How-to: PowerShell Functions and Filters

A block of code may be contained within a function for easy re-use.
To create a function, call the **function** keyword followed by a name for the function, then include your code inside a pair of curly braces.

```
function Add-Numbers {
 $args[0] + $args[1]
}
```

```
PS C:\> Add-Numbers 5 10
15
```

A similar function with named parameters:

```
function Output-SalesTax {
 param( [int]$Price, [int]$Tax )
 $Price + $Tax
}
```

```
PS C:\> Output-SalesTax -price 1000 -tax 38
1038
```

To display the definition of a function several methods can be used:

`cat function:Add-Numbers`
or
`${function:Add-Numbers}`
or
`(get-command Add-Numbers).definition`

To list all functions in the current session: `get-command -CommandType function`

N.B in a block of code you need to define the function **before** you call it.

Don't add brackets around the function parameters:

```
$result = Add-Numbers (5, 10) --Wrong!
$result = Add-Numbers 5 10    --Right
```

A function will automatically accept positional parameters in the order they are declared, or you can pass named parameters in any order. This also works when calling via an alias.

How to run a function

The most common way to use a function is to include it within your PowerShell script. The function must be defined *before* any calls to it, so this is a valid script:

```
function Add-Numbers {
 $args[0] + $args[1]
}
Add-Numbers 5 10
```

For small functions and/or when testing it is also possible to type (or copy and paste) an entire function at the PowerShell command line. This is not really practical for longer scripts because it fills up much of your command history.

PowerShell does not save functions or filters permanently by default. So if you close and reopen PowerShell, the function/filter will no longer be available. To make a function permanently available, add it to your PowerShell $Profile it will then be loaded for every new PowerShell session and can be called from multiple different scripts without having to be explicitly included.

Dot sourcing a script is very similar to using the PowerShell $Profile but rather than the always loaded $Profile, you save the function in a script of your choosing and then load it (by dot sourcing) only when it is needed:

```
. C:\Scripts\Tools.ps1
Add-Numbers 5 10
```

Another method of making a function available to multiple scripts/sessions is to include it as part of a PowerShell **Module**. When saved in either the system Module folder or the user Module folder, automatic cmdlet discovery (in PowerShell v3 and above) will then ensure that those functions are available.

When you call the function name, the code within the function will run, A function can accept imported values either as arguments or through the pipeline. If the function returns any values, they can be assigned to variables or passed to other functions or cmdlets.

Choose a good function name

The built-in PowerShell alias names will take precedence over a function (or script) with the same name. To avoid conflicts always check if the name is already in use with `help name`. Choosing a good descriptive name for your function will help with this

A best practice is to use choose a name in **Verb-Noun** format, by virtue of containing a '-' character, this will not conflict with any built in Aliases. You can generate a list of the approved PowerShell verbs by running the cmdlet `Get-Verb`

Function or Filter definition:

```
function [scope_type:]name
 {
  [ param(param_list) ]
   script_block
 }

filter [scope_type:]name
 {
  [ param(param_list) ]
  script_block
 }
```

The difference between a filter function and a regular function is the way they handle items passed through the pipeline:

With a regular function, pipeline objects are bound to the `$input` automatic variable, and execution is blocked until all input is received. The function then begins processing the data.

With a filter function, data is processes while it is being received, without waiting for all input. A filter receives each object from the pipeline through the `$_` automatic variable, and the script block is processed for each object.

The `param_list` is an optional list of comma separated parameter names, these may also be preceded by their data types in brackets. This makes the function more readable than using `$args` and also gives you the option to supply default values.

Advanced function

An Advanced PowerShell function contains the `[cmdletbinding()]` attribute (or the `Parameter` attribute). This adds several capabilities such as additional parameter checking, and the ability to easily use Write-Verbose.
A function with cmdletbinding will throw an error if unhandled parameter values appear on the command line.

Advanced PowerShell functions typically include Begin..Process..End blocks for processing the input data, documentation and auto-help, including the parameters.

Variable Scope

By default, all variables created in functions are local, they only exist within the function, though they are still visible if you call a second function from within the first one.

To persist a variable, so the function can be called repeatedly and the variable will retain it's last value, prepend `$script:` to the variable name, e.g. `$script:myvar`

To make a variable global prepend `$global:` to the variable name, e.g. `$global:myvar`

Example Functions

A function to find all files on the C: drive owned by a particular user:

```
function Get-ByOwner
{
  Get-ChildItem -recurse C:\ | get-acl | where {$_.Owner -match $args[0]}
}
```

```
PS C:\> Get-ByOwner JackFrost
```

A function to display PowerShell help on SS64.com, this will launch your default web browser:

```
function Get-Help2
{
  param([string]$command)
  Start-process -filepath "https://ss64.com/ps/$command.html"
}
```

You can then do:
```
PS C:\> Get-Help2 -command set-acl
```

or because there is only one parameter:
```
PS C:\> Get-Help2 set-acl
```

A filter to display only files smaller than a given size:

```
filter FileSizeBelow($size) { if ($_.length -le $size) { $_ } }
```

```
PS C:\> gci \\server64\workgroups -filter | FileSizeBelow 200kb
PS C:\> gci -recurse \\server64\workgroups | ?{!$_.PSIsContainer}
| FileSizeBelow 100mb
```

A function with default values:

```
 function write-price
 {
  param([string]$description = "unknown",
        [int]$price = 100)
  Write-Output "$description ..... $price"
 }
```

```
PS C:\> write-price -price 250 -description Hammer
Hammer ..... 250
```

A filter to find files owned by a specific user:

```
 filter ownedbyme
 {
  if ($_.Owner -match "JackFrost") {$_}
 }
```

```
PS C:\> gci -recurse C:\ | Get-Acl | where {$_ | ownedbyme}
```

"The function of the imagination is not to make strange things settled, so much as to make settled things strange" ~ G. K. Chesterton

Related PowerShell Cmdlets:

Advanced functions - include cmdlet features, common parameters -verbose, -whatif, -confirm etc.
Assertions - Using a filter to *assert* certain prerequisite conditions.
New-Alias - Create a new (short) alias name for your function.
Begin..Process..End - Function Input Processing.
Scriptblock - A collection of statements.
Ref vars - Passing a reference variable to a function.