# Optional Exercise Creating New Objects

When I was parsing log files with PowerShell, I often had to collect data from a loop or from multiple steps. One way to collect data is to store it in an object. We will create a new object that has an email property in addition to the domain, firstname, and lastname properties. This is most easily done in a script, or in the ISE.

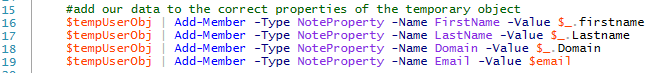
First, we’ll need to create an empty object (variable) to save our new data. It is an array because it will be an array of objects—one object for each user. The “@” means array, and “()” says it’s empty, for now.  
$ $usermail = @()

Inside our Foreach-Object loop, we’ll use this to temporarily save the email address we generate.  
$email = $\_.firstname + “.” + $\_.lastname + “@” + $\_.domain + “.edu”

Then we’ll create a temporary object to hold the lastname, firstname, domain, and email, and assign the values we want. For example for the first user, domain is Starwars, email is [Luke.Skywalker@Starwars.com](mailto:Luke.Skywalker@Starwars.com), lastname is Skywalker, and firstname is Luke.

This creates the object.  
$tempUserObj = New-Object System.Object

These lines showing $tempUserObj | Add-Member … add the values to the new object. (It’s shown in a screenshot so you can’t copy and paste the entire script. After all, ISE has tab-complete so it’s not too bad.)

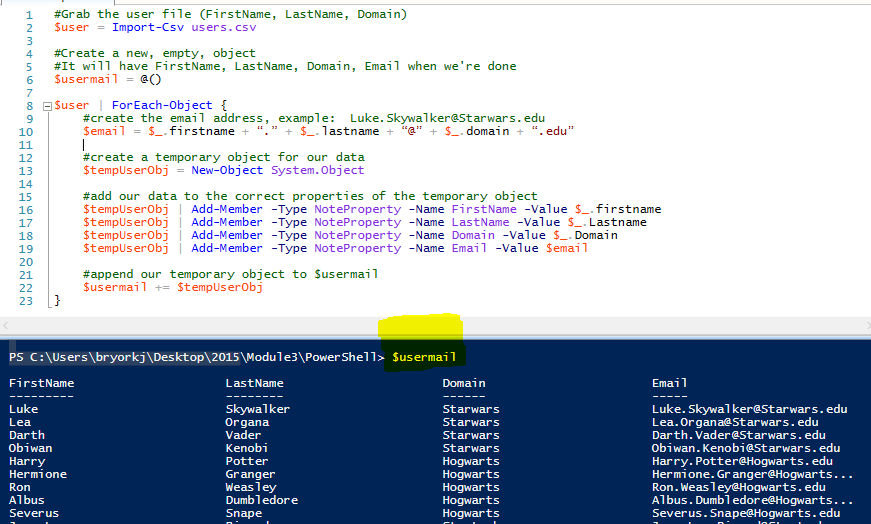


Finally we’ll append our temporary object to the $usermail array of objects.  
$usermail += $tempUserObj

When this is done, we have a new object (variable) that includes the properties we had before plus an email address. We can sort it, use Where-Object to select certain users, export it to a .csv file, and all will have our original properties plus the new email address we’ve created.

To see what $usermail holds, you can just type $usermail (after the script has run successfully.)

Here’s the entire script.



## Another Way—using Foreach instead of Foreach-Object

We used the pipeline to stream our $user object (variable) into the code into our block of script that made a new object that included email addresses. We used the current pipeline object, $\_, to represent the user we were working on at the moment.

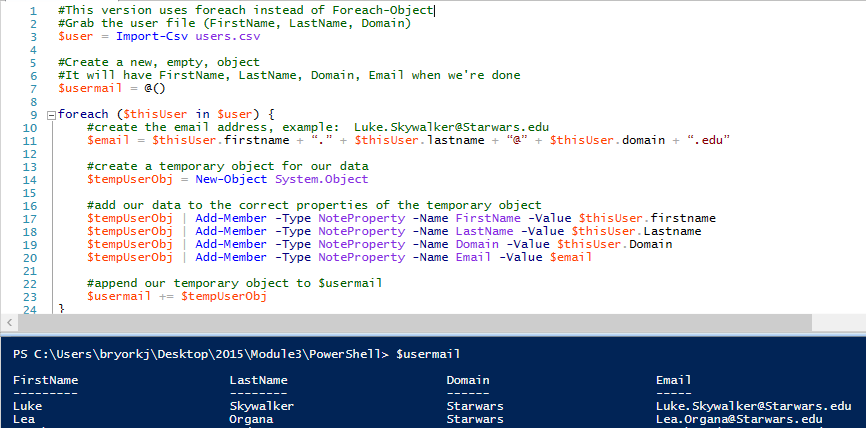
There is another Foreach that works more like what you have seen in other languages. The differences are explained in <https://blogs.technet.microsoft.com/heyscriptingguy/2014/07/08/getting-to-know-foreach-and-foreach-object/>. The syntax looks like this.

Foreach ($tempVariable in $dataArray) { script block}

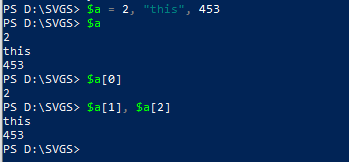
To use foreach instead of Foreach-Object, we need to

1. not pipe $user and change Foreach-Object to foreach ($thisUser in $user) {
2. change current pipeline object $\_ to $thisUser

The main differences between foreach and Foreach-Object are that foreach uses more memory and runs faster.



## Hash Tables

The repeated use of $tempUserObj | Add-Member -Type NoteProperty is a bit awkward, especially for people who don’t like to type. It’s also a good excuse to introduce the concept of hash tables. Hash tables appear in most programming languages, not just PowerShell, because they are very useful. (Python implements hash tables as a “dictionary.”) In a simple array, the indexes are integers, and each index has a value associated with it. For example, in the array, 2, “this”, 453  
 a[0] = 2, a[1] = “this”, and a[2] = 453. The indexes are 0, 1, and 2. As a reminder, here is an array in PowerShell.  


In a hash table, the indexes are called “keys” and can anything—integers, strings, whatever. In PowerShell, hash tables begin with the “@” character and are enclosed in curly braces “{}”. An empty hash table looks like this @{} while an empty array looks like this @(). Curly braces are for a hash table, parenthesis are for arrays. Here is one entry in what could be an address book.

$addressbook = @{

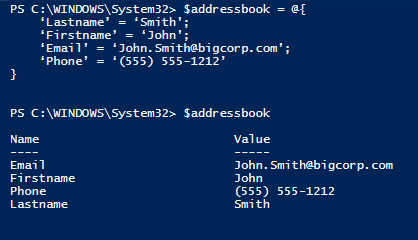
‘Lastname’ = ‘Smith’;

‘Firstname’ = ‘John’;

‘Email’ = ‘John.Smith@bigcorp.com’;

‘Phone’ = ‘(555) 555-1212’

}



You may notice that the key/value pairs of a hash table look a lot like an object’s properties, which have a name and a value. If this were a real address book, you would have an object for every person in the book.

Now that we know something about hash tables, we can add objects in a way that looks cleaner. This can replace the New-Object and Add-Member statements in the earlier example.

This version works with the foreach example above.

$tempUserObj = New-Object psobject -Property @{

FirstName = $thisUser.FirstName

LastName = $thisUser.LastName

Domain = $thisUser.Domain

Email = $email

}

This version works with the first example using For-EachObject.

$tempUserObj = New-Object psobject -Property @{

FirstName = $\_.FirstName

LastName = $\_.LastName

Domain = $\_.Domain

Email = $email

Here’s a finished example.

# Grab the user file (FirstName, LastName, Domain)

$user = Import-Csv D:\SVGS\4.Scripting\PowerShell\users.csv

# Create a new empty array object

# It will have FirstName, LastName, Domain, Email when we are done

$usermail = @()

$user | ForEach-Object {

# Create the email address, ex Luke.Skywalker@Starwars.edu

$email = $\_.FirstName + '.' + $\_.LastName + '@' + $\_.Domain + '.edu'

# Create a temporary object that holds data for one user

$tempUserObj = New-Object psobject -Property @{

FirstName = $\_.FirstName

LastName = $\_.LastName

Domain = $\_.Domain

Email = $email

}

# Continued below

# Append our temporary object to $usermail

$usermail += $tempUserObj

}

# Display the result

$usermail