Assignment 4

You will be required to create 2x script files. Following the instructions for each one and upload them as individual scripts to the autograder.

Exercise 1

Create a powershell script named `A4E1.ps1` that:

1. Accepts a filepath as an argument and stores it in the variable `$search_directory`
2. Search all CSV files in the given path for email addresses and stores it in the variable `$email_rows`. Use the csv files in `assignment4_data.zip`.
3. Count the number of email objects and store it as email_count.
4. Displays the sample output with the variables displaying their proper values.

How to count the emails:

```
$email_count = ($email_rows | Measure-Object).count
```

Expected display format:

```
Searching all .txt files in $search_directory for email addresses...
$email_rows
$email_count emails were found.
```

Submission

Upload only `A4E1.ps1` to the autograder. You do not need to upload any csv files, ones will be generated similar (but not identical) to the ones on D2L.

Important Notes and Requirements

- Ensure your text is exactly as shown above where N is the number of email addresses and PATH is the path entered as an argument (in other words, its the value of `$search_directory`).
- Use the regular expression for matching emails from the PDF. Do NOT use one from the internet as the results will vary slightly and your results must match exactly to the ones that match from that
- Ensure you use either `Join-Path` the user input before searching. To verify this, test your script by entering a path both with and without a trailing slash, as an example (that will not work on your computer, its just an example): `D:\Code\Samples` AND `D:\Code\Samples\`. Using `Join-Path` will solve this trailing slash problem for you with the `$search_directory` variable.
- For extra challenge, you may also attempt to accept a **named parameter** with a default value instead of using the special argument variable.

For demonstration, your output should look like this, but with different paths and a different number of emails but in this format when your script is complete:

```
PS D:\Code\Sample> .\Find-Emails.ps1 ..\Data\
Searching all .txt files in ..\Data\ for email addresses...

D:\Code\Data\output_1.csv:2:"Franklin","Mason","Hockey","email","masonfranklin@hot
mail.com"
D:\Code\Data\output_1.csv:3:"Brian","Dean","Volleyball","email","brian.dean@netnav
igators.org"
D:\Code\Data\output_1.csv:4:"Piper","Wood","Tennis","email","silverlion365@hotmail
.com"
D:\Code\Data\output_1.csv:5:"Peter","Fields","Hockey","email","greenfrog642@webwar
den.org"
D:\Code\Data\output_1.csv:6:"Jessie","Steward","Tennis","email","smallladybug723@n
etcrafters.org"
D:\Code\Data\output_1.csv:9:"Lucas","Hughes","Tennis","email","lucas_hughes@global
grid.biz"
D:\Code\Data\output_1.csv:13:"Bertha","Mckinney","Golf","email","mckinney.bertha@b
usiness.net"
D:\Code\Data\output_1.csv:14:"Patsy","Palmer","Golf","email","palmerpatsy@globalda
taflow.net"
D:\Code\Data\output_1.csv:16:"Oliver","Pelletier","Rugby","email","pelletier.olive
r@quantumcode.net"
D:\Code\Data\output_1.csv:17:"Adrian","Webb","Soccer","email","webb.adrian@netnavi
gator.com"
10 emails were found.

PS D:\Code\Sample>
```

Bonus Hint

If you're really stuck, you're probably looking for a line like this:

```powershell
# Combine the directory path and the wildcard filter to search only in .txt files
$files_to_search = Join-Path $search_dire "*.csv"
# Search all .txt files in the specified directory for valid email addresses
$email_rows = Select-String -Path $files_to_search -Pattern "{REGEX GOES HERE}"
```

Exercise 2

Create a powershell script named `A4E2.ps1` that is created from the template below. Each item has a statement and a corresponding regex that will complete the requested pattern:

```
```

Essential Phrase

> It is a shiny day in 2024-02-06 in COMP86,
> where code becomes art and pixels dance,
> even computers can't help but admire his tricks,
> Abe teaches not just with logic but with great pants.
> The colours #FFEE09 and #5522AA don't rhyme with anything.

Basic Regex Exercises

Here are 10 basic regex exercises suitable for students to practice with. Each exercise is designed to target a specific aspect of regex, gradually increasing in complexity.

1. **Match "a" or "A"**: Find any instance of "a" regardless of case.
2. **Match the number 86**: Match the number 86.
3. **Match any vowel**: Match from the set `a e i o u`
4. **Match the first letter of a line**: Match the first letter in any line. Hint: `(?ms)REGEX` [https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-options#multiline-mode](Read More)
5. **Match Words with Exactly 5 Letters**: Find words that are exactly 5 letters long. Hint: `\bREGEX\b` [https://www.regular-expressions.info/wordboundaries.html](Read more)
6. **Match All Numbers**: Match any sequence of digits in a string.
7. **Find dates in "YYYY-MM-DD" Format**: Match any date in the format of "YYYY-MM-DD". Don't worry about exactly matching 12 for month, if someone puts a date of `0010-83-48` that's their own fault, today.
8. **Match Hexadecimal values**: Find any 6 character hexadecimal codes.

PowerShell Script Template

Below is a sample PowerShell script template. Students can fill in the regex for each exercise within the script. This script demonstrates how to run regex against a predefined set of strings or files, capturing the matches and displaying them.

```
# Assignment 4 - Exercise 2
# Name: Your name here
# Student ID: Your ID here

# This powershell script solves world hunger through basic regular expression
exercises

$testString = @"
It is a shiny day in 2024-02-06 in COMP86,
where code becomes art and pixels dance,
even computers can't help but admire his tricks,
Abe teaches not just with logic but with great pants.
The colours #FFEE09 and #5522AA don't rhyme with anything.
"@

# Function to test regex
function Test-Regex {
```

```
    param(
        [string]$regex,
        [string]$string,
        [int]$exercise
    )

    if ($string -match $regex) {
        "Exercise $exercise - $($Matches.Count) Matches found: $($Matches.Values)"
    } else {
        "Exercise $exercise - No match found for $regex."
    }
}


# Testing Exercise 1
$regex1 = "[aA]"
Test-Regex -regex $regex1 -string $testString -exercise 1

# Testing Exercise 2
$regex2 = ""
Test-Regex -regex $regex2 -string $testString -exercise 2

# Testing Exercise 3
$regex3 = ""
Test-Regex -regex $regex3 -string $testString -exercise 3

# Testing Exercise 4
$regex4 = ""
Test-Regex -regex $regex4 -string $testString -exercise 4

# Testing Exercise 5
$regex5 = ""
Test-Regex -regex $regex5 -string $testString -exercise 5

# Testing Exercise 6
$regex6 = ""
Test-Regex -regex $regex6 -string $testString -exercise 6

# Testing Exercise 7
$regex7 = ""
Test-Regex -regex $regex7 -string $testString -exercise 7

# Testing Exercise 8
$regex8 = ""
Test-Regex -regex $regex8 -string $testString -exercise 8
```

There is no need to modify this script in any way except your name and to include the appropriate regular expressions. This is actually ridiculously difficult to create in an autograder, so if you believe something in error please don't hesitate to email about incorrect results or interpretation from the autograder.