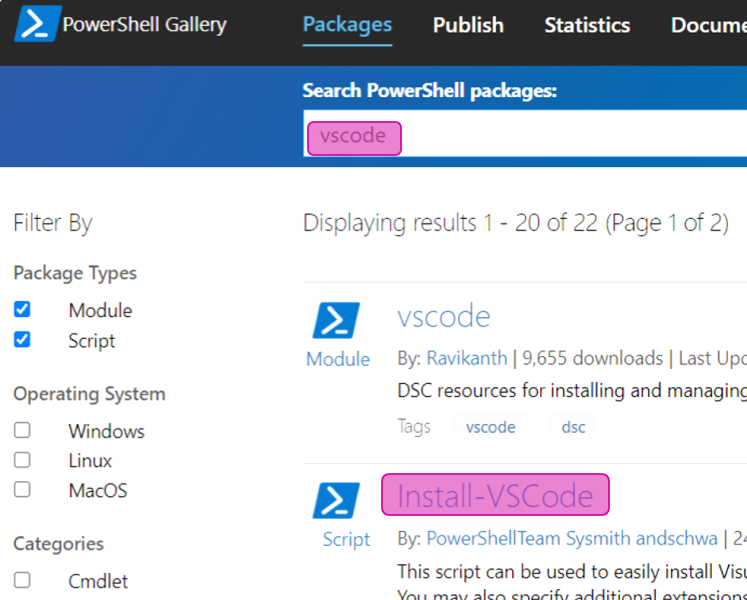
Visual Studio Code Intro

Visual Studio Code is a free Integrated Development Environment from Microsoft. It is an excellent option for assisting you in your PowerShell learning and development.

In this lab, we will use the **Install-VSCode** script from the PowerShell Gallery to easily install VS Code with the PowerShell extension already installed.

Note: If you are running your VMs from a macOS with the new M1 chip (ARM architecture) you will need to download and install VSCode from [here](https://code.visualstudio.com/download) instead of from the PowerShell gallery (select the **Apple Silicon** option). Also, install the PowerShell extension and described [here](https://learn.microsoft.com/en-us/powershell/scripting/dev-cross-plat/vscode/using-vscode?view=powershell-7.3#install-vs-code-and-the-powershell-extension). Skip the instructions about the PowerShell gallery below.

Search for vscode from the [PowerShell Gallery](https://www.powershellgallery.com/) and click on **Install-VSCode** to learn how to install this helper script.



Use the following command to install the VSCode install script:

Install-Script -Name Install-VSCode

When that completes, simply run Install-VSCode.

Install-VSCode

That was easy! Now you have a shortcut on your desktop to launch Visual Studio Code. Let’s take it for a test drive.

A picture containing text

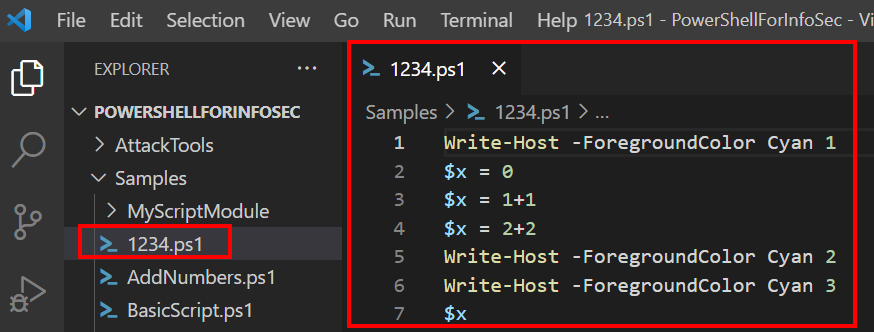
Description automatically generated

Start VS Code and select “Open Folder” from the File Menu. Select the **C:\Users\IEUser\PowerShellForInfoSec** folder. Now you should see the class files in the Explorer as shown below.

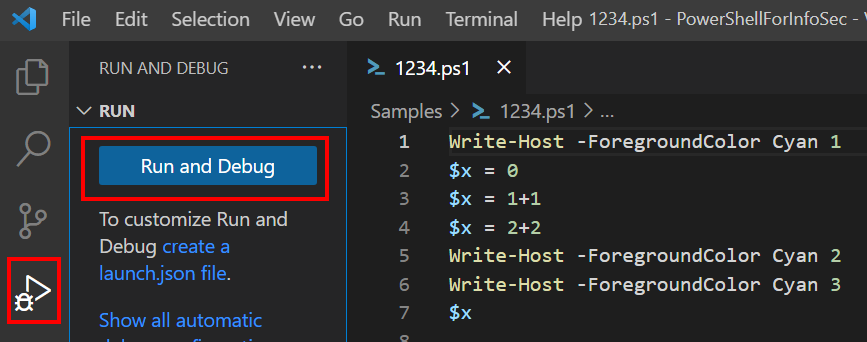
Text

Description automatically generated

Open the **1234.ps1** script by double clicking it from the Samples folder in the Explorer.



You can run this script by going to the debug menu and clicking **Run and Debug**.



When you run this script, you will see its output in the Terminal window on the bottom right of VSCode.

Text

Description automatically generated

This gives us a quick and easy way to run a script that we are developing. But what is even better is that it lets us debug an application, or incrementally step through the program one step at a time and inspect variable values along the way.

Click to the left of line 3, to set a breakpoint, which is represented by a red dot as shown below.

Text

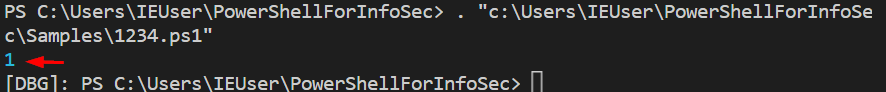
Description automatically generated

A breakpoint indicates a point in the code where we want execution to stop so that we can inspect values and step through the program slowly. Now press **Run and Debug** and you will see program execution stop after it executes line 2. The yellow shape around the red dot indicates the location where execution is currently stopped.

Graphical user interface, text, application, chat or text message

Description automatically generated

Look in the terminal window, how many of the numbers have been printed out at this point? Only the number 1.



Hover your mouse pointer over the $x variable (any one of them) and it will show you the current value of x. At this point it the script, it has a value of 0.

A screenshot of a computer

Description automatically generated

Click the **Step Over** button on the debug menu bar to execute line three and advance to line four.

Graphical user interface, text, application, chat or text message

Description automatically generated

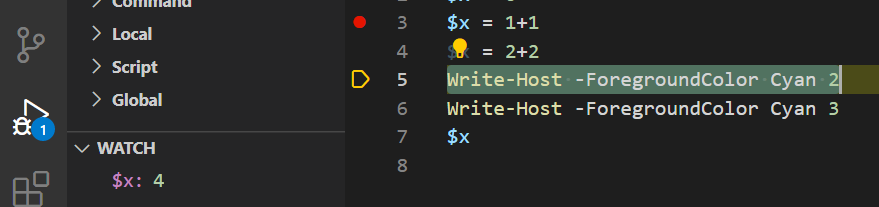
Hover your mouse over the x variable now. Does it have a value of 2 now (1+1) as expected?

Instead of hovering over a variable to see its value, you can also add it to the watch list in the left menu, for quicker reference.

Graphical user interface, application

Description automatically generated

Click the plus sign in the watch window and enter **$x** as the variable name. Now you can watch the value of x change as you step through the code one line at a time.

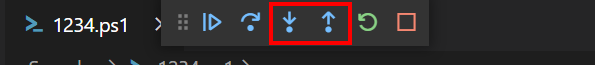


You can set as many breakpoints in the script as you like. You can use the **Continue** button to jump from one breakpoint all the way down to the next.

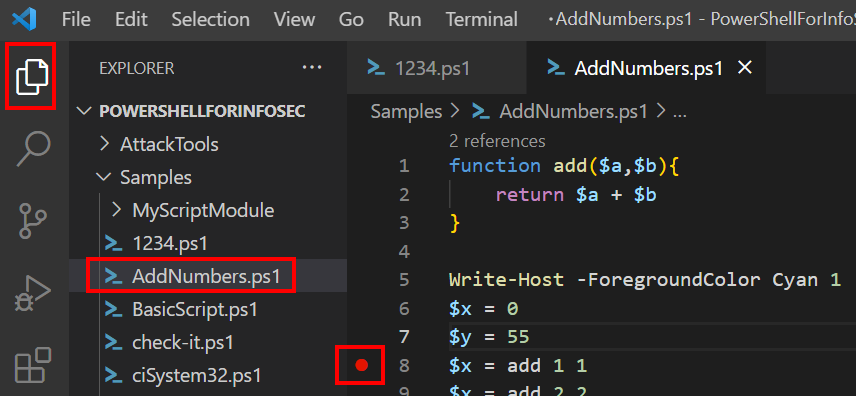
Text

Description automatically generated

The **Step In** and **Step Out** controls are helpful when your script calls a function.



Open the **AddNumbers.ps1** script and set a breakpoint in from of line eight.



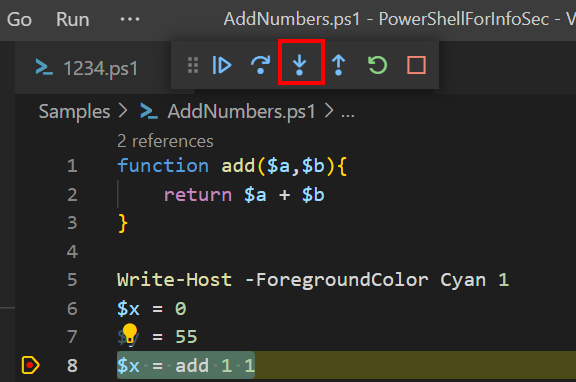
Click Run and Debug from the debug menu to execute the script up until the breakpoint.

Note: Sometimes VS Code loses its mind. If you get an unexplainable error as shown below, just close and reopen VS Code.

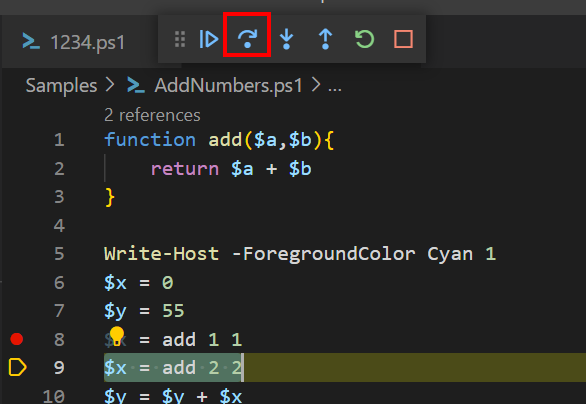
Graphical user interface, text, application

Description automatically generated

You should now be stopped at your breakpoint on line eight. Use the **Step Into** button and notice where it takes you.



It lets you step through the add function on lines one through three. Debug the script again, but this time hit the **Step Over** button and compare the results.



In this case, it skipped right over the add function and went directly to line nine. You choose which option you prefer by deciding what you are interested in reviewing.

Continue to play with the VSCode code completion and debugging options to get more familiar with this tool. I have found VS Code to be extremely helpful for all my work with PowerShell and I hope that you will too!