Advanced Scripting   
Variables

Last Updated: 3/13/2024 9:09 AM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

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

# Overview

This exercise presents a few practice drills involving PowerShell variables.

# Setup

## 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?Click or tap here to enter text.
2. Try this:  
   $null -eq $foo
   1. What was the result? Click or tap here to enter text.
   2. What can you infer about unassigned variables? Click or tap here to enter text.
3. To see all the variables defined in your PowerShell instance, enter the command:  
   Get-ChildItem variable:   
   What is the value of the **PROFILE** variable? Click or tap here to enter text.
4. Use **Test-Path** to see if a variable exists. To test whether the variable pwd exists, enter:  
   Test-Path Variable:pwd   
   What is the result? Click or tap here to enter text.
   1. Does the variable **foo** exist? Click or tap here to enter text.
   2. Does the variable **shell** exist? Click or tap here to enter text.
5. Create the variable **foo**. Enter:  
   $foo=1
6. Get the value of **foo**:  
   $foo
   1. What is the value of **foo**? Click or tap here to enter text.
7. Use Test-Path to see if **foo** exists. Your result: Click or tap here to enter text.

# 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:   
   $foo.gettype().name
   1. What data type is $foo? Click or tap here to enter text.
3. Change foo’s type. Enter:  
   $foo='yyz'   
   Then use the **GetType()** method to see what type foo is. What datatype is **$foo**? Click or tap here to enter text.
4. Enter:  
   $foo=get-childitem
   1. What datatype is $foo? Click or tap here to enter text.
   2. Notice the square brackets at the end of the type name. What do the square brackets indicate? Click or tap here to enter text.
5. You can create a *type-constrained* (a.k.a. *strongly typed*) variable by prefixing the variable with the datatype you want to allow in the variable. Enter:  
   [int]$age=21   
   What type is in **$age**? Click or tap here to enter text.
   1. Enter:  
      $age=65   
      What type is in **$age**? Click or tap here to enter text.
   2. Enter:  
      $age='old'   
      What was the result? Copy the error message here: Click or tap here to enter text.
      1. What is in **$age**? Click or tap here to enter text.
      2. What type is in **$age**? Click or tap here to enter text.
   3. Enter:  
      $age='35'
      1. What was the result? Click or tap here to enter text.
      2. What is in **$age**? Click or tap here to enter text.
      3. What type is in **$age**? Click or tap here to enter text.
6. Enter:  
   [string]$age='old'
   1. What is in **$age**? Click or tap here to enter text.
   2. What type is in **$age**? Click or tap here to enter text.
7. Enter:  
   $age=44
   1. What is in **$age**? Click or tap here to enter text.
   2. What type is in **$age**? Click or tap here to enter text.
8. In your own words, what did you learn about variable assignment and types when the variable is type-constrained? Click or tap here to enter text.

# Task 3—Attribute Constrained Variables

PowerShell also allows you to further constrain the values 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'   
   What was the result? Click or tap here to enter text.
3. Enter:  
   $color='orange'   
   What was the result? Click or tap here to enter text.  
   *We will learn more about declarative programming and attributes in later exercises.*

# Task 4—Variable Names

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

A *best practice* for 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 *curly brace* ( **{}** ) characters. The curly braces direct PowerShell to interpret the variable name's characters as literals.

Special character variable names can contain any Unicode character, with these 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 many reserved variables such as **$$**, **$?**, **$^**, and **$\_**. For more information, see the **about\_Automatic\_Variables** help topic.

## Steps

1. Create a standard (best practice) variable name:  
   $Sensor\_1=42
2. Create a variable with special characters:  
   ${Value at x,y with 90% Accuracy}=8675309
3. Access the variable  
   ${Value at x,y with 90% Accuracy}
4. ***Windows Only*:** You can use a path in the filesystem as a variable name. If a variable name is a path then the value is stored in that *file*.
   1. Here is an example.  
      **${c:\temp\data.dat}=42**
   2. You should now have a file in your temp directory named data.dat. You can verify with PowerShell.   
      **dir c:\temp**
   3. You can also read the file with PowerShell.   
      **Get-Content c:\temp\data.dat**
   4. Finally use your favorite text editor to edit the **data.dat** file to have something else in it, such as the words **forty two** instead of the numeral **42**. After you have altered the file’s contents, access the variable:  
      **${c:\temp\data.dat}**
   5. What was the result? Click or tap here to enter text.

# 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 **=**, but if you need special behavior for the variable you can create it with New-Variable or Set-Variable.
   1. Create a *ReadOnly* variable. Enter (all on one line):  
      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**   
      What was the result? Click or tap here to enter text.
2. Clear 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, then get a list of all of the variables:   
      Clear-Variable Stone   
      Get-Variable
      1. Does Stone still exist? Click or tap here to enter text.
      2. What is the value of Stone? Click or tap here to enter text.
   4. Remove the variable:   
      Remove-Variable Stone
      1. Now what is the value of stone? Click or tap here to enter text.
      2. Does stone still exist? Click or tap here to enter text.
      3. What command line did you use to determine whether it existed? Click or tap here to enter text.

# Task 6—Reference Types

Variables can be of two types, value type and reference type. Most primitive variables are value types in PowerShell. Reference type variables do not contain the variable’s value; they contain a memory 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. Let’s see the effects of 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 **$a**? Click or tap here to enter text.
   4. Of **$b**? Click or tap here to enter text.
   5. Now change variable **b**’s value:  
      $b=3
   6. Did that change the value in **$a**? Click or tap here to enter text.
2. Now let’s see how this works with reference types.
   1. Create a variable named **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**? Click or tap here to enter text.
   5. Change the value of **c** to 1000  
      $c=1000
   6. Now what is the value of **d**? Click or tap here to enter text.

# Deliverable

Upload this document with completed answers to I-Learn Canvas.