Ejercicios de PowerShell

disk.ps1 **Solución**

```
# disk.ps1

# Obtener la información de la partición C:
$disk = Get-PSDrive -Name C

# Calcular el porcentaje de espacio usado
$porcentaje = ($disk.Used / ($disk.Used + $disk.Free)) * 100

# Mostrar el porcentaje de espacio usado
Write-Output ("C: ocapada al {0:N2}%" -f $porcentaje)
```

mem.ps1 **Solución**

```
# mem.ps1

# Obtener la memoria física total y la memoria libre
$totalMemory = (Get-CimInstance Win32_ComputerSystem).TotalPhysicalMemory / 1ME
$freeMemory = (Get-CimInstance Win32_OperatingSystem).FreePhysicalMemory / 1MB

# Calcular la memoria ocupada
$usedMemory = $totalMemory - $freeMemory

# Obtener la fecha y hora actual para el registro
$timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"

# Crear el mensaje que se va a escribir en el archivo
$logEntry = "$timestamp - Memoria ocupada: $([math]::round($usedMemory, 2)) MB"

# Escribir la entrada en el archivo free.log, sin borrar el contenido anterior
Add-Content -Path "free.log" -Value $logEntry
```

cpu.ps1 **Solución**

```powershell
# cpu.ps1

# Obtener el nombre del procesador
$cpuName = Get-CimInstance -ClassName Win32_Processor | Select-Object -ExpandPr

# Mostrar el nombre del procesador en la pantalla
Write-Host "CPU: $cpuName"
```

mac.ps1 **Solución** Solución (foreach)

```powershell
# mac.ps1

Get-NetAdapter | Where-Object { $_.Status -eq 'Up'} | Format-Table Name,MacAddr
```

edad.ps1 **Solución**

```powershell
# Script: edad.ps1

# Solicitar al usuario que ingrese su año de nacimiento
$añoNacimiento = Read-Host "Ingrese el año en que naciste"

# Obtener el año actual
$añoActual = (Get-Date).Year

# Calcular la edad
$edad = $añoActual - $añoNacimiento

# Mostrar la edad
Write-Output "Tienes $edad años."
```

tres_numeros.ps1 Solución **Solución (usanmos decimales)**

```
echo "Dame un número"
[Double] $x=Read-Host
echo "Dame un número"
[Double] $y=Read-Host
echo "Dame un número"
[Double]$z=Read-Host

$r=$x+$y+$z

echo  $("{0:F2}" -f $r)
```

mult.ps1 **Solución**

```
# mult.ps1

$number = Read-Host "Introduce un número para mostrar su tabla de multiplicar"
$n = [int] $number

Write-Host "Tabla de multiplicar de $n"
Write-Host "--------------------------"
for ($i = 1; $i -le 10; $i++) {
    $r = $n * $i
    Write-Host "$n x $i = $r"
}
```

ext.ps1 **Solución**

```
# ext.ps1
foreach ($i In $(Get-ChildItem -Filter *.dat )){
  $newName = $i.Name -replace '\.dat$', '.txt'
  Rename-Item $i.Name $newName
  }
```

puerta.ps1 **Solución (if)** Solución (switch)

```powershell
# Script: puerta_if.ps1

$codigo = Read-Host "
1 - puerta roja
2 - puerta azul
3 - puerta verde
4 - puerta amarilla

Seleccione una puerta"

if ($codigo -eq 1) {
    Write-Output "Has seleccionado la puerta roja."
}
elseif ($codigo -eq 2) {
    Write-Output "Has seleccionado la puerta azul."
}
elseif ($codigo -eq 3) {
    Write-Output "Has seleccionado la puerta verde."
}
elseif ($codigo -eq 4) {
    Write-Output "Has seleccionado la puerta amarilla."
}
else {
    Write-Output "Puerta incorrecta."
}
```

puerta2.ps1 **Solución**

```powershell
# Script: puerta.ps1

# Solicitar al usuario que seleccione una puerta ingresando un número del 1 al
$codigo = Read-Host "Seleccione una puerta (1 - puerta roja, 2 - puerta azul, 3

# Usar switch para seleccionar la puerta según el código ingresado
switch ($codigo) {
    1 { Write-Output "Has seleccionado la puerta roja. Has perdido." }
    2 { Write-Output "Has seleccionado la puerta azul. Has perdido." }
    3 {
        Write-Output "Has seleccionado la puerta verde."

        # Solicitar al usuario que seleccione entre cara (0) o cruz (1)
        $moneda = Read-Host "Lanza la moneda: selecciona cara (0) o cruz (1)"

        # Verificar el resultado de la moneda
        if ($moneda -eq "1") {
            Write-Output "Has ganado."
        } else {
            Write-Output "Has perdido."
        }
    }
    4 { Write-Output "Has seleccionado la puerta amarilla. Has perdido." }
    Default { Write-Output "Puerta incorrecta." }
}
```

usuario.ps1 | **Solución (args)** | Solución (args2) | Solución (param)

```powershell
$Nombre = $args[1]
$Apellido = $args[3]
$Usuario = $args[5]
$Nacimiento = [int]$args[7]

$AñoActual = (Get-Date).Year

$Edad = $AñoActual - $Nacimiento

if ($Edad -ge 14) {
    Write-Output "Se ha creado a $Nombre $Apellido el usuario $Usuario"
} else {
    Write-Output "No se puede crear el usuario $Usuario a $Nombre $Apellido por
}
```

monedas.ps1 | **Solución** | Solución (n monedas)

```
$numero_tiradas=1000
$tres_caras=0
for ($i=0;$i -lt $numero_tiradas;$i++){
    $m1=$(Get-Random -Minimum 0 -Maximum 2)
    $m2=$(Get-Random -Minimum 0 -Maximum 2)
    $m3=$(Get-Random -Minimum 0 -Maximum 2)

    if (($m1 -eq 0) -and ($m2 -eq 0) -and ($m3 -eq 0)){
        $tres_caras++
        }
}

echo "La probabilidad es ($($tres_caras/$numero_tiradas))"
```

rnd.ps1 | **Solución (while)** | Solución (do while) | Solución (do while)

```powershell
# rnd.ps1

# Generar un número aleatorio entre 1 y 20
$NumeroAleatorio = Get-Random -Minimum 1 -Maximum 21

# Inicializar el contador de intentos
$contador = 0

Write-Output "He generado un número entre 1 y 20. ¡Intenta adivinarlo!"

# Inicializar la respuesta fuera del rango válido para entrar al bucle
[int] $respuesta = -1

while ($respuesta -ne $NumeroAleatorio) {

    [int] $respuesta = Read-Host "Introduce tu suposición"
    $contador++

    if ($respuesta -lt $NumeroAleatorio) {
        Write-Output "El número es mayor que $respuesta. Intenta de nuevo."
    }

    if ($respuesta -gt $NumeroAleatorio) {
        Write-Output "El número es menor que $respuesta. Intenta de nuevo."
    }

    if ($respuesta -eq $NumeroAleatorio) {
        Write-Output "¡Felicidades! Has adivinado el número."
        Write-Output "Número de intentos: $contador"
    }
}
```

dados.ps1 | **Solución (args)** | Solución (param)

```powershell
# dados.ps1

function Tirar_dados($Tiradas){
    "dado1,dado2,suma" > tiradas.csv
    for ($i = 1; $i -le $Tiradas; $i++) {
        $dado1 = Get-Random -Minimum 1 -Maximum 7
        $dado2 = Get-Random -Minimum 1 -Maximum 7
        $suma = $dado1 + $dado2
        "$dado1,$dado2,$suma" >> tiradas.csv
        }
        Write-Output "$Tiradas tiradas generadas y guardadas en tiradas.csv"
    }

function Show-Help {
    Write-Output "Uso del script dados.ps1:"
    Write-Output ".\dados.ps1 <número_de_tiradas>"
    Write-Output "Ejemplo: .\dados.ps1 100"
    Write-Output "Si no se proporciona ningún argumento, se preguntará cuántas
    Write-Output "Si se usa el argumento 'help', se mostrará esta ayuda."
    }

[int] $Tiradas = 0

if ($args.Length -eq 0) {

    $Tiradas = Read-Host "¿Cuántas tiradas quieres hacer?"
    Tirar_dados($Tiradas)

    }elseif ($args[0] -eq "help"){
        Show-Help
    }else{
        $Tiradas = [int] $args[0]
        Tirar_dados($Tiradas)
    }
```

analisis.ps1 | **Solución (variables)** | Solución (arrays)

```
# analisis.ps1

$A = Import-Csv -Path tiradas.csv
$total=$A.Count
echo "leemos $total tiradas"

$s2=0
$s3=0
$s4=0
$s5=0
$s6=0
$s7=0
$s8=0
$s9=0
$s10=0
$s11=0
$s12=0

foreach ($i in $A.Suma){
  if ($i -eq 2) {$s2++}
  if ($i -eq 3) {$s3++}
  if ($i -eq 4) {$s4++}
  if ($i -eq 5) {$s5++}
  if ($i -eq 6) {$s6++}
  if ($i -eq 7) {$s7++}
  if ($i -eq 8) {$s8++}
  if ($i -eq 9) {$s9++}
  if ($i -eq 10) {$s10++}
  if ($i -eq 11) {$s11++}
  if ($i -eq 12) {$s12++}

}

$salida="2 ($([math]::Round(($s2 / $total) * 100, 0))%), "
$salida+="3 ($([math]::Round(($s3 / $total) * 100, 0))%), "
$salida+="4 ($([math]::Round(($s4 / $total) * 100, 0))%), "
$salida+="5 ($([math]::Round(($s5 / $total) * 100, 0))%), "
$salida+="6 ($([math]::Round(($s6 / $total) * 100, 0))%), "
$salida+="7 ($([math]::Round(($s7 / $total) * 100, 0))%), "
$salida+="8 ($([math]::Round(($s8 / $total) * 100, 0))%), "
$salida+="9 ($([math]::Round(($s9 / $total) * 100, 0))%), "
$salida+="10 ($([math]::Round(($s10 / $total) * 100, 0))%), "
$salida+="11 ($([math]::Round(($s11 / $total) * 100, 0))%), "
$salida+="12 ($([math]::Round(($s12 / $total) * 100, 0))%)"

echo $salida
```

tragaperras.ps1 **Solución**

```powershell
$rulos = @('limon', 'manzana', 'platano', 'siete')
$numero_tiradas=1000

function Girar-Rulos {
    $ganacia = -1
    $rulo1=$rulos[(Get-Random -Minimum 0 -Maximum $rulos.Length)]
    $rulo2=$rulos[(Get-Random -Minimum 0 -Maximum $rulos.Length)]
    $rulo3=$rulos[(Get-Random -Minimum 0 -Maximum $rulos.Length)]

    if( ($rulo1 -eq $rulo2) -and ($rulo1 -eq $rulo3)  ){
        $ganacia+=1
        if($rulo1 -eq "siete"){
            $ganacia+=99
            }
        }

    #Write-Output "$rulo1,$rulo2,$rulo3"
    return $ganacia
    }

$jugador=10
for ($j=0; $j -lt $numero_tiradas; $j++){
    $jugador+=Girar-Rulos
    Write-Output $jugador
    if ($jugador -lt 0){
        Write-Output "se arruina con $j tiradas "
        #exit
        $j=$numero_tiradas
        }
    }
```

csv.ps1 | **Solución** | -delete | Solución

```powershell
param (
    #[Parameter(Mandatory = $true)] en el caso de que ingreses usuario, te lo p
    [string]$usuario,

    [string]$grupo="A"
)

$archivoCSV = "usuarios.csv"


if(! $usuario){ #caso de no recibir nombre de usuario se para
echo "no me han dado usuario"
exit
}


function GenerarPassword {
    $chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#
    $salida=""
    for ($i=0;$i -lt 8;$i++){
     $r = Get-Random -Minimum 0 -Maximum $chars.Length
     $salida+=$chars[$r]
    }
    Write-Output $salida
}

# Si no existe crear el encabezado
if (!(Test-Path $archivoCSV)) {
    Write-Output "usuario,grupo,password" >  $archivoCSV
}

# Si el usuario ya existe
$usuariosExistentes = Import-Csv -Path $archivoCSV

if ($usuariosExistentes | Where-Object { $_.usuario -eq $usuario })
{
    Write-Output "El usuario $usuario ya existe, no se puede crear."
}else{
    $password = GenerarPassword
    Write-Output "$usuario,$grupo,$password" >> $archivoCSV
    Write-Output "El usuario $usuario ha sido creado con la password $password
}
```

crear_usuarios_grupos.ps1 **Solución**

```powershell
# crear_usuarios_grupos.ps1

# Bucle para crear los grupos GPWS01 a GPWS09
for ($i = 1; $i -le 9; $i++) {

    $nombre_grupo = "GPWS{0:D2}" -f $i
    $grupo_existe =  Get-LocalGroup | Select-String -Pattern $nombre_grupo -Qui

    if ($grupo_existe) {
        Write-Output "El grupo $nombre_grupo existe, no se crea"
    } else {
        # Si el grupo no existe, crear el grupo
        New-LocalGroup -Name $nombre_grupo -Description "Grupo $nombre_grupo cr
        Write-Output "El grupo $nombre_grupo no existe, se crea"
    }
}
```

crear_usuarios_grupos.ps1 **Solución**

```powershell
# crear_usuarios_grupos.ps1

$nombre_base = "tunombre_gpws"

#Si da problemas poner una contraseña más compleja
$contraseña = "alumno"

# Bucle para crear los grupos GPWS01 a GPWS09
for ($i = 1; $i -le 9; $i++) {
    # Formatear el nombre del grupo
    $nombre_grupo = "GPWS{0:D2}" -f $i

    # Verificar si el grupo ya existe
    $grupo_existe = Get-LocalGroup | Where-Object { $_.Name -eq $nombre_grupo }

    if ($grupo_existe) {
        Write-Output "El grupo $nombre_grupo existe, no se crea"
    } else {
        New-LocalGroup -Name $nombre_grupo -Description "Grupo $nombre_grupo cr
        Write-Output "El grupo $nombre_grupo no existe, se crea"
    }

    # Bucle para crear los usuarios tunombre_gpwsXX_01 a tunombre_gpwsXX_09
    for ($j = 1; $j -le 9; $j++) {
        # Formatear el nombre del usuario
        $nombre_usuario = "{0}{1:D2}_{2:D2}" -f $nombre_base, $i, $j

        # Verificar si el usuario ya existe
        $usuario_existe = Get-LocalUser | Where-Object { $_.Name -eq $nombre_us

        if ($usuario_existe) {
            Write-Output "El usuario $nombre_usuario existe, no se crea"
        } else {
            # Crear el usuario con la contraseña especificada
            New-LocalUser -Name $nombre_usuario -Password (ConvertTo-SecureStri
            Write-Output "El usuario $nombre_usuario con grupo $nombre_grupo nc
        }

        # Verificar si el usuario pertenece al grupo específico
        $miembro_del_grupo = Get-LocalGroupMember -Group $nombre_grupo | Where-

        if ($miembro_del_grupo) {
            Write-Output "El usuario $nombre_usuario ya está en el grupo $nombr
        } else {
            Add-LocalGroupMember -Group $nombre_grupo -Member $nombre_usuario
            Write-Output "El usuario $nombre_usuario no está en el grupo $nombr
        }

        # Verificar si el usuario pertenece al grupo Usuarios, para que pueda
        $miembro_de_usuarios = Get-LocalGroupMember -Group "Usuarios" | Where-O

        if ($miembro_de_usuarios) {
            Write-Output "El usuario $nombre_usuario ya está en el grupo Usuari
        } else {
            Add-LocalGroupMember -Group "Usuarios" -Member $nombre_usuario
            Write-Output "El usuario $nombre_usuario no está en el grupo Usuari
        }
```

```
        }
    }
```

crear_usuarios_grupos_csv.ps1 **Solución**

```powershell
# Script: crear_usuarios_grupos_csv.ps1

# Leer el archivo CSV
$users = Import-Csv -Path "users.csv"

# Recorrer cada usuario en el archivo CSV
foreach ($user in $users) {
    $nombre_usuario = $user.usuario
    $nombre_grupo = $user.grupo
    $password = $user.password
    $entorno_grafico = $user.EntGraf

    # Verificar si el grupo existe
    if (-Not (Get-LocalGroup -Name $nombre_grupo -ErrorAction SilentlyContinue)
        Write-Output "El grupo $nombre_grupo no existe, se crea"
        New-LocalGroup -Name $nombre_grupo
    } else {
        Write-Output "El grupo $nombre_grupo existe, no se crea"
    }

    # Verificar si el usuario existe
    $usuarioExiste = Get-LocalUser -Name $nombre_usuario -ErrorAction SilentlyC
    if ($usuarioExiste) {
        Write-Output "El usuario $nombre_usuario existe, no se crea"

        # Verificar si el usuario está en el grupo
        $miembroDelGrupo = Get-LocalGroupMember -Group $nombre_grupo | Where-Ob
        if ($miembroDelGrupo) {
            Write-Output "El usuario $nombre_usuario ya está en el grupo $nombr
        } else {
            Write-Output "El usuario $nombre_usuario no está en el grupo $nombr
            Add-LocalGroupMember -Group $nombre_grupo -Member $nombre_usuario
        }
    } else {
        # Crear el usuario
        Write-Output "El usuario $nombre_usuario con grupo $nombre_grupo no exi
        New-LocalUser -Name $nombre_usuario -Password (ConvertTo-SecureString $
        Add-LocalGroupMember -Group $nombre_grupo -Member $nombre_usuario
        $logMessage = "El usuario $nombre_usuario con $password en el grupo $nc
        $logMessage | Out-File -FilePath "lista_usuarios_creados.dat" -Append
    }

    # Verificar si el usuario debe tener acceso al entorno gráfico
    if ($entorno_grafico -eq "Y") {
        # Verificar si el usuario está en el grupo Usuarios
        $miembroUsuarios = Get-LocalGroupMember -Group "Usuarios" | Where-Objec
        if ($miembroUsuarios) {
            Write-Output "El usuario $nombre_usuario ya está en el grupo Usuari
        } else {
            Write-Output "El usuario $nombre_usuario no está en el grupo Usuari
            Add-LocalGroupMember -Group "Usuarios" -Member $nombre_usuario
        }
    } else {
        Write-Output "El usuario $nombre_usuario no requiere acceso al entorno
```

```
        }
    }
```

grupos.ps1 **Solución**

```powershell
# Script: usuarios.ps1

function Listar-Usuarios {
    Write-Output "Usuarios en el sistema:"
    Get-LocalUser | ForEach-Object { Write-Output $_.Name }
}

function Crear-Usuario {
    $nombreUsuario = Read-Host "Ingrese el nombre del nuevo usuario"
    $password = Read-Host "Ingrese la contraseña del nuevo usuario" -AsSecureSt

    if (Get-LocalUser -Name $nombreUsuario -ErrorAction SilentlyContinue) {
        Write-Output "El usuario $nombreUsuario ya existe."
    } else {
        New-LocalUser -Name $nombreUsuario -Password $password -FullName $nombr
        Write-Output "El usuario $nombreUsuario ha sido creado."
    }
}

function Eliminar-Usuario {
    $nombreUsuario = Read-Host "Ingrese el nombre del usuario a eliminar"

    if (Get-LocalUser -Name $nombreUsuario -ErrorAction SilentlyContinue) {
        Remove-LocalUser -Name $nombreUsuario
        Write-Output "El usuario $nombreUsuario ha sido eliminado."
    } else {
        Write-Output "El usuario $nombreUsuario no existe."
    }
}

function Modificar-Usuario {
    $nombreUsuario = Read-Host "Ingrese el nombre del usuario a modificar"
    $nuevoNombre = Read-Host "Ingrese el nuevo nombre para el usuario"

    if (Get-LocalUser -Name $nombreUsuario -ErrorAction SilentlyContinue) {
        Rename-LocalUser -Name $nombreUsuario -NewName $nuevoNombre
        Write-Output "El usuario $nombreUsuario ha sido renombrado a $nuevoNomb
    } else {
        Write-Output "El usuario $nombreUsuario no existe."
    }
}

do {
    Write-Host "`nMenú de Usuarios:"
    Write-Host "1. Listar usuarios"
    Write-Host "2. Crear usuario"
    Write-Host "3. Eliminar usuario"
    Write-Host "4. Modificar usuario"
    Write-Host "5. Salir"
    $opcion = Read-Host "Seleccione opción"

    switch ($opcion) {
        1 { Listar-Usuarios }
        2 { Crear-Usuario }
        3 { Eliminar-Usuario }
        4 { Modificar-Usuario }
        5 { Write-Host "Saliendo..." }
```

```
        Default { Write-Host "Opción incorrecta. Intente de nuevo." }
    }
} until ($opcion -eq 5)
```

grupos.ps1 **Solución**

```powershell
# Script: grupo.ps1

function Listar-Grupos {
    Write-Output "Grupos disponibles:"
    Get-LocalGroup | ForEach-Object { Write-Output $_.Name }
}

function Ver-MiembrosDeGrupo {
    $nombreGrupo = Read-Host "Ingrese el nombre del grupo"
    if (Get-LocalGroup -Name $nombreGrupo -ErrorAction SilentlyContinue) {
        Write-Output "Miembros del grupo $nombreGrupo:"
        Get-LocalGroupMember -Group $nombreGrupo | ForEach-Object { Write-Outpu
    } else {
        Write-Output "El grupo $nombreGrupo no existe."
    }
}

function Crear-Grupo {
    $nombreGrupo = Read-Host "Ingrese el nombre del nuevo grupo"
    if (Get-LocalGroup -Name $nombreGrupo -ErrorAction SilentlyContinue) {
        Write-Output "El grupo $nombreGrupo ya existe."
    } else {
        New-LocalGroup -Name $nombreGrupo
        Write-Output "El grupo $nombreGrupo ha sido creado."
    }
}

function Eliminar-Grupo {
    $nombreGrupo = Read-Host "Ingrese el nombre del grupo a eliminar"
    if (Get-LocalGroup -Name $nombreGrupo -ErrorAction SilentlyContinue) {
        Remove-LocalGroup -Name $nombreGrupo
        Write-Output "El grupo $nombreGrupo ha sido eliminado."
    } else {
        Write-Output "El grupo $nombreGrupo no existe."
    }
}

function Crear-MiembroDeGrupo {
    $nombreGrupo = Read-Host "Ingrese el nombre del grupo"
    $nombreUsuario = Read-Host "Ingrese el nombre del usuario a agregar"
    if (Get-LocalGroup -Name $nombreGrupo -ErrorAction SilentlyContinue) {
        Add-LocalGroupMember -Group $nombreGrupo -Member $nombreUsuario
        Write-Output "El usuario $nombreUsuario ha sido agregado al grupo $nomb
    } else {
        Write-Output "El grupo $nombreGrupo no existe."
    }
}

function Eliminar-MiembroDeGrupo {
    $nombreGrupo = Read-Host "Ingrese el nombre del grupo"
    $nombreUsuario = Read-Host "Ingrese el nombre del usuario a eliminar"
    if (Get-LocalGroup -Name $nombreGrupo -ErrorAction SilentlyContinue) {
        Remove-LocalGroupMember -Group $nombreGrupo -Member $nombreUsuario
        Write-Output "El usuario $nombreUsuario ha sido eliminado del grupo $nc
    } else {
        Write-Output "El grupo $nombreGrupo no existe."
    }
```

```
}

do {
    Write-Host "`nMenú de Grupos:"
    Write-Host "1. Listar grupos"
    Write-Host "2. Ver miembros de un grupo"
    Write-Host "3. Crear grupo"
    Write-Host "4. Eliminar grupo"
    Write-Host "5. Crear miembro de un grupo"
    Write-Host "6. Eliminar miembro de un grupo"
    Write-Host "7. Salir"
    $opcion = Read-Host "Seleccione opción"

    switch ($opcion) {
        1 { Listar-Grupos }
        2 { Ver-MiembrosDeGrupo }
        3 { Crear-Grupo }
        4 { Eliminar-Grupo }
        5 { Crear-MiembroDeGrupo }
        6 { Eliminar-MiembroDeGrupo }
        7 { Write-Host "Saliendo..." }
        Default { Write-Host "Opción incorrecta. Intente de nuevo." }
    }
} until ($opcion -eq 7)
```