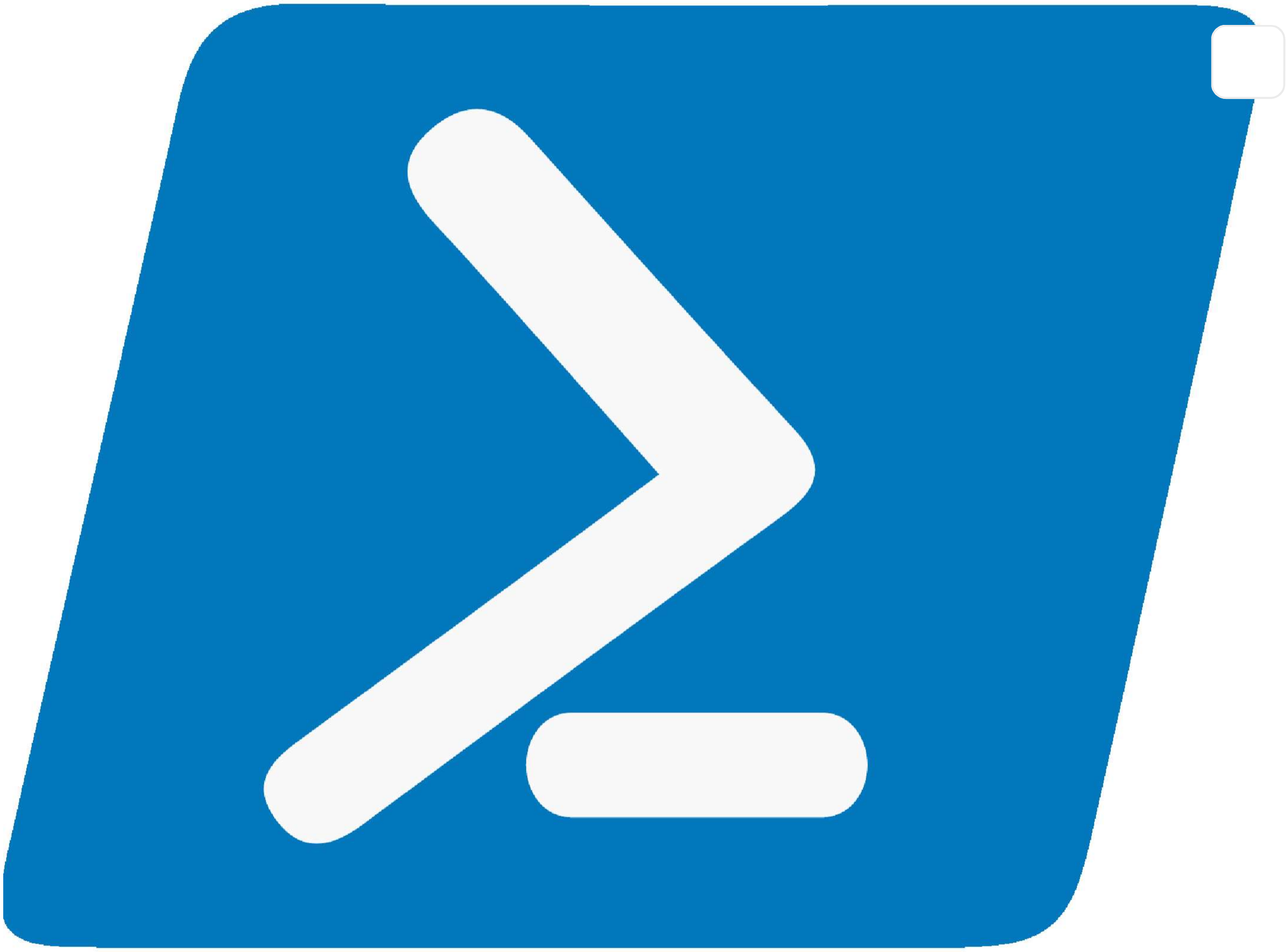
Intro to Object-Oriented Programming with PowerShell

In this post, I go over the fundamentals of object-oriented programming and some simple ways to interact with objects in PowerShell

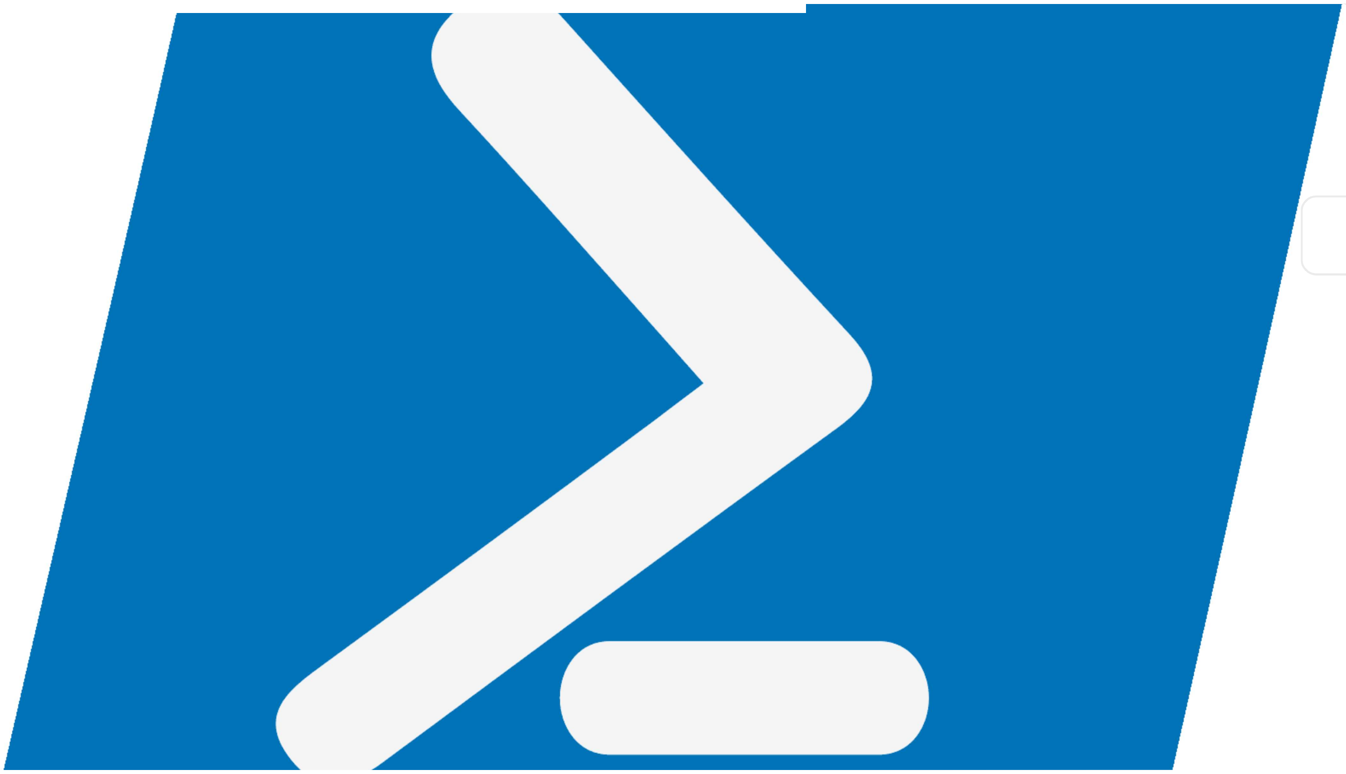
0xBEN
Jan 13, 2022 9 min read



In: PowerShell, PowerShell 101, Code

SHARE

In the previous post, we took a look at some fundamental concepts in programming – variables, data types, arrays, operators, functions, control flow logic, and loops. Knowing about these is a critical first step to becoming a better programmer.



An Intro to Programming with PowerShell

In this post, I go over the fundamentals of programming using PowerShell and demonstrate why it’s a great way to learn to code

0xBEN • • 0xBEN

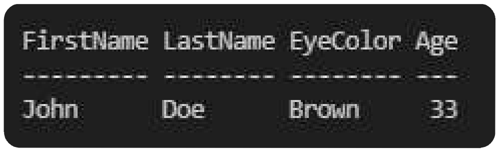
The next step up from there is to learn about **object-oriented programming (OOP)**. That's what *really* makes PowerShell a powerful shell, much more so than any Unix-style shell. Unix-style shells are text-based and are based on string input and output.

PowerShell deals with objects. When you create a script or function in PowerShell, **you should always ensure that you are outputting objects**. Any time you take input from the user, you should try to accept objects wherever applicable.

Classes and Objects

You can think of a `class` as a template for an `object`. Let me demonstrate this with some simple code.

```
class Person {
    [String]$FirstName
    [String]$LastName
    [String]$EyeColor
    [Int]$Age
}

$johnDoe = New-Object Person
$johnDoe.FirstName = 'John'
$johnDoe.LastName = 'Doe'
$johnDoe.EyeColor = 'Brown'
$johnDoe.Age = 33

# Print the object to output
$johnDoe
```



```
FirstName LastName EyeColor Age
--------- -------- -------- ---
John      Doe      Brown     33
```

I've created a `Person` class, so that any time I need to create an object about a `Person`, I can just use the template to build a new person. If I need to make changes, I just modify the template. Let's take a look at that.

```
class Person {
    [String]$FirstName
    [String]$LastName
    [String]$EyeColor
    [Int]$Age
    [String[]]$Nicknames
}
```

```
$johnDoe = New-Object Person
$johnDoe.FirstName = 'John'
$johnDoe.LastName = 'Doe'
$johnDoe.EyeColor = 'Brown'
$johnDoe.Age = 33
$johnDoe.Nicknames = @('johnny', 'jim', 'anonymous')

# Print the object to output
$johnDoe
```

I added the `[String[]]$Nicknames` property to the class. The `[String[]]` syntax – as opposed to `[String]` – indicates that this property will take **an array of strings**. I have populated it below on the `$johnDoe` object.

Object Properties and Methods

The most common parts of an object are its **properties** and **methods**. These are the **properties** of the `Person` class:

- `[String]$FirstName`
- `[String]$LastName`
- `[String]$EyeColor`
- `[Int]$Age`
- `[String[]]$Nicknames`

What is a Method?

A method is a function that is common to a particular type of class. In other words, **what is something that every person would do**? If you had a `Vehicle` class, what kinds of methods would you have?

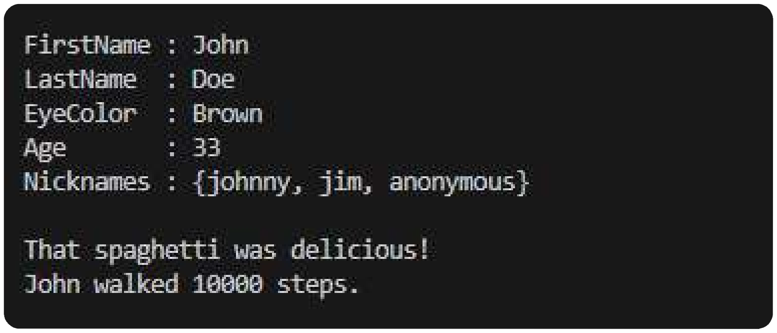
I am going to add some methods to my `Person` class.

- Eat
- Walk
- Sleep

These are things that every person would normally do.

```
class Person {
    [String]$FirstName
    [String]$LastName
    [String]$EyeColor
    [Int]$Age
    [String[]]$Nicknames

    [String] Eat([String]$Food) {
        return "That $Food was delicious!"
    }
    [String] Walk([Int]$Steps) {
        return $this.FirstName + " walked $Steps steps."
    }
    [Void] Sleep() {
        Start-Sleep -Seconds 10
    }
}

$johnDoe = New-Object Person
$johnDoe.FirstName = 'John'
$johnDoe.LastName = 'Doe'
$johnDoe.EyeColor = 'Brown'
$johnDoe.Age = 33
$johnDoe.Nicknames = @('johnny', 'jim', 'anonymous')

# Print the object to output
$johnDoe
```

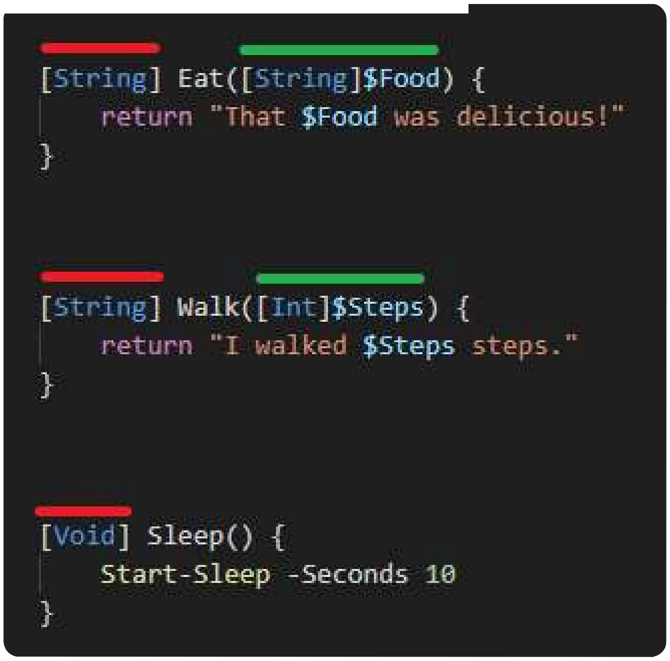
```
# Feed John Doe
# The Eat method takes a string
$johnDoe.Eat('spaghetti')

# John Doe needs a walk
# The Walk method take an integer
$johnDoe.Walk(10000)

# John Doe needs a nap
# The Sleep method does not take input
$johnDoe.Sleep()
```



```
FirstName : John
LastName  : Doe
EyeColor  : Brown
Age       : 33
Nicknames : {johnny, jim, anonymous}

That spaghetti was delicious!
John walked 10000 steps.
```

💡 The `$this` keyword shown in the `Walk` method tells the method to reference the `FirstName` property of itself when the method is called. It is dynamic, so what ever is stored in `FirstName`, `LastName`, `Age`, etc will always reflect the ***current value*** when the `$this` keyword is called.

Let's review the method structure.



```
[String] Eat([String]$Food) {
    return "That $Food was delicious!"
}


[String] Walk([Int]$Steps) {
    return "I walked $Steps steps."
}


[Void] Sleep() {
    Start-Sleep -Seconds 10
}
```

RED indicates the **kind of output** this method will return. **GREEN** indicates the **kind of input** this method will accept. The `Eat()` method accepts `[String]` input, whereas the `Walk()` method accepts `[Int]` input.

As you notice with the `Sleep()` method, it has a return type of `[Void]`, because there is no output returned from this method. Whereas with `Walk()` and `Eat()`, there is a return type of `[String]`.

Get-Member Cmdlet

The `Get-Member` cmdlet in PowerShell is very convenient for inspecting objects. With the `Get-Member` cmdlet, we can look at the various **properties and methods** of an object.

As I mentioned before, everything in PowerShell is an object – even files and directories. They all have properties and methods that can be inspected with the `Get-Member` cmdlet.

Let's inspect our `$johnDoe` object with the `Get-Member` cmdlet.

```
$johnDoe | Get-Member
```

```
   TypeName: Person

Name        MemberType Definition
----        ---------- ----------
Eat         Method     string Eat(string Food)
Equals      Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType     Method     type GetType()
Sleep       Method     void Sleep()
ToString    Method     string ToString()
Walk        Method     string Walk(int Steps)
Age         Property   int Age {get;set;}
EyeColor    Property   string EyeColor {get;set;}
FirstName   Property   string FirstName {get;set;}
LastName    Property   string LastName {get;set;}
Nicknames   Property   string[] Nicknames {get;set;}
```

The first thing you should notice is the `TypeName: Person` line. The `Get-Member` cmdlet tells us **what kind of object** we are dealing with. The next thing we can see is the object's **properties** and **methods**.

So, by looking at this, we can know that the `Person` class has the following:

Properties

- `Age`
- `EyeColor`
- `FirstName`
- `LastName`
- `Nicknames`

Methods

- `Eat`
- `Equals` *(automatically added by PowerShell)*
- `GetHashCode` *(automatically added by PowerShell)*
- `GetType` *(automatically added by PowerShell)*
- `Sleep`
- `ToString` *(automatically added by PowerShell)*
- `Walk`

Exploring with Get-Member

I am going to create a file on my desktop with this command: `New-Item -ItemType File -Name 'special-file.txt'`. Now, I'll store it in a variable: `$file = Get-Item ~/Desktop/special-file.txt`

```
$file | Get-Member
```

```
   TypeName: System.IO.FileInfo

Name                          MemberType     Definition
----                          ----------     ----------
LinkType                      CodeProperty   System.String LinkType{get=GetLinkType;}
Mode                          CodeProperty   System.String Mode{get=Mode;}
Target                        CodeProperty   System.Collections.Generic.IEnumerable`1[[System.Str
AppendText                    Method         System.IO.StreamWriter AppendText()
CopyTo                        Method         System.IO.FileInfo CopyTo(string destFileName), Syst
```

```
Create                    Method         System.IO.FileStream Create()
CreateObjRef              Method         System.Runtime.Remoting.ObjRef CreateObjRef(type req
CreateText                Method         System.IO.StreamWriter CreateText()
Decrypt                   Method         void Decrypt()
Delete                    Method         void Delete()
Encrypt                   Method         void Encrypt()
Equals                    Method         bool Equals(System.Object obj)
GetAccessControl          Method         System.Security.AccessControl.FileSecurity GetAccess(
GetHashCode               Method         int GetHashCode()
GetLifetimeService        Method         System.Object GetLifetimeService()
```

```
Open                      Method         System.IO.FileStream Open(System.IO.FileMode mode), S
OpenRead                  Method         System.IO.FileStream OpenRead()
OpenText                  Method         System.IO.StreamReader OpenText()
OpenWrite                 Method         System.IO.FileStream OpenWrite()
Refresh                   Method         void Refresh()
Replace                   Method         System.IO.FileInfo Replace(string destinationFileNam
SetAccessControl          Method         void SetAccessControl(System.Security.AccessControl.
ToString                  Method         string ToString()
PSChildName               NoteProperty   string PSChildName=special-file.txt
PSDrive                   NoteProperty   PSDriveInfo PSDrive=C
PSIsContainer             NoteProperty   bool PSIsContainer=False
PSParentPath              NoteProperty   string PSParentPath=Microsoft.PowerShell.Core\FileSy
PSPath                    NoteProperty   string PSPath=Microsoft.PowerShell.Core\FileSystem::(
PSProvider                NoteProperty   ProviderInfo PSProvider=Microsoft.PowerShell.Core\Fi
Attributes                Property       System.IO.FileAttributes Attributes {get;set;}
CreationTime              Property       datetime CreationTime {get;set;}
CreationTimeUtc           Property       datetime CreationTimeUtc {get;set;}
Directory                 Property       System.IO.DirectoryInfo Directory {get;}
DirectoryName             Property       string DirectoryName {get;}
Exists                    Property       bool Exists {get;}
Extension                 Property       string Extension {get;}
FullName                  Property       string FullName {get;}
IsReadOnly                Property       bool IsReadOnly {get;set;}
LastAccessTime            Property       datetime LastAccessTime {get;set;}
LastAccessTimeUtc         Property       datetime LastAccessTimeUtc {get;set;}
LastWriteTime             Property       datetime LastWriteTime {get;set;}
LastWriteTimeUtc          Property       datetime LastWriteTimeUtc {get;set;}
Length                    Property       long Length {get;}
Name                      Property       string Name {get;}
BaseName                  ScriptProperty System.Object BaseName {get=if ($this.Extension.Leng
VersionInfo               ScriptProperty System.Object VersionInfo {get=[System.Diagnostics.F
```

The first thing you should notice is the `TypeName: System.IO.FileInfo` line. This tells you the `class` that this file comes from. So, any file on your system inherits its properties and methods from its parent class.

As you can you see, the `System.IO.FileInfo` class has the following attributes:

- `CodeProperty`
- `Property`
- `NoteProperty`
- `ScriptProperty`
- `Method`

Let's try calling the `CreationTime` property from this object.

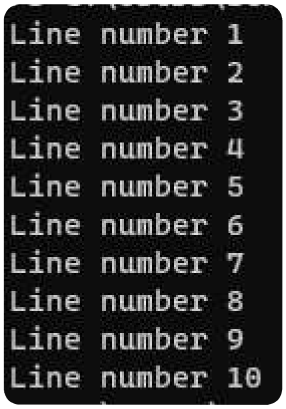
```
$file.CreationTime

Thursday, January 13, 2022 2:01:14 AM
```

Passing Objects Down the Pipeline

First, let me add some content to my test file.

```
foreach ($number in (1..10)) {
    "Line number $number" >> $file.FullName
}
```

```
Get-Content $file.FullName
```



```
Line number 1
Line number 2
Line number 3
Line number 4
Line number 5
Line number 6
Line number 7
Line number 8
Line number 9
Line number 10
```

Let's test passing objects down the pipeline.

```
$file | Get-Content

Line number 1
Line number 2
Line number 3
Line number 4
Line number 5
Line number 6
Line number 7
Line number 8
Line number 9
Line number 10
```

Why does that work? Because, the `Get-Content` cmdlet takes pipeline input and the pipeline input is expecting an object – specifically a file. This isn't a particularly impressive example of pipeline input and passing objects, but it is something that you should explore further.

NOTE: Not every cmdlet will accept pipeline input.

Selecting Object Properties

As you saw above, the `System.IO.FileInfo` class has a lot of properties that you can inspect. There are a couple of ways to select only certain properties that you wish to view.

With my `special-file.txt` file that is stored in the `$file` variable, I am particularly interested in viewing the `Length`, `CreationTime`, and `LastAccessTime` properties.

```
# One by one
$file.Length
304

$file.CreationTime
Thursday, January 13, 2022 2:01:14 AM

$file.LastAccessTime
Thursday, January 13, 2022 2:12:42 AM


# Together
$file | Select-Object Length, CreationTime, LastAccessTime

Length CreationTime          LastAccessTime
------ ------------          --------------
   304 1/13/2022 2:01:14 AM 1/13/2022 2:12:42 AM
```

Sorting Objects by Property

```
# Move to the Desktop folder
cd ~\Desktop

# Get all the files on the Desktop
$files = Get-ChildItem

# Sort files by name
$files | Sort-Object Name
```

Filtering Objects

Let's go back to the `Person` example from before using the custom class.

```
class Person {
    [String]$FirstName
    [String]$LastName
    [String]$EyeColor
    [Int]$Age
    [String[]]$Nicknames

    [String] Eat([String]$Food) {
        return "That $Food was delicious!"
    }
    [String] Walk([Int]$Steps) {
        return "I walked $Steps steps."
    }
    [Void] Sleep() {
        Start-Sleep -Seconds 10
    }
}

$johnDoe = New-Object Person
$johnDoe.FirstName = 'John'
$johnDoe.LastName = 'Doe'
$johnDoe.EyeColor = 'Brown'
$johnDoe.Age = 33
$johnDoe.Nicknames = @('johnny', 'jim', 'anonymous')
```

```
$janeDoe = New-Object Person
$janeDoe.FirstName = 'Jane'
$janeDoe.LastName = 'Doe'
$janeDoe.EyeColor = 'Brown'
$janeDoe.Age = 31
$janeDoe.Nicknames = @('janet', 'anonymous')

$johnSmith = New-Object Person
$johnSmith.FirstName = 'John'
$johnSmith.LastName = 'Smith'
$johnSmith.EyeColor = 'Blue'
$johnSmith.Age = 26
$johnSmith.Nicknames = @('JS', 'Bro')

$people = $johnDoe, $janeDoe, $johnSmith
```

So, now I have an array of people. Let's try filtering the objects using the `Where-Object` cmdlet.

```
# Select objects where the last name is Doe
$people | Where-Object {$_.LastName -eq 'Doe'}

FirstName : John
LastName  : Doe
EyeColor  : Brown
Age       : 33
Nicknames : {johnny, jim, anonymous}

FirstName : Jane
LastName  : Doe
EyeColor  : Brown
Age       : 31
Nicknames : {janet, anonymous}


# Select objects where the first name is John
$people | Where-Object {$_.FirstName -eq 'John'}

FirstName : John
LastName  : Doe
EyeColor  : Brown
Age       : 33
Nicknames : {johnny, jim, anonymous}

FirstName : John
LastName  : Smith
```

```
EyeColor  : Blue
Age       : 26
Nicknames : {JS, Bro}


# Select objects where the age is less than 30
$people | Where-Object {$_.Age -lt 30}

FirstName : John
LastName  : Smith
EyeColor  : Blue
Age       : 26
Nicknames : {JS, Bro}


# Select objects where the word 'Bro' occurs in the nicknames
$people | Where-Object {$_.Nicknames -contains 'Bro'}

FirstName : John
LastName  : Smith
EyeColor  : Blue
Age       : 26
Nicknames : {JS, Bro}
```

Final Project

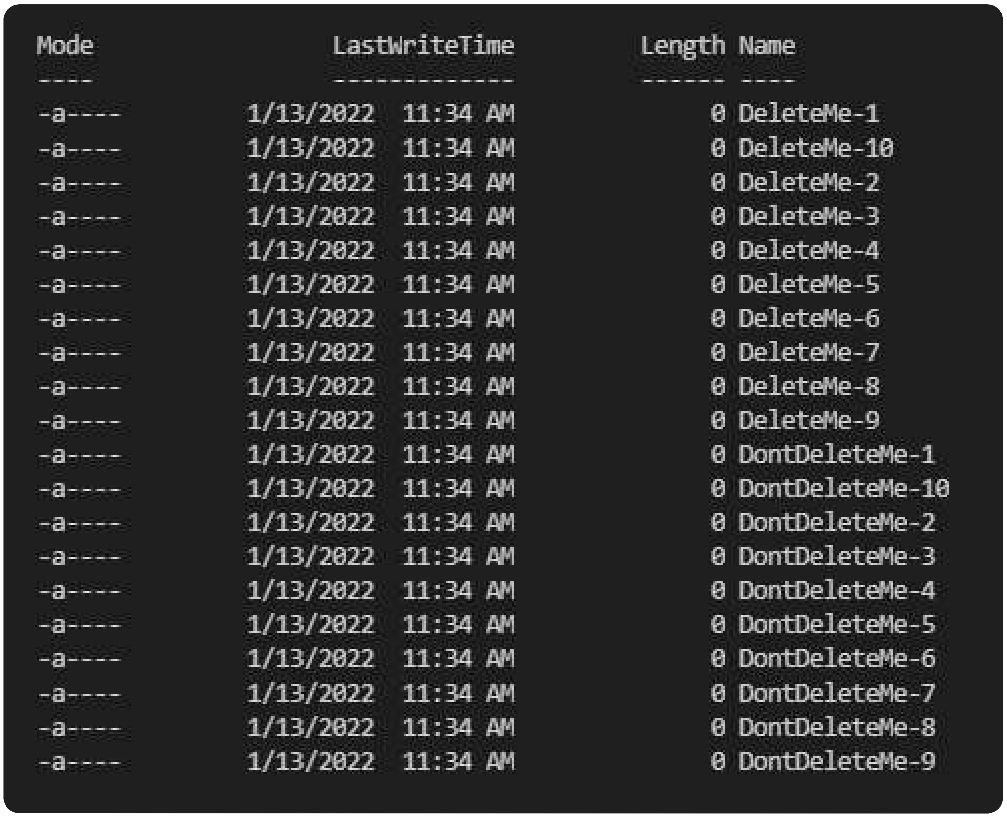
Project 1: Filter files on your desktop by passing them down the pipeline

```
cd ~\Desktop

# Get files on the Desktop
$files = Get-ChildItem

# Get files older than a week
$files | Where-Object {$_.CreationTime -lt (Get-Date).AddDays(-7)} | Sort-Object CreationTime
```

Project 2: Remove files by passing them down the pipeline

Create the files in a folder called `Test Folder` on your desktop

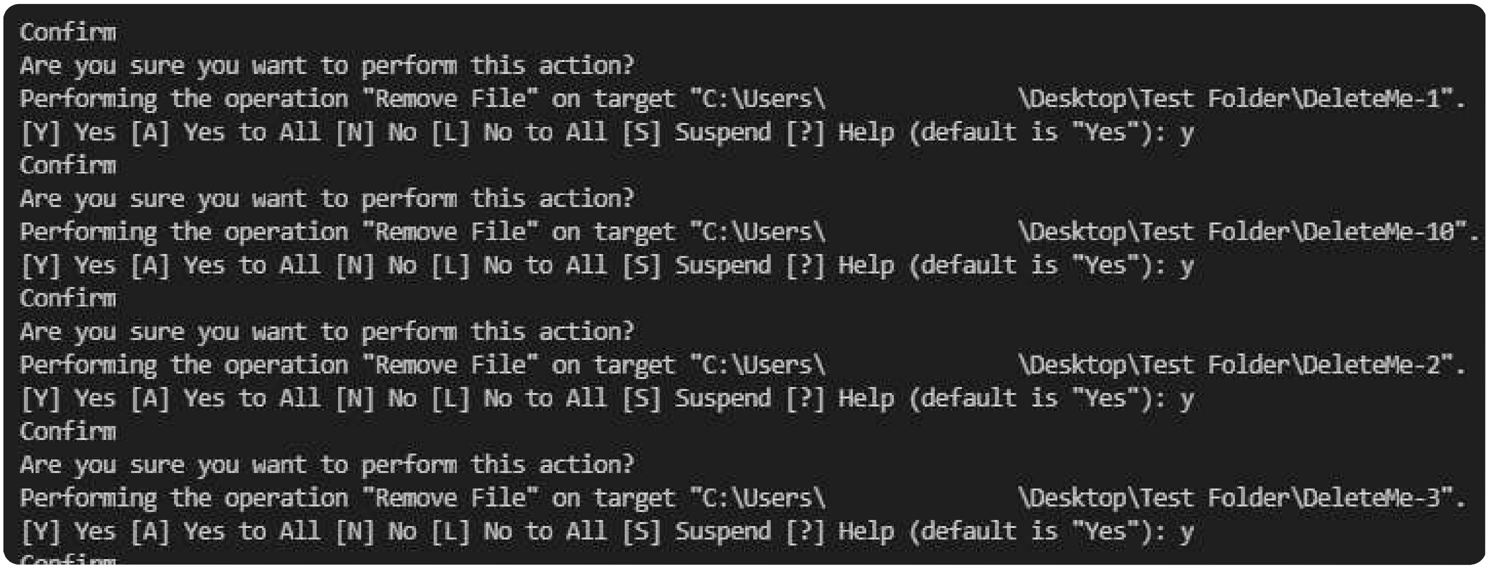
```
Set-Location ~\Desktop

# Create a folder on the Desktop
# Call it Test Folder
New-Item -ItemType Directory -Name 'Test Folder'

Set-Location 'Test Folder'

# Create some files
foreach ($number in (1..10)) {

    # Out-Null silences output from the cmdlet
    New-Item -ItemType File -Name "DeleteMe-$number" | Out-Null
    New-Item -ItemType File -Name "DontDeleteMe-$number" | Out-Null

}
```



```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/13/2022  11:34 AM              0 DeleteMe-1
-a----         1/13/2022  11:34 AM              0 DeleteMe-10
-a----         1/13/2022  11:34 AM              0 DeleteMe-2
-a----         1/13/2022  11:34 AM              0 DeleteMe-3
-a----         1/13/2022  11:34 AM              0 DeleteMe-4
-a----         1/13/2022  11:34 AM              0 DeleteMe-5
-a----         1/13/2022  11:34 AM              0 DeleteMe-6
-a----         1/13/2022  11:34 AM              0 DeleteMe-7
-a----         1/13/2022  11:34 AM              0 DeleteMe-8
-a----         1/13/2022  11:34 AM              0 DeleteMe-9
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-1
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-10
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-2
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-3
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-4
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-5
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-6
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-7
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-8
-a----         1/13/2022  11:34 AM              0 DontDeleteMe-9
```

Let's delete the files that start with `DeleteMe`.

```
# We have to wrap it in " "
# Double quotes because
# The name Test Folder
# Contains a space
cd "~\Desktop\Test Folder"

$files = Get-ChildItem
$deleteTheseFiles = $files | Where-Object {$_.Name -like 'DeleteMe-*'}
$deleteTheseFiles | Remove-Item -Confirm:$true
```



```
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\             \Desktop\Test Folder\DeleteMe-1".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): y
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\             \Desktop\Test Folder\DeleteMe-10".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): y
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\             \Desktop\Test Folder\DeleteMe-2".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): y
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\             \Desktop\Test Folder\DeleteMe-3".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): y
Confirm
```

Functional Programming with PowerShell

Once you're comfortable with the concepts here, move on to the next post in this series and learn the fundamentals of functional programming with PowerShell.