Invoke-Expression

Run a PowerShell expression. Accepts a string to be executed as code. It is essential that any user input is carefully validated.

```
Syntax
      Invoke-Expression [-command] string [CommonParameters]

Key
   -command string
       A literal string (or variable that contains a string) that is a
       valid PowerShell expression.
```

Standard Aliases for Invoke-Expression: **`iex`**

Invoke-Expression accepts a *string* and treats it as PowerShell code which allows the construction of dynamic code, this means that you have to be very careful about the string input.

You will have learned that PowerShell treats single and double quoted strings differently, with single quoted strings being interpreted literally, however `Invoke-Expression` will strip the quotes from `-command` completely, meaning this will work:

```
$a='Hello'
PS C:\> iex '$a'
Hello
```

If input is accepted from a user or any third party source, there is a possibility that they could inject unwanted additional PowerShell commands into the script with undesirable effects.

The -command run by Invoke-Expression can be a simple string, or a more complex scriptblock or a Dot Sourced Scriptblock.

Invoke-Expression performs string expansion, so for example `iex "echo 'Hello World $PsHome'"` will output
`Hello World C:\\System32\WindowsPowerShell\v1.0`

If the result of the expression is an empty array, invoke-expression will output `$null`

Using Invoke-Expression to run a set of commands is similar to using the `&` call operator but has a key difference in that invoke-expression does not create an additional scope, so any changes to variables made by the script block will remain visible.

Some examples to demonstrate the difference between `Invoke-Expression` and Call:

```
    PS C:\> $program = "Get-ChildItem"
    PS C:\> Invoke-expression $program
    > Directory listing…

    PS C:\> $program = "Get-ChildItem"
    PS C:\> & $program
    > Directory listing…
```

So far so good, they both work and appear to do the same thing, now lets add a parameter to filter the results:

```
PS C:\> $program = "Get-ChildItem *.txt"
PS C:\> Invoke-expression $program
> Directory listing

PS C:\> & $program
> & : The term 'Get-ChildItem *.txt' is not recognized as...
```

So using Call fails, but this is a good failure because we generally want to be specific about which command/cmdlet is being called and which parameters are being passed to it.
The correct way to do this with Call is passing the parameter as a separate string:

```
PS C:\> $program = "Get-ChildItem"
PS C:\> $progfilter = "*.txt"
PS C:\> & $program $progfilter
> Directory listing
```

Imagine a situation where you prompt the user for a file extension expecting that they will enter `.txt` or `.doc`, but instead they enter a semicolon ; command Separator followed by `;Remove-Item C:\Important folder\`
If you use Invoke-Item in this situation, both commands will be executed, but if you use Call (&) then `"Get-ChildItem"` is passed a parameter which won't make much sense, but nothing gets deleted.

Example

Create variables named $sorting and $MyExpr and use them to store the text of an expression, then use invoke-expression to actually run the expression:

```
PS C:\> $sorting = "sort-object Name"
PS C:\> $myExpr = "get-process | $sorting"
PS C:\> invoke-expression $myExpr
```

"Innovation is the distinction between a leader and a follower" ~ Steve Jobs

Related PowerShell Cmdlets:

Get-Command - Retrieve basic information about a command.
Invoke-Command - Run commands on local and remote computers.
Invoke-Item - Invoke an executable or open a file (START).
Invoke-History - Invoke a previously executed Cmdlet.
Start-Process - Start one or more processes, optionally as a specific user.
Trace-Command - Trace an expression or command.
--% - Stop parsing input as PowerShell commands.
. (source) - Run a command script in the current shell (persist variables and functions.)
& (call) - Run a command script.
Invoke-Expression considered harmful - PowerShell Team / devblogs.microsoft.com.