". The interface is divided into several sections:

- Hyper-V Manager:** Shows a virtual machine named "MS10" with the following details:

State	CPU Usage	Assigned Memory	Uptime
Running	0 %	1024 MB	00:06:17
- Notepad:** Displays a PowerShell script with the following content:

```
1:00 PM 7/28/2023 Zhaire Robinson  
State CPU Usage Assigned Memory Uptime  
Running 0 % 1024 MB 00:06:17  
he selected virtual machine has no checkpoints.  
7/29/2023 12:59:25 PM Clustered: No  
Version: 8.0 Heartbeat: No Contact  
2  
None  
Replication
```
- Instructions:** A sidebar on the right provides step-by-step guidance:
 1. In the ISE window, select the **Run** button, and observe the error messages.
One of the greatest challenges in developing effective scripts is to anticipate and account for errors. There are two ways you could approach this one.
 - You could change the value of the \$vmname variable so that the script creates a different VM.
 - Or you can enable the Try code block to check whether a VM of that name exists already.You'll take the second approach for this activity.
 2. For each line of code in 6 through 12 and 18, remove the # comment character from the start of the line. As you uncomment each line, observe how the ISE highlights errors in the structure of the code. When you uncomment the last line, the errors will be resolved.
 3. Save the file and then run it again.
 4. Switch to Hyper-V Manager. Verify that the machine uptime has reset. This is a new version of the VM.
 5. Select the **Score** button to validate this task:

The bottom of the screen shows a macOS dock with various application icons, including Finder, Mail, Photos, Messages, Safari, and others.

Lab 3: Improve a script with user input (1)

The screenshot displays a virtual machine environment with a Windows operating system. The main window shows a PowerShell console with the following commands and output:

```
13 catch{}
14 finally{
15     New-VM -Name $vmname -Path $path -Generation 2 -MemoryStart
16     Add-VMdvdDrive -VMName $vmname
17     Set-VMdvdDrive -VMName $vmname -Path $bootdisc
18     Set-VMFirmware $vmname -FirstBootDevice ( Get-VMdvdDrive -V
19     Get-VM $vmname | Start-VM
20 }
21 }
```

The output shows the VM running successfully:

Name	State	CPUUsage(%)	MemoryAssigned(M)	Uptime	Status
VM	Running	0	1024	00:08:52.4240000	Operating normally
VM	Off	0	0	00:00:00	Operating normally

On the right side, there is a lab interface titled "Assisted Lab: Implement a PowerShell Script" with a progress bar at 72% and a "Score" button. The lab instructions include:

- 8. Look at line 21. This bracket closes the **If** block.
- 9. Use the **Arrow** keys to move the cursor between the curly brackets in lines 20 and 21. Observe how the matching bracket is highlighted in the earlier code (line 21 matches with line 7, while line 20 matches with line 14).
- 10. Run the whole script again to test it. At the prompt, select **Cancel**.
- 11. Run the script again, selecting **OK** at the prompt.
- 12. Select the **Score** button to validate this task:

The "Score" button is highlighted, and a message indicates "VM with 4.626 seconds uptime found ... Task complete".

Lab 4: Improve a script with control structures (1)

The screenshot shows a web-based lab environment titled "Assisted Lab: Implement a PowerShell Script". The lab interface includes a terminal window, a command palette, and a sidebar with instructions and a progress bar.

Terminal Window:

```
vm3.ps1 X
1 Virtual Machines to
2
3 n10.iso'
4
5 }::LoadWithPartialName('Microsoft.VisualBasic') | Out-Null
6 isualBasic.Interaction)::InputBox('This script will create and start bet
7
8 $i=1
9 while ($i -le 10) {
10     $vmname=$i -ErrorAction Stop
11     " $vmname$1" -Turnoff:$true -Confirm:$false
12     me "$vmname$1" -Force
13     ath = "$vmname$1" -Recurse -Force
14 }
15
```

Command Palette:

Commands X

Modules: All Refresh

Name:

A:

- Add-AppClientConnectionGro
- Add-AppClientPackage
- Add-AppPublishingServer
- Add-AppPackage
- Add-AppProvisionedPackage
- Add-AppVolume
- Add-SCDataCacheExtension
- Add-BitLockerKeyProtector
- Add-BitsFile
- Add-CertificateEnrollmentPolicy
- Add-ClusterSCSITargetServerRc
- Add-Computer
- Add-Content
- Add-DfsrConnection
- Add-DfsrMember

Instructions:

programming languages when it comes to variable data types. The execution engine will try to dynamically change (or cast) a variable to fit the use to which it is being put. This can lead to unexpected results, however. If the script uses a variable in a way that depends on a certain data type, it is better to explicitly declare the variable as that type. In PowerShell, you can set the type explicitly by using square brackets and the type, such as `[Int]` or `[String]`.

- 5. In line 8, observe that the `If` block is now testing an integer, rather than a string. The `-le` operator means "less than or equal to."
- 6. In line 9, observe the use of a `For` block to create a looping structure within the code.

The variable `$i` is used to count the number of iterations, starting at 1. The use of `$i++` within the condition causes the value of `$i` to increase by 1 for each iteration of the loop. When `$i` exceeds the value of `$prompt`, the loop exits.

- 7. Observe the use of `$i` in the cmdlet parameters to change the name used for each VM.
- 8. Run the script a few times, entering different values to test that it works reliably.
- 9. Select the **Score** button to validate this task:

Score

90% Tasks Complete

< Previous Next: Complete comprehensive... >