# PowerShell Lab 5

Let’s continue working with our example from the last lab:

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

$user = Import-Csv users.csv

#Create a new, empty, object

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

$usermail = @()

foreach ($thisUser in $user) {

#create the email address, example: Luke.Skywalker@Starwars.edu

$email = $thisUser.firstname + “.” + $thisUser.lastname + “@” + $thisUser.domain + “.edu”

#create a temporary object for our data

$tempUserObj = New-Object System.Object

#add our data to the correct properties of the temporary object

$tempUserObj | Add-Member -Type NoteProperty -Name FirstName -Value $thisUser.firstname

$tempUserObj | Add-Member -Type NoteProperty -Name LastName -Value $thisUser.Lastname

$tempUserObj | Add-Member -Type NoteProperty -Name Domain -Value $thisUser.Domain

$tempUserObj | Add-Member -Type NoteProperty -Name Email -Value $email

#append our temporary object to $usermail

$usermail += $tempUserObj

}

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 using “dictionary.”) In a simple array, the indexes are integers and each index has a value associated with it. For example, a[0] = 2, a[1] = “this”, a[2] = 453, is an array with indexes 0, 1, 2, and values 2, “this”, and 453. In a hash table, the indexes are called “keys” and can anything. In PowerShell, hash tables are designated with the “@” character. For example this is a hash table that could be made into 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.