Creating Advanced PowerShell Functions: Step-by-Step Guide

This article demonstrates how to create and enhance PowerShell functions with various features, including parameter handling, validation, help documentation, and advanced options like WhatIf, Confirm, and ValidateSet.

---

1. Simple Function

A basic PowerShell function outputs the major version of PowerShell installed on your system.

```
function Get-MLOVersion {
   $PSVersionTable.PSVersion.Major
}
```

- **Explanation**: This function simply returns the major version of PowerShell.

---

2. Very Simple Function

A basic function outputs a welcome message.

```
Function Write-Welcome {
   Write-Host "Welcome Mr. Raymond"
}
```

- **Explanation**: The message is hardcoded.

---

3. Function with Parameters

A function with a parameter that customizes the output based on input.

```
Function Write-Welcome {
   param ($name)
   Write-Host "Welcome Mr. $Name"
}
(Get-Command -Name Write-Welcome).Parameters.Keys
Get-Command -Name Write-Welcome -Syntax
```

- **Explanation**:
    - The param keyword allows input for the $name variable.
    - Command outputs all parameter keys and syntax of the function.

---

4. Advanced Function

Using [CmdletBinding()] enables cmdlet-like behavior.

```
Function Welcome {
   [CmdletBinding()]
   param ($name)
   Write-Host "Welcome Mr. $Name"
}
(Get-Command -Name Welcome).Parameters.Keys
```

- **Explanation**: [CmdletBinding()] adds features like WhatIf and Confirm.

---

5. WhatIf and Confirm Support

Enable WhatIf and Confirm for safe execution.

```
Function Welcome {
   [CmdletBinding(SupportsShouldProcess)]
   param ($name)
   Write-Host "Welcome Mr. $Name"
}
(Get-Command -Name Welcome).Parameters.Keys
```

- **Explanation**: These switches allow testing and confirming operations without executing them.

---

6. Mandatory Parameter

Declare a parameter as required.

```
Function Write-Welcome {
   [CmdletBinding(SupportsShouldProcess)]
   param (
```

```
    [Parameter(Mandatory)]
    $name
  )
  Write-Host "Welcome Mr. $Name"
}
(Get-Command -Name Write-Welcome).Parameters.Keys
```

- **Explanation**: Mandatory enforces input for the $name parameter.

---

7. Default Parameter Value

Set a default value for a parameter.

```
Function Write-Welcome {
  [CmdletBinding(SupportsShouldProcess)]
  param (
    [ValidateNotNullOrEmpty()]
    [string[]]$name = "Raymond"
  )
  Write-Host "Welcome Mr. $Name"
}
```

- **Explanation**: If no input is provided, "Raymond" is used.

---

8. Verbose Output

Add verbose messaging for better traceability.

```
Function Write-Welcome {
  [CmdletBinding(SupportsShouldProcess)]
  param (
    [ValidateNotNullOrEmpty()]
    [string[]]$name = "Raymond"
  )
  Write-Verbose -Message "Welcoming Our Guest"
```

```
    Write-Host "Welcome Mr. $Name"
}
```

- **Explanation**: Use -Verbose to display additional details.

---

9. Adding Help Documentation

Provide detailed help for the function.

```
Function Write-Welcome {
<#
.SYNOPSIS
 Welcomes User
.DESCRIPTION
 Write-Welcome is a function that welcomes users.
.PARAMETER Name
 Name of the user.
.EXAMPLE
 Write-Welcome -Name "Raymond"
#>
    [CmdletBinding(SupportsShouldProcess)]
    param (
        [ValidateNotNullOrEmpty()]
        [string[]]$name = "Raymond"
    )
    Write-Verbose -Message "Welcoming Our Guest"
    Write-Host "Welcome Mr $Name"
}
```

- **Explanation**: Users can view help with Get-Help Write-Welcome.

---

10. Multiple Parameters

Accept multiple parameters for enhanced functionality.

```
Function Write-Welcome {
   [CmdletBinding(SupportsShouldProcess)]
   param (
      [ValidateNotNullOrEmpty()]
      [string[]]$Name = "Raymond",
      [ValidateNotNullOrEmpty()]
      [string[]]$Place = "India"
   )
   Write-Verbose -Message "Welcoming Our Guest"
   Write-Host "Welcome Mr. $Name to $Place"
}
```

- **Explanation**: Handles both name and place inputs.

---

11. Parameter Validation with Predefined Values

Restrict parameter values to predefined options.

```
Function Write-Welcome {
   [CmdletBinding(SupportsShouldProcess)]
   param (
      [ValidateSet("Services", "Process", "Events")]
      [string[]]$Item = ("Services", "Process", "Events")
   )
   Switch ($Item) {
      "Services" { Get-Service | Select -First 5 }
      "Process" { Get-Process | Select -First 5 }
      "Events" { Get-EventLog -LogName Application | Select -First 5 }
   }
}
```

- **Explanation**: Only accepts "Services," "Process," or "Events" as valid inputs.

---

12. Handling Single and Multiple Inputs

Allow single or multiple values for a parameter.

```
Function Write-Welcome {
   [CmdletBinding(SupportsShouldProcess)]
   param (
      [Parameter(Mandatory)]
      [string]$name
   )
   Write-Host "Welcome Mr $Name"
}
Write-Welcome -Name ("John", "Doe")
Write-Welcome -Name "Alice"
```

- **Explanation**: Handles both singular and array inputs.

By following these steps, you can create versatile PowerShell functions tailored to a wide range of scenarios.