Power Shell 101

**Module 7 Hands-on Activity – PowerShell in VSCode**

5/24/2019 Developed by Jin Chang and Sion Yoon

Center for Information Assurance (CIAE) at City University of Seattle



**Learning Outcomes**

* Learn on the difference between Write-Output vs. Write-Host
* Learn about Read-Host
* Learn how to make an InputBox

**Resources**

* Different ways to Write and Read-Host
  + <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/?view=powershell-6>
  + <https://www.youtube.com/playlist?list=PL6D474E721138865A>
* InputBox
  + <https://www.itprotoday.com/powershell/getting-input-and-inputboxes-powershell>

**Activities**

* Write
* Read-Host
* InputBox
* Q&A

**Write**

1. Open PowerShell in VSCode.
2. We will start by first learning the difference between Write-Output and Write-Host. Type:

**% Write-Output “Hello”**

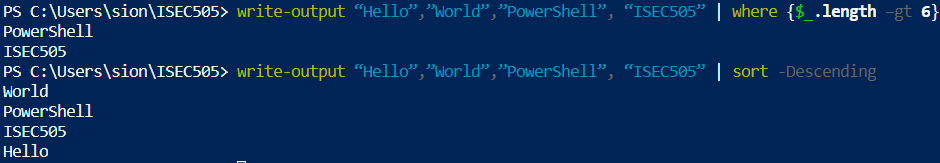
**% Write-Host “Hello”**

It looks like they are the same but they work in very different ways. The main difference between them is that Write-Output sends the specified object down the pipeline to the next command while Write-Host writes to the console itself.

1. Here is an example with Write-Output. Let’s put a few strings into a pipeline and filter to get only the strings that have a length greater than 6. Type:

**% write-output “Hello”,”World”,”PowerShell”, “ISEC505” | where {$\_.length –gt 6}**

**% write-output “Hello”,”World”,”PowerShell”, “ISEC505” | sort -Descending**



The only strings that have a length greater than 6 are “PowerShell” and “ISEC505” and they are the only one that made it through the command. You can try it with Write-Host, and it will not give you the same result because Write-Host does not send the objects down the pipeline.

1. As mentioned above, Write-Host writes to the console itself so you can specify background and foreground colors. Here is an example. Type:

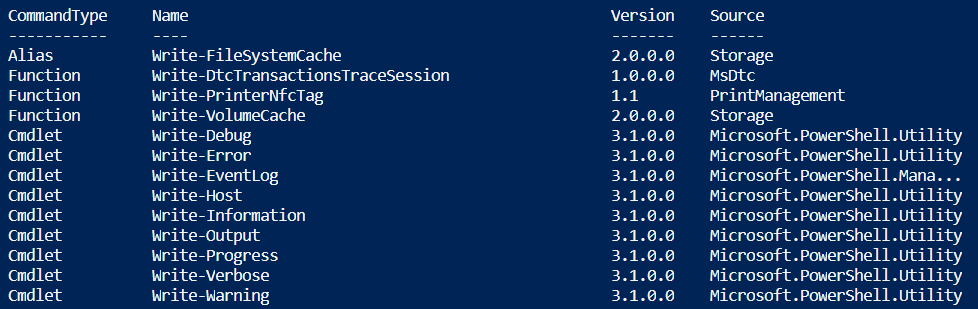
**% write-host "Hello" -BackgroundColor yellow -ForegroundColor blue**



However, it is not used for the output of your script and simply used for display and therefore, it is best not to use this command.

1. Now let’s see some other write cmdlets. Type:

**% get-command –verb write**



Here are the descriptions for each of the write cmdlets to keep in mind:

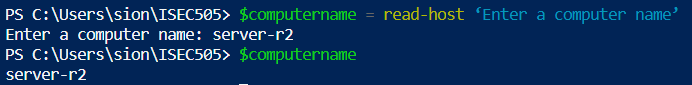
|  |  |
| --- | --- |
| **Cmdlet** | **Description** |
| Write-Debug | Writes debug messages to the console from a script or command. |
| Write-Error | Declares a non-terminating error. By default, errors are sent in the error stream to the host program to be displayed, along with output. |
| Write-EventLog | Writes an event to an event log. |
| Write-Information | Specifies how PowerShell handles information stream data for a command. |
| Write-Progress | Displays a progress bar in a Windows PowerShell command window that depicts the status of a running command or script. You can select the indicators that the bar reflects and the text that appears above and below the progress bar. |
| Write-Verbose | Writes text to the verbose message stream in PowerShell. |
| Write-Warning | Writes a warning message to the PowerShell host. The response to the warning depends on the value of the user's $WarningPreference variable and the use of the *WarningAction* common parameter. |

|  |
| --- |
| **Q1**  Search online and provide an example of how to use two of the Write cmdlets on the table above. Provide your screen captures below. |

**Read-Host**

1. One way to gather input from a user is to use the Read-Host cmdlet. This command accepts a prompt and allows the user to type a response. Create a variable, and have it equal to the read-host cmdlet and a prompt. Thereafter, provide a response. Note that the response is placed into the pipeline.

**% $computername = read-host ‘Enter a computer name’**



If you see the contents of the variable that you created, you will see that the response was placed into that variable.

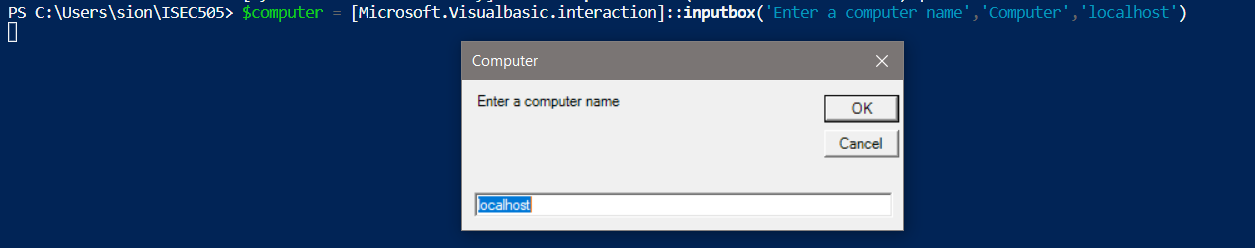
**InputBox**

1. There may be times when you need to display a popup dialog box. To do this, we need to use System.Reflection.Assembly .NET Framework class to load the ‘Microsoft.VisualBasic’. The following command loads the part of the .NET Framework into memory. The loading generates a success message, which is piped to Out-Null to suppress it.

**%[system.reflection.assembly]::loadwithpartialname('Microsoft.Visualbasic') | Out-Null**



1. Once this is done, create a variable and ask the interaction class of the Microsoft.VisualBasic part of the framework to use the InputBox() method. You can put up to three arguments in this order: (1) your prompt, (2) the title for the dialog box, and (3) a default value.



1. When you click enter, it launches the InputBox with the arguments that you set. Click OK. What is in the textbox will end up in the computer variable that we created. Type:

**% $computer**



|  |
| --- |
| **Q2**  Create a different example of the InputBox (using different arguments) and provide a screen capture of the InputBox and output of the variable that you created.    **Q3**  What is MsgBox() and how is it different to InputBox()?  Inputbox receiver an input during runtime from user. Msgbox displays a dialog box conainint a message. |

PUSH YOUR WORK TO GITHUB

Once you completed the Hands-on practice, **export this document to PDF**, save in the same Module 6 folder you cloned, then do the following to push your work to GitHub

Open the terminal from the VSCode by hitting the control + ~ key, make sure you are in the right path, for example: KimNguyen/Desktop/ISEC505/HOP07-KimNguyenMai/Module 7

Type the following command:

>>> **git add .** (to copy all changes you have made)

>>> **git commit -m “Submission for Module 7 – YOUR GITHUB USERNAME”** (To add a message to your submission)

>>> **git push origin master** (to upload your work to Github)