# PowerShell Functions: Pipelines, Parameters, and Templates

Download the latest version of this PowerShell™ wallchart and see the accompanying in-depth article ***Down the Rabbit Hole: A Study in PowerShell Pipelines, Functions, and Parameters*** from Simple-Talk at http://bit.ly/pIl44i

## Function Structure

Sub-blocks (begin, process, and end) support pipeline processing with pre- and post-processing components.

There are two abbreviated formats:

- All code in the **end** block is equivalent to the quintessential, simple **function**.
- All code in the **process** block can be simplified with the **filter** keyword.

**Standard Structure**

```
function <name> ( { <param> } )
{
  begin   { <statement list> }
  process { <statement list> }
  end     { <statement list> }
}
```

| Special Case | Implicit Format |
|---|---|
| `function <name> ( { <param> } ) { end { <statement list> } }` | `function <name> ( { <param> } ) { <statement list> }` |
| `function <name> ( { <param> } ) { process { <statement list> } }` | `filter <name> ( { <param> } ) { <statement list> }` |

## Syntax Pitfalls

There are 2 **right ways** to call a function and several wrong ways; watch out for **those that fail silently**!

| Calling Syntax | $a | $b | $c |
|---|---|---|---|
| **f(1,2,3)** | 1,2,3 | | |
| **f(1,2,3) [StrictMode]** | Syntax error–called like a method | | |
| **f (1,2,3)** | 1,2,3 | | |
| **f 1,2,3** | 1,2,3 | | |
| **f (1 2 3)** | Syntax error–unexpected token | | |
| **f 1 2 3** | 1 | 2 | 3 |
| **f –c 3 –a 1 –b 2** | 1 | 2 | 3 |

## References

Learning PowerShell, about_Functions, about_Functions_Advanced, about_Parameters, about_Functions_Advanced_Parameters, about_Pipelines, PowerShell gotchas, $input gotchas, Parentheses in PowerShell.

## Templates

When you send data into a function, what arrives depends on your structure and your invocation. For each template, the table below shows what arrives for all input combinations. The results are color-coded to green (pass) and red (fail) against these expectations: (a) "no input" should yield a default; and (b) a direct or pipeline input should reflexively yield back the input.

Group C obviously is the best choice—*but **only** if you need to satisfy **all** those expectations*. Group A templates may suffice if you need to handle either direct input or pipeline input *but not both*. Group B templates may suffice if you also are concerned with a default value.

| Direct Input Commands | Pipeline Input Commands |
|---|---|
| TestFunction | @() \| TestFunction |
| TestFunction $null | $null \| TestFunction |
| TestFunction "" | "" \| TestFunction |
| TestFunction "one" | "one" \| TestFunction |
| TestFunction "one","two" | "one","two" \| TestFunction |
| TestFunction "one",$null,"two","" | "one",$null,"two","" \| TestFunction |

Results of executing input class (*right*) against template (*below*)

### Group A

```
function Template-A_1_Basic($item)
{
    $item | % { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| $null | $null | "" | one | one,two | one, $null,two,"" |
| $null | $null | $null | $null | $null | $null |

```
function Template-A_2_Process($item)
{
    Process { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| $null | $null | $null | $null | $null | $null |
| $null | $null | "" | one | one,two | one, $null,two,"" |

```
function Template-A_3_Input($item)
{
    $input | % { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| $null | $null | $null | $null | $null | $null |
| $null | $null | "" | one | one,two | one, $null,two,"" |

### Group B

```
function Template-B_1_BasicDefault(
  [array]$item = "default")
{
    $item | % { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| default | $null | "" | one | one,two | one, $null,two,"" |
| default | default | default | default | default | default |

```
function Template-B_2_ProcessDefault(
  [array]$item = "default")
{
    Process { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| $null | $null | $null | $null | $null | $null |
| $null | $null | "" | one | one,two | one, $null,two,"" |

```
function Template-B_3_InputDefault(
  [array]$item = "default")
{
    $input | % { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| $null | $null | $null | $null | $null | $null |
| $null | $null | "" | one | one,two | one, $null,two,"" |

### Group C

```
function Template-C_1_ProcessDefault_A(
  [array]$item = "default")
{
    Process {
        if ($_) { $item = $_ }
        $item | % { return $_ }
    }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| default | $null | "" | one | one,two | one, $null,two,"" |
| $null | default | default | one | one,two | one,one,two,two |

Without properly indicating pipeline input (via ValueFromPipeline), you can add a kludge as shown to force pipeline input into $item. The kludge, however, fails to distinguish different types of input that evaluate to false, leading to the curious result for the last test!

```
function Template-C_2_ProcessDefault_B(
  [Parameter(ValueFromPipeline=$True)]
  [array]$item = "default")
{
    Process {
        $item | % { return $_ }
    }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| default | $null | "" | one | one,two | one, $null,two,"" |
| $null | $null | "" | one | one,two | one,$null,two,"" |

You can simplify this template by converting the function keyword to filter and then remove the explicit process block. Depending on what you deem correct for the standout test case, this template wins in elegance of coding compared to C_3/C_4.

```
function Template-C_3_InputDefault_A(
  [Parameter(ValueFromPipeline=$True)]
  [array]$item = "default")
{
  $list = @($input)
  if ($list.count) { $item = $list }
  if (!(test-path variable:\item))
     { $item = "default" }
  $item | % { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| default | $null | "" | one | one,two | one,null,two,"" |
| default | $null | "" | one | one,two | one,null,two,"" |

C_3 and C_4 yield identical results, though C_3 is technically more correct by indicating pipeline input (via ValueFromPipeline). However, that requires adding more code to prevent a runtime error on one test case. Strange but true: the function's own $item parameter is undefined if an empty pipeline feeds the function!

```
function Template-C_4_InputDefault_B(
  [array]$item = "default")
{
    $list = @($input)
    if ($list.count) { $item = $list }
    $item | % { return $_ }
}
```

| No Input | Null ($null) | Empty String ("") | Scalar (one) | List (one, two) | List with Null/Empty (one, $null, two, "") |
|---|---|---|---|---|---|
| default | $null | "" | one | one,two | one,null,two,"" |
| default | $null | "" | one | one,two | one,null,two,"" |

Both C_3 and C_4 have elegance in results—the outputs from direct input exactly mirror those from pipeline input.