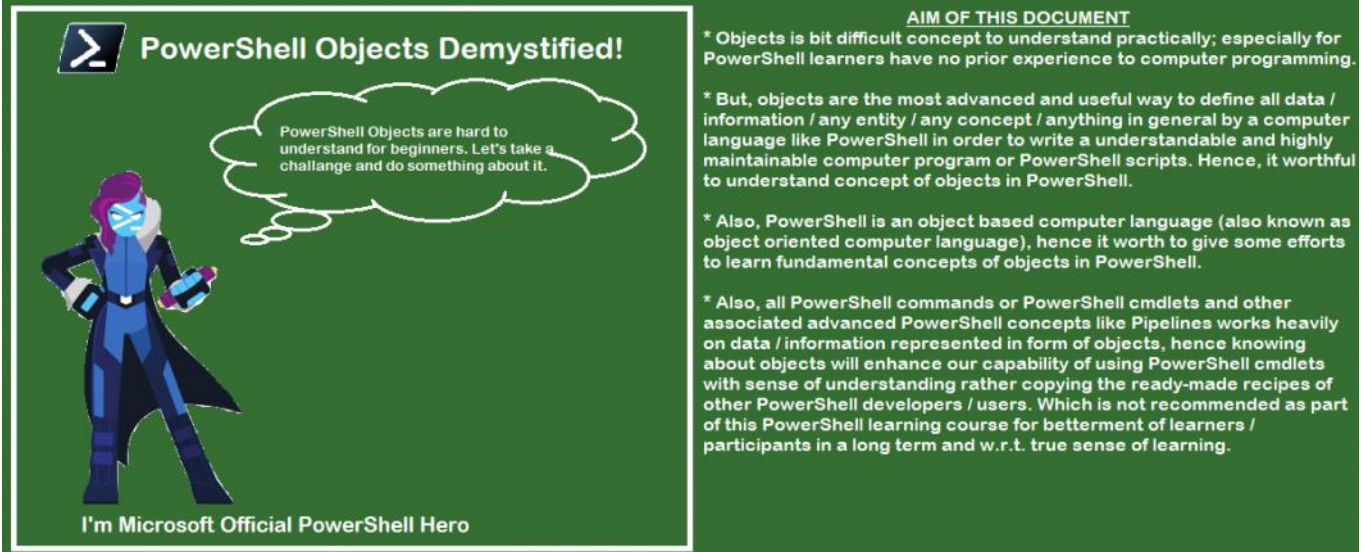


PowerShell Objects Demystified!

Friday, April 29, 2022 2:08 PM

Instructor	Himanshu Singh
Date	29-Apr-2022
Version	1.0

- **Aim of this document:**

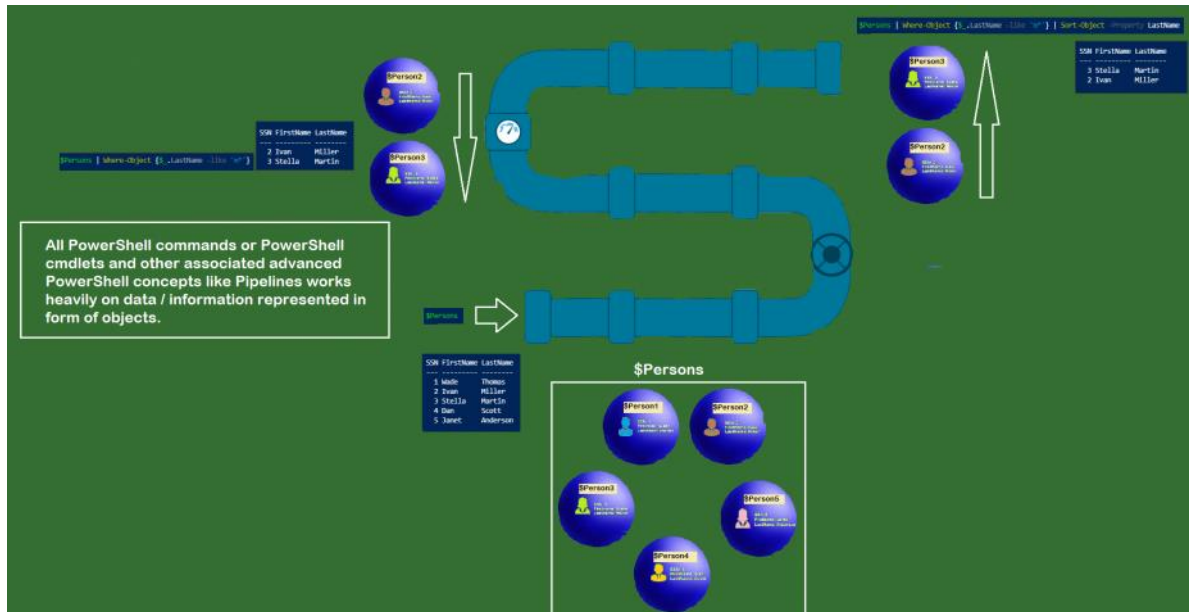
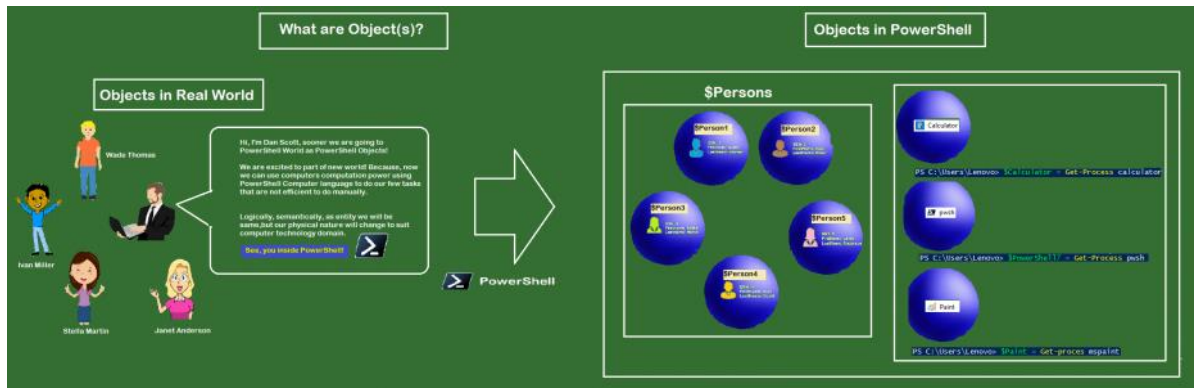


AIM OF THIS DOCUMENT

- * Objects is bit difficult concept to understand practically; especially for PowerShell learners have no prior experience to computer programming.
- * But, objects are the most advanced and useful way to define all data / information / any entity / any concept / anything in general by a computer language like PowerShell in order to write a understandable and highly maintainable computer program or PowerShell scripts. Hence, it worthwhile to understand concept of objects in PowerShell.
- * Also, PowerShell is an object based computer language (also known as object oriented computer language), hence it worth to give some efforts to learn fundamental concepts of objects in PowerShell.
- * Also, all PowerShell commands or PowerShell cmdlets and other associated advanced PowerShell concepts like Pipelines works heavily on data / information represented in form of objects, hence knowing about objects will enhance our capability of using PowerShell cmdlets with sense of understanding rather copying the ready-made recipes of other PowerShell developers / users. Which is not recommended as part of this PowerShell learning course for betterment of learners / participants in a long term and w.r.t. true sense of learning.

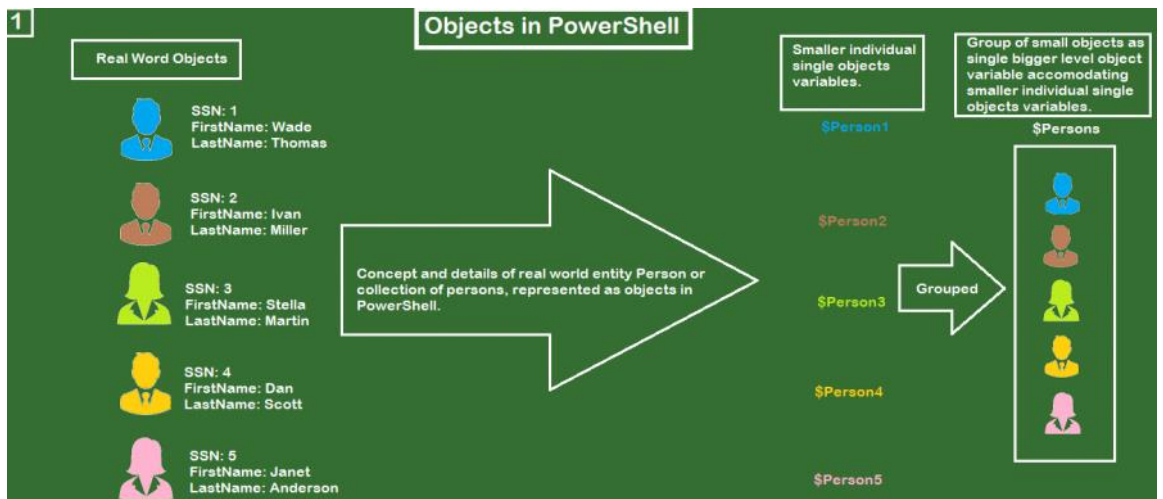
- **What are object(s) in PowerShell and in general?**

- Objects are computer language representation of world concepts and any real world entities to work inside computers using computers programs / applications / PowerShell scripts that are written / coded by us / computer programmer / software developer / script developer / system administrator.
- Examples of an valid PowerShell Object:
 - Any generic concept and details of a real world entity 'person' or 'collection of persons', are represented as an objects in PowerShell.
 - Concept and details of a real world entity like a 'car' or 'collection of cars', are represented as an objects in PowerShell.
 - In more realistic scenario with respect to PowerShell primary use in System Administration automation tasks, concept and details of all running computer programs / applications like (notepad = notepad.exe, Paint = mspaint.exe and PowerShell 7 itself = pwsh.exe etc.) running in our computers are technically w.r.t. computer science theory and computer programming are also represented as same concept of objects in PowerShell.



○ **Understanding objects in PowerShell step by step:**

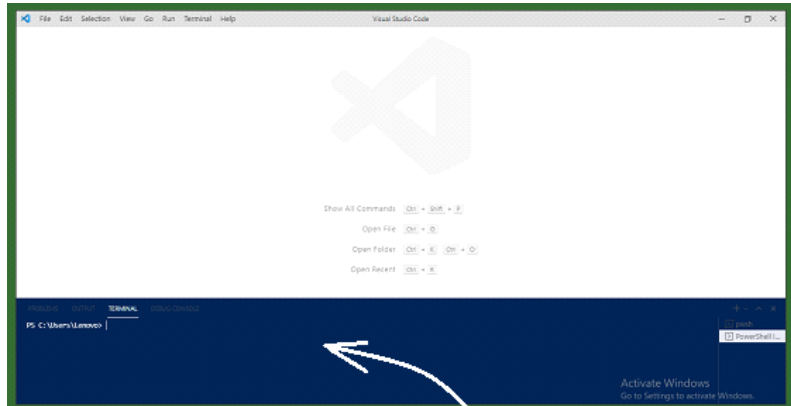
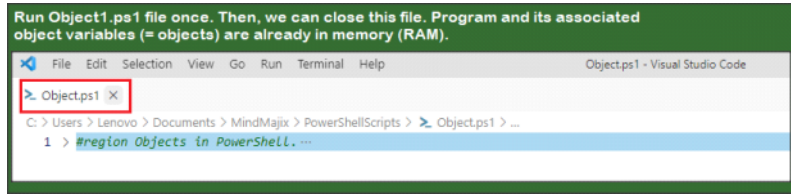
- Concept and details of real world entity Person or collection of persons, represented as an objects in PowerShell.



▪ **PowerShell objects lab setup:**

- Download and run Object1.ps1 program to create PowerShell objects lab environment. Open the downloaded file in Visual Studio Code or PowerShell ISE code editor , run this program file once (press F5) and then close it because objects that are created by program will be still present inside computer memory (RAM) to experiment with them. It should be noted that after

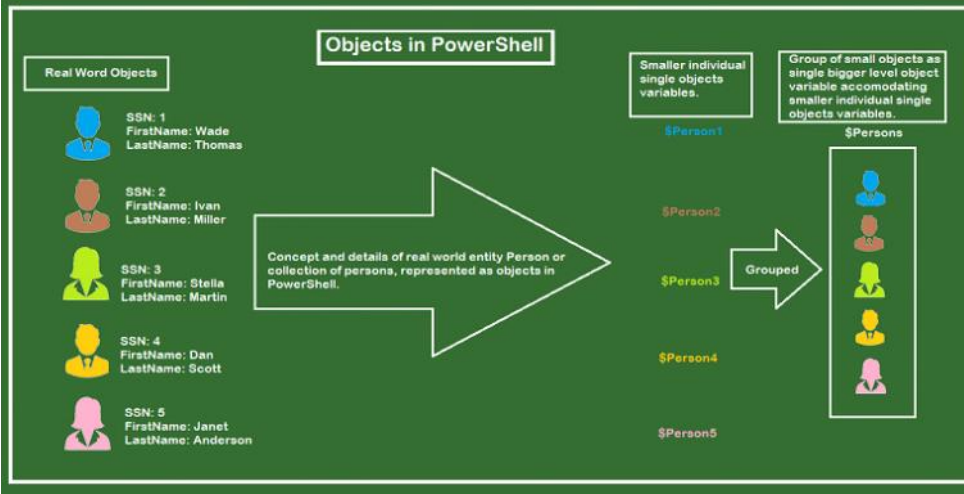
running the program objects will only remain in computer memory (RAM) until code editor is running and not closed.



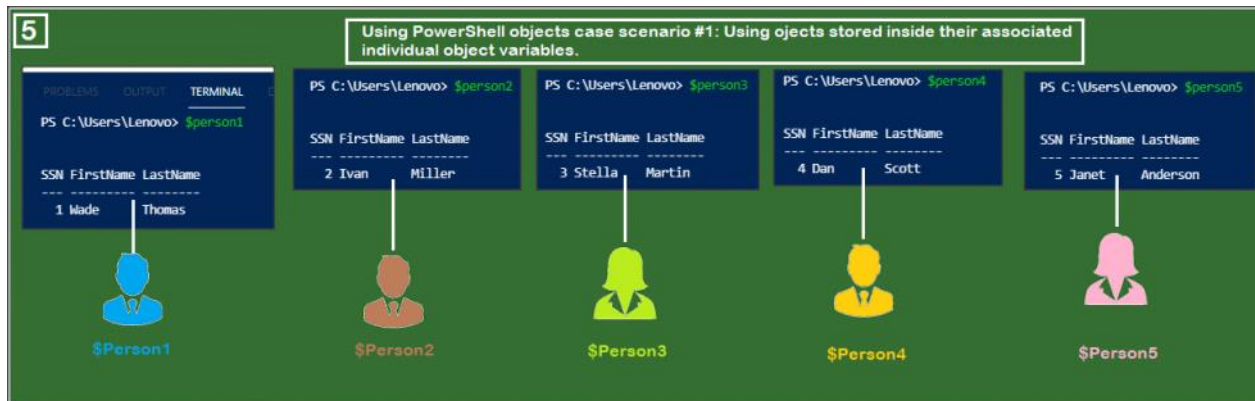
Go to Visual Studio Code / ISE Console Pane after running and then closing Object1.ps1 program. We are ready at this point to use and experiment with our created objects. We can test all case scenarios of object concept available in PowerShell in most simplified, technically correct, but still easy to understand way.

2

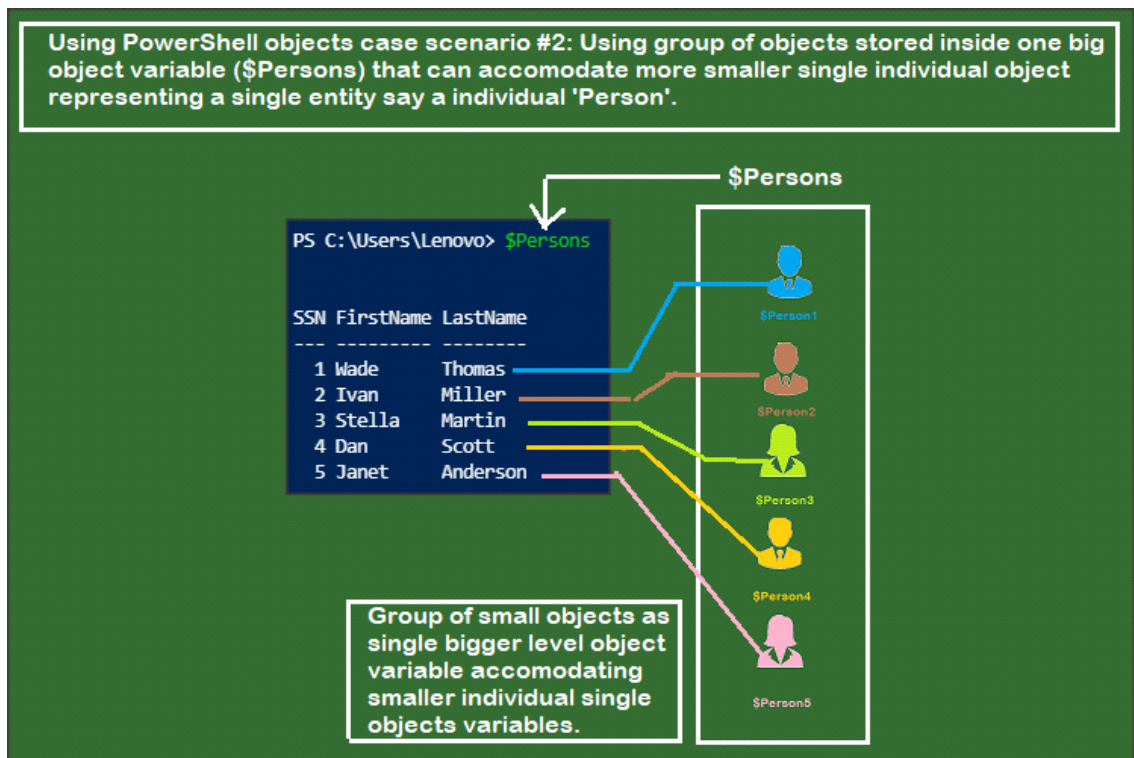
These objects are created as result of running Object1.ps1 PowerShell computer program file (press F5)



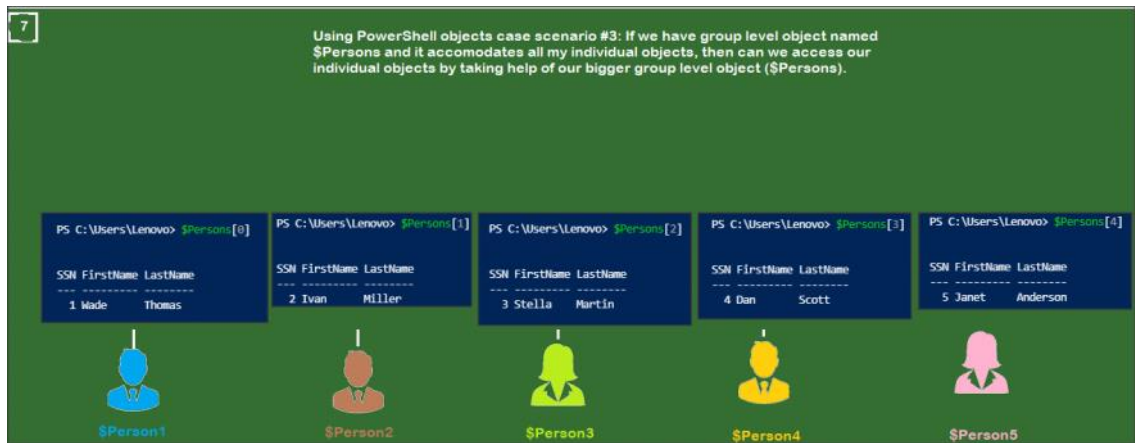
- Using PowerShell Objects - case scenario #1: Using objects that are stored inside their individual single object variable.



- Using PowerShell Objects - case scenario #2:** Using a group of single individual person objects that are stored inside one big group representing object variable (\$Persons). This group level object (\$Persons) is bigger in size in computer memory and can accommodate more smaller single individual object (\$Person1, \$Person2,...,\$Person5) inside it.



- Using PowerShell Objects - case scenario #3:** If, we have a group level object named \$Persons and we know that it accommodates all of my individual person representing objects, then can we access our individual objects by taking help of our bigger group level object i.e. \$Persons. This case scenario #3 is another approach of accomplishing same goal that is accomplished in case scenario #1 but by defining dedicated variables for each individual single person representing object.



○ **Application of PowerShell objects:**

- In earlier section, we learned what are objects in PowerShell. We, are also able to create a simple yet powerful (able to cover all case scenarios of using objects) lab environment to work with PowerShell objects.
- While setting up the objects lab by running PowerShell computer program named 'Object.ps1' actually, we have made us capable to capture and realize the concept and details of the real world entity - Person or collection of persons as PowerShell objects.
- This object.ps1 program has created few individual person representing objects for us to work with. Each individual person representing object is represented by and present inside its associated dedicated variable viz. \$Person1, \$Person2, \$Person3, \$Person4 \$Persons5.
- There is also one more group level bigger size object variable with name \$Persons that capability to accommodate all smaller individual objects inside it and allow PowerShell user to access individual person representing objects that it is storing with help of group level variable \$Persons instead of using dedicated variables (\$Person1, \$Person2,...,\$Person5) to refer individual object in a group / collection. Well and good till this point.
- Now, as we have objects to work with (rather I better say play with!) lets apply that knowledge to work with PowerShell objects using various computer programming concepts / methods / techniques available in PowerShell computer language and accomplish some goal or solution to a problem / requirement in hand.
- For getting the grasp of core concepts related to PowerShell objects and for simplicity reasons we have created in earlier discuss PowerShell object lab individual known persons and their group as PowerShell objects.
- It is should be noted here that, all PowerShell computer programming techniques are applicable on sample PowerShell objects created and mentioned above as they are applicable any type of PowerShell objects we will encounter in real case scenarios. For e.g. all PowerShell objects created as part of firing 'Get-Process' cmdlet. In context of Get-Process cmdlet, we have our presently running computer programs / applications / processes represented as PowerShell objects.
- **Objects application #1:** Finding count of total number of individual single objects (\$Person1, \$Person2,..., \$Person5)

present inside bigger group level object (\$Person.

- Command: \$Persons.Count

```
PS C:\Users\Lenovo> $Persons.Count
```

-

```
5
```

- **Objects application #2:** Find all available properties of each individual person object.

- What are object properties?

- ◆ In real world, every object has some particular characteristics associated to it that makes that object uniquely identified in the world. For e.g. person can have characteristic like SSN (Social Security Number), first name and last name as its characteristics. We can identify or refer a person in the world among other present objects uniquely using them. For e.g. if some person has associated characteristics as SSN = 1546, first name = 'Stella' and last name = 'Martin', then using these characteristics we can uniquely refer to this particular person in the real world.

- ◆ Like real world object, PowerShell objects also have associated characteristics associated to it. For e.g. in our previously discussed PowerShell object lab, we have single individual object \$Person3 which refers to person 'Stella Martin'. This object has three characteristics associated with it viz. SSN, FirstName and LastName. Each object characteristic has a value associated with it like person 'Stella Martin' has characteristic SSN with associated value 3, has characteristic FirstName with associated value 'Stella' and has characteristic 'LastName' with associated value 'Martin'. In PowerShell we call object characteristics as its properties. Each property has associated value to it.

- Let's say we have real person 'Stella Martin' represented as an object (\$Person1) using PowerShell computer language in a PowerShell computer program / PowerShell script. To make this person uniquely identifiable / locate / refer / distinguish in a group (\$Persons) of other similar individual objects present in that group we have associated / attached few properties / characteristics with associated values to each individual object. If we have access to these properties and their associated values then seeing the details of these properties in form of that value we as human as well as computer programs are able to identify and work of individual object per se. Without properties and their associated values; it would have been hard to human as well as machine / computer programs to distinguish objects created by computer languages like PowerShell. So, now as we know our person in concern ('Stella Martin') may have some properties and their associated values let's access / find / retrieve those values with help of object that represents 'Stella Martin' in PowerShell computer language (i.e. \$Person3 or using group level object we can also access 'Stella Martin' single individual person object representation as Persons[2]). So, to access all properties of 'Stella Martin' single individual object do following:

◆  SSN: 3
 FirstName: Stella
 LastName: Martin

All 4 ways accomplish same goal: Successfully able to select particular interested individual single person object representation of 'Stella Martin' person and shows all its associated properties and each property associated and corresponding value. Important!!

```
PS C:\Users\Lenovo> $Person3 | Select-Object -Property *
```

SSN	FirstName	LastName
3	Stella	Martin

Using individual single object containing object type variable

```
PS C:\Users\Lenovo> $Persons[2] | Select-Object -Property *
```

SSN	FirstName	LastName
3	Stella	Martin

Using group level object variable (\$Persons) containing individual single object inside it.

```
PS C:\Users\Lenovo> $MyNewGroupObj = Get-Person
PS C:\Users\Lenovo> $MyNewStellaMartinObject = $MyNewGroupObj[2]
PS C:\Users\Lenovo> $MyNewStellaMartinObject | Select-Object -Property *
```

SSN	FirstName	LastName
3	Stella	Martin

Using 'Get-Person' function. Similar to using 'Get-Process' cmdlet.

```
PS C:\Users\Lenovo> $Person3 | Format-List *
```

```
SSN      : 3
FirstName : Stella
LastName  : Martin
```

Using Format-List Cmdlet to list all properties of an individual single object containing object type variable.

- **Objects application #3:** Let's keep on focus on our particular individual object 'Stella Martin'. Now we know to print all the properties of object, let's print only selected interested properties of object

```
PS C:\Users\Lenovo> $Person3 | Select-Object -Property SSN
```

SSN
3

□

```
PS C:\Users\Lenovo> $Person3 | Select-Object -Property SSN, LastName
```

SSN	LastName
3	Martin

- **Objects application #4:** Let's keep on focus on our particular individual object 'Stella Martin'. Now we know that we have access to particular interested person object 'Stella Martin', we also want to enquire details about this particular individual object but, we don't want to see all the properties of the object, rather we only want to see few interested properties of the interested individual object 'Stella Martin'. Problem is we don't the exact names of the properties of this object to specify using previously learned command. There is a command to accomplish this goal:

Properties of individual person object 'Stella Martin' represented as single object variable \$Person3

```
PS C:\Users\Lenovo> $Person3 | Get-Member -MemberType Property
```

Name	MemberType	Definition
FirstName	Property	string FirstName {get;set;}
LastName	Property	string LastName {get;set;}
SSN	Property	byte SSN {get;set;}

\$Person3

Properties of 'Stella Martin' person represented as individual object.

Values associated with each available property of Properties of 'Stella Martin' person represented as individual object.

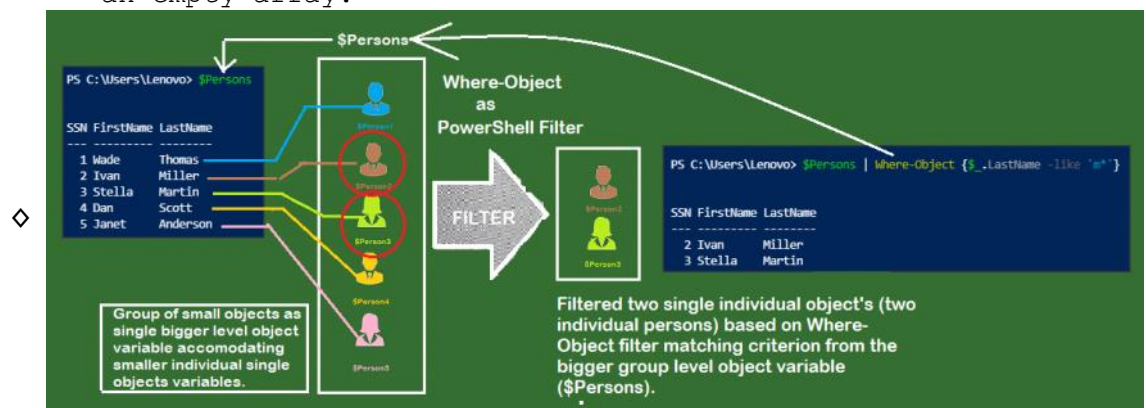
Both ways accomplish same goal.

- Now we know all properties name, we can specify interested property name to show interested property with its associated value for our focus and PowerShell selected object as discussed in previous command.
- **Objects application #5:** Let's widen our focus. Suppose, we have bigger group level object variable (like \$Persons) which accumulates inside it and also representing a collection or group smaller individual object representing single individual person (in our case \$Person1, \$Person2,...,\$Person5). Now, the we have situation. We, want to display each individual single object (i.e. \$Person1, \$Person2,...,\$Person5)present inside group level object (\$persons) but not based on their properties / characteristics names also not all the objects present inside the group (\$Persons have in our case 5 total individual single Person representing objects), but only few of those individual single objects that have 'values of the particular interested property of the individual single Person object' matching our 'Object Filtering' criterion. This technique / method in PowerShell is called 'Filtering of objects'. We have Cmdlet in PowerShell to do just that work in very efficient fashion - Where-Object cmdlet. Few examples given below to showcase various category of requirements in which Where-Obj cmdlet can be applied to filter desired single individual objects from a group level bigger object variable like in our case (\$Person)
 - **Where-Object example #1:** From group of Persons represented by bigger group level object (\$Persons), I want only those individual single object whose interested property say 'LastName' value starting with character 'm' that is also in case-insensitive fashion i.e. 'martin' or Martin' as LastName (= Surname) property value will be considered valid as per criterion of using Where-Object for filtering individual single object.
 - Theory of Where-Object cmdlet because it is very useful for applications with PowerShell objects.
 - ◆ Where-Object in general: The Where-Object command is used to filter objects based on any of their properties. Where-Object follows a consistent pattern that looks like:
 - ◆ Where-Object {\$_.PropertyName -ComparisonType FilterValue}
 - ◆ The PropertyName is the name of the object's property that you are filtering.
 - ◆ ComparisonType or Comparison operators is a short keyword for what type of comparison you are doing. Some examples are "eq" for equals, "gt" for greater than, "lt" for less than, and "like" for a wildcard

search. Finally, the FilterValue is the value you are comparing the object's property against.

Where-Object Comparison operators list.		
Type	Operator	Comparison test
Equality	-eq	equals
	-ne	not equals
	-gt	greater than
	-ge	greater than or equal
	-lt	less than
	-le	less than or equal
Matching	-like	string matches wildcard pattern
	-notlike	string does not match wildcard pattern
	-match	string matches regex pattern
	-notmatch	string does not match regex pattern
Replacement (advanced)	-replace	replaces strings matching a regex pattern
Containment	-contains	collection contains a value
	-notcontains	collection does not contain a value
	-in	value is in a collection
	-notin	value is not in a collection
Type (advanced)	-is	both objects are the same type
	-isnot	the objects are not the same type

► Note: By default, string comparisons are case-insensitive. The equality operators have explicit case-sensitive and case-insensitive forms. To make a comparison operator case-sensitive, add a c after the -. For example, -ceq is the case-sensitive version of -eq. To make the case-insensitivity explicit, add an i after -. For example, -ieq is the explicitly case-insensitive version of -eq. When the input of an operator is a scalar value, the operator returns a Boolean value. When the input is a collection / simple a bigger group level variable like in case \$Persons containing many smaller individual single objects inside it as group level object variable, the operator returns only those filtered elements of the collection that match the right-hand value of the expression. If there are no matches in the collection, comparison operators return an empty array.

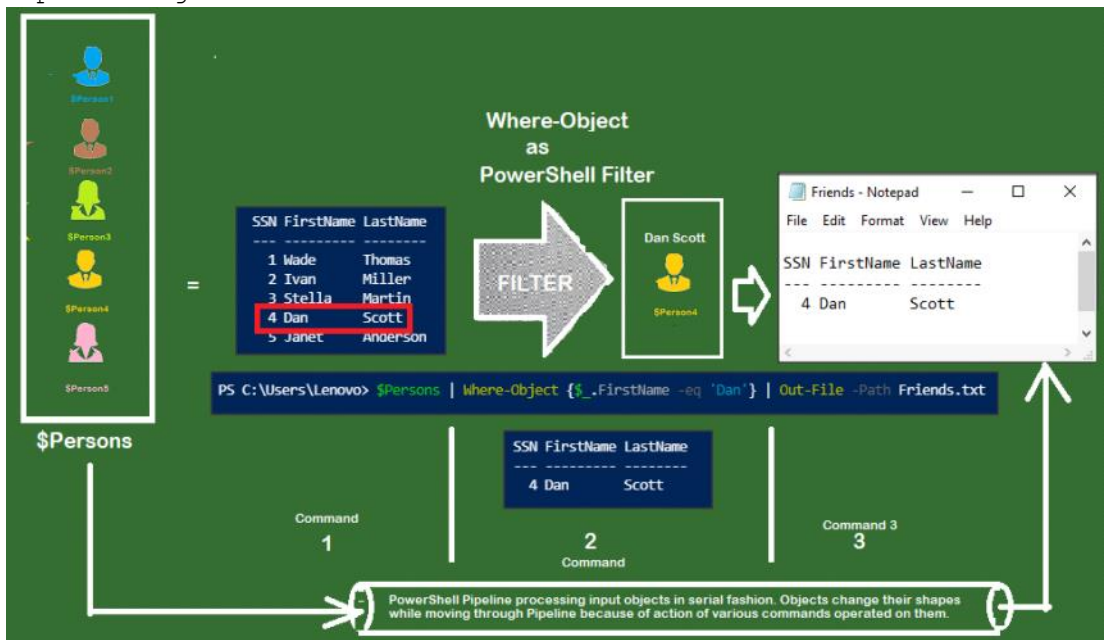


□ **Where-Object example #2:** From group of Persons represented by bigger group level object (\$Persons), I want only those individual single object whose interested property say 'SSN' value (a numeric value) is greater than 3 or equal to 3. We will be using equality Comparison operators as greater than equal to here in this case which is written as 'ge'.

```
PS C:\Users\Lenovo> $Persons | Where-Object {$_.SSN -ge 3}
```

SSN	FirstName	LastName
3	Stella	Martin
4	Dan	Scott
5	Janet	Anderson

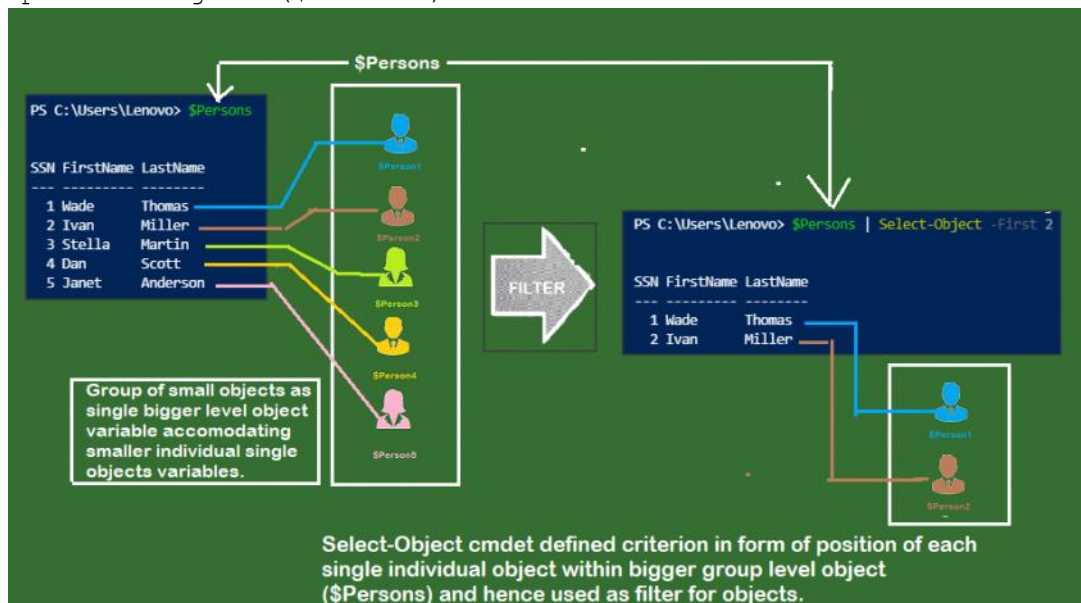
- Where-Object example #3:** From group of Persons represented by bigger group level object (\$Persons), I want only one individual single object by specifying a very precise and strict criterion that is my single individual object interested property say 'FirstName' should be exactly equal to property value of 'Dan'. If I give this criterion then single individual object representation of person 'Dan Scott' will be filtered out from group of Persons represented by bigger group level object (\$Persons). We also want to write the final filtered object representing person 'Dan Scott' in some text file for logging purpose, data analysis purpose and for future reference using PowerShell pipeline concept adding third command as 'Out-File' cmdlet.



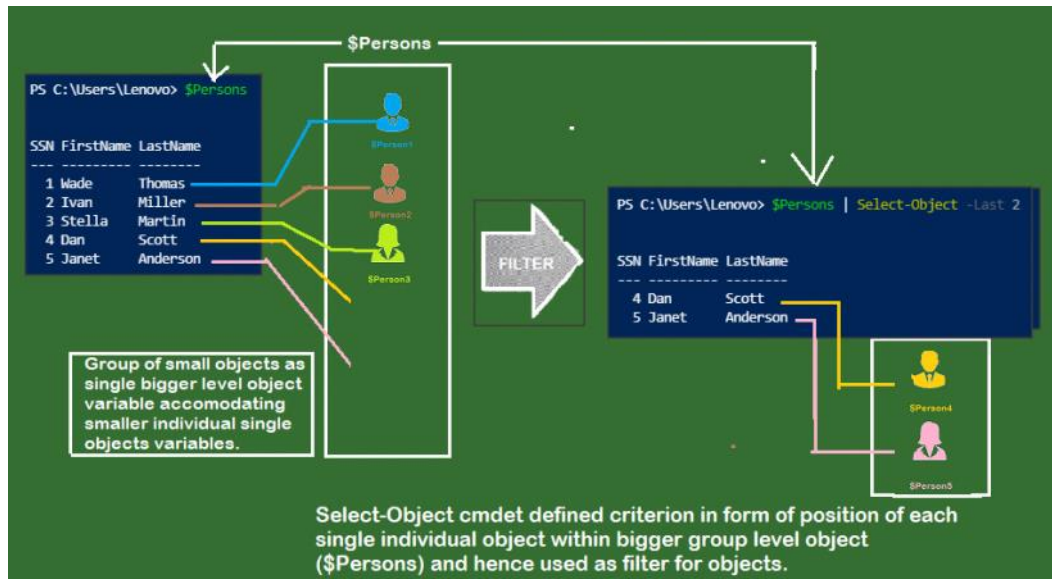
- Objects application #6:** Let's still keep widen our focus. Suppose, we have bigger group level object variable (like \$Persons) which accumulates inside it smaller individual objects which are representing single individual objects of type person (\$Person1, \$Person2,...,\$Person5). Now, let's consider a situation. We, want to display each individual single object (i.e. \$Person1, \$Person2,...,\$Person5) present inside group level object (\$persons) but this time, based on individual single objects that are representing a person properties / characteristics name or individual single objects other higher level details like what is the position particular individual single object while being present inside a bigger group level object variable (\$Persons). It should be noted that both Select-Object cmdlet and Where-Object cmdlets filter-objects but the degree of level of details of each individual single object (i.e. \$Person1, \$Person2,...,\$Person5) that these two different cmdlets take care differ. Select-Object confines

its selection and search criterion for the each individual single object till the level of each single individual object's property name and its position within bigger group level object variable (\$Persons) while Where-Object goes little more deep while defining its selection and its search criterion for the each individual single object present inside within bigger group level object variable (\$Persons) to the level of matching what are the details of each single individual object based on the values of properties of each single individual object. Both cmdlets have use in different case scenarios and PowerShell requirements. Apart from filtering each single individual objects within bigger group level object variable (\$Persons) Select-Object is also used to reduce / put a limit on amount of details shown / displayed / exposed / revealed / retrieved in form of all available properties and their associated values for each individual single object.

- **Select-Object example #1:** From group of Persons represented by bigger group level object (\$Persons), I want only those individual single object whose position in its containing bigger group level object (\$Persons) should be less than or equal to 2 in other words we want those single individual object(s) that are among first 2 within their accumulating bigger group level object (\$Persons).



- **Select-Object example #2:** From group of Persons represented by bigger group level object (\$Persons), I want only those individual single object whose position in its containing bigger group level object (\$Persons) should be among last 2 within their accumulating bigger group level object (\$Persons).



- **Select-Object example #3:** As discussed earlier, we can also use Select-Object to limit details shown about each selected / retrieved / displayed single individual object. For examples we want limited details of all single individual objects in form its only single property named 'SSN' and its associated value. As discussed earlier, we have group of Persons which are represented by bigger group level object (\$Persons) which actually accumulates individual single object (i.e. \$Person1, \$Person2,...,\$Person5) inside it.

```
PS C:\Users\Lenovo> $Persons | Select-Object -Property SSN
```

```
SSN
---
```

```
1
2
3
4
5
```

- **Select-Object example #4:** As discussed earlier, we can also use Select-Object to limit details shown about each selected / retrieved / displayed about each single individual object (i.e. \$Person1, \$Person2,...,\$Person5) accessed from the group level object (\$Persons). For examples we want limited details of all single individual objects in form its all those properties whose names we are not fully clear but we know that our interested set of properties for each single individual object (i.e. \$Person1, \$Person2,...,\$Person5) accumulated / present inside bigger group level object (\$Persons) have 'Name' word present in last of our interested Property Name (for e.g. FirstName, LastName as our interested property names). So, in this case scenario in place of exact property name we can specify approximate value of property name using wildcards (using *).

▪ **Topic: Counting Objects using new cmdlet Measure-object in Pipeline while are using Select-Object or Where-Object also**

- Topic details: Find how many individual single objects (i.e. \$Person1, \$Person2,...,\$Person5) are there in count which have the property with name 'LastName' and with its property value that starts with character 'm' ends with then anything (i.e. * in wildcards usage terms).

```
PS C:\Users\Lenovo> $Persons | Where-Object {$_.LastName -like 'm*'}
```

SSN	FirstName	LastName
2	Ivan	Miller
3	Stella	Martin

```
PS C:\Users\Lenovo> $Persons | Where-Object {$_.LastName -like 'm*' } | Measure-Object | Select-Object -Property Count
```

Count
2

▪ **Topic: Using Select-Object cmdlet with -Last parameter, in pipeline, for filtering and sorting using Sort-Object, real world scenario.**

- Topic details: Select 5 most largest amount of computer memory (RAM) using processes currently running inside computer.

- ◆ Get-Process | Sort-Object -Property WS | Select-Object -Last 5

▪ **Topic: Select-Object with use of -Unique parameter, used in pipeline, for filtering object**

```
PS C:\Users\Lenovo> $a = "Peter", "Sam", "Peter", "Christopher", "Sam", "Lucy", "Mark"
```

```
PS C:\Users\Lenovo> $a | Select-Object -Unique
```

- Peter
- Sam
- Christopher
- Lucy
- Mark

▪ **Topic: Select-Object use with -Index parameter, used in pipeline, for filtering object**

```
PS C:\Users\Lenovo> $a = Get-ChildItem | Sort-Object -Property LastWriteTime
```

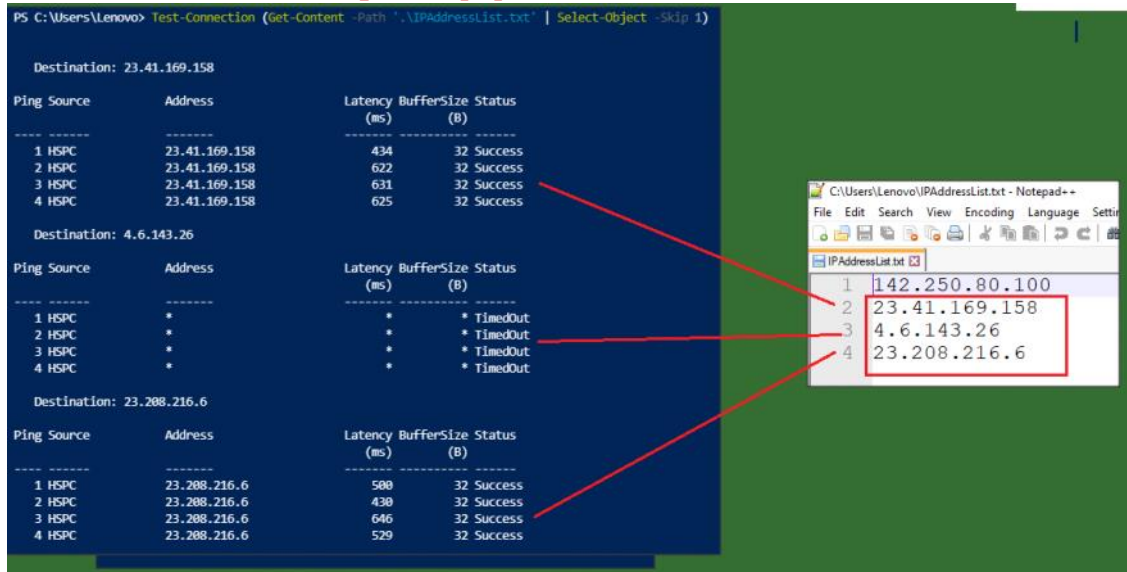
Mode	LastWriteTime	Length	Name
d-r--	1/17/2021 9:52 AM		3D Objects
d-r--	1/17/2021 9:52 AM		Contacts
d-r--	1/17/2021 9:52 AM		Favorites
d-r--	1/17/2021 9:52 AM		Saved Games
d-r--	1/17/2021 9:52 AM		Links
d-r--	2/13/2021 5:30 PM		Videos
d-r--	7/5/2021 1:08 PM		Searches
d-r--	7/7/2021 3:20 PM		Pictures
d----	7/12/2021 5:25 AM		.vscode
d----	7/16/2021 4:53 PM		Tracing
d-r--	7/19/2021 8:09 PM		Music
-a---	7/24/2021 10:21 PM	2719	Get-TrainingStatus.ps1
d----	7/29/2021 4:39 PM		.cache
d----	7/29/2021 6:34 PM		.dbus-keyrings
d-r--	7/30/2021 7:02 AM		Desktop
dar--	8/8/2021 11:45 PM		OneDrive
d----	8/9/2021 12:57 AM		.VirtualBox
d-r--	8/12/2021 12:03 PM		Downloads
-a---	8/12/2021 9:35 PM	73	Friends.txt
d-r--	8/12/2021 10:03 PM		Documents

```
PS C:\Users\Lenovo> $a | Select-Object -Index 0, ($a.Count - 1)
```

Mode	LastWriteTime	Length	Name
d-r--	1/17/2021 9:52 AM		3D Objects
d-r--	8/12/2021 10:03 PM		Documents

▪ **Topic: Reading lines from text file (in particular, IP addresses written in a text file with one IP address per**

line), with use of pipeline, give all read lines as collection of read lines objects to next command in pipeline i.e. Select-Object, then using Select-Object with -Skip parameter and finally using Test-Connection cmdlet to work on lines / line objects / in particular IP addresses filtered out by the pipeline used commands.



- Topic: Enumerating objects (moving to each object explicitly without use of pipeline), using ForEach-Object cmdlet, simple example.

```
Process[0]: PS C:\Users\Lenovo> 1, 2, 3 | ForEach-Object {$_}
```

```
1
```

```
2
```

```
3
```

- Topic: Enumerating objects, using ForEach-Object cmdlet while using pipeline, here using ForEach-Object we are moving to each process object collection got from previous command executed / run in the pipeline and for each object doing some processing when we have reach particular object during the overall movement. The Processing done is written inside {} and details of processing are: write this in this format for each object 'ID = Process Name = Company has developed me.'

```
PS C:\Users\Lenovo> Get-Process | Where-Object {$_.Company -ne $null} | ForEach-Object {Write-Host $_.Id = $_.ProcessName. $_.Company has developed me.}
```

```
9992 = ApplicationFrameHost . Microsoft Corporation has developed me.
10432 = backgroundTaskHost . Microsoft Corporation has developed me.
11036 = Calculator . Microsoft Corporation has developed me.
2060 = Code . Microsoft Corporation has developed me.
3480 = Code . Microsoft Corporation has developed me.
3500 = Code . Microsoft Corporation has developed me.
5532 = Code . Microsoft Corporation has developed me.
7032 = Code . Microsoft Corporation has developed me.
7036 = Code . Microsoft Corporation has developed me.
7084 = Code . Microsoft Corporation has developed me.
10900 = Code . Microsoft Corporation has developed me.
11188 = Code . Microsoft Corporation has developed me.
11156 = CompPkgSrv . Microsoft Corporation has developed me.
3532 = conhost . Microsoft Corporation has developed me.
10156 = conhost . Microsoft Corporation has developed me.
6284 = dllhost . Microsoft Corporation has developed me.
2320 = DuckCapture . DuckLink Software has developed me.
6648 = explorer . Microsoft Corporation has developed me.
124 = firefox . Mozilla Corporation has developed me.
636 = firefox . Mozilla Corporation has developed me.
1304 = firefox . Mozilla Corporation has developed me.
1372 = firefox . Mozilla Corporation has developed me.
6216 = firefox . Mozilla Corporation has developed me.
9440 = firefox . Mozilla Corporation has developed me.
9908 = firefox . Mozilla Corporation has developed me.
10760 = firefox . Mozilla Corporation has developed me.
```

- Topic: Enumerating objects, using ForEach-Object cmdlet, in pipeline, this time doing some computation when we are moving through each object using ForEach-Object cmdlet and

doing some computation / processing when are at particular object during overall movement. In this example \$Files variable contains files inside C:\Users\Lenovo folder.

```
PS C:\Users\Lenovo> $Files = Get-ChildItem -File
PS C:\Users\Lenovo> $Files | ForEach-Object -Process {$_ .Length / 1024}
64.234375
0.0712890625
2.6552734375
0.0537109375
0.240234375
0.2822265625
```

Friends	TXT File	1 KB
Get-TrainingStatus	PowerShell Source File	3 KB
IPAddressList	TXT File	1 KB
Set-MindMajixDir	PowerShell Source File	1 KB
Set-MindMajixScrip..	PowerShell Source File	1 KB
ChalkboardOriginal	FastStone PNG File	65 KB

© 2021 Himanshu Singh.