Power Shell 101

**Module 9 Hands-on Activity – PowerShell in VSCode**

5/25/2019 Developed by Jin Chang, Sion Yoon

5/25/2019 Tested by Sion Yoon

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



**Learning Outcomes**

* Describe how to create and use functions in PowerShell

**Resources**

* Functions
  + <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions?view=powershell-6>

**Activities**

* How to define a function using the PowerShell syntax.

**Functions**

1. In this activity, we’ll create several functions in a file.
2. Go ahead create a folder under c:\isec505\module8 in a command line and go to the folder

**mkdir isec505\module8**

**cd isec505\module8**

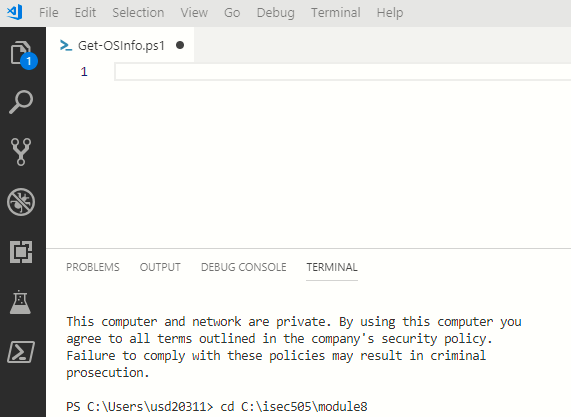
Make sure you’re in module8 folder

1. To create a cmdlet file, type the following

**code add-numbers.ps1**

This will open Visual Studio Code with the file tap named Add-Numbers.ps1

Wait utill the PowerShell integrated console shows up the type cd isec505\module8 to move your current directory as shown below.



1. We have learned various cmdlets for individual actions. What if you have a set of cmdlets that have been used multiple times. Wouldn’t be handy if we can create a way that you don’t need to copy and paste or type all the time? Functions are very useful to save copying and pasting the same code over and over again. Define once and use it everywhere you need. “Function is a series of commands (or cmdlets) that are logically grouped together to perform a specific task”
2. The following is a full syntax of a function in PowerShell

In short, after a function is declared, it (1) takes the inputs to complete the task defined, (2) executes the body of function in statements, (3) returns the value to the caller if needed and exits.

**function** [<**scope**:>]<**name**> **[([type]$parameter1[,[type]$parameter2])]**

**{**

**param([type]$parameter1 [,[type]$parameter2])**

**dynamicparam {<statement list>}**

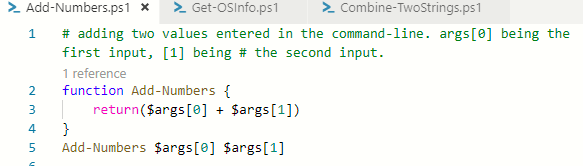
**begin {<statement list>}**

**process {<statement list>}**

**end {<statement list>}**

**}**

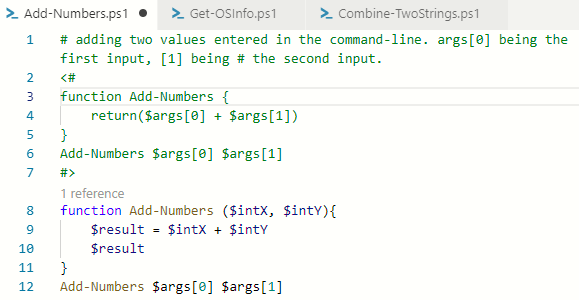
1. There is a key word ‘function’ followed by its name. The name should be meaningful and represent what the actual task is about the operation. Try to use a form of “Verb-Noun” as in Do-Something such as Add-Numbers. Don’t use a generic name like function A {} or function ABC {}.
2. Input values can be passed into a function in several ways. Go ahead and type the following functions in the file, Add-Numbers.ps1. In general, $args[] is a string object therefore the user’s input in string and converted to loosely typed variable into integer if it is a number.



In your console window. Type .\Add-Numbers.ps1 99 1



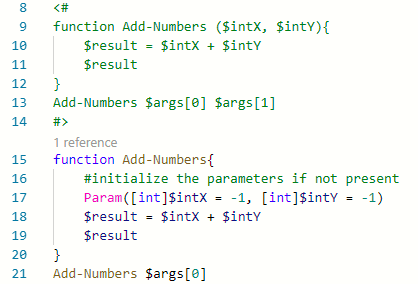
1. Go ahead and use comment block <# and #> to block the function you defined in the previous step. Now, here is another way to pass the input values.



In your console window. Type .\Add-Numbers.ps1 99 1



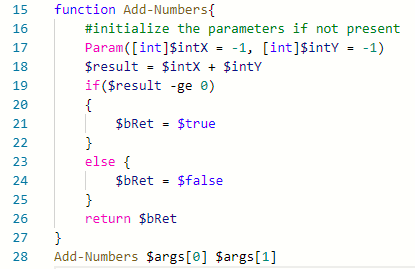
1. Again, use the comment block the functions defined in the previous steps. You can use “Param” block to pass the values while initializing input values to a specific type (aka strongly typed as opposed to loosely typed). It is a good practice to initialize the input values if missing. We will go over more detail of this “Param” in more detail in a later section.



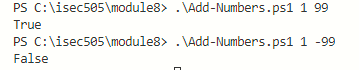
In your console window. Type .\Add-Numbers.ps1 100 by passing only one input. Note that the second parameter is initialized to -1 since it is missing not passed in.



1. Let’s look at the return statement. The primary goal of putting return in a function is to signal ‘exit’. It is not mandatory in PowerShell but depending on the overall operations, a value or a set of values need to be returned for the subsequent steps.



In your console window. Enter positive or negative values to ensure the function returns the value correctly.

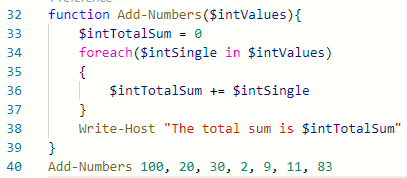


1. Finally, to display new cmdlet’s object methods and properties, you can also use other cmdlets on other existing cmdlets, for instance, ‘get-member’.



|  |
| --- |
| **Q1** **Write a function that (1) takes two input strings from the command-line, (2) combines two strings then (3) convert each string in a uppercase and finally (4) prints the output. Write this function as a cmdlet in a file called “Combine-TwoStrings.ps1. Note that (2) and (3) can be swapped. Insert the screenshot of the code and its output. Feel free to look up the previous group activity documents to concatenate two strings using “+” operation.**  **(e.g.) INPUT: hello world 🡪 [Your Function] 🡪 OUTPUT: HELLO WORLD** |

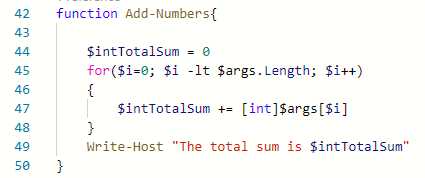
1. Let’s look at more examples. You can also pass an array to operate the items in it. Comment-block the previous functions, and type the following examples in the file.



Enter your cmdlet to calculate the total sum.



1. Comment-block the previous function, and type the following example that shows the dynamic list of numbers passing to the function.



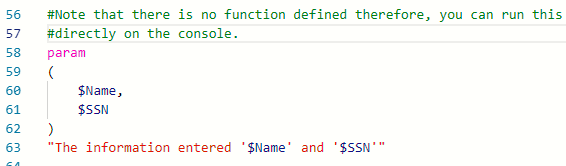
Try number of input values in the script followed by running your cmdlet in the console.

🡪 

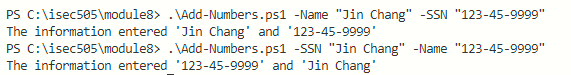
 🡪 

 🡪 

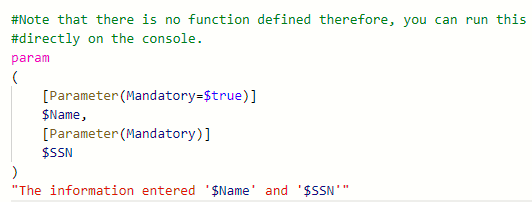
1. In the command-line, we often express a switch or a parameter using “-” followed by an actual parameter value such as “**-LogName Application**”. Let’s take try the following example. Go ahead and comment-block all the previous functions and type as below.



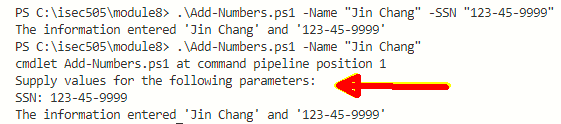
1. Go to the console and type as below. As the output shows, a value is assigned to the value to defined parameter if you specify the parameter names such as “-Name” or “-SSN”.



1. We can make those parameters “Mandatory” if needed. The following example shows how, and type the example in your script file.



1. Test out the script without passing one of the mandatory parameters and to see what you get in the command-line. Indeed. It forces you to enter the parameter value to be entered. Note that (Mandatory = $true) and (Mandatory) are the same since its default value is set to $true.



|  |
| --- |
| Q2 You will write two functions. First function stores random numbers to 2 arrays. Create two arrays, one named your last name $CHANG (*use your\_lastname)* and the other with your partner’s last name $SMITH (*use\_your partner’s lastname*). Then pass them to the second function that compares each number in both arrays. After comparison, count the total number of wins (win = greater value) and print out who the winner is.     1. Define and initialize two arrays. 2. Fill each array with 10 random numbers from 1 to 100.   You can do this as following using Get-Random cmdlet:  **1..100 | Get-Random –Count 10**   1. Call your second function by passing the arrays   #calling a function with parameters can be done as following:  **Your-SecondFunction $CHANG $SMITH**   1. In the second function, you then compare the arrays, and claim who the winner is. 2. Repeat your script 5 times and get the screen shot then insert below.   Insert the screen shot of your functions🡪  Insert the screen shot of your results🡪 |