Functions, Parameters, User Input, Providers, Modules Summer 2022

Powershell

Script Parameters

- Scripts can have parameters
- Use the param statement as the first line in your script to add parameters to your script
- The param statement adds variables to your script with the variable names being the parameter names
- The value of the parameter variables comes from the command line when the user enters those parameters
  param ($MyParameter, $AnotherParameter)
- Multiple parameters are separated by commas, and each parameter can have a type, default value, and additional attributes

Function Parameters

- Parameters can be specified for functions using the param statement at the start of a function definition
  e.g.

```
function myfunc {
    param ([int]$myinteger = 1,
           [string[]]$mystrings,
           $something)
    ...
}
```

- Types and default values can be specified when defining parameters
- The special type switch is used for parameters to indicate they do not require an object on the command line

Parameter Attributes

- Parameters can have attributes specified using the [parameter()] declaration immediately preceding the name of the parameter, this makes them into "Advanced Functions" and means you can use the common parameters with them
  e.g.

```
function myfunc {
    param ([parameter(Mandatory=$true,
                      Position=0,
                      ParameterSetName="MySet1",
                      ValueFromPipeline=$true,
                      ValueFromPipelineByPropertyName=$true,
                      HelpMessage="The MyParameter parameter can have any value you like",
                      Alias=("mp","MyParameter"))]
           $MyParameter)
    ...
}
```

- Various validation attributes are available to further enforce parameter rules, see help about_functions_advanced

Parameter Attribute Table

- The attribute table, found in cmdlet help, is a detailed listing of the parameters a cmdlet will accept and how to specify them in a command
- help -full or -parameter or -online will show the parameter attribute table
- e.g

```
-path <string[]>
    Specifies a path of one or more locations. Wildcard characters are
    permitted. The default location is the current directory (.).

Required?                    false
Position?                    1
Default value                Current directory
Accept pipeline input?       true (ByValue, ByPropertyName)
Accept wildcard characters?  true
```

Common Parameters

- Cmdlets support common parameters which allow generic specification of common options
- Common parameters can be added to your scripts automatically by adding CmdletBinding() to your script or function before the param statement
- verbose (vb - $verbosepreference, used with write-verbose)
  debug (db - $debugpreference, used with write-debug)
- warningaction (wa - $warningactionpreference)
  erroraction (ea - $erroractionpreference)
  warningvariable (wv)
  errorvariable (ev)
  outvariable (ov)
- whatif (wi)
  confirm (cf)
- For more common parameters or more detail, refer to help about_commonparameters

Parameter Exercises

- Save the example below to a file
- Try running the command without any parameters, then with one of the parameters, then with both
- Try running the command without putting in the parameter names

```
Param ([Parameter(Mandatory=$true,position=1)][string]$SourceFile,
       [Parameter(Mandatory=$true,position=2)][string]$DestinationFile )
"SourceFile was '$sourcefile'"
$objtypename = $sourcefile.gettype().name
"SourceFile was a $objtypename object"
"DestinationFile was '$destinationfile'"
$objtypename = $destinationfile.gettype().name
"DestinationFile was a $objtypename object"
```

Working With Files Example

- This example will show a gridview listing of large document files in a folder specified by the user, having a minimum size specified by the user, with both parameters available to use from the command line
- It shows the script having 2 parameters, and a function with one parameter

FILE: bigdocs.ps1

```
param ([String]$Path=".", [long]$MinimumSize = 0)
function Get-Docs ([string]$DocsPath=".") {
    Get-ChildItem -Path $DocsPath `
                  -Include *.txt,*.doc,*.docx,*.pdf,*.xls,*.ppt,*.ps1 `
                  -Recurse `
                  -ErrorAction SilentlyContinue
}
Get-Docs -DocsPath $path |
    Where-Object { $_.length -ge $minimumsize } |
    Select-Object FullName, LastAccessTime, Length |
    Sort-Object -Descending Length |
    Out-GridView
```

User Input

- User input can be obtained using the read-host cmdlet
  e.g.
```
$UserInput = read-host
[int]$Num = read-host -prompt "Give me a number "
$pass = read-host -prompt "Password: " -AsSecureString
```

- SecureString objects are designed to be used as part of credentials objects
  Extracting the original string from a securestring:
  (New-Object PSCredential -ArgumentList "someusername",securestringvariable).GetNetworkCredential().Password

- get-content will get data from a file as a generic object array

User Input Exercise

- Create a rolldice script
- Accept a count of dice, and a number of sides per die
- Default to 6 sides and 2 dice
- Use get-random to generate the random numbers
- Ask the user for the numbers if they don't give them on the command line, preserving the defaults

Modules

- A module is at the minimum a collection of functions stored in a file with a .psm1 extension
- If you put the module file in $env:HOMEPATH/Documents/WindowsPowerShell/Modules/ModuleFileNameWithoutExtension/ ($PSModulePath), it will be automatically imported
- get-module -listavailable can be used to show modules not yet imported
- get-command -module modulename can be used to see what commands are in a module
- remove-module modulename can be used to remove a module from memory (e.g. you update the module file and want it to be imported again)
- Beware of name conflicts when creating modules, use common verbs whenever possible
- See https://msdn.microsoft.com/en-us/library/dd878340(v=vs.85).aspx for more information on creating modules

Providers

- Providers allow us to use drive name semantics to access different types of storage spaces
- get-psprovider shows the list of providers currently loaded in memory
- get-psdrive shows the list of drives using currently loaded providers with some summary information, very limited compared to WMI classes
- get-childitem (ls) can be used to view what is accessible via the providers by using the provider name as a drive name (e.g ls env: variable: alias: function:)
- Creating items of the types stored by these providers automatically stores them in that provider's storage

Lab 5 - Parameters and Modules