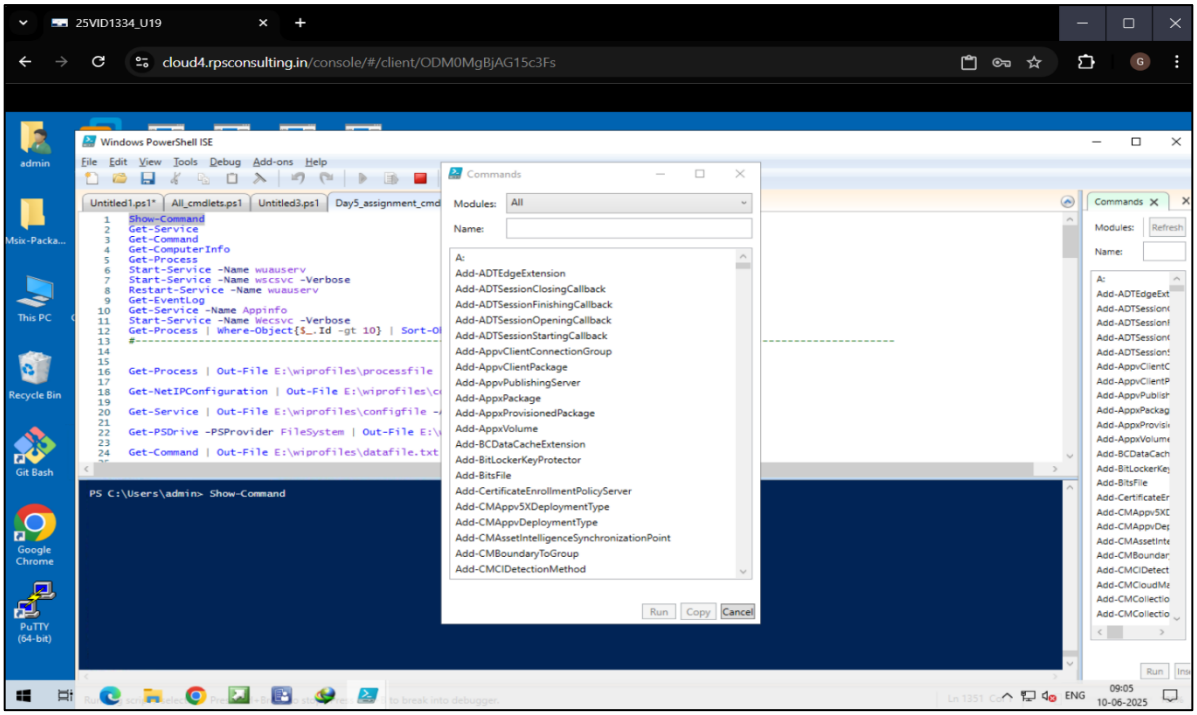
Show, Get, Start and Restart Commands

List of commands:

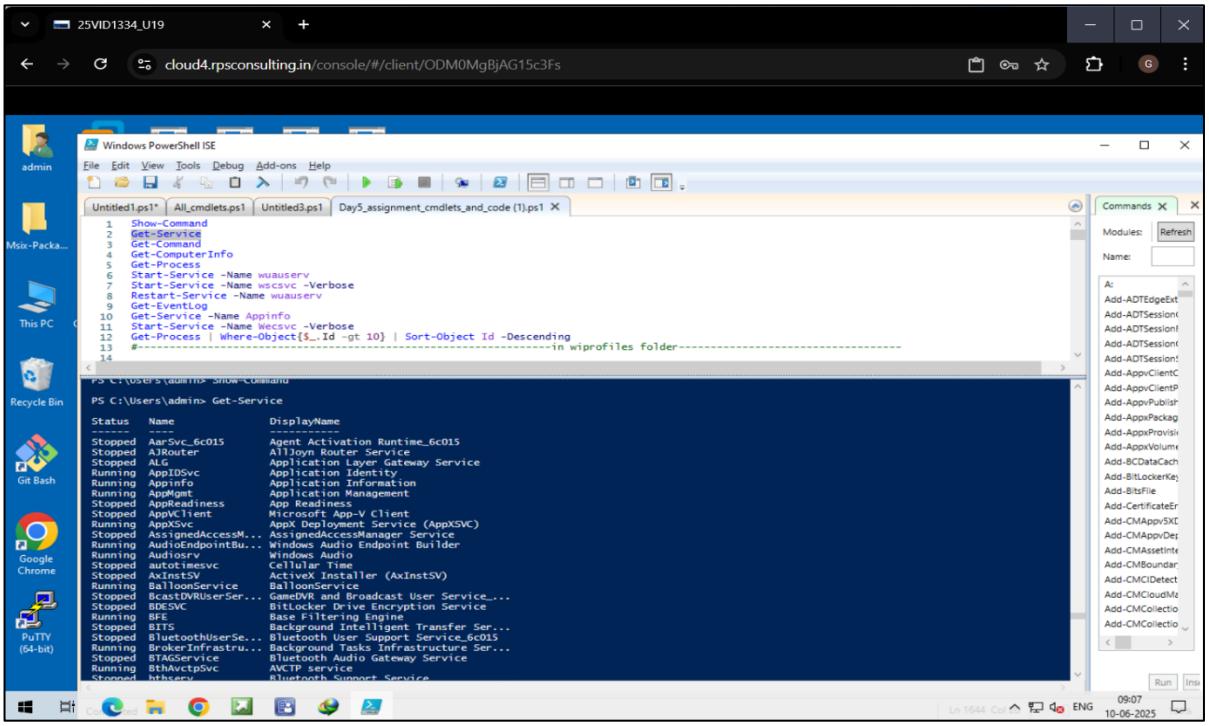
1. **Show-Command:** Show-Command opens a **graphical window** that displays the PowerShell cmdlets.

- **Show-Command**



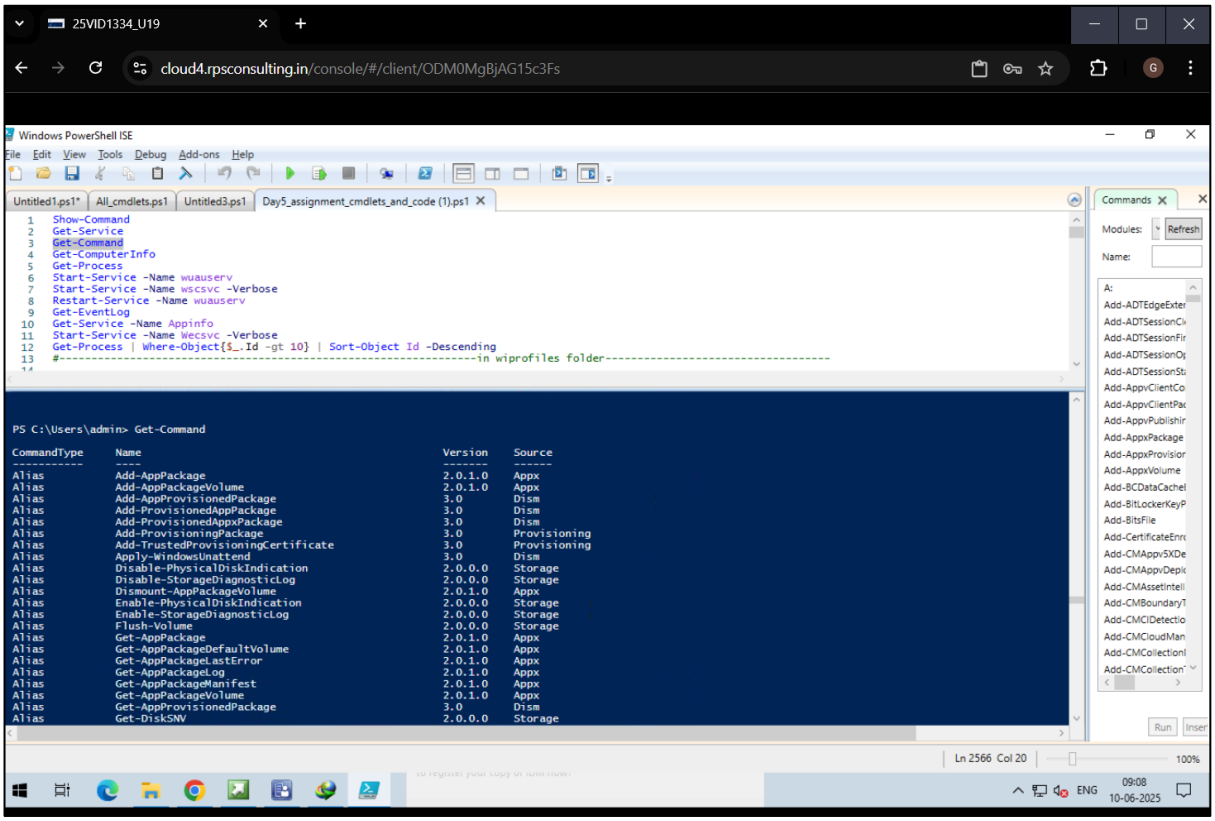
2. **Get-Service:** The Get-Service cmdlet in PowerShell retrieves the **services installed on a local or remote computer**, including their status, name, and display name.

- **Get-Service**



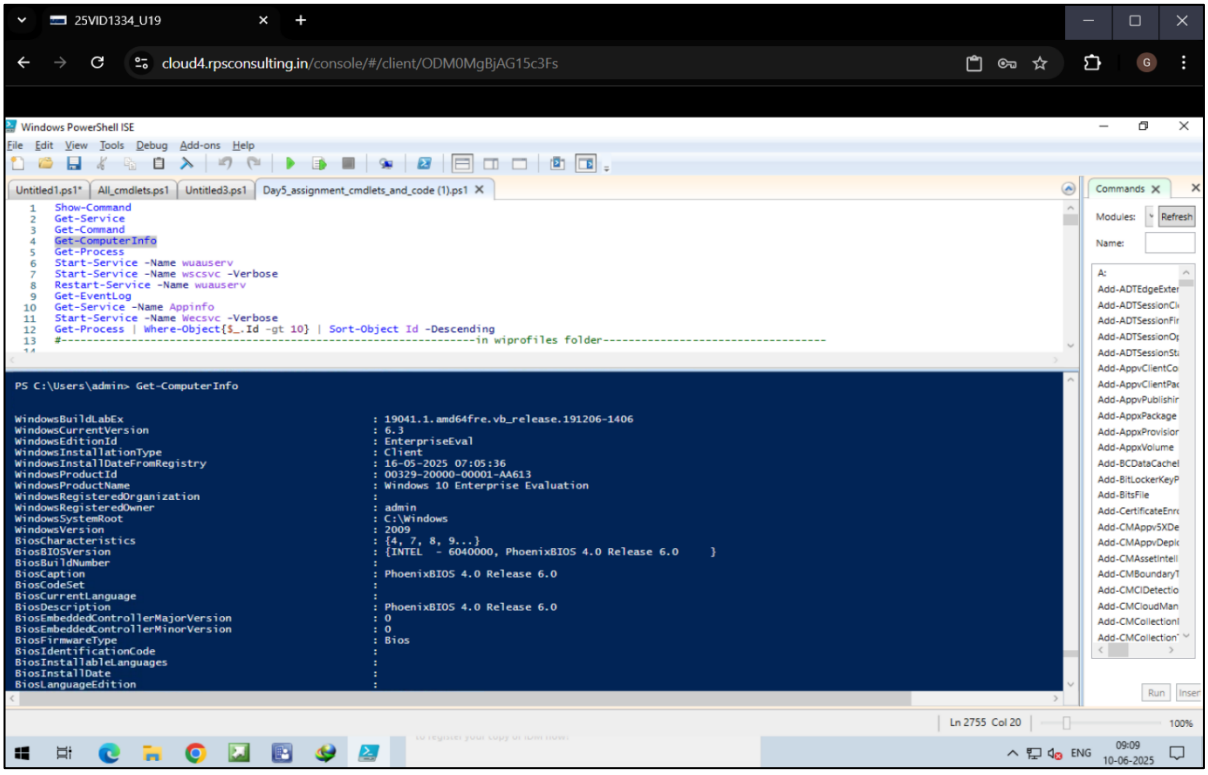
3. **Get-command:** The Get-Command cmdlet retrives **list of all available commands**.

- **Get-command**



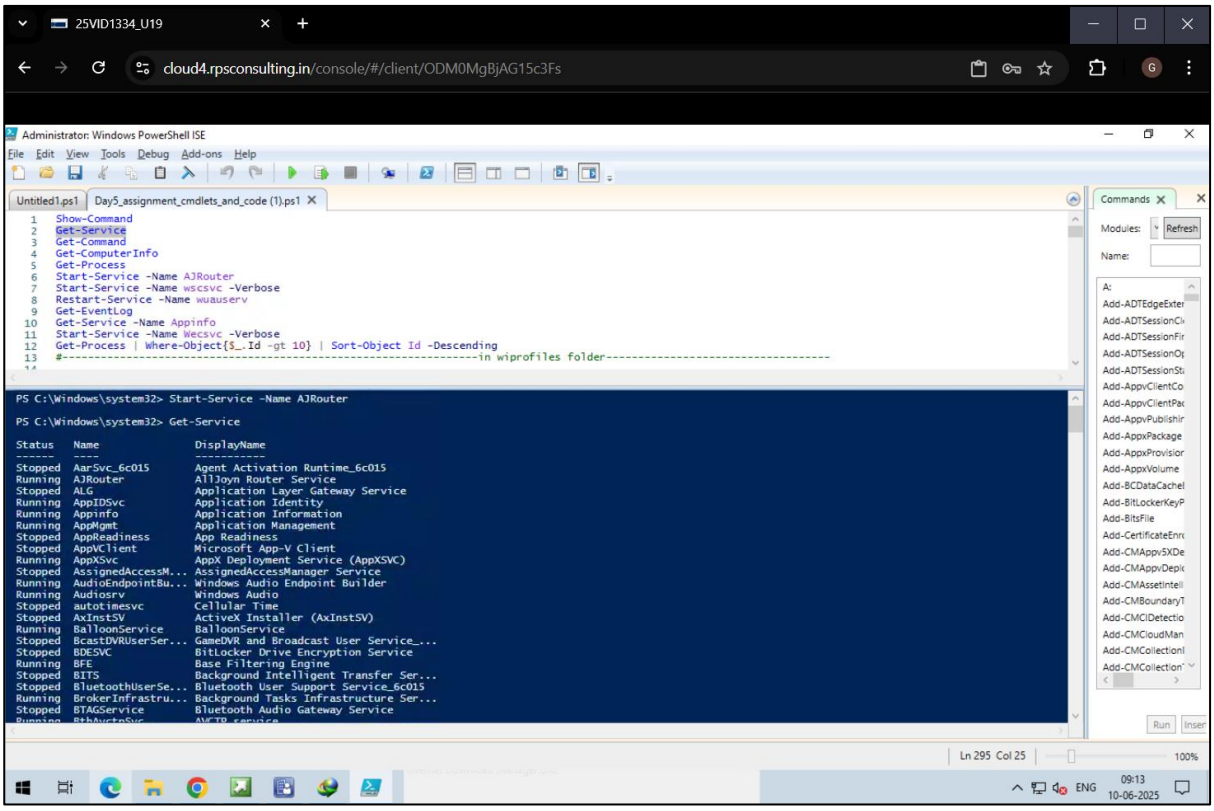
4. **Get-ComputerInfo**: The Get-ComputerInfo cmdlet in PowerShell retrieves a **consolidated** object containing system and operating system properties.

- **Get-ComputerInfo**



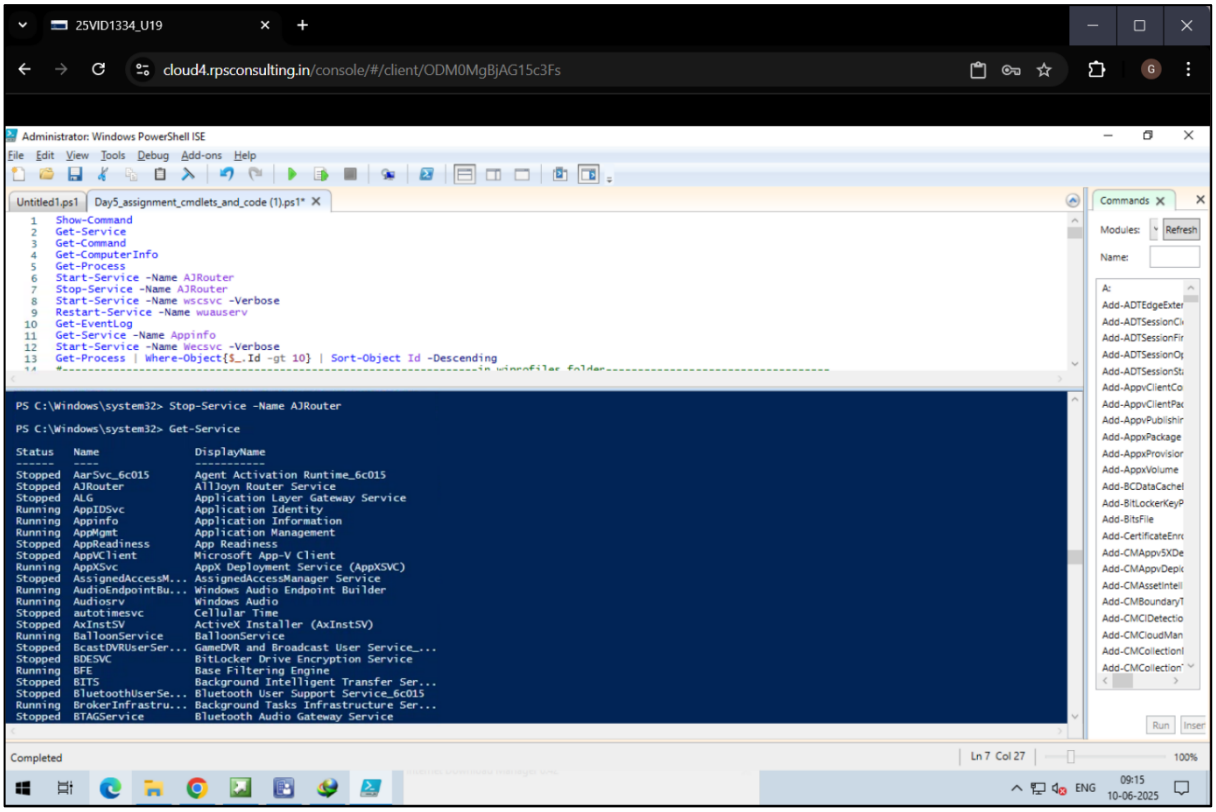
5. **Start-Service:** Starts a specific **service** with the name parameter.

- **Start-Service -Name AjRouter**



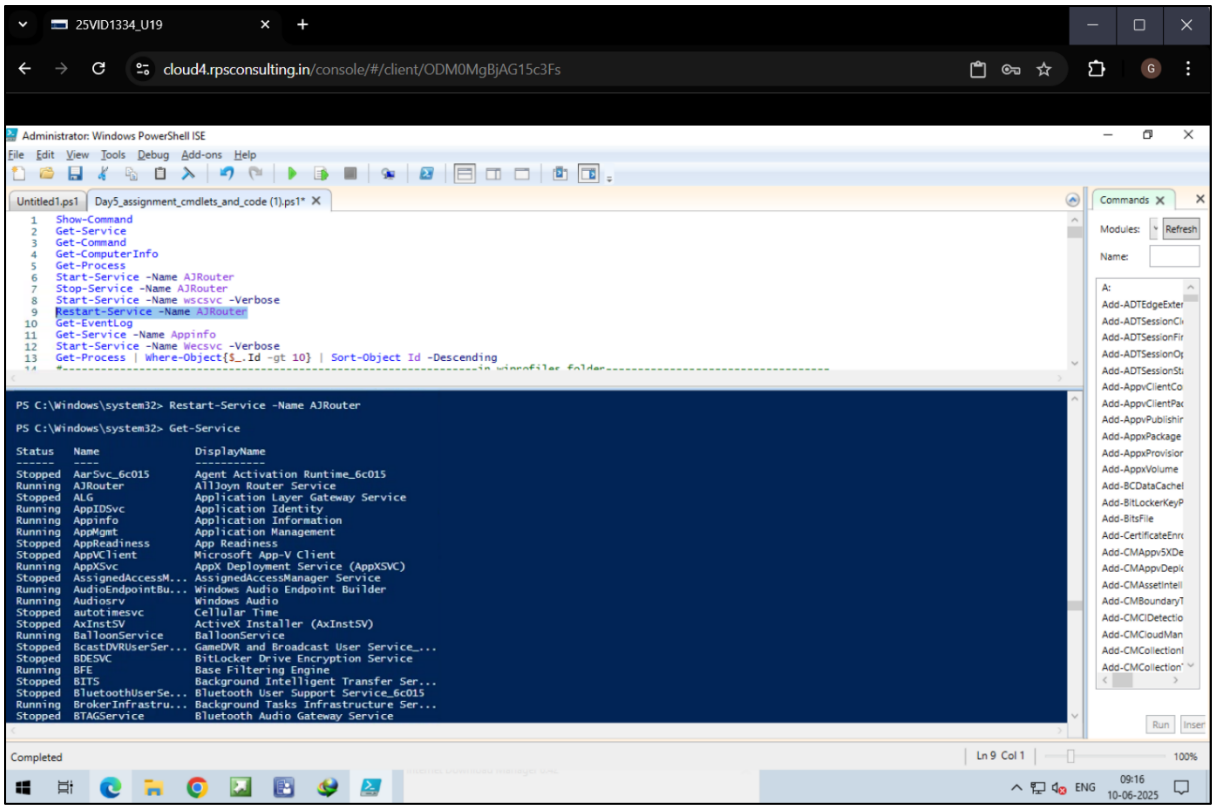
6. **Stop-Service:** Stops the **service** with the name parameter.

- **Stop-Service -Name AjRouter**



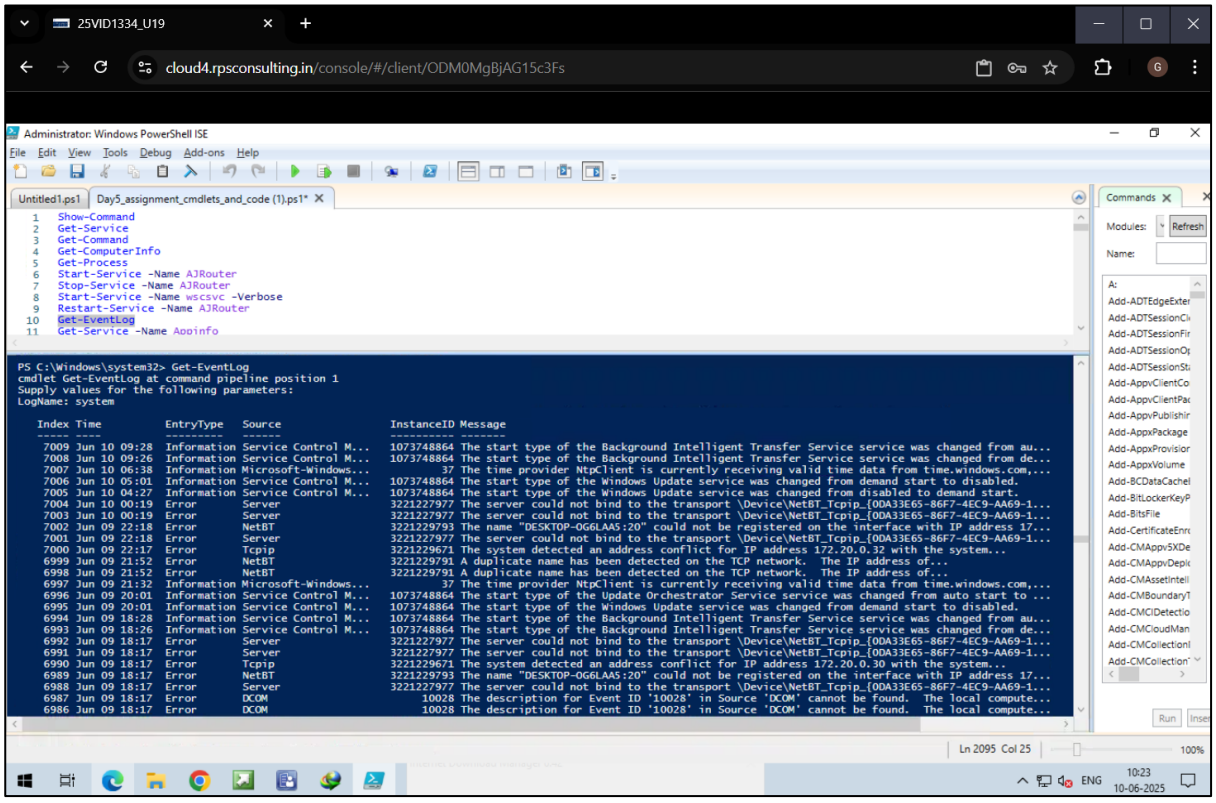
7. **Restart-Service:** Restarts the **service** with the name parameter

- **Restart-Service -Name AjRouter**



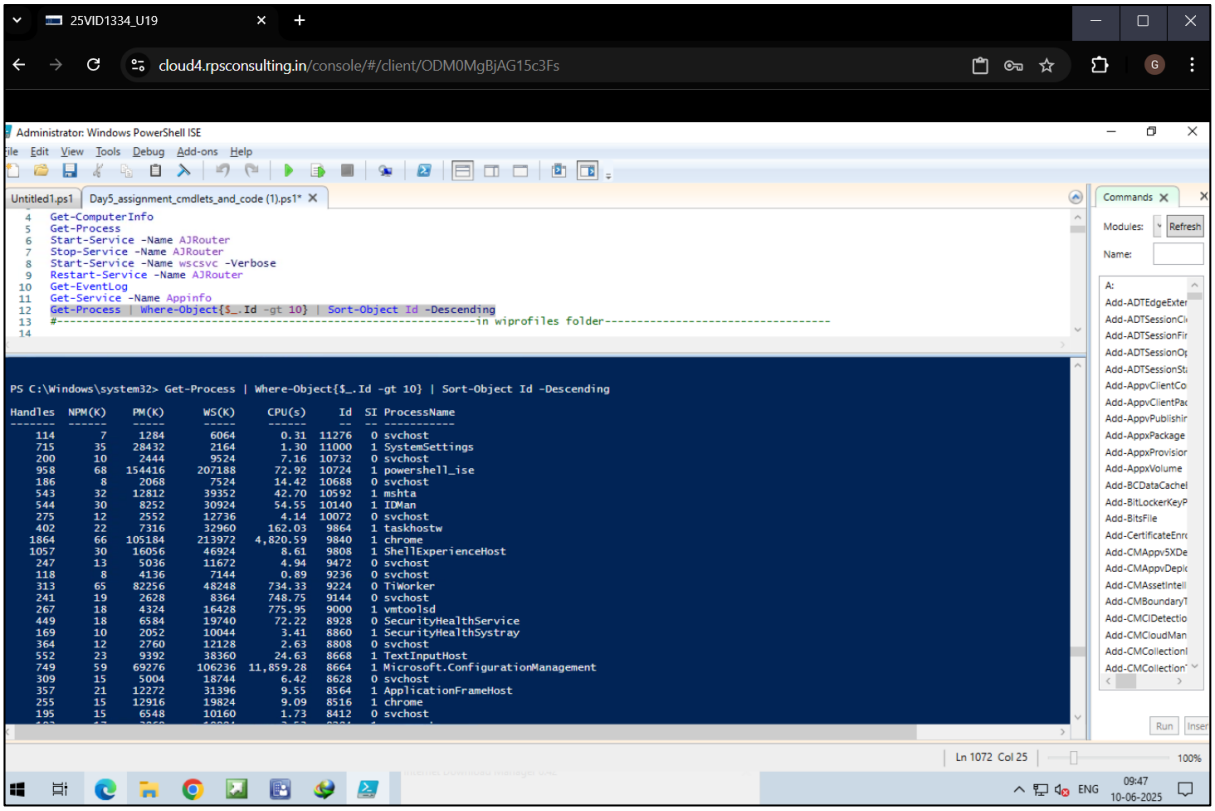
8. **Get-EventLog:** Get-EventLog retrieves **event log entries** from **Windows Event Viewer** useful for checking system, application, and security logs.(in the LogName give system)

- **Get-EventLog**



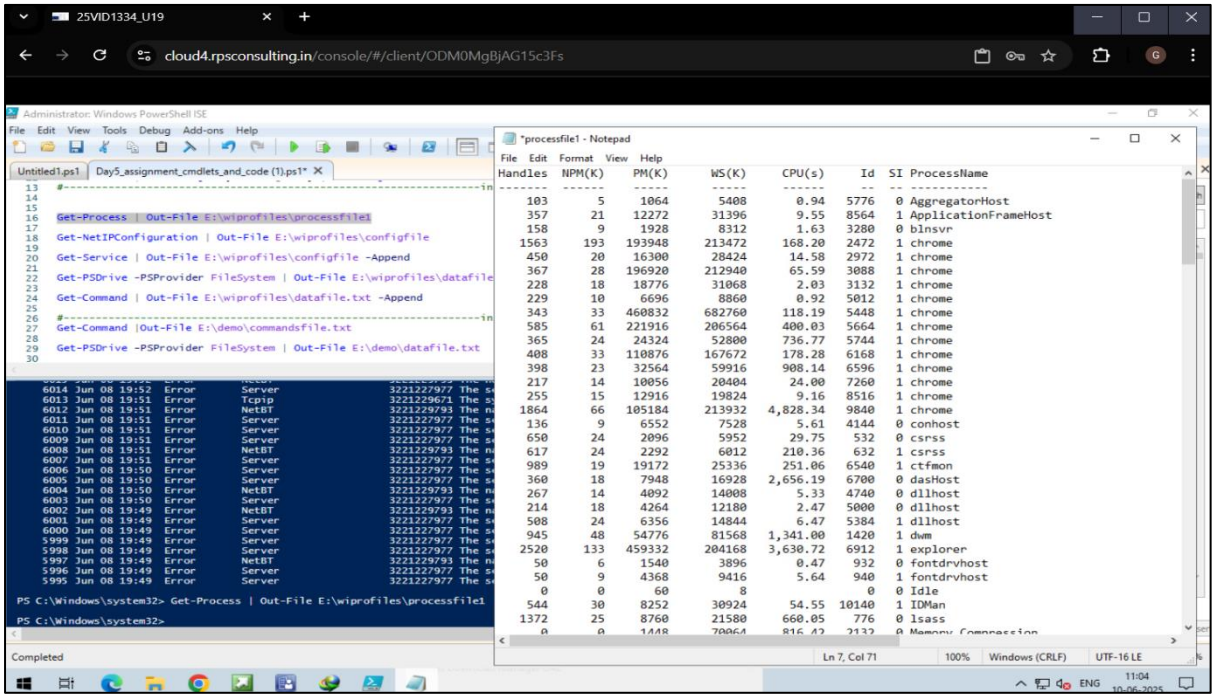
9. **Get-Process:** Retrives the currently running process and **where-object** helps to filter the process and **sort-object** helps to sort the process in descending.

- **Get-Process | Where-Object{$_.Id -gt 10} | Sort-Object Id -Descending**

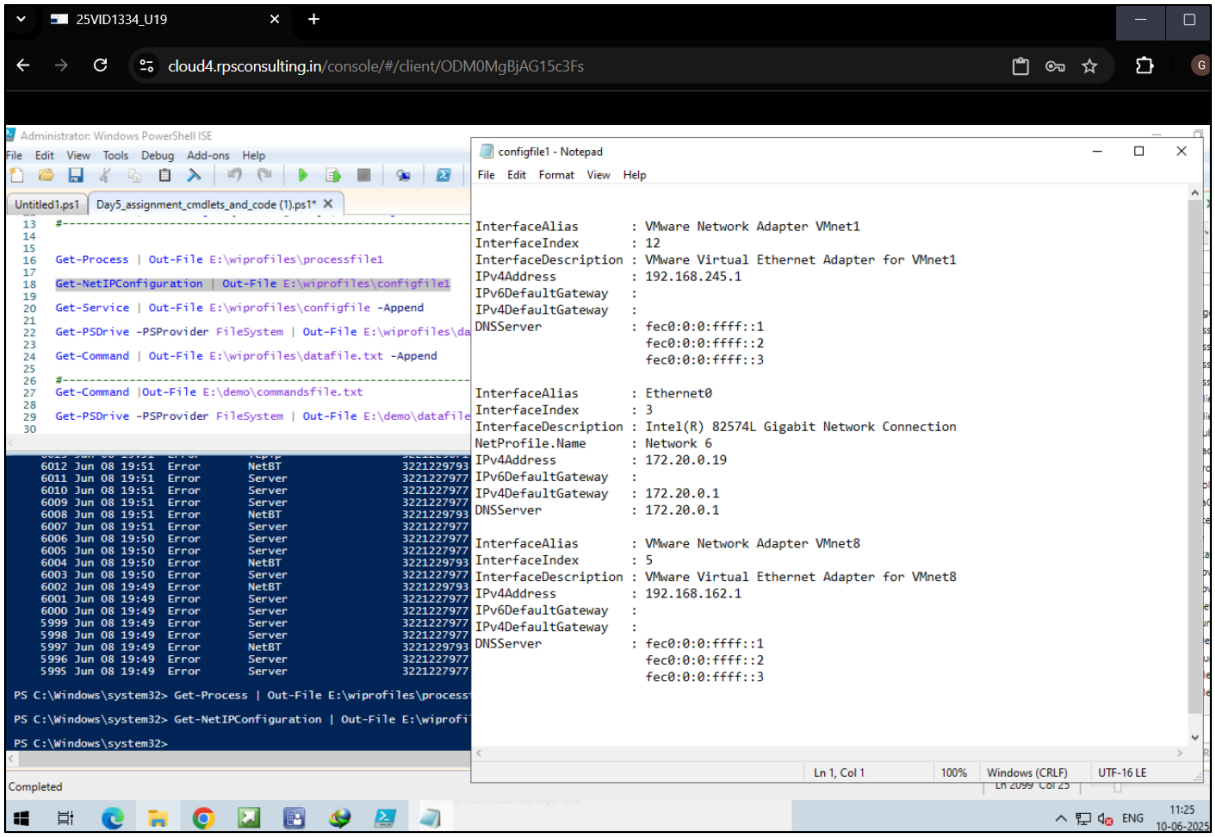


10. **Out-File:** The Out-File cmdlet in PowerShell is used to send output to a file.

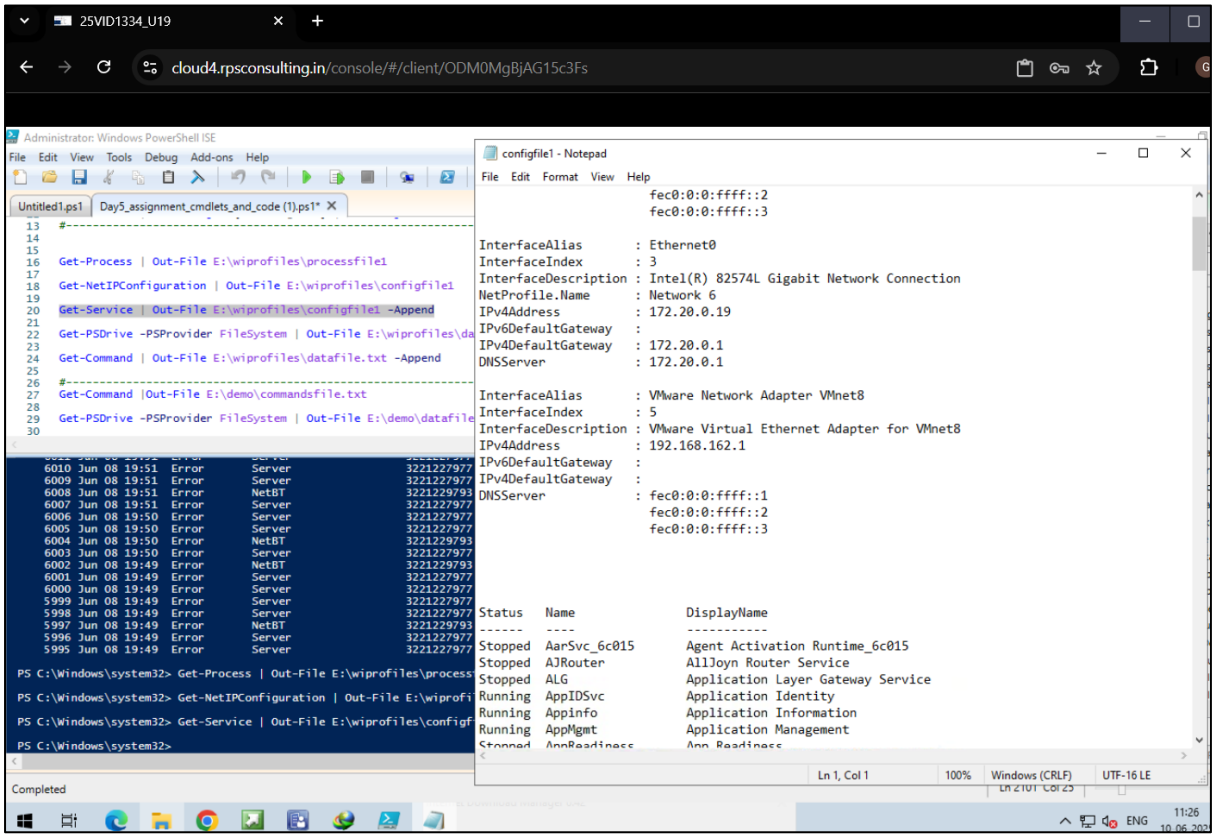
- **Get-Process | Out-File “E:\wiprofiles\processfiles1”**



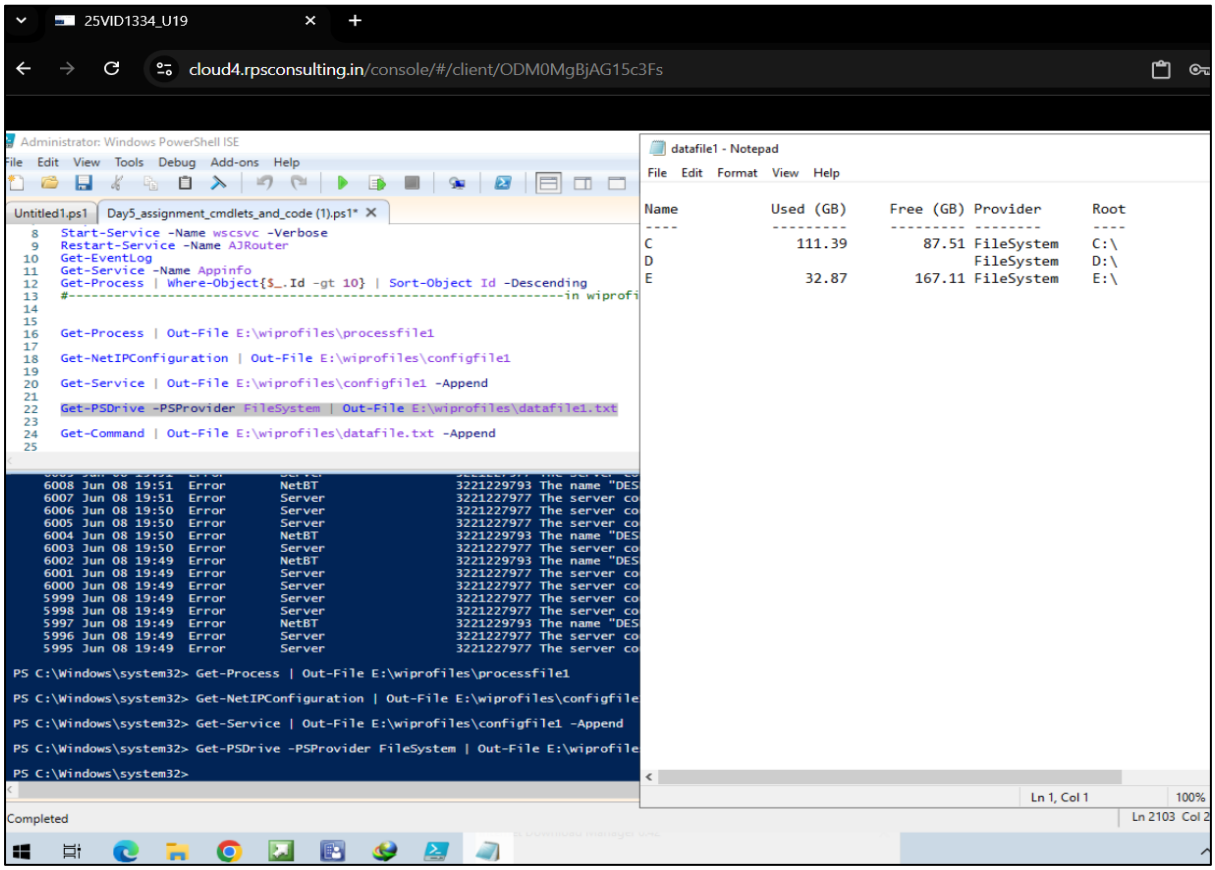
- **Get-NetIPConfiguration | Out-File E:\wiprofiles\configfiles**



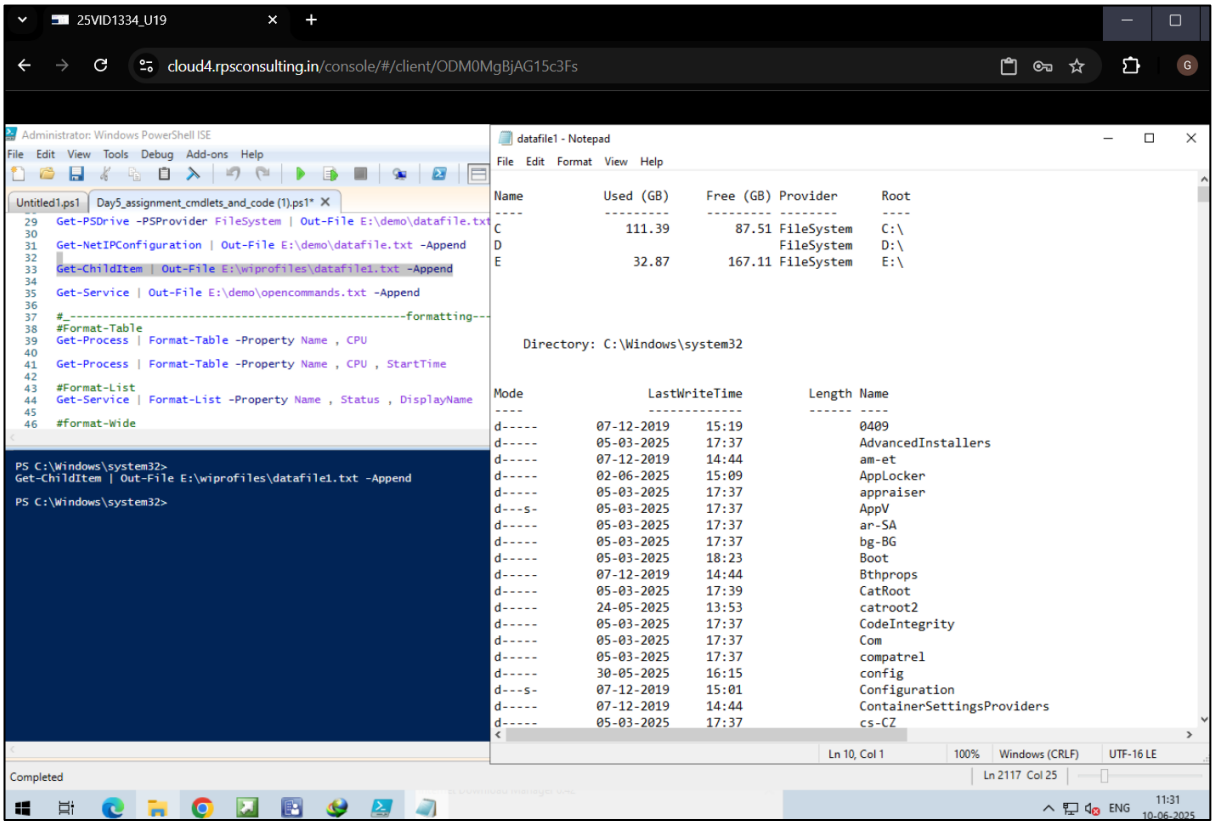
- **Get-Service |Out-File E:\wiprofiles\configfile1 -Append**



- **Get-PSDrive -PSProvider FileSystem | OutFile E:\wiprofiles\datafile1.txt**

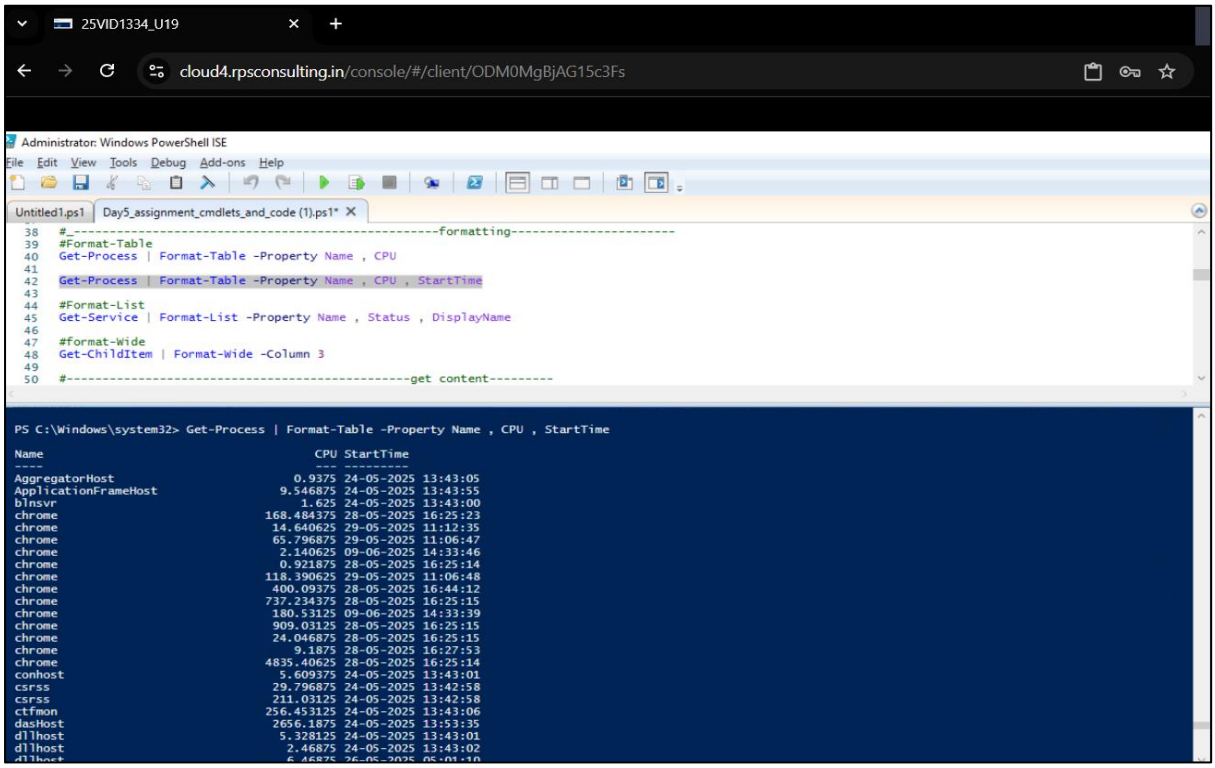


- **Get-ChildItem | OutFile E:\wiprofiles\datafile1.txt -Append**



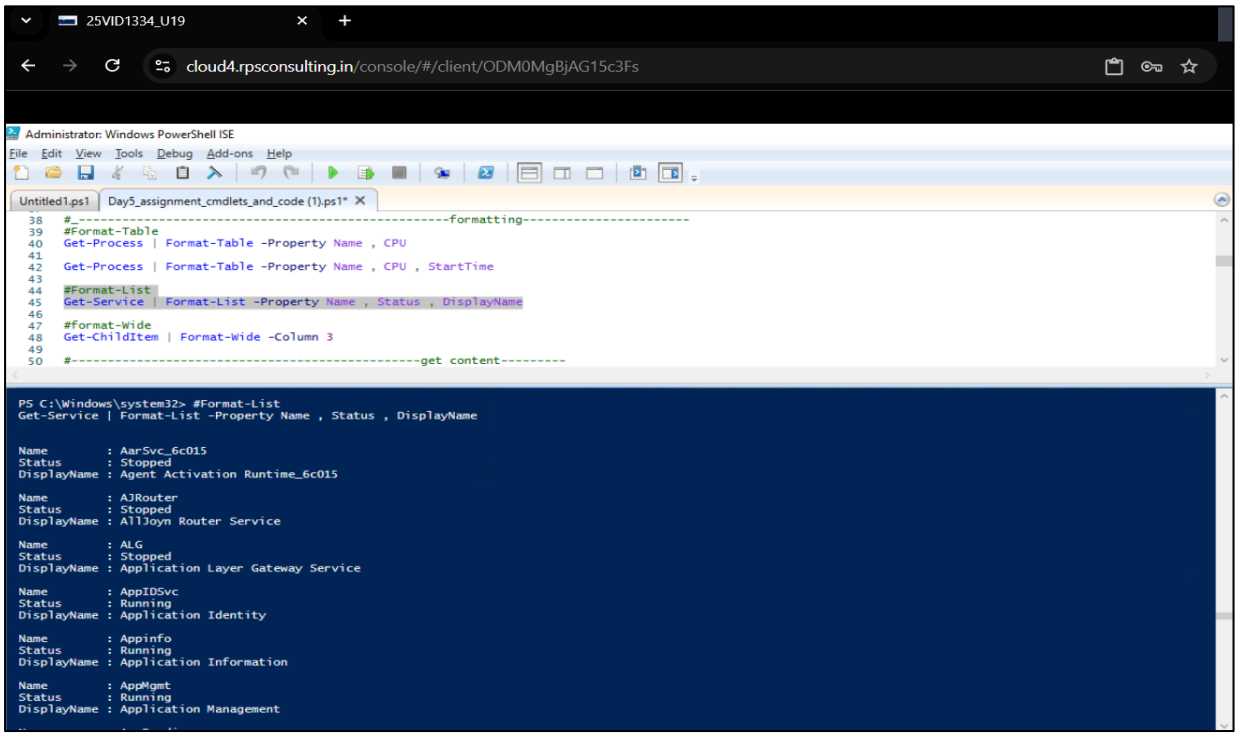
11. **Format-Table:** The Format-Table cmdlet in PowerShell formats the output of a command as a table with the selected properties of the object in each column.

- **Get-Process | Format-table -Property Name,CPU,StartTime**



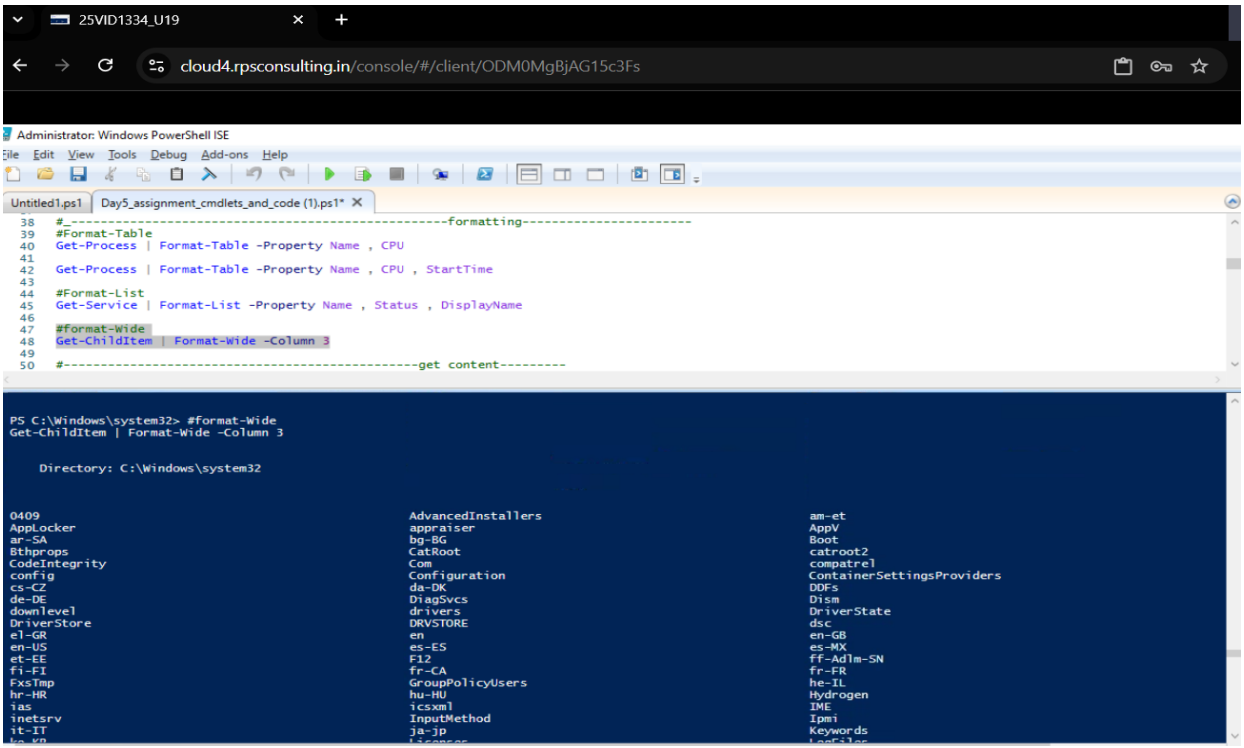
12. **Format-List:** The Format-List cmdlet in PowerShell formats the output of a command as a list of properties, displaying each property on a separate line.

- **Get-Service | Format-List -Property Name,Status,DisplayName**



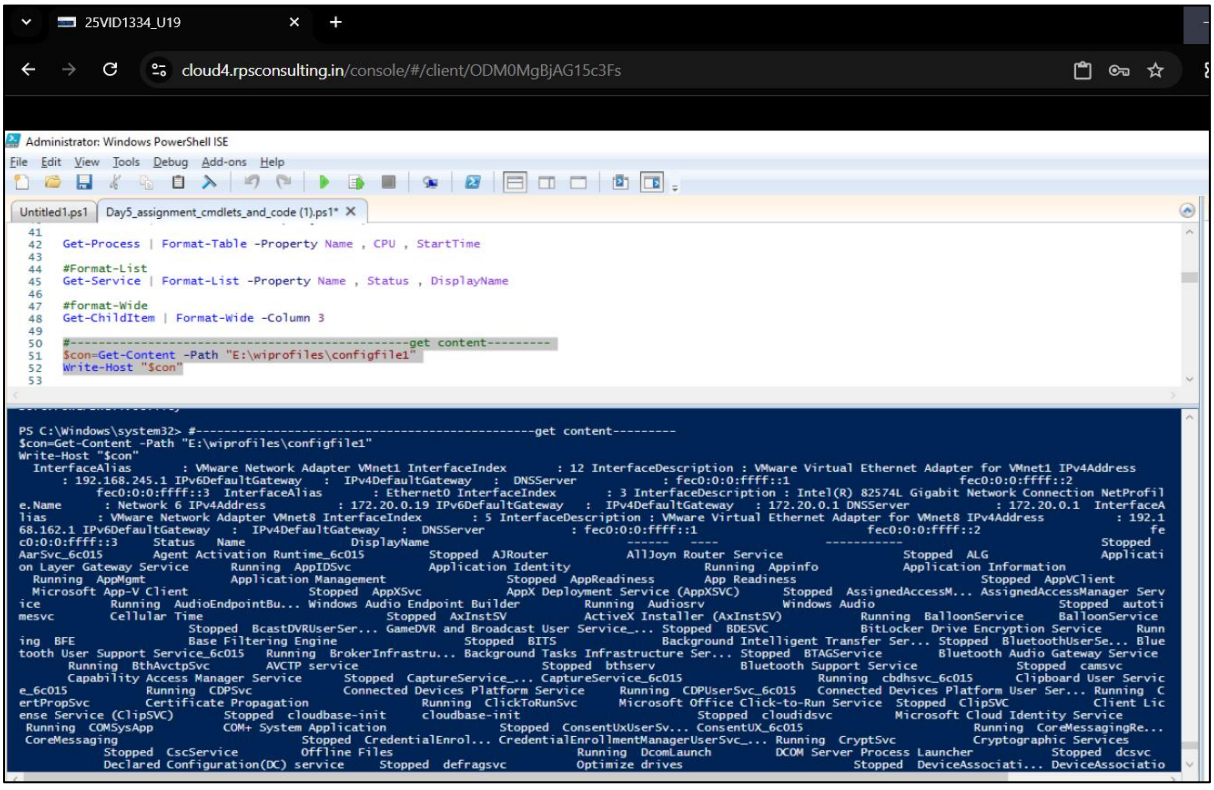
13. **Format-Wide:** The Format-Wide cmdlet in PowerShell formats objects as a wide table that displays only one property of each object.

- **Get-ChildItem | Format-Wide -Column 3**



14. **Get-Content:** Retrives the information from the file provided in the -Path parameter

- **Get-Content -Path "E:\wiprofiles\canfigfiles"**



Objects, Arrays, Variables, Scripting Constraints

➢ Variable

Variables are used to store values. They are denoted by $ symbol followed by name. They are not case sensitive. They can include letters, numbers and underscore. There is no need to declare datatypes of variables. However, we can use type casting if needed.

Code Example:

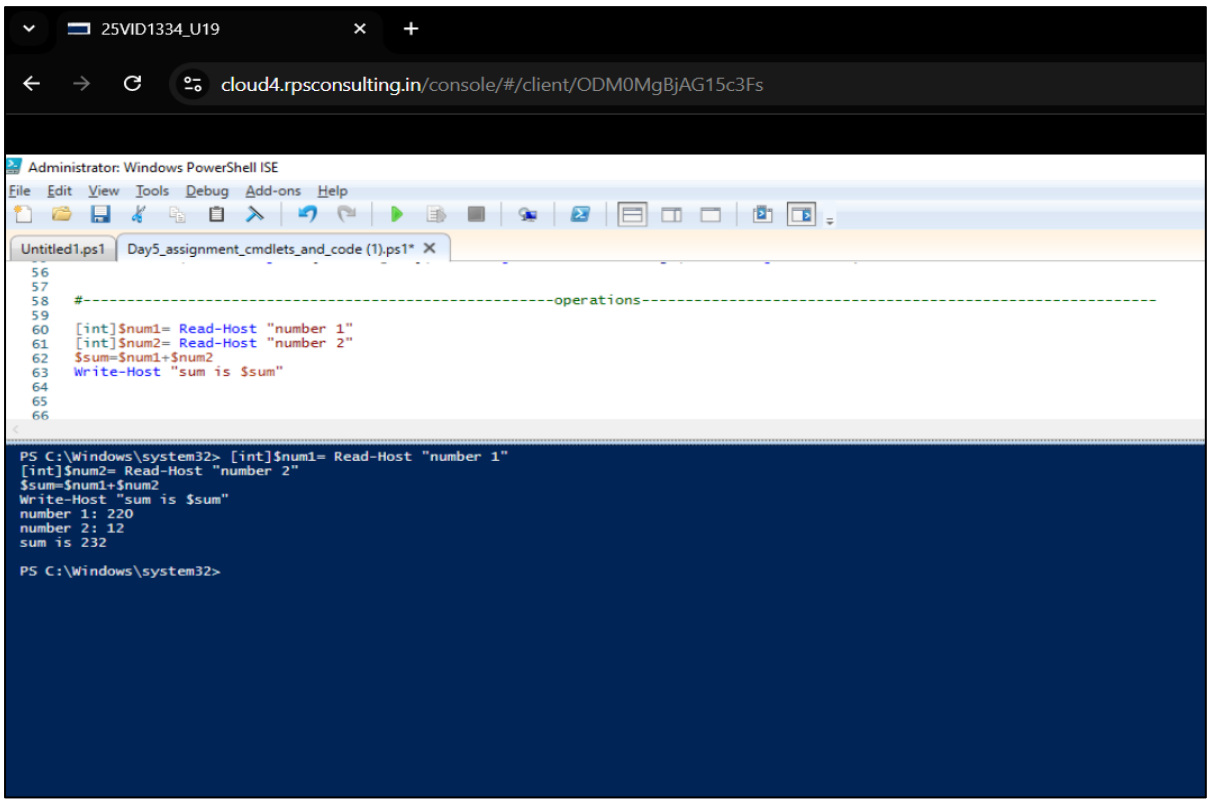
```
$name=”Ghanshyam”

$age=22

$isTrue=$true

[int]$number=”123”
```

➢ Input and Output Formatting

1. **Read-Host**: This prompts the user for input and stores it as a string.
2. **Write-Host**: Displays output directly to the console.

Code Example:

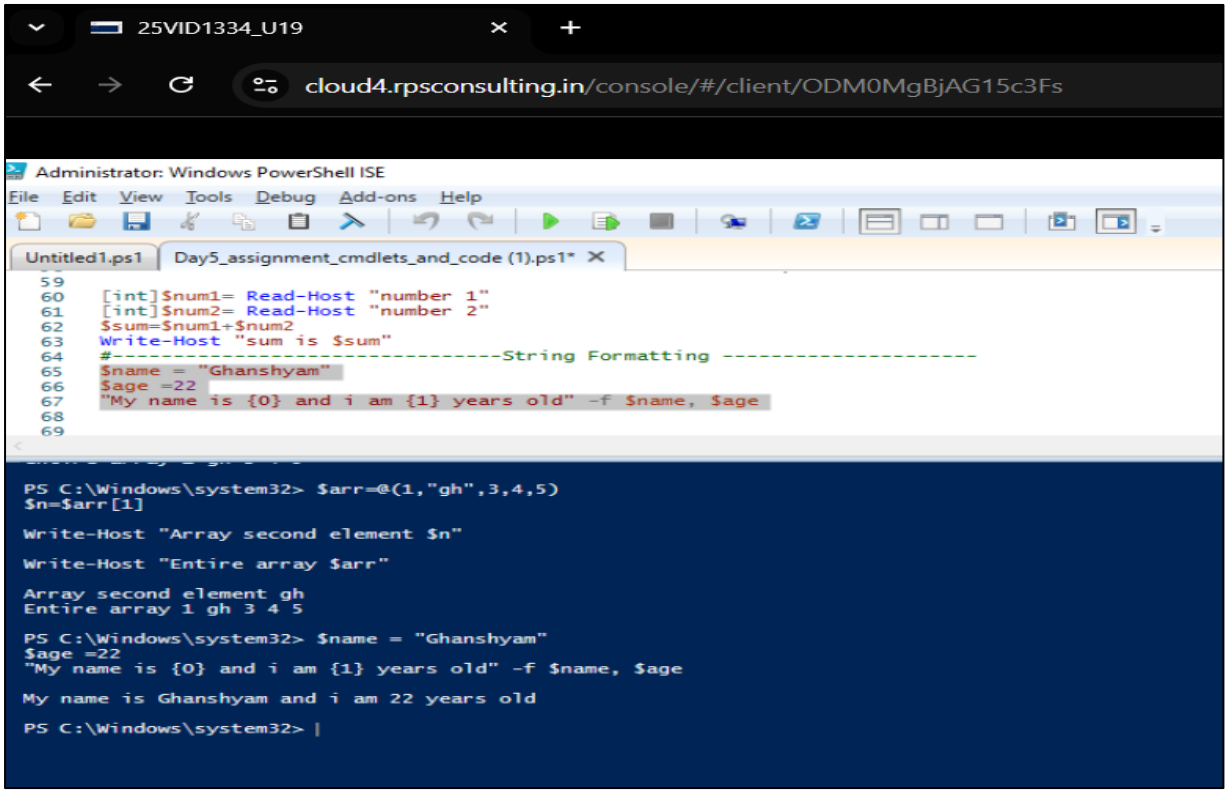
```
[int]$num1= Read-Host "number 1"

[int]$num2= Read-Host "number 2"

$sum=$num1+$num2

Write-Host "sum is $sum"
```



- **String Formatting:** -f format operator allows for composite formatting.

Code Example:

```
$name=”Ghanshyam”

$age=22

“My name is {0} and I am {1} years old” -f $name,$age
```

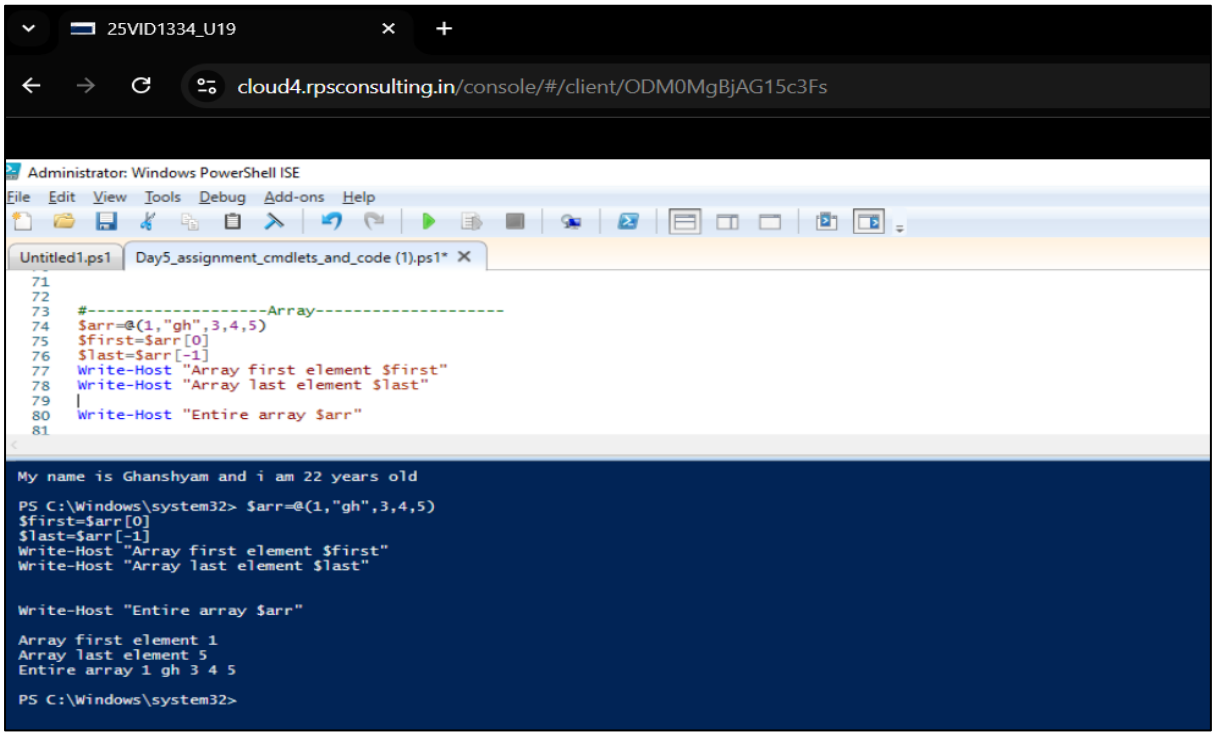


- **Array**

Arrays are used to store collections of items. An array can hold multiple objects of the same or different types. We can also declare them directly with @(elements).

Code Example:

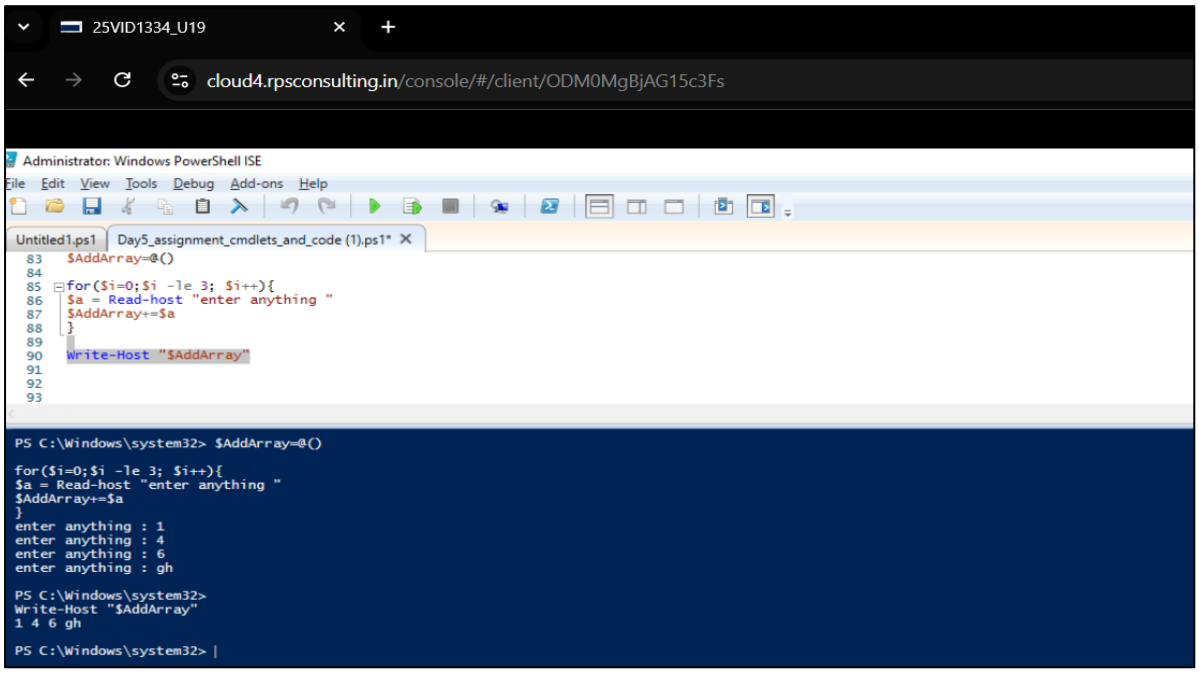
```
$arr=@(1,2,3,"gh")

$first=$arr[0]

$last=$arr[-1]

Write-Host "Array first element is $first"

Write-Host "Array last element is $last"

Write-Host "Entire array $arr"
```



Size of an array can be adjusted. We can add elements **using += operator**.

Code Example:

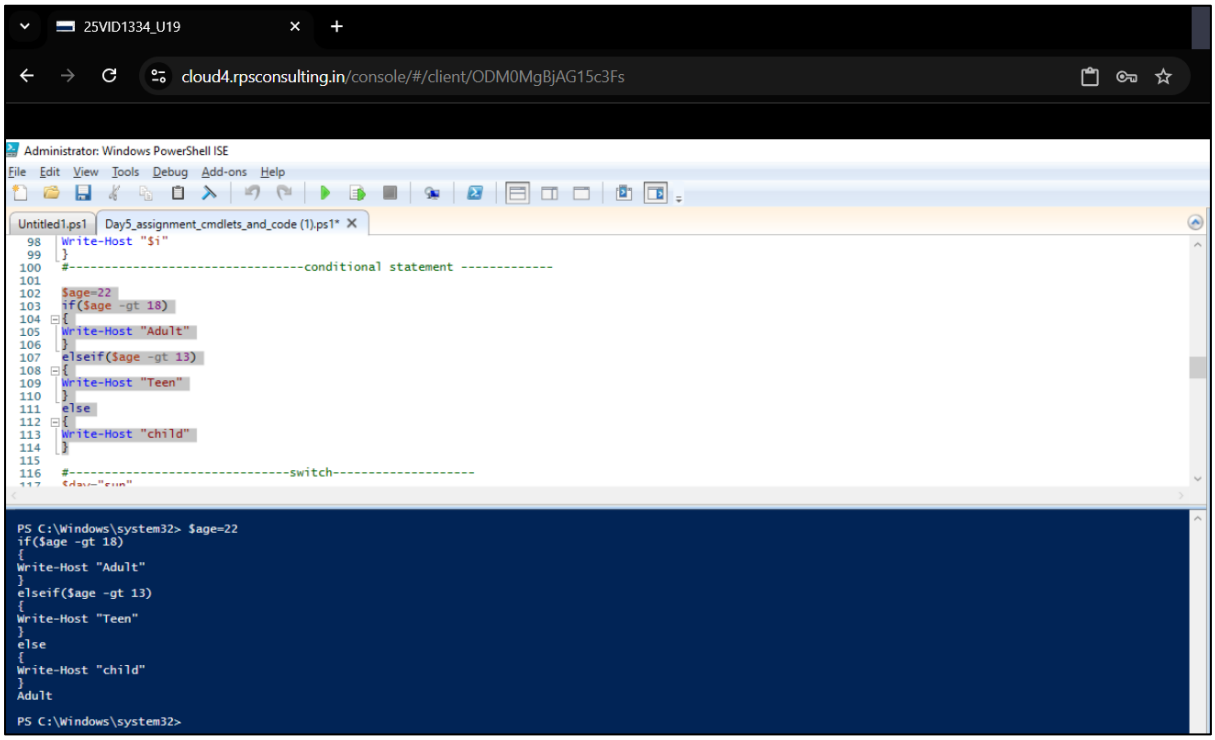
```
$AddArray=@()

for($i=0;$i -le 3; $i++){

$a=Read-Host "Enter anything"

$AddArray+=$a}
```



- **Conditional Statements: if, elseif, else:** These are used to execute code blocks based on conditions.

Code Example:

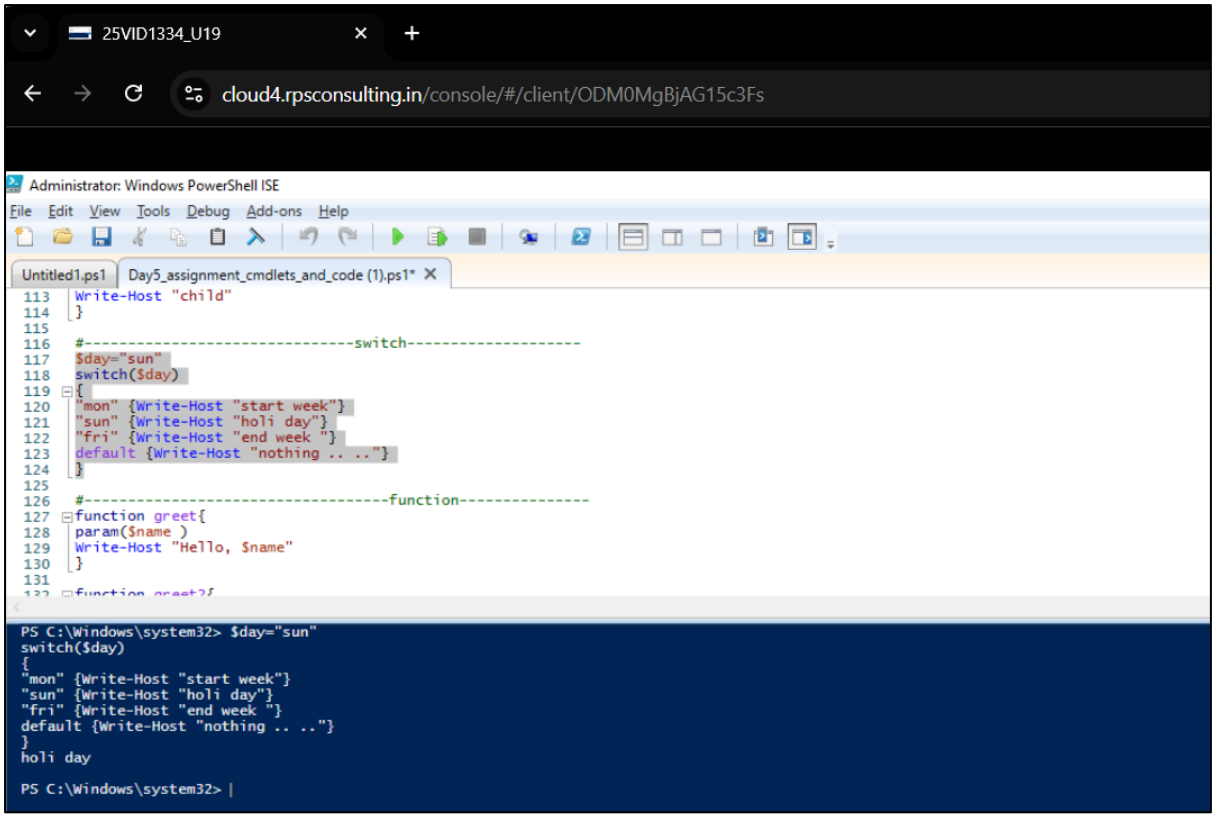
```
$age=25

if($age -ge 18){Write-Host “Adult”}

elseif($age -ge 13){Write-Host “Teenager”}

else{Write-Host “Child”}
```



- **Switch:** Switch case efficiently handles multiple conditions.

Code Example:

```
$day="sun"

switch($day){

"mon" {Write-Host "Start week"}

"sun" {Write-Host "holi day"}

"fri" {Write-Host "end week"}

default {Write-Host "nothing...."}

}
```

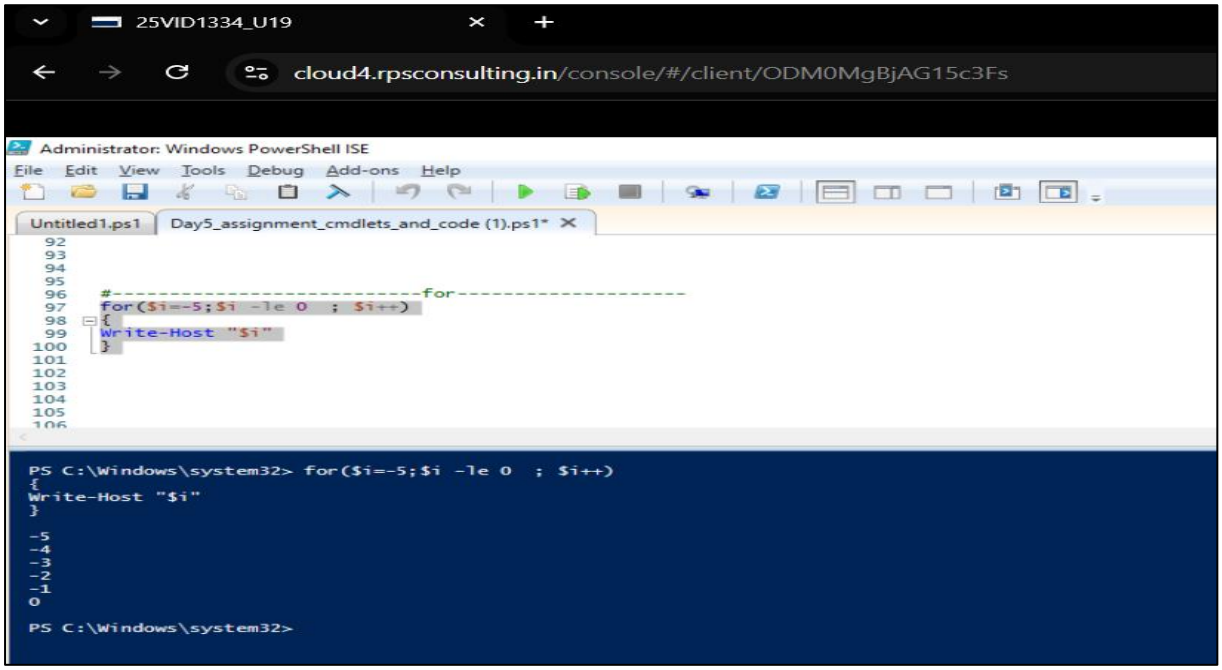


➢ Looping Statements

1. **for:** For loop iterates a specific number of times.

Code Example:

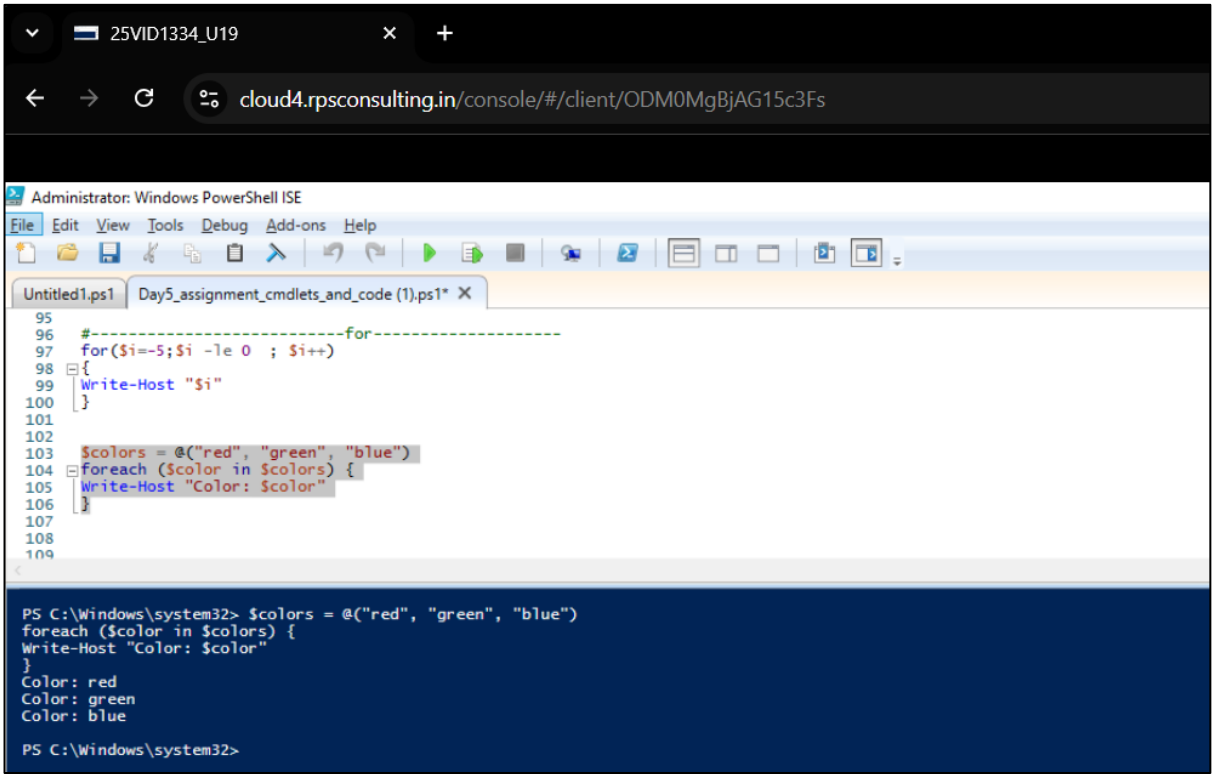
```
for($i=-5;$i -le 0;$i++){

Write-Host "$i"

}
```



2. **foreach:** Foreach loop iterates through a collection.

Code Example:

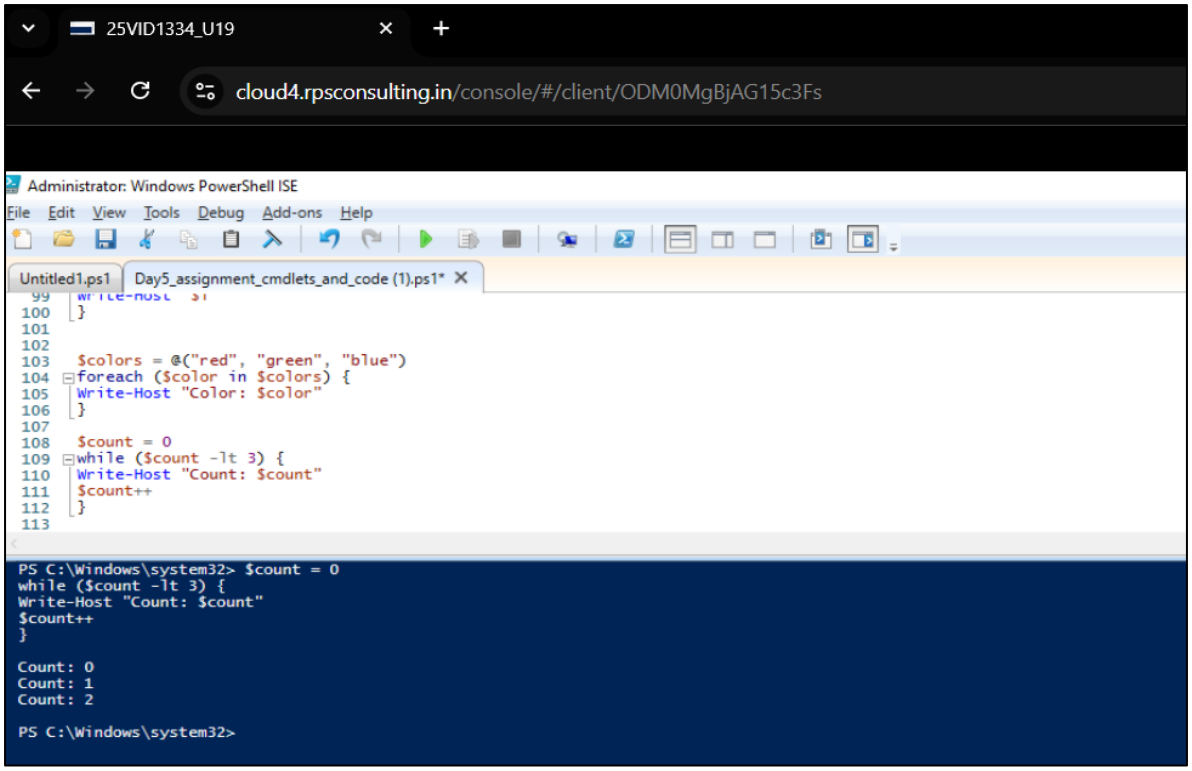
```
$colors=”red”,”green”,”blue”
foreach($elem in $colors){
Write-Host "$elem"
}
```



3. **while:** While loop repeats as long as a condition is true.

Code Example:

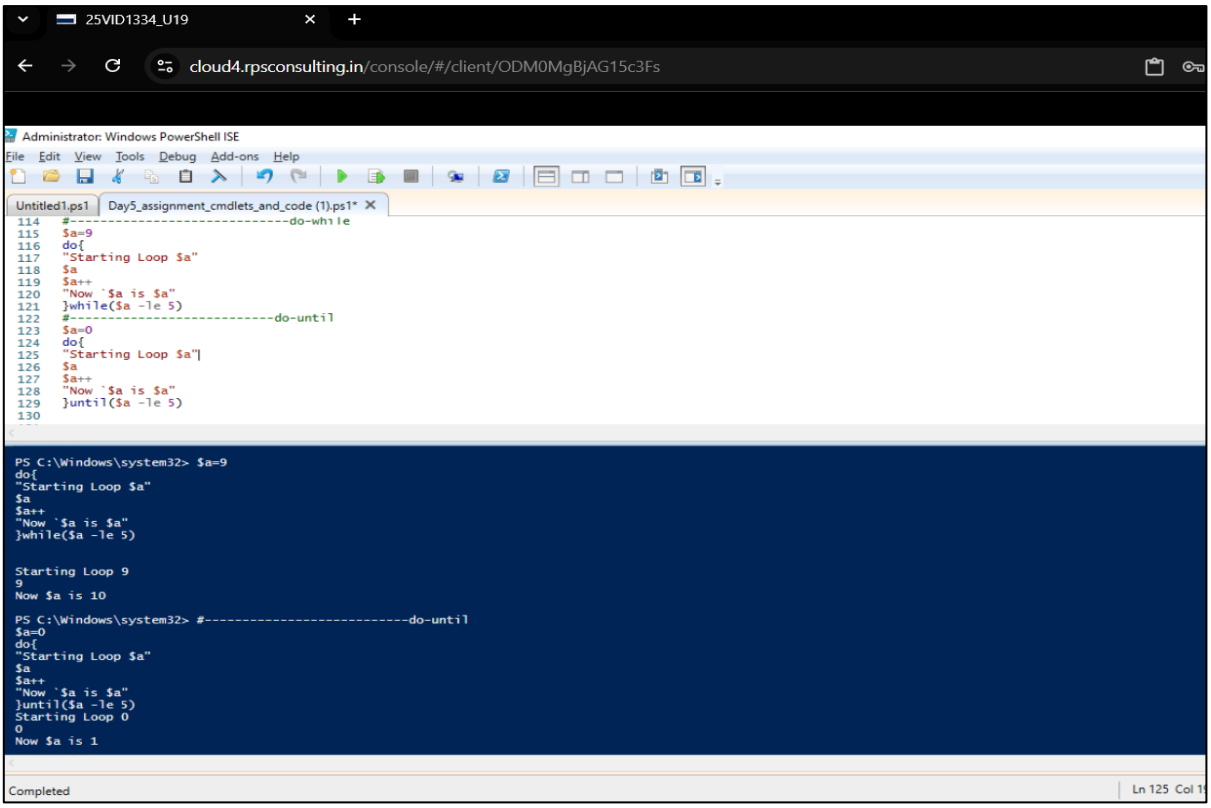
```
$count=0
while($count -lt 3){
Write-Host "Count: $count"
$count++
}
```



4. **do-while** & **do-until:** They are similar to while but the condition is checked at the end.

Code Example:

```
$a=9
do{
"Starting Loop $a"
$a
$a++
"Now `$a is $a"
}while($a -le 5)

#----------------------------------do-until

$a=0

do{

"Starting Loop $a"

$a

$a++

"Now `$a is $a"

}until($a -le 5)
```



- **Functions:** PowerShell functions are blocks of code designed to perform specific tasks making scripts more modular, reusable and easier to maintain. To define a basic function, we use the function keyword followed by the function name and a block of code enclosed in curly braces { }. To invoke or call a function, simply use the function's name. Parameters can be added to the function declaration to make them more flexible. The keyword param is used to declare all parameters in the first function statement. To invoke a function with arguments, the values are passed to the function as parameters or list arguments after the function call.

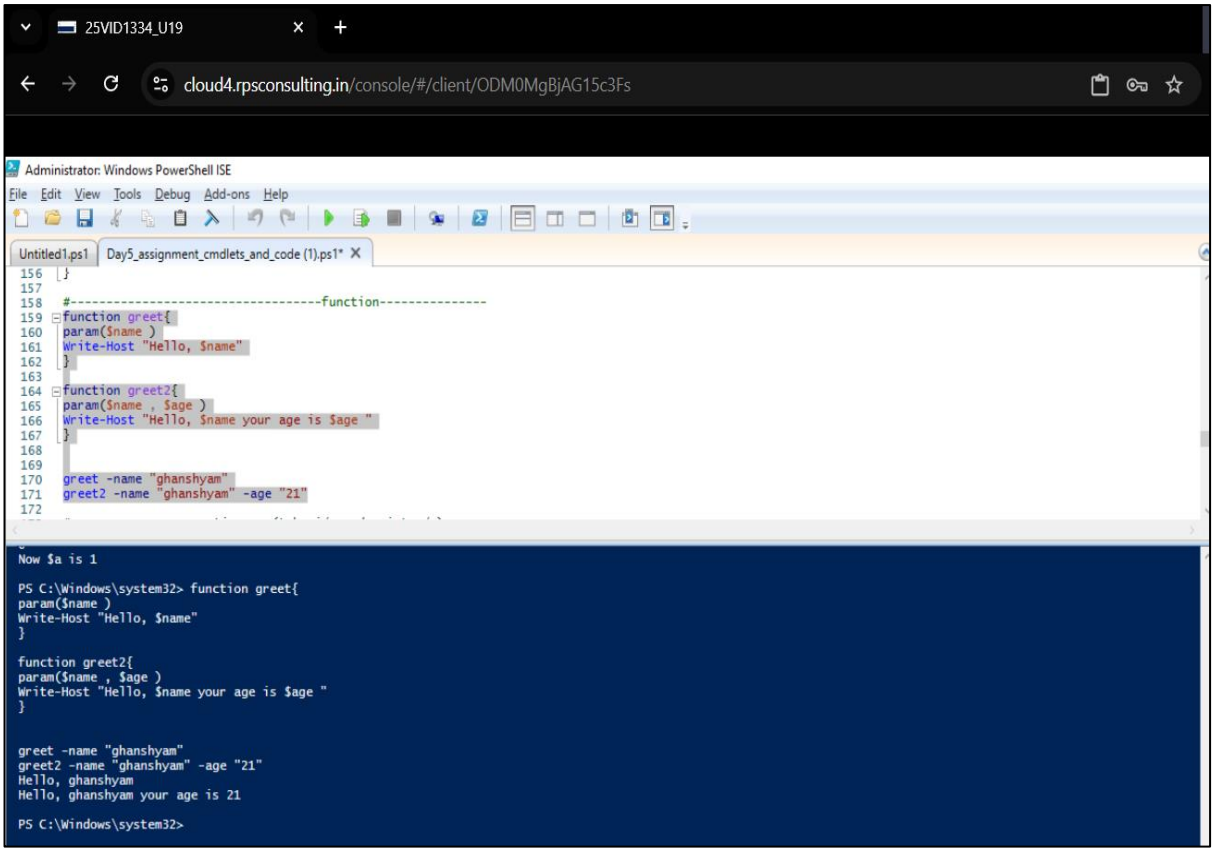
Code Example:

```
function greet {
param($name)
Write-Host "Hello, $name"

}
```

- greet -name "ghanshyam"

```
 function greet2 {
param($name, $age)
Write-Host "Hello, $name your age is $age"

}
```

- greet -name "ghanshyam" -age 21



- **Comment**: PowerShell allows for two types of comments: single-line and multi-line (block) comments. Single-line comments are created using the hash symbol (#) whereas Multi-line comments are created using the <#.....#> delimiters.

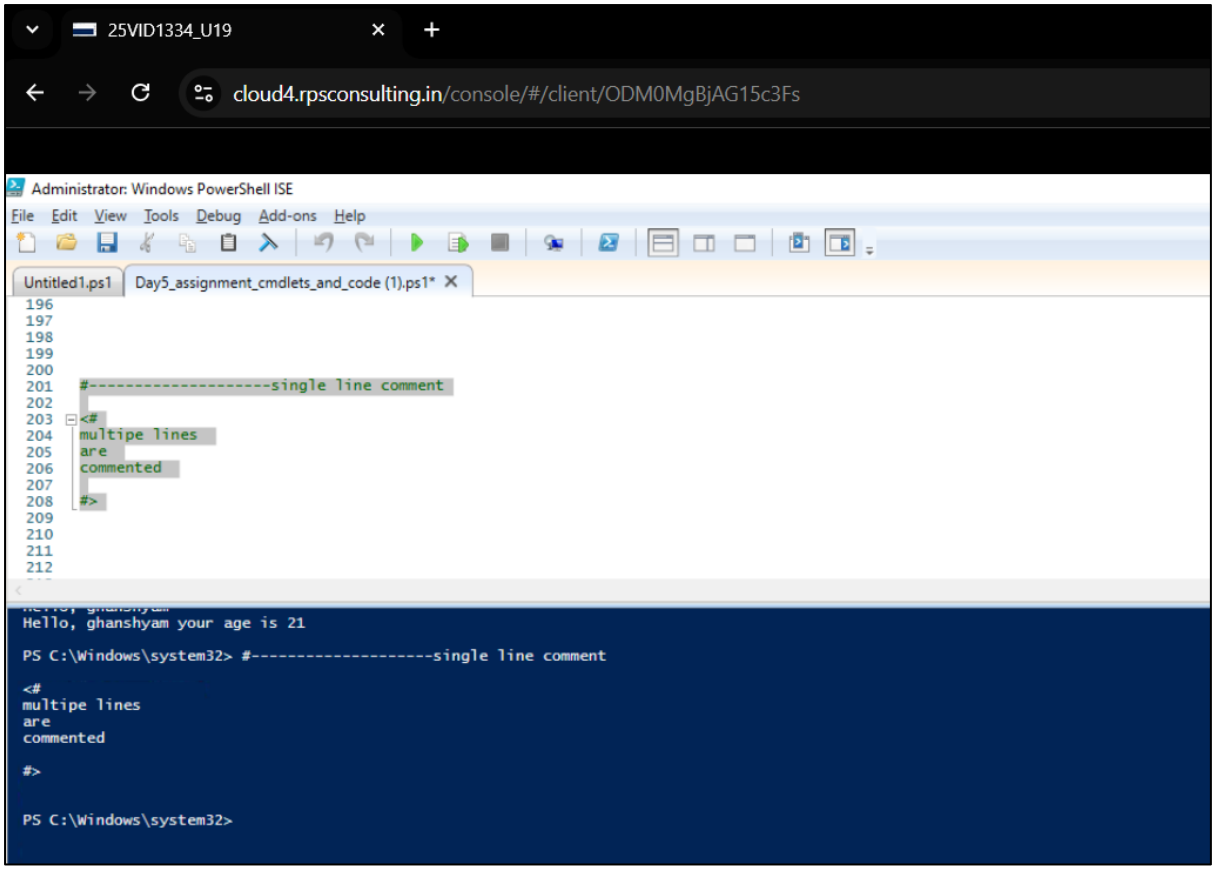
Code Example:

Single-line comment

```
#heyyyyyyyyyyyyyyyy
```

Multi-line comment

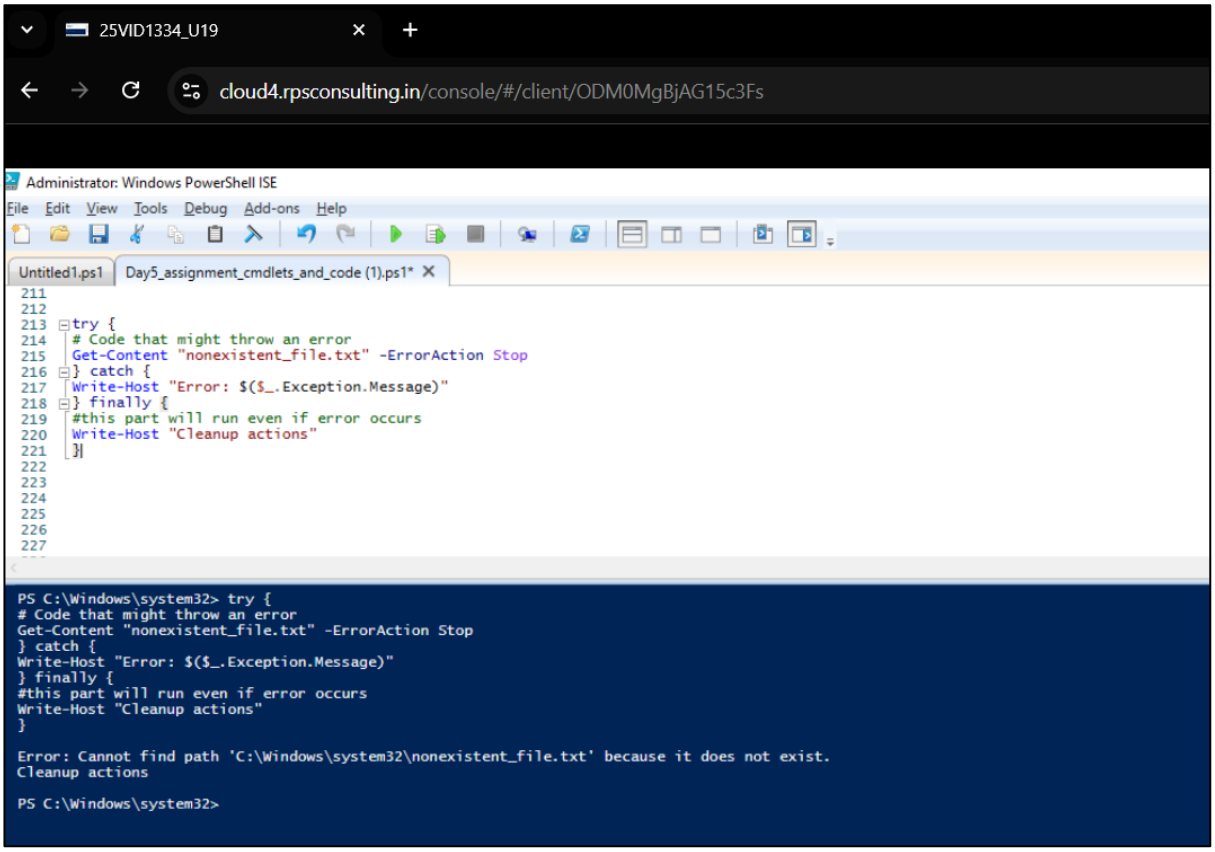
```
<#
First line
Second line
.
.
N lines
#>
```



- **Error Handling:** try, catch, finally blocks to manage exceptions

Code Example:

```
try {
 # Code that might throw an error
 Get-Content "nonexistent_file.txt" -ErrorAction Stop
}
catch {
Write-Host "Error: $($_.Exception.Message)"
}
finally {
#This block will run even if error occurs
 Write-Host "Cleanup actions"
}
```


25VID1334_U19
cloud4.rpsconsulting.in/console/#/client/ODM0MgBjAG15c3Fs
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Untitled1.ps1
Day5_assignment_cmdlets_and_code (1).ps1*
211
212
213 try {
214 # Code that might throw an error
215 Get-Content "nonexistent_file.txt" -ErrorAction Stop
216 } catch {
217 Write-Host "Error: $($_.Exception.Message)"
218 } finally {
219 #this part will run even if error occurs
220 Write-Host "Cleanup actions"
221 }
222
223
224
225
226
227
PS C:\Windows\system32> try {
# Code that might throw an error
Get-Content "nonexistent_file.txt" -ErrorAction Stop
} catch {
Write-Host "Error: $($_.Exception.Message)"
} finally {
#this part will run even if error occurs
Write-Host "Cleanup actions"
}
Error: Cannot find path 'C:\Windows\system32\nonexistent_file.txt' because it does not exist.
Cleanup actions
PS C:\Windows\system32>