Functions in PowerShell

Functions in PowerShell are reusable blocks of code designed to perform specific tasks. They help make scripts modular, easier to read, and maintain. Here's a quick overview of how to create and use them:

1. Basic Syntax

A function is defined using the `function` keyword, followed by a name and a script block:

```
function FunctionName {
    # Code to execute
    Write-Output "Hello, PowerShell!"
}
```

You can call the function by its name:

```
FunctionName
```

2. Adding Parameters

Parameters allow you to pass data into the function:

```
function Greet-User {
    param (
        [string]$Name
    )
    Write-Output "Hello, $Name!"
}

# Call the function with a parameter
Greet-User -Name "Alice"
```

3. Return Values

Functions can return values using `return` or simply outputting data:

```
function Add-Numbers {
    param (
        [int]$a,
        [int]$b
    )
    return $a + $b
}

# Call the function and store the result
$result = Add-Numbers -a 5 -b 10
Write-Output $result  # Output: 15
```

4. Advanced Features

- **Parameter Validation**: Ensure inputs meet specific criteria.

```
function Set-Age {
    param (
        [ValidateRange(1, 120)]
        [int]$Age
    )
    Write-Output "Your age is $Age."
}

Set-Age -Age 25  # Valid
# Set-Age -Age 150  # Throws an error
```

- **Pipeline Input**: Allow functions to accept input from the pipeline.

```
function Get-Square {
    param (
        [Parameter(ValueFromPipeline)]
        [int]$Number
    )
    process {
        $Number * $Number
    }
}

1..5 | Get-Square  # Outputs: 1, 4, 9, 16, 25
```

5. Best Practices

- Use **descriptive names** for functions (e.g., `Get-Data`, `Set-Config`).
- Follow the **Verb-Noun** naming convention.
- Include **comments** to explain complex logic.
- Use **error handling** (e.g., `try/catch`) for robustness.

Functions are a powerful way to organize and reuse code in PowerShell, making your scripts more efficient and maintainable!