Learning Power Shell

QUALITY THOUGHT
INFO SYSTEMS (INDIA) PVT. LTD

Commitment

"To learn a skill it must be used, to learn a language it must be spoken."

- Joshua B. Jones

- "The more that you read, the more things you will know. The more that you learn, the more places you'll go."

- Dr. Seuss

Lab Setup

- Windows 10 Pro version

Getting Started with Powershell

What is Powershell

- PowerShell is a mixture of a command line, a functional programming language, and an object-oriented programming language. PowerShell is based on Microsoft .NET, which gives it a level of open flexibility that was not available in Microsoft's scripting languages (such as VBScript or batch) before this.
- PowerShell is an explorer's scripting language. With built-in help, command discovery, and with access to much of the .NET Framework, it is possible to dig down through the layers

Quick Reference

Powershell

Cheatsheet:

http://ramblingcookiemonster.github.io/images/Cheat-Sheets/powershell-basic-cheat-sheet2.pdf

QUALITY THOUGHT
INFO SYSTEMS (INDIA) PVT. LTD

Comments

Line comment	#	Copy `# This is a line comment`

Block comment	<#
	#>
		Copy
		`<#`
		`This is a block or multi-line comment`
		`#>`

Special Characters

Statement separator	;	Copy `Get-Command Get-Process; Get-Command Get-Help`

Call operator	&	Copy
		`& 'Get-Process'  # Invoke the string as a command`
		`& { Get-Process -Id $PID }  # Invoke the script block`

Dot-source operator	.	Copy `. C:\script.ps1  # Execute the script in the current scope (instead of its own scope)`

Tick in PowerShell

- A tick may be used as a line continuation character. Consider the following example:

```
'one' -replace 'o', 't' `
      -replace 'n', 'w' `
      -replace 'e', 'o'
```

Description	String	ASCII character code
Null	`0	0
Bell sound	`a	7
Backspace	`b	8
New page form feed	`f	12
Line feed	`n	10
Carriage return	`r	13
Horizontal tab	`t	9
Vertical tab	`v	11

Common Operators

Equal to	`-eq`	Copy
		`1 -eq 1 # Returns $true`
		`1 -eq 2 # Returns $false`

Not equal to	`-ne`	Copy
		`1 -ne 2 # Returns $true`
		`1 -ne 1 # Returns $false`

And	`-and`	Copy
		`$true -and $true # Returns $true`
		`$true -and $false # Returns $false`
		`$false -and $false # Returns $false`

Or	`-or`	Copy
		`$true -or $true # Returns $true`
		`$true -or $false # Returns $true`
		`$false -or $false # Returns $false`

Addition and concatenation	`+`	Copy
		`1 + 1 # Equals 2`
		`"one" + "one" # Equals oneone`

Subexpression operator	`$( )`	Copy
		`"Culture is $($host.CurrentCulture)"`
		`"Culture is $(Get-Culture)"`

Creating arrays and hashtables

Using the array operator	`@()`	Copy
		`$array = @()  # Empty array`
		`$array = @(1, 2, 3, 4)`

Implicit array	`Value1, Value2, Value3`	Copy
		`$array = 1, 2, 3, 4`
		`$array = "one", "two", "three", "four"`

Using the `hashtable` operator	`@{}`	Copy
		`$hashtable = @{}  # Empty hashtable`
		`$hashtable = @{Key1 = "Value1"}`
		`$hashtable = @{Key1 = "Value1"; Key2 = "Value2"}`

Strings

Expanding string	" "	Copy "Value" $greeting = "Hello"; "$greeting World" # Expands variable
Expanding here-string	@" "@	Copy $one = 'One' @" Must be opened on its own line. This string will expand variables like $var. Can contain other quotes like " and '. Must be closed on its own line with no preceding white space. "@
Non-expanding string	' '	Copy 'Value' '$greeting World' # Does not expand variable

Strings (contd)

Non-expanding here-string	@’ ‘@	Copy @’ Must be opened on its own line. This string will not expand variables like $var. Can contain other quotes like “ and ‘. Must be closed on its own line with no preceding white space. ‘@
Quotes in strings	“ `” “ “ ““ “ ‘ `’ ‘ ‘ ‘‘ ‘	Copy “Double-quotes may be escaped with tick like `”.” “Or double-quotes may be escaped with another quote ““.” ‘Single-quotes may be escaped with tick like `’.’ ‘Or single-quotes may be escaped with another quote like ‘‘.’

Common reserved variables

Errors	`$Error`	Copy `$Error[0]    # The last error`

Formats the enumeration limit. Dictates the number of elements displayed for objects with properties based on arrays.
The default is `4` .
	`$FormatEnumerationLimit`	Copy
		`$object = [PSCustomObject]@{`
		`Array = @(1, 2, 3, 4, 5)`
		`}`
		`$object    # Shows 1, 2, 3, and 4`
		`$formatenumerationlimit = 1`
		`$object    # Shows 1`

Holds data of the current PowerShell host.	`$Host`	Copy
		`$host`
		`$host.UI.RawUI.WindowTitle`

The matches found when using the `-match` operator.	`$Matches`	Copy
		`'text' -match '.*'`
		`$matches`

Common reserved variables

The output field separator.
The default is a single space.
Dictates how arrays are joined when included in an expandable string.
	`$OFS`	Copy
		`$arr = 1, 2, 3, 4`
		`"Joined based on OFS: $arr"`
		`$ofs = ', '`
		`"Joined based on OFS: $arr"`

Current PowerShell process ID.	`$PID`	Copy `Get-Process -Id $PID`

Holds the path to each of the `profile` files.	`$PROFILE`	Copy
		`$profile.AllUsersAllHosts`
		`$profile.AllUsersCurrentHost`
		`$profile.CurrentUserAllHosts`
		`$profile.CurrentUserCurrentHost`

PowerShell version information.	`$PSVersion Table`	Copy `$PSVersionTable.PSVersion`
Present working directory.	`$PWD`	Copy `$PWD.Path`

Quick commands and hot keys

`ise` `ise <file>`	Opens PowerShell ISE. Opens a file with ISE if a filename is given.
`code` `code <file or folder>`	If Visual Studio Code is installed (and in `%PATH%`). Opens the VS Code. Opens a file or folder with the VS Code.
`Get-History` `history`	Shows command history for the current session.
`<Text><Tab>`	Autocompletes in context. `Tab` can be used to complete command names, parameter names, and some parameter values.
`#<Text><Tab>`	Autocompletes against history (beginning of the line). Typing `#get-` and repeatedly pressing **Tab** will cycle through all commands containing `Get-` from your history.
`ii`	`ii` is an alias for the `invoke-item`. Opens the current directory in Explorer.
`start iexplore`	`start` is an alias for the start-process. Opens Internet Explorer.
`start <name> -verb runas`	Runs a process as administrator.

Powershell Playgrounds

- Powershell console

  Console is the default Powershell Terminal

- Powershell ISE :

Windows PowerShell ISE . **ISE** stands for Integrated Scripting Environment , and it is a graphical user interface that allows you to easily create different scripts without having to type all the commands in the command line

Command naming and discovery

- Commands in PowerShell are formed around verb and noun pairs in the form verb-noun.
- Verbs :
  - Get-Verb
  - Complete list of Verbs https://msdn.microsoft.com/en-us/library/ms714428(v=vs.85).aspx
- Nouns:
  - The noun provides a very short description of the object the command is expecting to act on.
  - The noun part may be a single word, as is the case with Get-Process, New-Item, or Get-Help or more than one word, as seen with Get-ChildItem, Invoke-WebRequest, or Send-MailMessage.

Finding commands

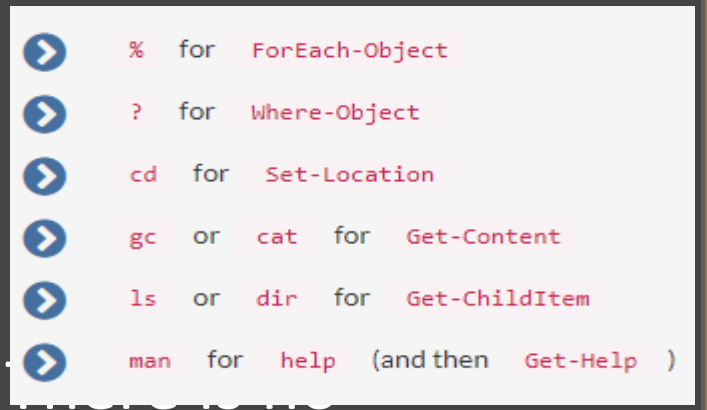
- The verb-noun pairing can make it a lot easier to find commands (without resorting to search engines).

```
PS> Get-Command Get-*Firewall* -Module NetSecurity

CommandType Name                                Version Source
----------- ----                                ------- ------
Function    Get-NetFirewallAddressFilter        2.0.0.0 NetSecurity
Function    Get-NetFirewallApplicationFilter    2.0.0.0 NetSecurity
Function    Get-NetFirewallInterfaceFilter      2.0.0.0 NetSecurity
Function    Get-NetFirewallInterfaceTypeFilter 2.0.0.0 NetSecurity
Function    Get-NetFirewallPortFilter           2.0.0.0 NetSecurity
Function    Get-NetFirewallProfile              2.0.0.0 NetSecurity
Function    Get-NetFirewallRule                 2.0.0.0 NetSecurity
Function    Get-NetFirewallSecurityFilter       2.0.0.0 NetSecurity
Function    Get-NetFirewallServiceFilter        2.0.0.0 NetSecurity
Function    Get-NetFirewallSetting              2.0.0.0 NetSecurity
```

Aliases

- An alias in PowerShell is an alternate name for a command. A command may have more than one alias.
- The list of aliases may be viewed using Get-Alias
- Get-Alias dir
- Get-Alias -Definition Get-ChildItem
- An alias does not change how a command is used. [illegible] practical difference between the following two following commands:
  - cd $env:TEMP
  - Set-Location $env:TEMP
- New aliases are created with the New-Alias
- “New-Alias grep -Value Select-String”



Parameters

- When viewing help for a command, we can see many different approaches to different parameters.
- Optional parameters
- Optional parameters are surrounded by square brackets. This denotes an optional parameter that requires a value (when used):
  - SYNTAX
  - Get-Process [-ComputerName <String[]>] ...
  - Optional positional parameters
  - It is not uncommon to see an optional positional parameter as the first parameter:
  - SYNTAX
  - Get-Process [[-Name] <String[]>] ...
  - In this example, we may use either of the following:
  - Get-Process -Name powershell
  - Get-Process powershell

Mandatory parameters

- A mandatory parameter must always be supplied and is written as follows:
  - SYNTAX
  - Get-ADUser -Filter <string> ...
- In this case, the Filter parameter must be written and it must be given a value. For example, to supply a Filter for the command, the Filter parameter must be explicitly written:
  - Get-ADUser -Filter { sAMAccountName -eq "SomeName" }

Switch parameters

- Switch parameters have no arguments (values); the presence of a switch parameter is sufficient; for example, Recurse is a switch parameter for Get-ChildItem:
  - SYNTAX
  - Get-ChildItem ... [-Recurse] ...
  - As with the other types of parameters, optional use is denoted by square brackets.
  - Switch parameters, by default, are false (not set). If a switch parameter is true (set) by default, it is possible to set the value to false using the notation, as shown in the following code:
  - Get-ChildItem -Recurse:$false

Parameter values

- Value types of arguments (the type of value expected by a parameter) are enclosed in angular brackets, as shown in the following example:
  - <string>
  - <string[]>
  - If a value is in the <string> form, a single value is expected. If the value is in the <string[]> form, an array (or list) of values is expected.
  - For example, Get-CimInstance accepts a single value only for the ClassName parameter: Get-CimInstance -ClassName Win32_OperatingSystem
  - Get-Process -Name powershell, explorer, smss

Confirm, WhatIf, and Force

- The Confirm, WhatIf, and Force parameters are used with commands that make changes (to files, variables, data, and so on). These parameters are often used with commands that use the verbs Set or Remove, but the parameters are not limited to specific verbs.
- Confirm:

```
PS> Set-Location $env:TEMP
New-Item IMadeThisUp.txt -Force
Remove-Item .\IMadeThisUp.txt -Confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\whoami\AppData\Local\Temp\IMadeThisUp.txt".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"):
```

WhatIf

- By employing PowerShell, and appending the -WhatIf switch, you get a preview of would happen without risking any damage.

```
PS> Set-Location $env:TEMP
New-Item IMadeThisUp.txt -Force
Remove-Item .\IMadeThisUp.txt -WhatIf

Confirm
Are you sure you want to perform this action?
What If: Performing the operation "Remove File" on target "C:\Users\whoami\AppData\Local\Temp\IMadeThisUp.txt".
```

Providers

- Providers in PowerShell present access to data that is not normally easily accessible. There are providers for the filesystem, registry, certificate store, and so on. Each provider arranges data so that it resembles a filesystem.

```
PS> Get-PSProvider

Name           Capabilities                       Drives
----           ------------                       ------
Registry       ShouldProcess, Transactions        {HKLM, HKCU}
Alias          ShouldProcess                      {Alias}
Environment    ShouldProcess                      {Env}
FileSystem     Filter, ShouldProcess, Credentials {C, D}
Function       ShouldProcess                      {Function}
Variable       ShouldProcess                      {Variable}
Certificate    ShouldProcess                      {Cert}
WSMan          Credentials                        {WSMan}
```

Drives using providers

- The output from Get-PSProvider shows that each provider has one or more drives associated with it.
- As providers are presented as a filesystem, accessing a provider is similar to working with a drive. Let's look at the following example:

```
PS C:\> Set-Location Cert:\LocalMachine\Root

PS Cert:\LocalMachine\Root> Get-ChildItem

    Directory: Microsoft.PowerShell.Security\Certificate::LocalMachine\Root


Thumbprint                                Subject
----------                                -------
CDD4EEAE6000AC7F40C3802C171E30148030C072 CN=Microsoft Root Certif...
BE36A4562FB2EE05DBB3D32323ADF445084ED656 CN=Thawte Timestamping C...
A43489159A520F0D93D032CCAF37E7FE20A8B419 CN=Microsoft Root Author...
```

QUALITY THOUGHT
INFO SYSTEMS (INDIA) PVT. LTD

Modules

Powershell

What is module?

A module may be binary, script, dynamic, or manifest:

- **Binary module:** This is written in a language, such as C# or VB.NET, and then compiled into a library (DLL)
- **Script module:** This is a collection of functions written in the PowerShell language. The commands typically reside in a script module file (PSM1)
- **Dynamic module:** This does not have files associated with it. This is created using the `New-Module` command. The following command creates a very simple dynamic module that adds the `Get-Number` command:

What is the PowerShell Gallery?

- In February 2016, Microsoft made the PowerShell Gallery public.
- The PowerShell Gallery may be searched using https://powershellgallery.com
- Useful commands are Import-Module, Get-Module, Remove-Module, Install-Module

Working with Objects in PowerShell

Pipelines

Members

Enumerating and filtering

Selecting and sorting

Grouping and measuring

Comparing

Importing, exporting, and converting

Pipelines

- The pipeline is used to send output from one command into another command
- The object pipeline
  - Languages such as Batch scripting (on Windows) or Bash scripting (ordinarily on Linux or Unix) use a pipeline to pass text between commands. It is up to the next command to figure out what the text means.
  - PowerShell, on the other hand, sends objects from one command to another.
  - The pipe (|) symbol is used to send the standard output between commands.

```
Get-Process | Where-Object WorkingSet -gt 50MB
```

Members

- The Get-Member command
  - The Get-Member command is used to view the different members of an object. For example, it can be used to list all of the members of a process object

```
Get-Process -Id $PID | Get-Member
```

- Get-Member offers filters using its parameters (MemberType, Static, and View). For example, if we wished to view only the properties of the PowerShell process, we might run the following:

```
Get-Process -Id $PID | Get-Member -MemberType Property
```

Members – Accessing Properties

- Properties of an object in PowerShell may be accessed by writing the property name after a period. For example, the Name property of the current PowerShell process may be accessed by the following:

```
(Get-Process -Id $PID).Name
```

```
$process = Get-Process -Id $PID
$process.Name
```

```
(Get-Process -Id $PID).StartTime.DayOfWeek
```

Members – using methods

- Methods are called using the following notation in PowerShell:

```
<Object>.Method(Argument1, Argument2)
```

```
PS> $date = Get-Date "01/01/2010"
$date.ToLongDateString()
01 January 2010
```

Enumerating and filtering

- Enumerating, or listing, the objects in a collection in PowerShell does not need a specialized command. For example, if the results of Get-PSDrive were assigned to a variable, enumerating the content of the variable is as simple as writing the variable name and pressing Return:

```
PS> $drives = Get-PSDrive
$drives
Name     Used (GB) Free (GB) Provider   Root
----     --------- --------- --------   ----
Alias                        Alias
C           319.37   611.60  FileSystem  C:\
Cert                         Certificate \
Env                          Environment
...
```

`ForEach-Object` may be used where something complex needs to be done to each object.

`Where-Object` may be used to filter results.

The ForEach-Object command

- ForEach-Object is most often used as a loop (of sorts). For example, the following command works on each of the results from Get-Process in turn:

```
Get-Process | ForEach-Object {
    Write-Host $_.Name -ForegroundColor Green
}
```

- In the preceding example, a special variable, $_, is used to represent each of the processes in turn.
- ForEach-Object may also be used to get a single property, or execute a single method on each of the objects. For example, ForEach-Object may be used to return only the Path property when using Get-Process:

```
Get-Process | ForEach-Object Path
```

Where-Object command

- Filtering the output from commands may be performed using Where-Object. For example, we might filter processes that started after 5 p.m. today:

```
Get-Process | Where-Object StartTime -gt (Get-Date 17:00:00)
```

```
Get-Process | Where-Object -Property StartTime -Value (Get-Date 17:00:00) -gt
```

- However, it is far easier to read StartTime is greater than <some date>, so most examples tend to follow that pattern.

- Where-Object will also accept filters using the FilterScript parameter.

Selecting and sorting

- Select-Object allows a subset of data to be returned when executing a command. This may be a more restrictive number of elements, or a smaller number of properties.
- Sort-Object can be used to perform both simple and complex sorting.

```
Get-Process | Select-Object -Property Name, Id
```

```
Get-Process | Select-Object -Property Name, *Memory
```

```
Get-Process | Select-Object -Property * -Exclude *Memory*
```

```
Get-ChildItem C:\ -Recurse | Select-Object -First 2
```

```
Get-ChildItem C:\ | Select-Object -Last 3
```

```
1, 1, 1, 3, 5, 2, 2, 4 | Select-Object -Unique
```

```
Get-Process | Select-Object -Property Name, Id,
    @{Name='FileOwner'; Expression={ (Get-Acl $_.Path).Owner }}
```

```
PS> 5, 4, 3, 2, 1 | Sort-Object
1
2
3
4
5
```

```
PS> 'ccc', 'BBB', 'aaa' | Sort-Object
aaa
BBB
ccc
```

Comparing

```
PS> Compare-Object -ReferenceObject 1, 2, 3, 4 -DifferenceObject 1, 2 -IncludeEqual

InputObject SideIndicator
----------- -------------
          1 ==
          2 ==
          3 <=
          4 <=
```

```
PS> Compare-Object -ReferenceObject 1, 2, 3, 4 -DifferenceObject 1, 2 -ExcludeDifferent -IncludeEqual -PassThru
1
2
```

```
$reference = Get-ChildItem C:\Windows\System32 -File
$difference = Get-ChildItem C:\Windows\SysWOW64 -File
Compare-Object $reference $difference -Property Name, Length -IncludeEqual -ExcludeDifferent
```

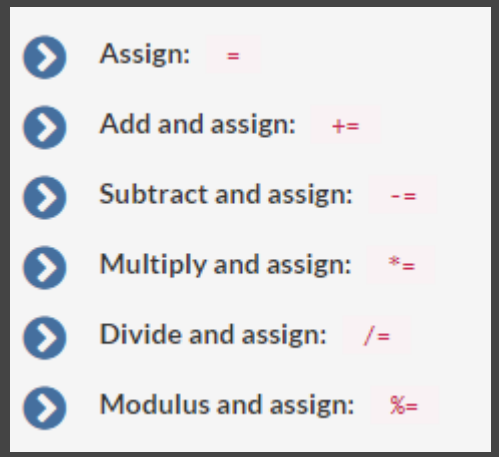
Arithmetic operators

- Arithmetic operators are used to perform numeric calculations. The operators available are the following

- Addition: `+`
- Subtraction: `-`
- Multiplication: `*`
- Division: `/`
- Modulus: `%`
- Shift left: `-shl`
- Shift right: `-shr`

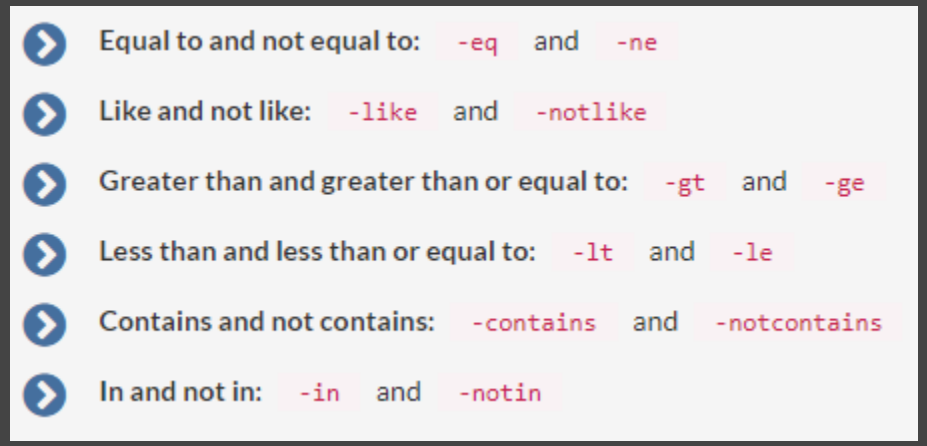
Assignment operators

- Assignment operators are used to give values to variables. The assignment operators available are the following:



Comparison operators

- PowerShell has a wide variety of comparison operators:



Regular-expression-based operators

- The following operators use regular expressions:

Match:	`-match`
Not match:	`-notmatch`
Replace:	`-replace`
Split:	`-split`

Logical operators

- And: `-and`
- Or: `-or`
- Exclusive or: `-xor`
- Not: `-not and !`

Type operators

- As: `-as`
- Is: `-is`
- Is not: `-isnot`

```
PS> if (-not ('System.Web.HttpUtility' -as [Type])) {
    Write-Host 'Adding assembly' -ForegroundColor Green
    Add-Type -Assembly System.Web
}
Adding assembly
```

```
"1" -as [Int32]
'String' -as [Type]
```

```
'string' -is [String]
1 -is [Int32]
[String] -is [Type]
123 -isnot [String]
```

Other Operators

- Call: `&`
- Comma: `,`
- Format: `-f`
- Increment and decrement: `++` and `--`
- Join: `-join`

```
$command = 'ipconfig'
& $command
```

```
$scriptBlock = { Write-Host 'Hello world' }
& $scriptBlock
```

```
PS> "a,b,c,d" -split ',' -join "`t"
a b c d
```

Variables

- Variables in PowerShell are preceded by the dollar symbol ($)

```
$MyVariable
```

- Variable-Commands

```
Clear-Variable

Get-Variable

New-Variable

Remove-Variable

Set-Variable
```

Type and type conversion

- Type conversion in PowerShell is used to switch between different types of a value.

```
PS> [String](Get-Date)
10/27/2016 13:14:32
```

```
PS> [DateTime]"01/01/2016"

01 January 2016 00:00:00
```

Arrays

- Creating an array “$myArray = @()”
- $myGreetings = "Hello world", "Hello sun", "Hello moon“
- $myGreetings = @("Hello world", "Hello sun", "Hello moon")
- Selecting elements from an array
- $myArray = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- $myArray[0]
- $myArray[1]

HashTable

- Creating a hashtable “$hashtable = @{}”

  $hashtable = @{Key1 = "Value1"; Key2 = "Value2"}

- Adding and changing elements to a hashtable

```
$hashtable = @{} ; $hashtable.Add("Key1", "Value1")
$hashtable = @{ Existing = "Old" }
$hashtable["New"] = "New"          # Add this
$hashtable["Existing"] = "Updated"   # Update this
```

- Selecting elements from a hashtable “$hashtable["Key1"]”

Conditional statements

```
if (<condition>) {
   <statements>
}
if (<first-condition>) {
   <first-statements>
} else {
   <second-statements>
}
if (<first-condition>) {
   <first-statements>
} elseif (<second-condition>) {
   <second-statements>
} elseif (<last-condition>) {
    <last-statements>
}
```

Loops

```
foreach (<element> in <collection>) {
    <body-statements>
}
for (<intial>; <exit condition>; <repeat>){
    <body-statements>
 }
while (<condition>) {
    <body-statements>
}
```