

SOC (Security Operations Center)

Módulo 1



PowerShell Scripting - Windows

Ejercicios

Sheila Fernández Cisneros – 18/05/2024

Tarea 1:

Almacenar la cadena “Hola Mundo” en una variable, y posteriormente mostrar por pantalla el contenido de la variable.

Para realizar este ejercicio, primero abrimos PowerShell. Esto se puede hacer buscando “PowerShell” en el menú de inicio y seleccionando la opción correspondiente. No es necesario abrir PowerShell en modo administrador para este ejercicio. Una vez en PowerShell, podemos ejecutar los siguientes comandos para almacenar la cadena “Hola Mundo” en una variable y luego mostrar su contenido por pantalla.

Para almacenar la cadena "Hola Mundo" en una variable en PowerShell, utilizamos el símbolo “\$” seguido del nombre de la variable y el operador de asignación “=” seguido de la cadena. Luego, para mostrar el contenido de la variable por pantalla, usamos el comando “Write-Output” seguido del nombre de la variable.

Administrador: Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/pssc0re6

PS C:\Windows\system32> set-location C:\Users\soced\Desktop
PS C:\Users\soced\Desktop> $mensaje = "Hola Mundo"
PS C:\Users\soced\Desktop> write-output $mensaje
Hola Mundo
PS C:\Users\soced\Desktop>
```

Tarea 2:

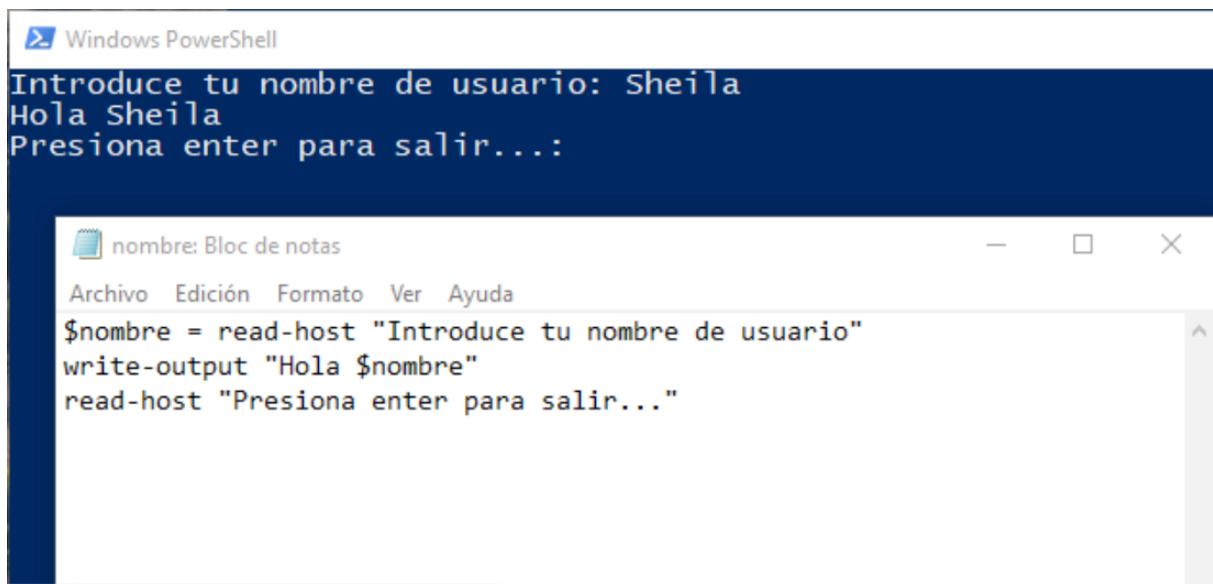
Escribir un programa que pregunte el nombre de usuario y tras introducirlo, lo muestra por pantalla con la cadena “Hola <nombre>”.

Para cumplir con la tarea de solicitar el nombre de usuario y luego mostrar un saludo personalizado, creamos un script en PowerShell llamado “nombre.ps1” utilizando el Bloc de notas.

En este script, se ha utilizado el comando “Read-Host”, el cual es un comando de PowerShell que solicita al usuario que introduzca datos desde la consola. En este caso, se muestra el mensaje “Introduce tu nombre de usuario”, y la entrada del usuario se guarda en la variable \$nombre. Luego, el script muestra un saludo personalizado utilizando el nombre ingresado por el usuario mediante el comando “Write-Output”. Finalmente, para mantener la ventana abierta hasta que el usuario esté listo para salir, se utiliza “Read-Host” nuevamente con el mensaje “Presiona enter para salir...”. Esto permite que el usuario vea el saludo antes de que el programa termine de ejecutarse.

Una vez guardado el script, podemos ejecutarlo haciendo clic derecho sobre él y seleccionando la opción “Ejecutar con PowerShell”. Esto abrirá una ventana de PowerShell donde se solicitará al usuario que introduzca su nombre. Después de ingresar el nombre y presionar “Enter”, el

script mostrará el saludo personalizado “Hola <nombre>” en la consola. Finalmente, el programa esperará a que el usuario presione “Enter” nuevamente para salir.



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". Inside, the command "Introduce tu nombre de usuario: Sheila" is displayed, followed by "Hola Sheila" and "Presiona enter para salir....". Below the window, a Notepad window titled "nombre: Bloc de notas" contains the PowerShell script:

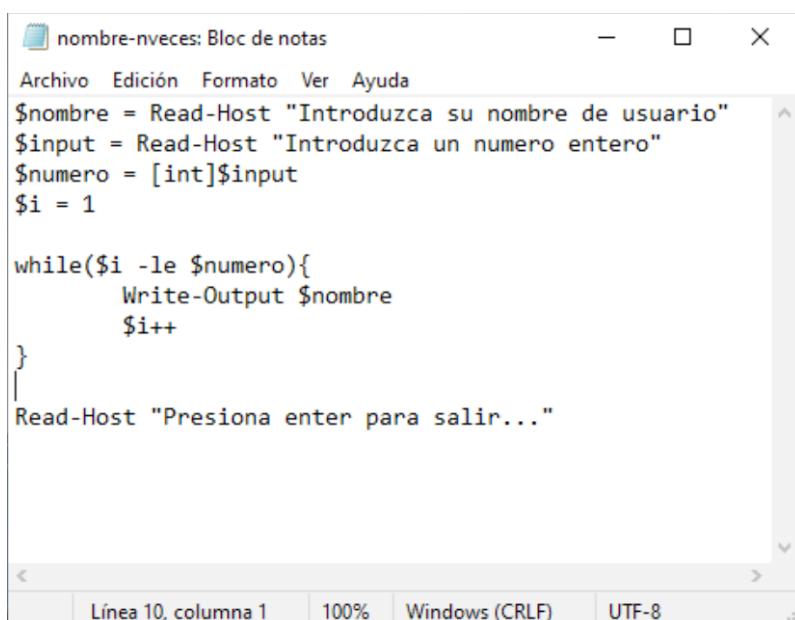
```
$nombre = read-host "Introduce tu nombre de usuario"
write-output "Hola $nombre"
read-host "Presiona enter para salir..."
```

Tarea 3:

Escribir un programa que pregunte el nombre del usuario y un número entero e imprima por pantalla en líneas distintas el nombre del usuario tantas veces como indique el número.

Para completar la tarea de solicitar el nombre del usuario y un número entero, y luego imprimir el nombre del usuario tantas veces como indique el número, creamos un script en PowerShell llamado “nombre-nveces.ps1”.

El contenido del script es el siguiente:



The screenshot shows a Notepad window titled "nombre-nveces: Bloc de notas". It contains the following PowerShell script:

```
Archivo Edición Formato Ver Ayuda
$nombre = Read-Host "Introduzca su nombre de usuario"
$input = Read-Host "Introduzca un numero entero"
$numero = [int]$input
$i = 1

while($i -le $numero){
    Write-Output $nombre
    $i++
}

Read-Host "Presiona enter para salir..."
```

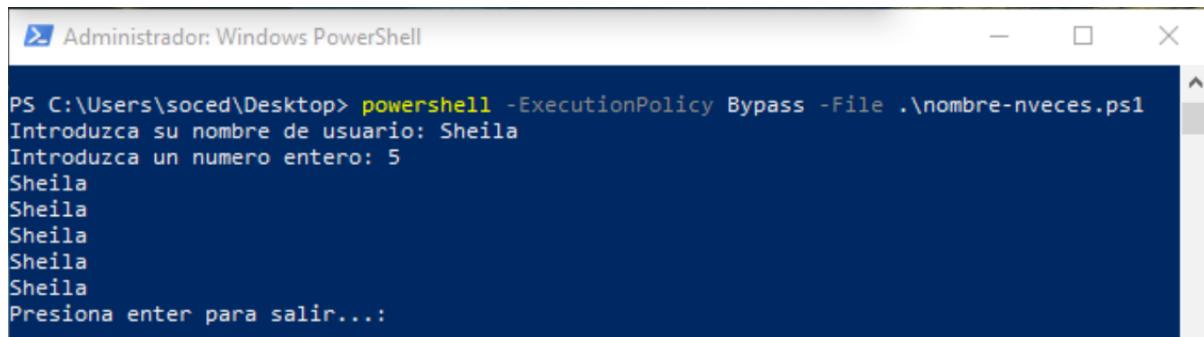
En este script, se utiliza “Read-Host” para solicitar al usuario que introduzca su nombre y un número entero. El número ingresado por el usuario se convierte a un tipo de dato entero utilizando [int] para que pueda ser tratado como un número y no como una cadena de texto.

Luego, se utiliza un bucle while para imprimir el nombre del usuario tantas veces como indique el número ingresado. El contador \$i se inicializa en 1 y se incrementa en cada iteración del bucle hasta que alcanza el número indicado por el usuario. En cada iteración, se muestra el nombre del usuario utilizando Write-Output.

Para ejecutar este script en PowerShell, se puede usar el comando:

```
powershell -ExecutionPolicy Bypass -File nombre-nveces.ps1
```

Esto permitirá que el script se ejecute correctamente, incluso si la política de ejecución de scripts de PowerShell está restringida.



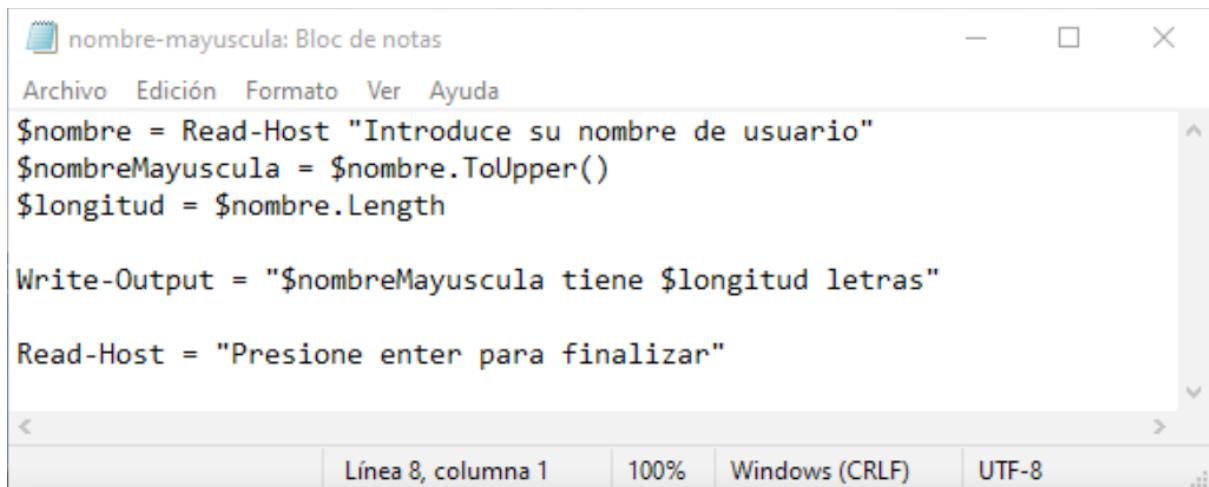
```
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\nombre-nveces.ps1
Introduzca su nombre de usuario: Sheila
Introduzca un numero entero: 5
Sheila
Sheila
Sheila
Sheila
Sheila
Presiona enter para salir....
```

Tarea 4:

Escribir un programa que pregunte el nombre de usuario en consola y al introducirlo, se muestre por pantalla: <NOMBRE> tiene <n> letras, donde <NOMBRE> es el nombre del usuario en mayúsculas y <n> el número de letras que tiene el nombre. Puedes ayudarte del comando \$nombre.ToUpper (para pasar a mayúsculas) y \$nombre.length (para calcular la longitud del nombre).

Para desarrollar el programa requerido vamos a utilizar el método “ToUpper()” para convertir el nombre ingresado por el usuario a mayúsculas y la propiedad “length” para calcular la longitud del nombre.

El contenido del script es el siguiente:



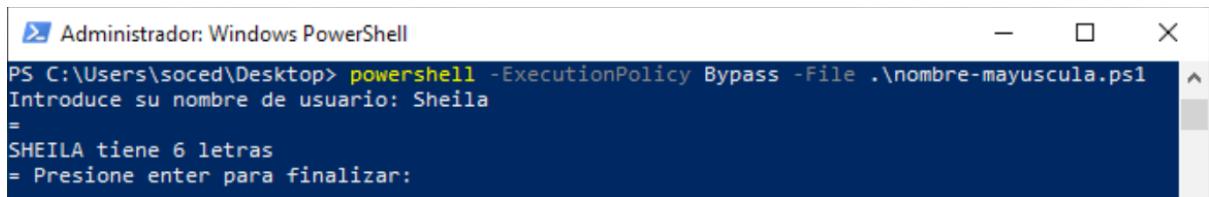
```
$nombre = Read-Host "Introduce su nombre de usuario"
$nombreMayuscula = $nombre.ToUpper()
$longitud = $nombre.Length

Write-Output = "$nombreMayuscula tiene $longitud letras"

Read-Host = "Presione enter para finalizar"
```

Línea 8, columna 1 | 100% | Windows (CRLF) | UTF-8

Este script primero solicita al usuario que introduzca su nombre en la consola con el comando “Read-Host”. Luego, convierte ese nombre a mayúsculas utilizando el método “ToUpper()” y lo guarda en la variable \$nombreMayuscula. Posteriormente, calcula la longitud del nombre utilizando la propiedad “length” y guarda el dato en la variable \$longitud. Finalmente, muestra el resultado por pantalla en el formato solicitado usando el comando “Write-Output”.



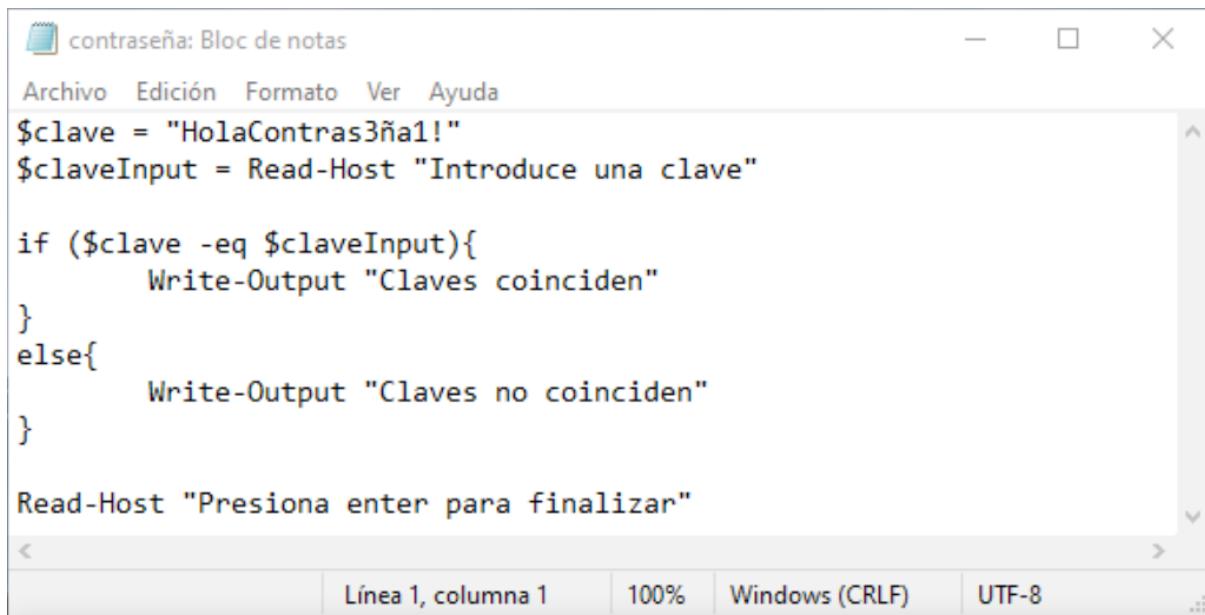
```
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\nombre-mayuscula.ps1
Introduce su nombre de usuario: Sheila
=
SHEILA tiene 6 letras
= Presione enter para finalizar:
```

Tarea 5:

Escribir un programa que almacene una contraseña en una variable. Luego, pregunta al usuario que introduzca una contraseña. Por último, imprime por pantalla si las contraseñas coinciden o no.

Se ha creado un script de PowerShell llamado “contraseña.ps1” para comparar una contraseña almacenada con una ingresada por el usuario.

Primero, almacenamos una contraseña en una variable llamada “\$clave”. Luego, solicitamos al usuario que introduzca una contraseña utilizando “Read-Host”. A continuación, comparamos la contraseña almacenada con la ingresada por el usuario utilizando un condicional “if” y el operador “-eq”. Si las contraseñas coinciden, se imprime “Claves coinciden”; si no coinciden, se imprime “Claves no coinciden”. Finalmente, utilizamos “Read-Host” para que el usuario presione “Enter” antes de finalizar el programa, permitiendo al usuario ver el resultado de la comparación.



```
contraseña: Bloc de notas
Archivo Edición Formato Ver Ayuda
$clave = "HolaContras3ña1!"
$claveInput = Read-Host "Introduce una clave"

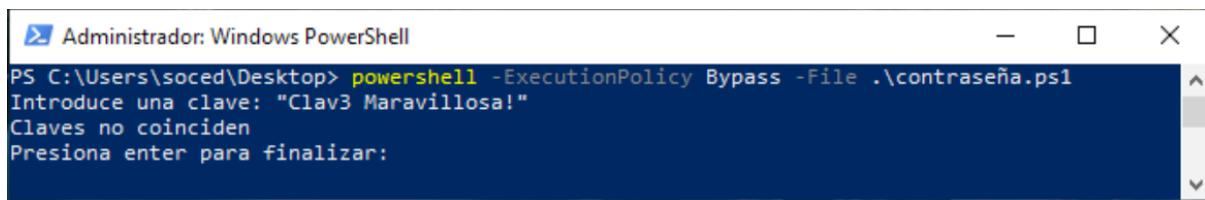
if ($clave -eq $claveInput){
    Write-Output "Claves coinciden"
}
else{
    Write-Output "Claves no coinciden"
}

Read-Host "Presiona enter para finalizar"

```

Línea 1, columna 1 | 100% | Windows (CRLF) | UTF-8

El resultado de la ejecución puede verse a continuación:



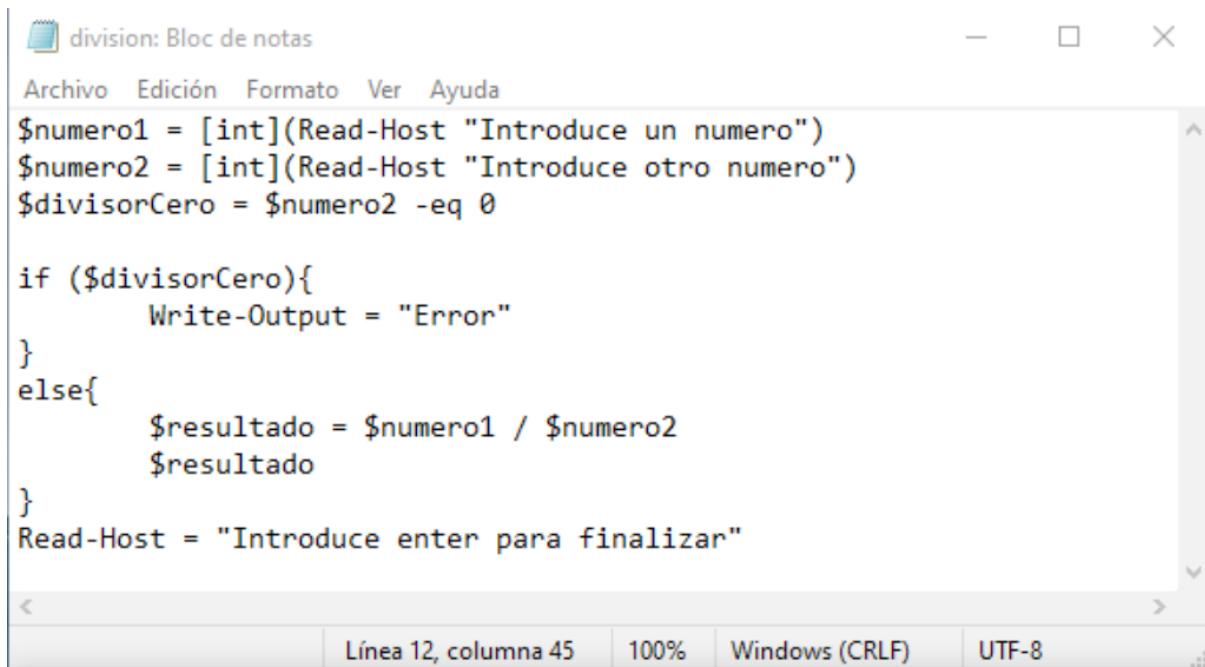
```
Administrador: Windows PowerShell
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\contraseña.ps1
Introduce una clave: "Clav3 Maravillosa!"
Claves no coinciden
Presiona enter para finalizar:
```

Tarea 6:

Escribir un programa que pida al usuario dos números y muestre por pantalla la división de ambos. Si el divisor es 0, el programa debe mostrar un error. Recomendación: usar casting [int]\$numero1 al leer los números introducidos por usuario para que se almacenen como valores numéricos.

Para la tarea de pedir al usuario dos números y mostrar la división de ambos, se ha creado el script “división.ps1”.

El script primero solicita al usuario que introduzca dos números. Utiliza “Read-Host” para leer las entradas y realiza un casting a entero [int] para asegurarse de que los valores se almacenen como números. Luego, se verifica si el divisor es cero mediante una variable llamada “\$divisorCero”. Si el divisor es cero, se muestra un mensaje de “Error”. De lo contrario, se realiza la división y se muestra el resultado. Finalmente, el script espera a que el usuario presione “Enter” antes de finalizar, permitiéndole ver el resultado o el mensaje de error.



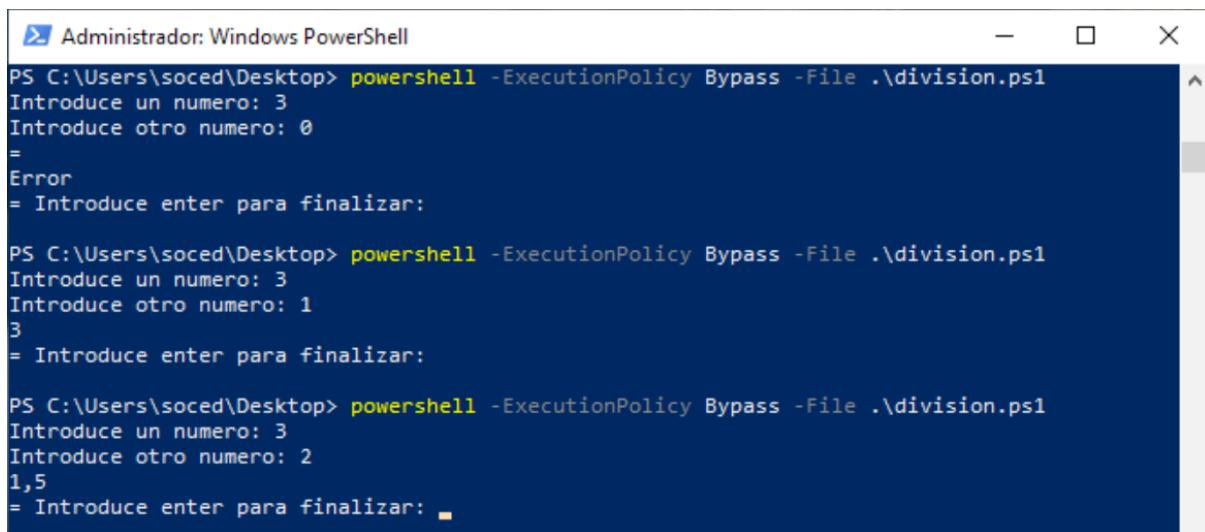
division: Bloc de notas

```
Archivo Edición Formato Ver Ayuda
$numero1 = [int](Read-Host "Introduce un numero")
$numero2 = [int](Read-Host "Introduce otro numero")
$divisorCero = $numero2 -eq 0

if ($divisorCero){
    Write-Output = "Error"
}
else{
    $resultado = $numero1 / $numero2
    $resultado
}
Read-Host = "Introduce enter para finalizar"
```

Línea 12, columna 45 | 100% | Windows (CRLF) | UTF-8

Algunos ejemplos de ejecución pueden verse a continuación:



```
Administrator: Windows PowerShell
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\division.ps1
Introduce un numero: 3
Introduce otro numero: 0
=
Error
= Introduce enter para finalizar:

PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\division.ps1
Introduce un numero: 3
Introduce otro numero: 1
3
= Introduce enter para finalizar:

PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\division.ps1
Introduce un numero: 3
Introduce otro numero: 2
1,5
= Introduce enter para finalizar: ■
```

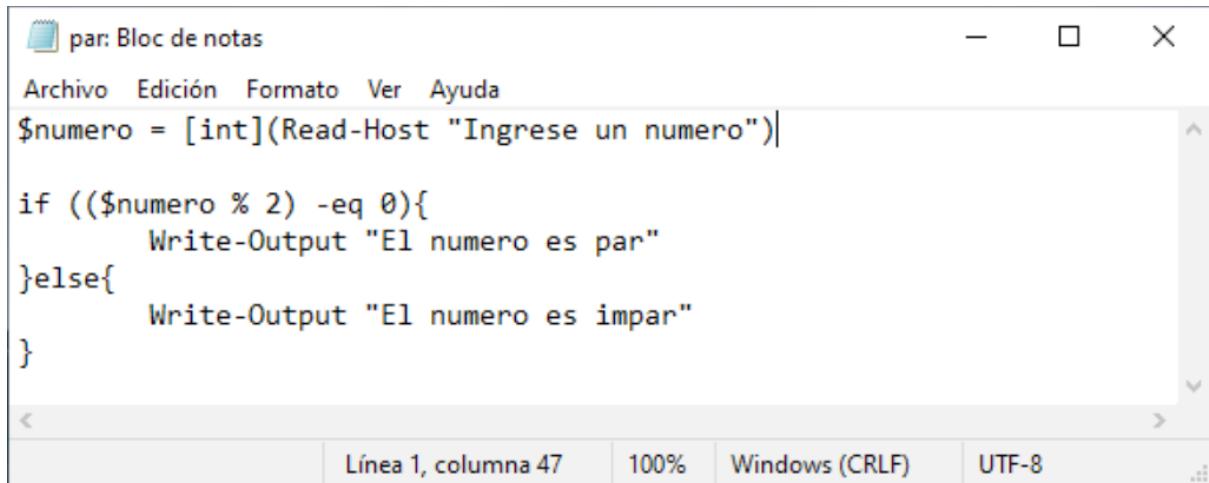
Tarea 7:

Escribir un programa que pida al usuario un número entero y muestre por pantalla si es par o impar.

Para la tarea de determinar si un número es par o impar, se ha creado un script de PowerShell llamado “par.ps1”.

El script solicita al usuario que ingrese un número entero. Utiliza “Read-Host” para leer la entrada y realiza un casting a entero [int] para asegurarse de que el valor se almacene como número. Luego, utiliza un condicional “if” para verificar si el número es par o impar utilizando

el operador módulo “%”. Si el número es divisible por 2 (es decir, el resto es 0), se considera par y se muestra el mensaje correspondiente. De lo contrario, se muestra que el número es impar.



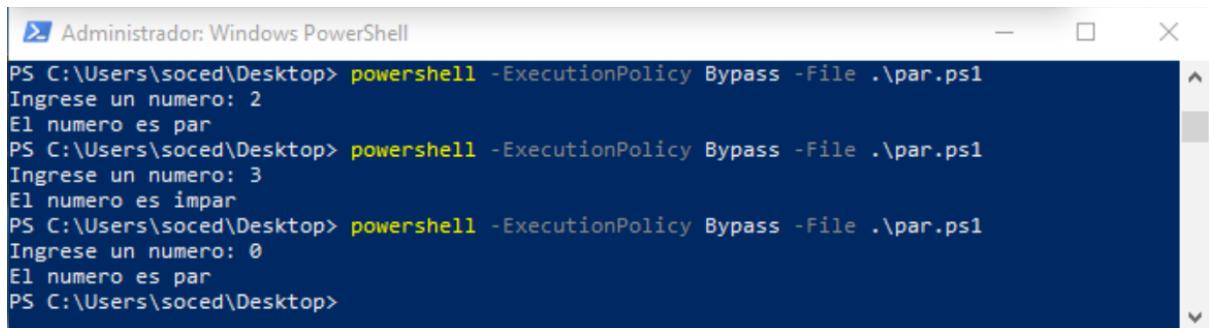
```
par: Bloc de notas

Archivo Edición Formato Ver Ayuda
$numero = [int](Read-Host "Ingrese un numero")

if (($numero % 2) -eq 0){
    Write-Output "El numero es par"
} else{
    Write-Output "El numero es impar"
}

Línea 1, columna 47 | 100% | Windows (CRLF) | UTF-8
```

Algunos ejemplos de ejecución pueden verse a continuación:



```
Administrador: Windows PowerShell
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\par.ps1
Ingrese un numero: 2
El numero es par
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\par.ps1
Ingrese un numero: 3
El numero es impar
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\par.ps1
Ingrese un numero: 0
El numero es par
PS C:\Users\soced\Desktop>
```

Tarea 8:

Escribir un programa que pida un número entero al usuario y muestre por pantalla si es primo o no.

Para la tarea de determinar si un número es primo, hemos creado un script de PowerShell llamado “primo.ps1”.

Un número es primo si solo es divisible por 1 y por sí mismo. Para optimizar la comprobación, partimos de la base de que, si un número es divisible por un número mayor que su raíz cuadrada, también lo es por algún número menor que su raíz cuadrada. Por tanto, no es necesario hacer comprobaciones por encima de su raíz cuadrada.

Consideraciones iniciales, si el número es igual o inferior a 1, no es primo. Si el número es 2, es primo (siendo este el menor número primo y el único par).

Por tanto, siendo n el número ingresado por el usuario, comprobamos la divisibilidad para cada número i desde 2 hasta la raíz cuadrada del número ingresado, si el número es divisible por i (su módulo es cero), entonces no es primo. Si no es divisible por ningún número en ese rango, entonces es primo.

En esta ocasión he querido hacer mi primera función en PowerShell para realizar este ejercicio. Es un script bastante sencillo en el que se va comprobando en un bucle “while” las condiciones necesarias para determinar si un número es primo o no.

En PowerShell, se puede declarar una función utilizando la palabra clave “function” seguida del nombre de la función. En nuestro caso, la función se llama “esPrimo”.

Dentro de la función, utilizamos el parámetro “\$n” para recibir el número que el usuario introduce. La palabra clave “param” en PowerShell se utiliza dentro de funciones y scripts para definir parámetros que pueden aceptar argumentos.

Primero, comprobamos si el número es menor o igual a 1. Si es así, devolvemos “false” indicando que no es primo. A continuación, utilizamos un bucle “while” en el que se compara el valor de la variable “\$i”, inicializada a 2, con el valor de la raíz cuadrada del número, calculada usando la clase “math”. Si el módulo de “\$n % \$i” es cero, la función retorna “false”. Si no, el valor de “\$i” se incrementa en uno hasta alcanzar el valor de la raíz cuadrada. Si no se encuentra ningún divisor, la función retorna “true”, indicando que el número es primo.

Posteriormente, se llama a la función en un condicional “if” que utiliza “Write-Output” para mostrar por pantalla el mensaje correspondiente indicando si el número es primo o no.

```
$n = [int](Read-Host "Ingrese un numero")
$i = [int]2

function esPrimo{
    param ($n)
    if ($n -le 1){
        return $false
    }
    while ($i -le [math]::sqrt($n)){
        if ($n % $i -eq 0){
            return $false
        }
        $i++;
    }
    return $true
}

$resultado = esPrimo($n)

if ($resultado -eq $false){
    Write-Output "El numero no es primo"
}
else{
    Write-Output "El numero es primo"
}
```

Línea 7, columna 16 | 100% | Windows (CRLF) | UTF-8

A continuación, se añaden algunos test para comprobar el funcionamiento del script.

```
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 0
El numero no es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 1
El numero no es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 2
El numero no es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 3
El numero es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 4
El numero no es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 5
El numero es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 24
El numero no es primo
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\primo.ps1
Ingrese un numero: 23
El numero es primo
PS C:\Users\soced\Desktop>
```

Tarea 9:

(OPCIONAL) Escribir un programa que genere un número aleatorio entre 1 y 100, de tal forma que el programa pida al usuario que introduzca un número para adivinar el generado por el programa en el mínimo número de intentos. El programa tiene que indicar si el número introducido por el usuario es mayor que el número aleatorio generado o si es menor. Cuando el usuario adivine el número, se le tiene que decir en cuántos intentos lo ha conseguido.

Para realizar esta tarea, se ha creado un script llamado “adivinar.ps1”.

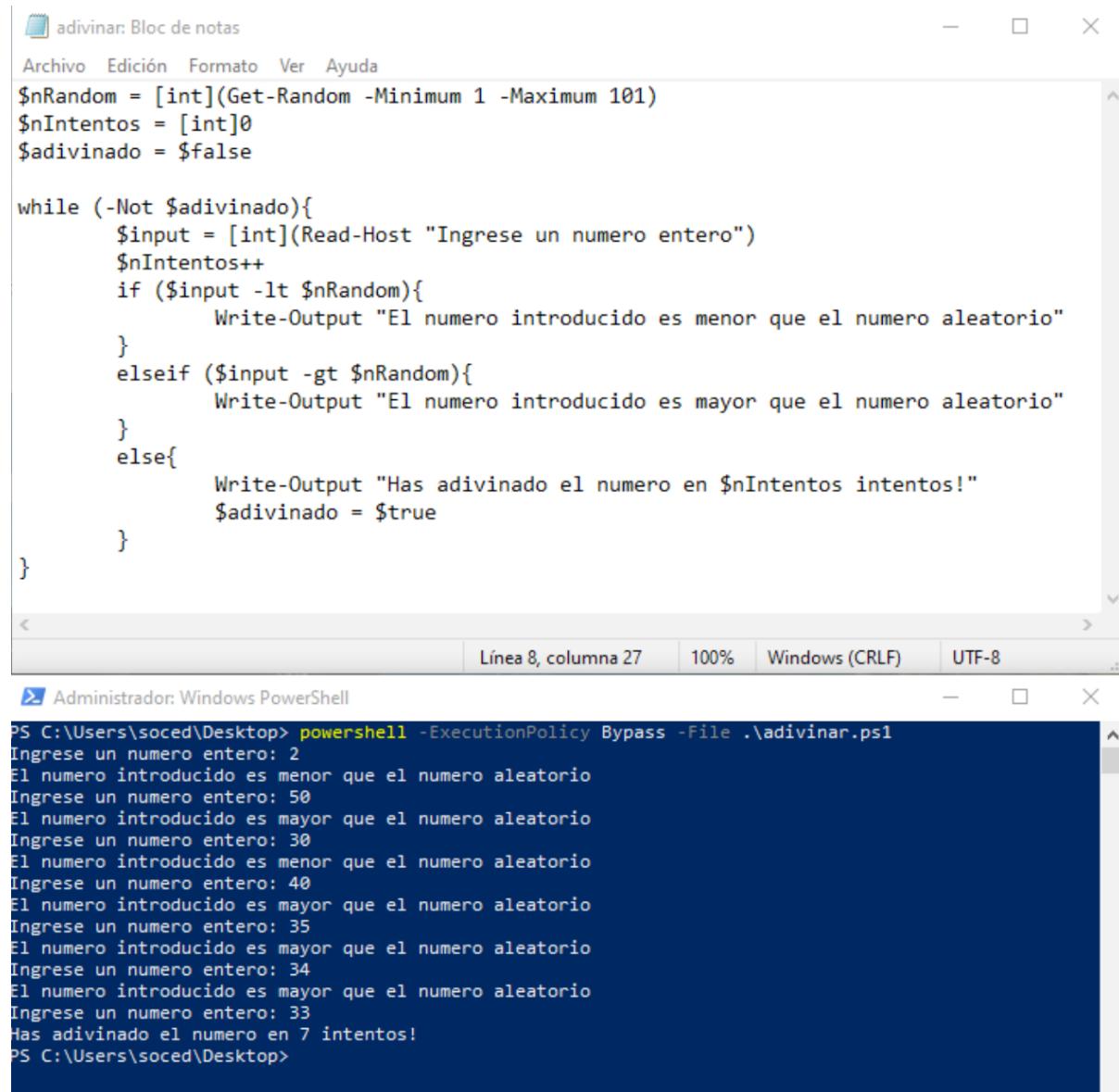
El programa comienza generando un número aleatorio entre 1 y 100 utilizando el comando “Get-Random”. Este número se almacena en una variable llamada “\$nRandom”. Además, se inicializan dos variables: “\$nIntentos” en 0 (la cual no es necesario castearla a int ya que al almacenar un número se convertirá en int32), para contar el número de intentos realizados por el usuario, y “\$adivinado” en false, para controlar el bucle que permite los intentos del usuario.

El bucle “while” se ejecuta mientras la variable “\$adivinado” sea false. En cada iteración del bucle, el programa solicita al usuario que introduzca un número entero mediante “Read-Host”, y este se almacena en una variable llamada “\$input”. Cada vez que el usuario introduce un número, el contador de intentos, “\$nIntentos”, se incrementa en uno.

El programa compara el número introducido por el usuario con el número aleatorio generado. Si el número introducido es menor que el número aleatorio, se muestra un mensaje indicando que el número es menor. Si el número introducido es mayor que el número aleatorio, se muestra un mensaje indicando que el número es mayor. Si el número introducido coincide con el número aleatorio, el programa muestra un mensaje indicando que el usuario ha adivinado el

número y en cuántos intentos lo ha conseguido. En este punto, la variable “\$adivinado” se establece en true, lo que provoca la salida del bucle “while” y finaliza el programa.

A continuación se muestra el funcionamiento del programa.



The screenshot shows two windows side-by-side. The left window is a 'Bloc de notas' (Notepad) titled 'adivinar: Bloc de notas' containing the PowerShell script 'adivinar.ps1'. The right window is a 'Administrador: Windows PowerShell' window showing the execution of the script.

```
$nRandom = [int](Get-Random -Minimum 1 -Maximum 101)
$nIntentos = [int]0
$adivinado = $false

while (-Not $adivinado){
    $input = [int](Read-Host "Ingrese un numero entero")
    $nIntentos++
    if ($input -lt $nRandom){
        Write-Output "El numero introducido es menor que el numero aleatorio"
    }
    elseif ($input -gt $nRandom){
        Write-Output "El numero introducido es mayor que el numero aleatorio"
    }
    else{
        Write-Output "Has adivinado el numero en $nIntentos intentos!"
        $adivinado = $true
    }
}

Línea 8, columna 27 | 100% | Windows (CRLF) | UTF-8
```

```
PS C:\Users\soced\Desktop> powershell -ExecutionPolicy Bypass -File .\adivinar.ps1
Ingrese un numero entero: 2
El numero introducido es menor que el numero aleatorio
Ingrese un numero entero: 50
El numero introducido es mayor que el numero aleatorio
Ingrese un numero entero: 30
El numero introducido es menor que el numero aleatorio
Ingrese un numero entero: 40
El numero introducido es mayor que el numero aleatorio
Ingrese un numero entero: 35
El numero introducido es mayor que el numero aleatorio
Ingrese un numero entero: 34
El numero introducido es menor que el numero aleatorio
Ingrese un numero entero: 33
Has adivinado el numero en 7 intentos!
PS C:\Users\soced\Desktop>
```