Advanced Scripting   
Variables

Last Updated: 9/23/2021 8:45 AM Version 1  
Document Prepared for: CIT361 Student

# Name Ryan Skidmore ID 615907798

# Instructions

Save a copy of this document. Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

Explore the loosely typed variable system in PowerShell. A variables type is set at assignment time.

# Requirements

* PowerShell

# Task 1—Variable Basics

Explore variable assignment and types.

## Steps

1. Variables are created when you assign a value to them. Try accessing a variable that does not exist. Start a new instance of PowerShell and enter:  
   $foo
   1. What was the result?[blank]
2. Try this:  
   $foo -eq $null
   1. What was the result? True
   2. What can you infer about unassigned variables? They are null
3. To see all the variables defined in your PowerShell instance you can enter the command:  
   Get-ChildItem variable:
   1. What is the value of the *$PROFILE* variable? C:\Users\Derek Ryan\Documents\WindowsPowerShell\Microsoft.PowerShell\_profile.ps1
4. Use Test-Path to see if a variable exists. Test to see if the variable pwd exists. Enter:  
   Test-Path Variable:pwd
   1. What is the result? true
   2. Does the variable foo exist? false
   3. Does the variable shelled exist? false
5. Create the variable foo. Enter:  
   $foo=1
6. Get the value of $foo Enter:  
   $foo
   1. What is the value of foo? 1
7. Use Test-Path to see if foo exists. Does foo exist? True

# Task 2—Variable Types

A variable’s type is defined at assignment. You can change a variable’s type by assigning the variable a value of a different data type. You can always get the data type of a variable by calling the gettype() method of the variable.

## Steps

1. Make sure $foo=1
2. To get the type of data in foo call foo’s gettype() method. Enter  
   $foo.gettype()
   1. What datatype is $foo? System.ValueType
3. Change foo’s type. Enter:  
   $foo=’yyz’
   1. Use the gettype() method to see what type foo is. What datatype is $foo? System.Object
4. Enter:  
   $foo=get-childitem
   1. What datatype is $foo? System.Array
5. You can create a type constrained variable by prefixing the variable with the datatype you want to allow in the variable. Enter:  
   [int]$age=21
   1. What type is in $age? System.ValueType
6. Enter:  
   $age=65
   1. What type is in $age? System.ValueType
7. Enter :  
   $age='old'
   1. What was the result? Cannot convert value "old" to type "System.Int32". Error: "Input string was not in a correct format."  
      At line:1 char:1  
      + $age='old'  
      + ~~~~~~~~~~  
       + CategoryInfo : MetadataError: (:) [], ArgumentTransformationMetadataException  
       + FullyQualifiedErrorId : RuntimeException
   2. What is in $age? 65
   3. What type is in age? System.ValueType
8. Enter:  
   $age='35'
   1. What was the result? accepted
   2. What is in $age? 35
   3. What type is in age? System.ValueType
9. Enter:  
   [string]$age='old'
   1. What is in $age? old
   2. What type is in age? System.object
10. Enter:  
    $age=44
    1. What is in $age? 44
    2. What type is in age? System.Object
11. What did you learn about variable assignment and types when the variable is type constrained? The variable being restrained keeps it from being changed to an unwanted type

# Task 3—Attribute Constrained Variables

PowerShell also allows you to constrain what values are allowed in a variable. This is accomplished by setting attributes on the variable that describe the behavior we desire of the variable.

## Steps

1. Create an attribute constrained string variable that only allows strings that are from 0-5 characters long. Enter:  
   [ValidateLength(0,5)] [string]$color=''
2. Enter:  
   $color='red'
   1. What was the result? blank
3. Enter:  
   $color='orange'
   1. What was the result? error
4. We will learn more about declarative programming and attributes later in the course.

# Task 4—Variable Names

From the PowerShell help topic **about\_variables**.  
Variable names begin with a dollar ($) sign and can include alphanumeric characters and special characters. The variable name length is limited only by available memory.

The best practice is that variable names include only alphanumeric characters and the underscore (\_) character. Variable names that include spaces and other special characters, are difficult to use and should be avoided.

* Alphanumeric variable names can contain these characters:
* Unicode characters from these categories: Lu, Ll, Lt, Lm, Lo, or Nd.
* Underscore (\_) character.
* Question mark (?) character.

The following list contains the Unicode category descriptions. For more information, see UnicodeCategory.

* Lu - UppercaseLetter
* Ll - LowercaseLetter
* Lt - TitlecaseLetter
* Lm - ModifierLetter
* Lo - OtherLetter
* Nd - DecimalDigitNumber

To create or display a variable name that includes spaces or special characters, enclose the variable name with the curly braces ({}) characters. The curly braces direct PowerShell to interpret the variable name's characters as literals.

Special character variable names can contain these characters:

* Any Unicode character, with the following exceptions:
* The closing curly brace (}) character (U+007D).
* The backtick (`) character (U+0060). The backtick is used to escape Unicode characters so they're treated as literals.

PowerShell has reserved variables such as $$, $?, $^, and $\_ that contain alphanumeric and special characters. For more information, see **about\_Automatic\_Variables**.

## Steps

1. Create a standard variable name:  
   $Sensor\_1=42
2. Create a variable with special characters:  
   ${Value at x,y with 90% Accuracy}=8675309
3. Access the variable  
   ${Value at x,y with 90% Accuracy}
4. **Windows Only.** You can use a path in the filesystem as a variable name. If a variable name is a path then the value is stored in that file. For this to work you will need to know the full path to your psfiles folder. Mine is d:\psfiles. You will need to adjust the commands to your paths.
   1. Here is an example.  
      **${d:\psfiles\data.dat}=42**
   2. You should now have a file in your psfiles directory named data.dat. You can verify with PowerShell   
      **dir d:\psfiles**
   3. You can also read the file with PowerShell  
      **Get-Content D:\psfiles\data.dat**
   4. Finally use your favorite text editor to edit the data.dat file to have something else in it. Then access the variable  
      **${d:\psfiles\data.dat}**
   5. What was the result? Something Else

# Task 5—Variable Cmdlets

PowerShell has several variable cmdlets to work with variables. They are Clear-Variable, Get-Variable, New-Variable, Remove-Variable, Set-Variable.

## Steps

1. Typically, you create variables with the assignment operator =. If you need special behavior for the variable you can create it with New-Variable or Set-Variable.
   1. Create a ReadOnly Variable. Enter:  
      New-Variable -name Pi -Value 3.1415927 -Description 'Pi to 7 digits precision' -Option ReadOnly
   2. Retrieve the value of Pi:  
      **$pi**
   3. Try to change pi:  
      **$pi=3**
      1. What was the result? Cannot overwrite
2. Clearing the contents of a variable with clear Variable.
   1. Create a variable:  
      $stone='ruby'
   2. View the contents of the variable:  
      $stone
   3. Clear the variable.  
      Clear-Variable stone
      1. What is the value of stone? null
      2. Does stone exist? yes
      3. What expression did you use to determine if it existed? Get-Childitem variable:
   4. Remove the variable  
      Remove-Variable stone
      1. What is the value of stone? null
      2. Does stone exist? no
      3. What expression did you use to determine if it existed? Get-Childitem variable:

# Task 6—Reference Types

Variables can be of two types, value type and reference type. Most variables are value types in PowerShell. Reference type variables do not contain the variable’s value, they contain the address of where the value is. Thus, if two variables have the same address in them, changing the value of one will be reflected in the other.

## Steps

1. See the effects of a value type variables.
   1. Create a variable  
      $a=1
   2. Create a second variable from the first variable  
      $b=$a
   3. What is the value of **$a** 1 and **$b** 1
   4. Now change variable b’s value  
      $b=3
   5. Did that change the value of $a? no
2. Now see how this works with reference types.
   1. Create the variable c  
      $c=10
   2. Now create a variable d that references c  
      $d=Get-Variable c
   3. Now set $d’s value  
      $d.value=100
   4. What happened to c? changed to 100
   5. Change the value of c to 1000  
      $c=1000
   6. What is the value of d? Name Value  
      ---- -----  
      c 1000

# Deliverable

Upload this document with completed answers to i-learn.