```
PS› function Get-PowershellBasics {
  Get-Instructors
  $course = Invoke-RestMethod $courseUrl
  Foreach ($topic in $course) {
      Write-Host $topic
  }
}

PS› Get-PowershellBasics
```

Roque Sosa
Eduardo Garcia

Brief Contents

Scope

Introduction

If we think about code you use on your automation that you did not write yourself, whomever wrote it might have used the same name for a variable or function than you. So how does PowerShell know the difference? Well, both implementations have a completely different scope.

In PowerShell Scripting, like in most other programming languages, scope plays a big part in managing variable states and conditions. Restricting functions and features from accessing variables that are outside of their scope.

In-Depth

Scope protects variables and functions from being accessed by limiting from which parts of the script they can be read, changed and emptied.

An item added to a scope can be accessed by its current scope and all child scopes, but it won't be able to be accessed from a different scope. To make this easier to understand, let's look at some examples:

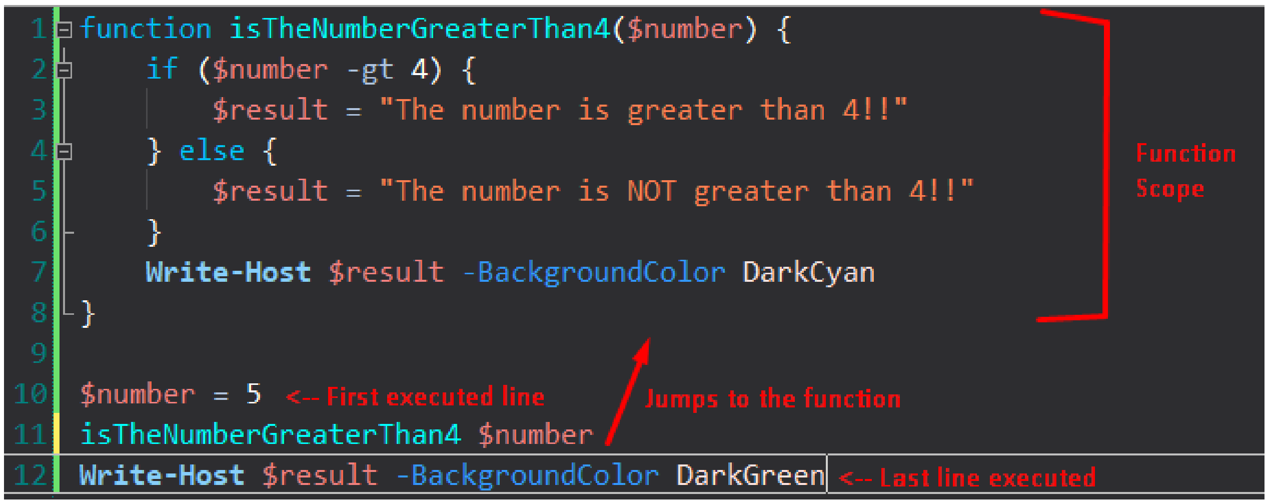
```
function isTheNumberGreaterThan4($number) {
	if ($number -gt 4) {
		$result = "The number is greater than 4!!"
	} else {
		$result = "The number is NOT greater than 4!!"
	}
	Write-Host $result -BackgroundColor DarkCyan
}

$number = 5
$result = “Test”
isTheNumberGreaterThan4 $number
Write-Host $result -BackgroundColor DarkGreen
```

Execution:

```
PS>& '.\Scope - Variable names.ps1'
The number is greater than 4!!
Test
```

As we can see in the execution, only one print of results occurs. That's because on the second print of `$result`, the variables are actually different but with the same name. This is possible because both are created on different scopes. This is the flow for this script.



```
function isTheNumberGreaterThan4($number) {
    if ($number -gt 4) {
        $result = "The number is greater than 4!!"
    } else {
        $result = "The number is NOT greater than 4!!"
    }
    Write-Host $result -BackgroundColor DarkCyan
}

$number = 5
isTheNumberGreaterThan4 $number
Write-Host $result -BackgroundColor DarkGreen
```

Functions

Introduction

In the process of making any script there comes a time when code needs to be reused, sometimes twice, thrice or dozens of times. Instead of just copying and pasting the same lines of code every time, and maybe making some minor edits to it, we can use functions. These are snippets of code that can be reused inside the Script as many times as needed and helps both in the modularity and the readability of the code.

In-Depth

Let's say we have a Script that uses the same lines of code, again and again, causing a lengthy Scrip. By using functions, the script would be much easier to read and more efficient. For example,

```
$ErrorActionPreference = 'Stop'

function Ask-ForValue { ←-- Function declaration
      try {
            [int32]$valueEntered = Read-Host -Prompt "What number
do you want to study?"
      } catch {
            Write-Host "You did not enter a variable castable to
[Int32]"
            exit
      }
      return $valueEntered
}

$counterTimes = 0

$value = Ask-ForValue ←-- Function call
while ($value -ne 0) {
      if ($value % 2 -ne 0) {
            Write-Host "$value is ODD"
      } else {
            Write-Host "$value is PAIR"
      }

      $counterTime++

      $value = Ask-ForValue ←-- Function call
}

"Total of numbers entered = " + $counterTimes | Write-Host
```

Here we can see the results of using functions on a script, is shorter and easier to read taking the user input code outside of the main logic. It's really important when writing a script that you look for code that can me moved to a function, but how do we create a function and what are the conventions?

```
Function name ($parameter1, $parameter2 … $parameterN){
      <statement-list>
}
```

That is a basic function. PowerShell offers tools to create advanced functions but we will not be exploring these. For the next and last topic on functions we have one keyword that always creates a discussion on the Powershell community, `return`.

The `return` keyword can be observed in our previous example. It is used to turn the flow of the execution and take the value passed to it back to where the function was called. In the example we see it returns the user input to the `$value` variable

Classes & Objects

Introduction

As it was explained in the previous lesson, everything in PowerShell is an object, but… what is an object?

An object is in its most basic definition, the programmatic representation of something. Just like a real-world object they possess properties and may possess methods which represent the actions it can perform.

In-Depth

Let’s look at the object as if it were something we would see in real life, for example a car. The car has characteristics, which are its **properties**; like color, number of doors, manufacturer, etc. It also can perform actions, which would represent its **methods**; it can turn on, turn off, accelerate and brake.

In order to create a custom object (called PSCustomObject) we can call the Cmdlet New-Object passing down the desired properties as a Hash Table

```
PS>$car = New-Object -TypeName PSCustomObject -Property
@{color = "blue" ; doors = 2 }
```

And add new Methods by using the Add-Member Cmdlet

```
PS>$car | Add-Member -MemberType ScriptMethod -Name "TurnOn"
-Value { "Vrrroooom" }
PS>$car | Get-Member

   TypeName: PSCustomObject

Name        MemberType
----        ----------
Equals      Method
GetHashCode Method
GetType     Method
ToString    Method
color       NoteProperty
doors       NoteProperty
TurnOn      ScriptMethod
```

Properties and Methods can be accessed at any time by writing down the object followed by dot (.) and the Method/Property Name

```
PS>$car.color
blue

PS>$car.TurnOn()
Vrrroooom
```

Now if this is an object and its type is PSCustomObject, how does [Int32],[String] and [System.Collections.ArrayList] exists and how were they created? They were created following the instructions of a Class.

Class

Classes are blueprints for an object. They have the properties of the object and its types (properties, including getter and setters), the instructions to create the objects correctly (constructors) and the actions you can perform using that object (methods).

To be completely honest, classes in powershell are not of great use, at least for now in version 5. This means that there is a lack of normal class functionality defined and used on all other Object Oriented Languages, but we can use Powershell classes to explain classes teorically:

Definition:
```
class Automobile {
     #Properties
     [String]$color
     [Int32]$doors

     #Constructor
     Automobile(
          [Int32]$doors,
          [String]$color
     ) {
          $this.doors = $doors
          $this.color = $color
     }

     #Methods
     [String]OpenDoors() {
          Return “Beep Beep”
     }

}
```

Usage:
```
PS>$car = [Automobile]::new(4, “red”)
PS>$car.color
red

PS>$car.OpenDoors()
Beep Beep

PS>$car.color = “blue”
PS>$car.color
blue
```

We can see the class itself it's never used after the declaration of it, but when you create the object you are creating an instance of that class, basically creating an object following the rules set in the class.

Modules

Introduction

A **module** is a set of functions created to solve issues that are similar, for example, if you create a script full of math functions (sum, multiple, divide, pow, square root) you can move those functions into a module.

In-Depth

Modules are also objects when imported, this means they have properties to explore. That's the way you discover the cmdlets that come packaged into a module.

We will not be exploring how to create/modify modules, it's not really something difficult but it's a lengthy explanation for something that is not really top priority. What it is though, is using modules created by other people who might have had the same issue than you and they've solved it, leaving you with a solution for your problems all neat and packaged for ease of use.

Exploring modules

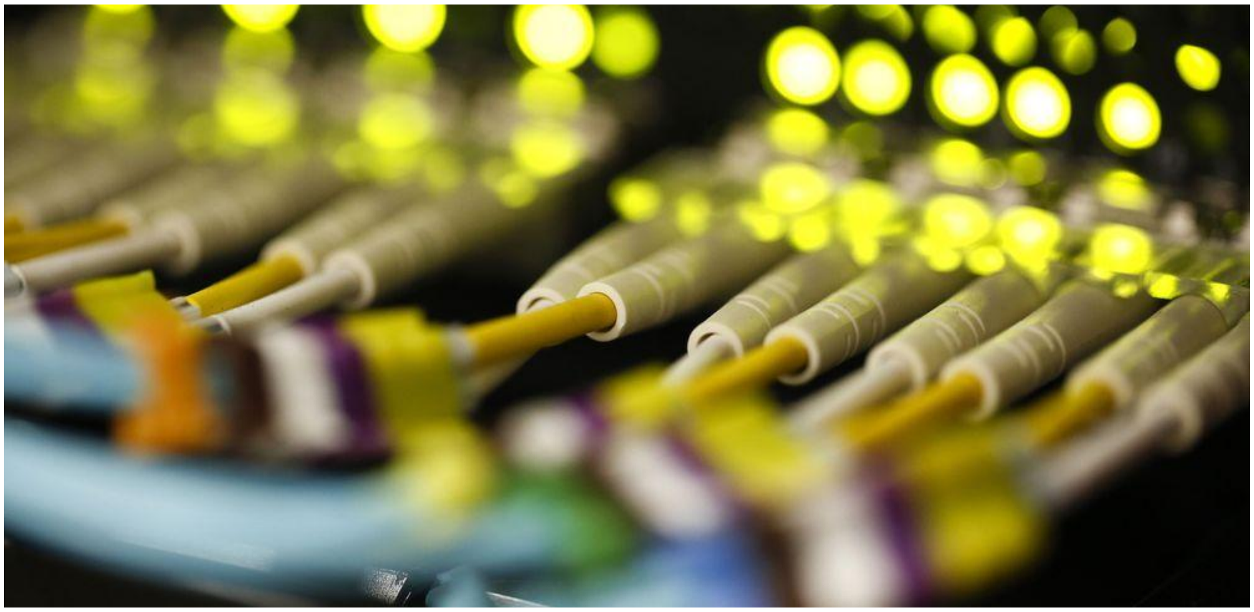
There are a couple of cmdlets to explore the functionality and composition of a module:

```
PS>Get-Command *Module* | select CommandType,Name,Version

CommandType Name                       Version
----------- ----                       -------
   Function Find-Module                1.0.0.1
   Function Get-InstalledModule        1.0.0.1
   Function Install-Module             1.0.0.1
   Function Publish-Module             1.0.0.1
   Function Save-Module                1.0.0.1
   Function Uninstall-Module           1.0.0.1
   Function Update-Module              1.0.0.1
   Function Update-ModuleManifest      1.0.0.1
     Cmdlet Export-ModuleMember        3.0.0.0
     Cmdlet Get-Module                 3.0.0.0
     Cmdlet Import-Module              3.0.0.0
     Cmdlet New-Module                 3.0.0.0
     Cmdlet New-ModuleManifest         3.0.0.0
     Cmdlet Remove-Module              3.0.0.0
     Cmdlet Test-ModuleManifest        3.0.0.0
```

There you can see Import and Install -Module. The thing is that inside JPMorgan's network the repository where these Cmdlets download the modules is blocked so in the course we will show a way to install modules without breaking company policy.

API Interactions



Introduction

Sometimes in the never-ending journey of automating our daily tasks and making processes more efficient, we need to interact with web resources through our scripts. These interactions are made easier through the use of APIs and invoking rest methods.

In-Depth

First of all, what is an API?

Application Programming Interface

The acronym A.P.I. stands for Application Programming Interface. An API basically consists of a set of methods defined with the purpose of allowing and aiding the communication between two entirely different applications. In this class, we will be talking specifically interest in REST APIs and how to interact with them

REST API

REST stands for **RE**presentational **S**tate **T**ransfer, without getting too much into the technicalities of what constitutes a REST API, the most important factor is that they provide communication with web services. This allows you to retrieve information, trigger processes and even have a certain degree of control over a web service through HTTP Methods

HTTP Methods

The most important methods you will be using are:

- **GET:** Requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.
- **POST:** Requests that the server accept the data enclosed in the request and perform a process with it.
- **PUT:** Requests that the enclosed entity be stored under the supplied URI.
- **DELETE:** Requests the deletion of the specified resource.

HTTP Responses

When you perform an HTTP Request, the service answers back by providing an HTTP Status Code and in many cases a message. These are some of the most common response codes, do keep in mind that you don't need to remember all of them:

- **200 (OK):** Request has succeeded
- **201 (Created):** Has succeeded in creating a resource
- **202 (Accepted):** Accepted for Processing
- **400 (Bad Request):** Could not be understood by the server
- **401 (Unauthorized):** Requires authentication
- **403 (Forbidden):** Has not been accepted and should not be repeated

- **404 (Not Found):** Server hasn't found anything on the target URL
- **500 (Internal Server Error):** Unexpected condition and couldn't fulfill request

Invoke-RestMethod

PowerShell provides you with a very useful Cmdlet to perform these interactions. This allows you to send a request to a REST API by providing the target **URL**, the chosen **Method**, the request **Body** (for use in case of a POST request), the required **Headers**, **Credentials** and many more less-frequently used parameters.

```
      PS>$response = Invoke-RestMethod
'https://jobs.github.com/positions.json?description=python&location=new
+york'

      PS>$response[0]

      id             : 056ce8a5-7520-448a-a0a6-fa954bb6e1f2
      type           : Full Time
      url            : https://jobs.github.com/positions...
      created_at     : Thu Jun 13 20:38:32 UTC 2019
      company        : BentoBox
      company_url    : https://getbento.com/
      location       : New York, NY
      title          : Senior Software Engineer
      description    : <p>BentoBox puts restaurants online. We are
transforming the hospitality industry by helping restaurants stay at
the forefront…
```

As simple as that we have the information we wanted, already formatted into a custom object for us to use. This is obviously a very simple example, you should explore the `Invoke-WebRequest` for a more complete interaction with the API.

Conclusion

This concludes the course. With these new set of tools you have been provided with, you should be able to start practicing automating your day to day tasks. Research and try new methods. Start with something small you do once or twice everyday, no matter how simple it looks at first, that way you'll build up confidence to start automating more difficult or complex processes. Google your doubts, if something breaks remember to debug it to find the failing feature. If you get stuck, take a breather and dive back into it later. This is just your first step in the thrilling journey of automation, enjoy the ride.