PowerShell Ateliers

Table of Contents

Préambule : Ces exercices sont tous réalisables avec le contenu du cours et l'aide intégrée de PowerShell, rien d'autre n'est nécessaire hormis votre cerveau.
Ces exercices peuvent tous être complété avec l'IA mais ce n'est pas elle qui passera l'examen a votre place et le jury tout comme vos futurs employeurs ne vous fourniront pas du travail par ce que vous savez faire un prompt correct. Merci de faire l'effort de vous en passer pour cette semaine.

1. Exercices Didactiques en PowerShell

Pour chaque exercice vous devrez livrer :
- Un organigramme
- Un script commenté

1.1. Exercice : Contrôler la connexion réseau

- **Objectif** : Identifier si une adresse IP répond au ping. (IF)
- **Instruction** :
  1. Demandez à l'utilisateur d'entrer une adresse IP.
  2. Effectuez une commande `Test-Connection` sur cette adresse.
  3. Utilisez une structure `If/Else` pour afficher si l'adresse est joignable ou non.
- **Indication** : Testez avec `-Quiet` pour obtenir un résultat booléen.

```
# Demander à l'utilisateur une adresse IP
$ip = Read-Host "Entrez une adresse IP"

# Tester la connexion avec Test-Connection
if (Test-Connection -ComputerName $ip -Count 2 -Quiet) {
    Write-Output "L'adresse $ip est joignable."
} else {
    Write-Output "L'adresse $ip n'est pas joignable."
}
```

1.2. Exercice : Analyse de réponses

- **Objectif** : Comprendre l'utilisation de `Switch` pour traiter des entrées multiples.
- **Instruction** :
  1. Demandez à l'utilisateur de choisir une option parmi "démarrer", "arrêter" ou "redémarrer".
  2. Utilisez une structure `Switch` pour afficher le message approprié pour chaque cas.
- **Indication** : Prévoyez une réponse par défaut pour les choix invalides.

```
# Demander une action à l'utilisateur
$action = Read-Host "Choisissez une action (démarrer, arrêter, redémarrer)"

# Utiliser Switch pour traiter les entrées
switch ($action) {
    "démarrer" { Write-Output "Le service démarre..." }
    "arrêter" { Write-Output "Le service s'arrête..." }
    "redémarrer" { Write-Output "Le service redémarre..." }
    default { Write-Output "Option invalide. Veuillez choisir une action valide." }
```

```
}
```

1.3. Exercice : Identifier une plage IP

- **Objectif** : Utiliser `Switch` pour associer une adresse IP à une catégorie (ex. publique ou autre, privée), respecter les informations de la RFC 1918.
- **Instruction** :
  1. Demandez à l'utilisateur d'entrer une adresse IP.
  2. En fonction des plages connues (ex. 192.168.x.x, 10.x.x.x), afficher le type d'adresse.
- **Indication** : l'utilisation de regex pourrait aider.

```
# Demander une adresse IP à l'utilisateur
$ip = Read-Host "Entrez une adresse IP"

# Vérifier la catégorie de l'IP
if ($ip -match "^192\.168\.") {
    Write-Output "L'adresse $ip est une adresse privée."
} elseif ($ip -match "^10\.") {
    Write-Output "L'adresse $ip est une adresse privée."
} elseif ($ip -match "^172\.(1[6-9]|2[0-9]|3[0-1])\.") {
    Write-Output "L'adresse $ip est une adresse privée."
} else {
    Write-Output "L'adresse $ip est une adresse publique."
}
```

1.4. Exercice : Créer un menu interactif

- **Objectif** : Utiliser `Switch` pour construire un menu permettant de choisir différentes actions.
- **Instruction** :
  1. Afficher un menu avec plusieurs options (par exemple : "Afficher la date", "Lister les fichiers", "Quitter").
  2. L'utilisateur choisit une option, et une action spécifique est exécutée.
  3. Répétez le menu jusqu'à ce que l'utilisateur choisisse de quitter.

```
do {
    Write-Host "1. Afficher la date"
    Write-Host "2. Lister les fichiers"
    Write-Host "3. Quitter"
    $choix = Read-Host "Votre choix"

    switch ($choix) {
        "1" { Get-Date }
        "2" { Get-ChildItem }
```

```
        "3" { break }
        default { Write-Host "Option invalide" }
    }
} while ($choix -ne "3")
```

1.5. Exercice : Surveillance CPU

- **Objectif** : Créer une boucle pour analyser des données système.
- **Instruction** :
  1. Répéter 5 fois une commande de surveillance CPU.
  2. Afficher les valeurs obtenues pour chaque itération.
  3. Une fois que cela fonctionne avec le CPU modifiez le script pour qu'il affiche également les informations pour la RAM et l'interface réseau. 4.
- **Indication** : Consultez l'aide de la cmdlet `Get-Counter`. Une pause peut etre introduite entre les itérations avec `Start-Sleep`.

```
Write-host "Affichage des performances de la machine $env:COMPUTERNAME"
for ($i=0; $i -lt 5; $i++) {
    Get-Counter -counter '\Processor(_Total)\% Processor Time'
    Get-Counter -counter '\memory\% committed bytes in use'
    Get-Counter -counter "\network interface(*)\bytes total/sec"
    Start-Sleep -Seconds 5
}
```

1.6. Exercice : Deviner un nombre

- **Objectif** : Maintenir une boucle tant qu'une condition n'est pas remplie.
- **Instruction** :
  1. le script définira aléatoirement un nombre entre 0 et 9.
  2. Afficher ce nombre (sera commenté par la suite, sert pour le debug).
  3. Afficher un texte qui invite le joueur a faire une proposition pour deviner le nombre.
  4. Afficher les deux nombres. (sera commenté par la suite, sert au debug)
  5. Tester si l'utilisateur a correctement deviné. Si il a gagné le script le félicite et quitte, sinon il l'invite a essayer a nouveau.
  6. Répéter jusqu'à ce que la réponse soit correcte.
- **Bonus** :
  1. Afficher le nombre d'essais nécessaire a l'utilisateur pour gagner.
  2. Les saisies incorrectes ne sont pas prise en compte dans le nombre d'essai.
  3. Le script proposera différent mode de jeux a l'utilisateur via un menu : La valeur a deviner doit elle être :

-comprise entre 0 et 9
-comprise entre 0 et 50
-comprise entre 0 et 100

- **Indication** : L'aleatoire en PowerShell : `Get-Random`. Affichez des indices (plus ou moins).

```
$random = Get-Random -Minimum 0 -Maximum 10
$essais = 0

while ($true) {
    $guess = Read-Host "Devinez le nombre entre 0 et 9"
    $essais++
    if ($guess -eq $random) {
        Write-Host "Bravo! Vous avez deviné en $essais essais." -ForegroundColor Green
        break
    } elseif ($guess -lt $random) {
        Write-Host "Trop bas!" -ForegroundColor Yellow
    } else {
        Write-Host "Trop haut!" -ForegroundColor Yellow
    }
}
```

1.7. Exercice : Suivi des services

- **Objectif** : Surveiller l'état d'un service.
- **Instruction** :
  1. Demander à l'utilisateur le nom d'un service.
  2. Utiliser une boucle `While` pour afficher toutes les 5 secondes l'état du service.
  3. Arrêter la boucle si l'état passe à "Stopped".
- **Indication** : Utilisez `Get-Service` pour suivre l'état.

```
## affichage des nom dess services
Get-Service | Select-Object Name | Format-Wide -AutoSize
## demande de saisie du service
$serviceName = Read-Host "Entrez le nom du service"
## boucle tant que le service n'est pas arreté
do {
    ## stockage du nom de service
    $service = Get-Service -Name $serviceName
    ## affichage de l'etat du service
    Write-Host "État du service: $($service.Status)"
    ## pause
    Start-Sleep -Milliseconds 50
    ## arret du service
    Stop-Service $serviceName
} while ($service.Status -ne "Stopped")
```

```
## affichage du service pour validation de l'arret
Write-Host "État du service: $($service.Status)"
```

1.8. Exercice : Contrôle de mot de passe

- **Objectif** : Tester une entrée utilisateur jusqu'à ce qu'elle soit valide.
- **Instruction** :
  1. Demander à l'utilisateur un mot de passe prédéfini (ex. "admin123").
  2. Répéter la demande tant que le mot de passe est incorrect.
  3. Afficher un message de validation si la réponse est correcte.
- **bonus** : Trouver comment faire pour que la demande de saisie d'un mot de passe n'affiche rien a l'écran.

```
## boucle
do {
    ## demande de saisie -ASsecureString permet de masquer ce qui est tappé (-
MaskInput aurait permit de le faire aussi mais sans chiffrer) et de le securiser
    $password = Read-Host "Entrez le mot de passe" -AsSecureString
    Write-Host "la variable `$password : $password"
    ## on retransforme le password en non chiffré
    $plainTextPassword = ConvertFrom-SecureString -SecureString $password -AsPlainText
    Write-Host "La variable `$plainTextPassword : $plainTextPassword"
} while ($plainTextPassword -ne "admin123")
Write-Host "Mot de passe correct!"
```

1.9. Exercice : Détection de port ouvert

- **Objectif** : Répéter une commande jusqu'à ce qu'une condition soit remplie.
- **Instruction** :
  1. Demander à l'utilisateur un numéro de port.
  2. Utiliser une commande pour tester si le port est ouvert (ex. `Test-NetConnection`).
  3. Répéter toutes les 3 secondes jusqu'à ce que le port soit ouvert.

```
## demande de saisie du port
$port = Read-Host "Entrez un numéro de port"
## boucle
do {
    ## recherche du port ouvert ou non
    $result = Test-NetConnection -Port $port -InformationLevel Quiet
    if ($result) {
        Write-Host "Le port $port est ouvert." -ForegroundColor Green
        break
    } else {
```

```
        Write-Host "Le port $port est fermé, nouvelle vérification dans 3s..."
        Start-Sleep -Seconds 3
    }
} while ($true) ## boucle infinie
```

1.10. Exercice : Liste de fichiers

- **Objectif** : Parcourir une liste avec `Foreach`.
- **Instruction** :
  1. Créer une liste de 5 noms de fichiers factices.
  2. Pour chaque nom dans la liste, afficher "Traitement de [nom]".
  3. Afficher un message final une fois la boucle terminée.
- **bonus** : Faites en sortes que le script test si le fichier est bien présent ou non dans l'arborescence souhaitée.

```
## definition du repertoire de travail
$rep = "C:\butch\script\"
<# soluce sans le bonus
$files = @("file1.txt", "file2.txt", "file3.txt", "file4.txt", "file5.txt")
foreach ($file in $files) {
    Write-Host "Traitement de $file"
}
Write-Host "Tous les fichiers ont été traités."
#>

<# Bonus
## creation du repertoire
Try { test-path $rep }
catch { New-Item -ItemType Directory -Name $rep }
## creation des fichiers :
$files = @("file1.txt", "file2.txt", "file3.txt", "file4.txt", "file5.txt")
foreach ($file in $files) {
    New-Item -ItemType File -name $file
    Write-Host "Le fichier $file a été créé."
}
#>

## simulation de traitement et verification de presence
$files = @("file1.txt", "file2.txt", "file3.txt", "file4.txt", "file5.txt")
foreach ($file in $files) {
    $verif = Test-Path -Path $rep$file
    if ( $verif -like "true" )
    {
    Write-Host "Traitement de $file"
    }
    Else { Write-Host "Fichier non present"}
}
```

```
Write-Host "Tous les fichiers ont été traités."
```

1.11. Exercice : Utilisation des adresses MAC

- **Objectif** : Parcourir les résultats d'une commande système.
- **Instruction** :
  1. Récupérer toutes les adresses MAC disponibles (ex. `Get-NetAdapter`).
  2. Pour chaque adaptateur, afficher son nom et son adresse MAC, si il a un adresse IP affichez la et le script indiquera si elle répond au ping ou non.
- **Indication** : Utilisez `Select-Object` pour cibler les propriétés.

```
# Récupérer tous les adaptateurs réseau
$adapters = Get-NetAdapter | Where-Object { $PSItem.Status -eq 'Up' }

foreach ($adapter in $adapters) {
    # Afficher le nom de l'adaptateur
    Write-Host "Adaptateur : $($adapter.Name)"
    # Afficher l'adresse MAC
    Write-Host "Adresse MAC : $($adapter.MacAddress)"
    # Récupérer les adresses IP associées à cet adaptateur
    $ipAddresses = Get-NetIPAddress -InterfaceIndex $adapter.InterfaceIndex | Where-
Object { $PSItem.AddressFamily -eq 'IPv4' }
    if ($ipAddresses) {
        foreach ($ip in $ipAddresses) {
            # Afficher l'adresse IP
            Write-Host "Adresse IP : $($ip.IPAddress)"
            # Tester la connectivité avec un ping
            $pingResult = Test-Connection -ComputerName $ip.IPAddress -Count 1 -Quiet
            if ($pingResult) {
                Write-Host "Ping $($ip.IPAddress) : Répond"
            } else {
                Write-Host "Ping $($ip.IPAddress) : Ne répond pas"
            }
        }
    } else {
        Write-Host "Aucune adresse IP associée."
    }
    Write-Output "-----------------------------"
}
```

2. Exercices scripting

2.1. Audit de fichiers

- **Objectif** : Audit de fichiers
- **Instruction** : Créez un script qui listera les fichiers non-modifié depuis un labs de temps donné. Il affichera au moins le nom, le chemin complet, le poids en MO avec seulement 2 caractères après la virgule, la date de dernière modification. Leur classement se fera par date puis par poids.

```
# Demande à l'utilisateur le dossier cible
$folderPath = Read-Host "Entrez le chemin du dossier à analyser"

# Vérification de l'existence du dossier
if (-Not (Test-Path -Path $folderPath -PathType Container)) {
    Write-Host "Le dossier spécifié n'existe pas." -ForegroundColor Red
    exit
}

# Demande du délai en jours
$days = Read-Host "Entrez le nombre de jours d'ancienneté des fichiers à rechercher"
$limitDate = (Get-Date).AddDays(-[int]$days)

# Récupération des fichiers non modifiés depuis le temps donné
$files = Get-ChildItem -Path $folderPath -File -Recurse | Where-Object {
$_.LastWriteTime -lt $limitDate }

# Vérification si des fichiers correspondent au critère
if ($files.Count -eq 0) {
    Write-Host "Aucun fichier trouvé avec cette ancienneté." -ForegroundColor Yellow
    exit
}

# Création d'un tableau d'objets pour affichage structuré
$results = $files | Select-Object Name, @{Name="Chemin Complet";
Expression={$_.FullName}},
    @{Name="Poids (Mo)"; Expression={[math]::Round($_.Length / 1MB, 2)}},
    @{Name="Dernière Modification"; Expression={$_.LastWriteTime}} |
    Sort-Object "Dernière Modification", "Poids (Mo)"

# Affichage des résultats sous forme de tableau
$results | Format-Table -AutoSize
```

2.2. Configuration IP

- **Objectif** : Configuration IP
- **Instruction** : Réalisez un script qui affichera les noms, adresse MAC, statut et description des équipements réseaux.
  Il devra permettre a l'utilisateur de choisir une carte afin de n'afficher que les informations qui

la concerne.
Faites en sortes que le script adapte le serveur DNS configuré en fonction de l'adresse IP de l'interface. (10.0 ou 10.100 selon le site).

- **Bonus** : Le script sera autonome et ne demandera pas de saisie utilisateur, se lancera a chaque démarrage du système.
- **Indication** : A tester sur vos VMs uniquement.

```
# Récupération des cartes réseau
$adapters = Get-NetAdapter | Where-Object { $_.Status -eq "Up" }

# Vérification s'il y a des interfaces réseau disponibles
if ($adapters.Count -eq 0) {
    Write-Host "Aucune interface réseau active trouvée." -ForegroundColor Red
    exit
}

# Affichage des informations sur les interfaces réseau
Write-Host "Interfaces réseau disponibles :"
$adapters | Format-Table -Property Name, MacAddress, Status, InterfaceDescription
-AutoSize

# Demande à l'utilisateur de choisir une carte réseau
$adapterName = Read-Host "Entrez le nom de l'interface que vous souhaitez voir en
détail"

# Vérification si l'interface existe
$selectedAdapter = $adapters | Where-Object { $_.Name -eq $adapterName }
if (-Not $selectedAdapter) {
    Write-Host "Interface non trouvée." -ForegroundColor Red
    exit
}

# Récupération des informations détaillées
$ipConfig = Get-NetIPAddress -InterfaceAlias $selectedAdapter.Name -AddressFamily IPv4
-ErrorAction SilentlyContinue

Write-Host "`nDétails de l'interface sélectionnée :"
Write-Host "Nom : $($selectedAdapter.Name)"
Write-Host "Adresse MAC : $($selectedAdapter.MacAddress)"
Write-Host "Statut : $($selectedAdapter.Status)"
Write-Host "Description : $($selectedAdapter.InterfaceDescription)"

if ($ipConfig) {
    Write-Host "Adresse IP : $($ipConfig.IPAddress)"
    # Détermination du serveur DNS à configurer
    if ($ipConfig.IPAddress -match "^10\.0\.") {
        $dnsServer = "10.0.0.1"
    } elseif ($ipConfig.IPAddress -match "^10\.100\.") {
        $dnsServer = "10.100.0.1"
```

```
    } else {
        $dnsServer = $null
    }

    # Configuration du serveur DNS si nécessaire
    if ($dnsServer) {
        Set-DnsClientServerAddress -InterfaceAlias $selectedAdapter.Name
-ServerAddresses $dnsServer
        Write-Host "Serveur DNS mis à jour : $dnsServer" -ForegroundColor Green
    } else {
        Write-Host "Aucune mise à jour du DNS nécessaire." -ForegroundColor Yellow
    }
} else {
    Write-Host "Pas d'adresse IP assignée à cette interface." -ForegroundColor Yellow
}

# ---- BONUS : Ajouter au démarrage du système ----
$taskName = "Auto-Configure-Reseau"
$scriptPath = "$PSScriptRoot\configure_reseau.ps1"

if (-Not (Get-ScheduledTask -TaskName $taskName -ErrorAction SilentlyContinue)) {
    Write-Host "Ajout du script au démarrage..." -ForegroundColor Cyan
    $action = New-ScheduledTaskAction -Execute "powershell.exe" -Argument "-
ExecutionPolicy Bypass -File `"$scriptPath`""
    $trigger = New-ScheduledTaskTrigger -AtStartup
    Register-ScheduledTask -TaskName $taskName -Action $action -Trigger $trigger
-RunLevel Highest -User "NT AUTHORITY\SYSTEM"
    Write-Host "Tâche planifiée créée : $taskName" -ForegroundColor Green
} else {
    Write-Host "Le script est déjà configuré pour s'exécuter au démarrage."
-ForegroundColor Yellow
}
```

2.3. Application

- **Objectif** : Vérification de la présence d'une application
- **Instruction** : Réalisez un script qui va vérifier si VMWare workstaion est présent sur la machine
  le test se fait sur le répertoire d'installation.
  le test se fait par rapport a la base de registre.
- **Bonus** : on souhaite en plus vérifier si la version est a jour et alerter l'administrateur si ce n'est pas le cas.
- **Indication** : il y'a peut être une variable qui stocke l'information des chemins des programmes installés ?

```
## a reprendre
# Recherche du chemin d'installation de VMware Workstation dans les variables
```

```
d'environnement
$installPath = ($env:Path -split ";" | Where-Object { $PSItem -match "VMware
Workstation" }) -join ";"

# Si le chemin n'est pas trouvé dans $env:Path, utilisation du chemin par défaut
if (-not $installPath) {
    $installPath = "C:\Program Files (x86)\VMware\VMware Workstation"
}

$registryPath = "HKLM:\SOFTWARE\WOW6432Node\VMware, Inc.\VMware Workstation"
$latestVersionURL =
"https://customerconnect.vmware.com/en/downloads/details?downloadGroup=WKST-PLAYER-
1703&productId=1039"

# Vérification du dossier d'installation
if (Test-Path $installPath) {
    Write-Host "VMware Workstation détecté dans : $installPath" -ForegroundColor Green
} else {
    Write-Host "VMware Workstation non trouvé dans le dossier d'installation."
-ForegroundColor Red
}

# Vérification via la base de registre
$vmwareInstalled = $false
$installedVersion = $null
if (Test-Path $registryPath) {
    $installedVersion = (Get-ItemProperty -Path $registryPath -Name
"DisplayVersion").DisplayVersion
    Write-Host "VMware Workstation trouvé dans la base de registre." -ForegroundColor
Green
    Write-Host "Version installée : $installedVersion"
    $vmwareInstalled = $true
} else {
    Write-Host "VMware Workstation non trouvé dans la base de registre."
-ForegroundColor Red
}

# Vérification de la dernière version disponible
if ($vmwareInstalled) {
    Write-Host "Vérification de la dernière version disponible..."

    # Simulation d'une requête pour obtenir la dernière version
    $latestVersion = "17.0.3"  # À mettre à jour si nécessaire

    if ($installedVersion -lt $latestVersion) {
        Write-Host "Votre version ($installedVersion) est obsolète !" -ForegroundColor
Yellow
        Write-Host "La dernière version disponible est : $latestVersion"
        Write-Host "Téléchargez la mise à jour ici : $latestVersionURL"
    } else {
        Write-Host "VMware Workstation est à jour." -ForegroundColor Green
```

```
    }
}
```

2.4. Utilisateurs

- **Objectif** : Audit des utilisateurs
- **Instruction** : Réalisez un script qui affichera la liste des utilisateurs puis demandera de choisir entre les utilisateurs actifs ou inactifs.
  Dans un second temps il affichera la liste des utilisateurs correspondant au critère choisi uniquement.
  Afficher la liste des comptes dont le mot de passe n'expire jamais et les afficher sous forme de warning. + Afficher les comtpes ne s'etant pas connectés depuis 1 mois.
- **Bonus** : Le script doit fonctionner aussi bien avec que sans AD.
  Trouver les ordinateurs qui n'ont pas été utilisé depuis 6 mois dans l'AD.
- **Indication** : le warning en PowerShell est un flux particulier

```
# Vérification de la présence d'Active Directory
$adModule = Get-Module -ListAvailable -Name ActiveDirectory
$useAD = $adModule -ne $null

if ($useAD) {
    Import-Module ActiveDirectory
    Write-Host "Module Active Directory détecté et chargé." -ForegroundColor Green
} else {
    Write-Host "Aucun module Active Directory détecté. Passage en mode local."
-ForegroundColor Yellow
}

# Fonction pour obtenir la liste des utilisateurs en fonction du mode
function Get-Users {
    if ($useAD) {
        Get-ADUser -Filter * -Properties Enabled, PasswordNeverExpires, LastLogonDate
    } else {
        Get-LocalUser | Select-Object Name, Enabled, PasswordExpires
    }
}

# Récupération des utilisateurs
$users = Get-Users

# Sélection actif/inactif
$choice = Read-Host "Voulez-vous voir les utilisateurs Actifs (A) ou Inactifs (I) ?"
if ($choice -match "^[Aa]$") {
    $filteredUsers = $users | Where-Object { $_.Enabled -eq $true }
    Write-Host "Utilisateurs actifs :" -ForegroundColor Green
} elseif ($choice -match "^[Ii]$") {
    $filteredUsers = $users | Where-Object { $_.Enabled -eq $false }
```

```
    Write-Host "Utilisateurs inactifs :" -ForegroundColor Red
} else {
    Write-Host "Choix invalide." -ForegroundColor Yellow
    exit
}

# Affichage des utilisateurs sélectionnés
$filteredUsers | Select-Object Name, Enabled, LastLogonDate | Format-Table -AutoSize

# Comptes dont le mot de passe n'expire jamais
$pwdNeverExpires = $users | Where-Object { $_.PasswordNeverExpires -eq $true }
if ($pwdNeverExpires) {
    Write-Host "Comptes dont le mot de passe n'expire jamais :" -ForegroundColor
Yellow
    $pwdNeverExpires | Select-Object Name, PasswordNeverExpires | Format-Table
-AutoSize
}

# Comptes ne s'étant pas connectés depuis 1 mois
$oldUsers = $users | Where-Object { $_.LastLogonDate -lt (Get-Date).AddMonths(-1) -and
$_.LastLogonDate -ne $null }
if ($oldUsers) {
    Write-Host "Comptes ne s'étant pas connectés depuis plus d'un mois :"
-ForegroundColor Cyan
    $oldUsers | Select-Object Name, LastLogonDate | Format-Table -AutoSize
}

# Recherche des ordinateurs inactifs depuis 6 mois (AD uniquement)
if ($useAD) {
    $oldComputers = Get-ADComputer -Filter * -Properties LastLogonDate | Where-Object
{ $_.LastLogonDate -lt (Get-Date).AddMonths(-6) -and $_.LastLogonDate -ne $null }
    if ($oldComputers) {
        Write-Host "Ordinateurs n'ayant pas été utilisés depuis 6 mois :"
-ForegroundColor Red
        $oldComputers | Select-Object Name, LastLogonDate | Format-Table -AutoSize
    }
}
```

2.5. Certificat

- **Objectif** : Audit Certificat
- **Instruction** : Réalisez un script qui fait la liste des certificats présent sur la machine et remonte les certificats qui vont expirer dan le mois.
  Faites en sortes que le message affichez a l'écran ait un format que l'on pourrait facilement intégrer a un mail.

```
# Script PowerShell pour lister les certificats et identifier ceux qui expirent dans
le mois
```

```
# Récupération des certificats dans le magasin personnel et racine
$certs = Get-ChildItem Cert:\LocalMachine\My, Cert:\LocalMachine\Root | Where-Object {
$_.NotAfter -ne $null }

# Date actuelle et date d'expiration dans un mois
$today = Get-Date
$expirationThreshold = $today.AddMonths(1)

# Liste des certificats expirant dans moins d'un mois
$expiringCerts = $certs | Where-Object { $_.NotAfter -lt $expirationThreshold }

# Format de sortie compatible pour intégration dans un mail
$output = @()
$output += "Liste des certificats sur la machine :"
$output += "-------------------------------------"
foreach ($cert in $certs) {
    $output += "Sujet : $($cert.Subject)"
    $output += "Émetteur : $($cert.Issuer)"
    $output += "Expiration : $($cert.NotAfter)"
    $output += "Thumbprint : $($cert.Thumbprint)"
    $output += "-------------------------------------"
}

if ($expiringCerts) {
    $output += "\nCertificats expirant dans moins d'un mois :"
    $output += "-------------------------------------"
    foreach ($cert in $expiringCerts) {
        $output += "Sujet : $($cert.Subject)"
        $output += "Émetteur : $($cert.Issuer)"
        $output += "Expiration : $($cert.NotAfter)"
        $output += "Thumbprint : $($cert.Thumbprint)"
        $output += "-------------------------------------"
    }
} else {
    $output += "\nAucun certificat n'expire dans le mois."
}

# Affichage des résultats
$output | ForEach-Object { Write-Output $_ }
```

2.6. Systeme de fichier

- **Objectif** : Système de fichiers
- **Instruction** : Faire un script qui liste le contenu d'un répertoire, il affichera en bleu les dossiers et on conservera l'affichage normal pour les fichiers.
  Il affichera en plus les fichiers avec l'attribut archives en vert.
- **Bonus** il affichera en rouge le fichier et le répertoire ayant le poids le plus élevé

```
param (
    [string]$Path = "."
)

# Vérifier si le chemin existe
if (-Not (Test-Path -Path $Path)) {
    Write-Host "Le chemin spécifié n'existe pas."
    exit
}

# Récupérer les éléments du répertoire
$items = Get-ChildItem -Path $Path -Force

# Vérifier s'il y a des fichiers et des dossiers
if ($items.Count -eq 0) {
    Write-Host "Le répertoire est vide."
    exit
}

# Initialisation des variables pour le plus gros fichier et dossier
$largestFile = $null
$largestFolder = $null
$maxFileSize = 0
$maxFolderSize = 0

# Fonction pour obtenir la taille d'un dossier
function Get-FolderSize {
    param ([string]$folderPath)
    $size = (Get-ChildItem -Path $folderPath -Recurse -File | Measure-Object -Property
Length -Sum).Sum
    return $size
}

foreach ($item in $items) {
    if ($item.PSIsContainer) {
        # Afficher les dossiers en bleu
        Write-Host "$($item.FullName)" -ForegroundColor Blue
        # Vérifier la taille du dossier
        $folderSize = Get-FolderSize -folderPath $item.FullName
        if ($folderSize -gt $maxFolderSize) {
            $maxFolderSize = $folderSize
            $largestFolder = $item
        }
    } else {
        # Vérifier si le fichier a l'attribut archive
        if ($item.Attributes -match "Archive") {
            Write-Host "$($item.FullName)" -ForegroundColor Green
        } else {
            Write-Host "$($item.FullName)"
        }
```

```
        # Vérifier la taille du fichier
        if ($item.Length -gt $maxFileSize) {
            $maxFileSize = $item.Length
            $largestFile = $item
        }
    }
}

# Afficher le plus gros fichier en rouge
if ($largestFile) {
    Write-Host "Le fichier le plus volumineux : $($largestFile.FullName) - Taille :
$([math]::Round($largestFile.Length / 1MB, 2)) MB" -ForegroundColor Red
}

# Afficher le plus gros dossier en rouge
if ($largestFolder) {
    Write-Host "Le dossier le plus volumineux : $($largestFolder.FullName) - Taille :
$([math]::Round($maxFolderSize / 1MB, 2)) MB" -ForegroundColor Red
}
```

2.7. Creation de fichier

- **Objectif** : Création de fichiers
- **Instruction** : Faire un script qui crée des fichiers avec différentes extensions dans un répertoire donné. (ex : desktop de l'utilisateur actuel)
  Les extensions seront : .xlsx, .docx, .jpg, .gif, .ps1.
  Le script demandera a l'utilisateur ne nombre souhaité de fichiers par extensions.
- **Bonus** : Il sera possible de passer ses paramètres au lancement du script.
- **Indication** : `$#, $[1-9], $@`

```
# Définition du paramètre optionnel pour le nombre de fichiers
param(
    [int]$NumberOfFiles = 0
)

# Fonction pour créer des fichiers avec une extension donnée
function Create-Files {
    param (
        [string]$Extension,  # Extension du fichier à créer
        [int]$Count,         # Nombre de fichiers à créer
        [string]$Path        # Chemin où créer les fichiers
    )

    # Boucle pour créer le nombre spécifié de fichiers
    for ($i = 1; $i -le $Count; $i++) {
        $fileName = "File_$i$Extension"
```

```
        # Création du fichier sans afficher de sortie
        New-Item -Path $Path -Name $fileName -ItemType File -Force | Out-Null
    }
    # Affichage d'un message de confirmation
    Write-Host "Creation de $Count fichiers avec l'extension $Extension"
}

# Liste des extensions de fichiers à créer
$extensions = @('.xlsx', '.docx', '.jpg', '.gif', '.ps1')

# Obtention du chemin du bureau de l'utilisateur actuel
$desktopPath = "$env:HOMEDRIVE$env:HOMEPATH\Desktop"

# Si le nombre de fichiers n'est pas spécifié en paramètre, demander à l'utilisateur
if ($NumberOfFiles -eq 0) {
    $NumberOfFiles = Read-Host "Entrer le nombre de fichiers souhaités par extensions"
}

# Création des fichiers pour chaque extension
foreach ($ext in $extensions) {
    Create-Files -Extension $ext -Count $NumberOfFiles -Path $desktopPath
}

# Message de fin d'exécution
Write-Host "Fin du script. fichiers créés dans $DesktopPath."
```

2.8. Range ton bureau

- **Objectif** : Déplacement de fichiers
- **Instruction** : Faire un script qui affiche la liste des extensions des fichiers présent sur le bureau de l'utilisateur et sans qu'une extension apparaissent plusieurs fois.
  Le script déplace les fichiers avec différentes extensions dans un répertoire portant le nom de l'extension.
- **Bonus** : Le script proposera une fois terminé de déplacer les répertoires et leurs contenus dans sur un emplacement plus approprié pour le stockage de documents que le bureau.

```
# Obtenir le chemin du bureau de l'utilisateur
$desktopPath = "$env:HOMEDRIVE$env:HOMEPATH\Desktop"

# Récupérer tous les fichiers du bureau
$files = Get-ChildItem -Path $desktopPath -File

# Obtenir la liste unique des extensions
$uniqueExtensions = $files | Select-Object -ExpandProperty Extension -Unique

# Afficher la liste des extensions uniques
Write-Host "Extensions uniques trouvées sur le bureau :"
$uniqueExtensions | ForEach-Object { Write-Host $_ }
```

```
# Créer des dossiers pour chaque extension et déplacer les fichiers
foreach ($ext in $uniqueExtensions) {
    if ($ext) {
        $folderName = $ext.TrimStart(".")
        $folderPath = Join-Path -Path $desktopPath -ChildPath $folderName

        # Créer le dossier s'il n'existe pas
        if (-not (Test-Path $folderPath)) {
            New-Item -Path $folderPath -ItemType Directory | Out-Null
        }

        # Déplacer les fichiers dans le dossier correspondant
        $files | Where-Object { $_.Extension -eq $ext } | Move-Item -Destination
$folderPath
    }
}

Write-Host "Les fichiers ont été organisés dans des dossiers par extension."

# Bonus : Proposer de déplacer les dossiers vers un emplacement plus approprié
$response = Read-Host "Voulez-vous déplacer les dossiers vers un emplacement plus
approprié ? (O/N)"
if ($response -eq "O") {
    $destinationPath = Read-Host "Entrez le chemin de destination"
    if (Test-Path $destinationPath) {
        Get-ChildItem -Path $desktopPath -Directory | Move-Item -Destination
$destinationPath
        Write-Host "Les dossiers ont été déplacés vers $destinationPath"
    } else {
        Write-Host "Le chemin de destination n'existe pas. Opération annulée."
    }
}
```

2.9. copie de fichier

- **Objectif** : Copie de fichiers
- **Instruction** : Faire un script qui liste le contenu d'un répertoire renseigné par l'utilisateur.
  Puis copie le contenu de ce répertoire dans un répertoire choisi par l'utilisateur.
  Il permettra de renommer chaque fichier selon un pattern choisi.
  Exemple : renommer des photos en les préfixant d'une indication de lieu ou d'événement suivi de la date puis du nom du fichier.
- **Bonus** : lorsque le script a terminé la copie et vérifié qu'il a bien traité tout les fichiers, il proposera de supprimer les fichiers d'origines.

```

```

2.10. Audit Machine

- **Objectif** : Audit de machine sur une plage IP
- **Instruction** : Faire un script en PowerShell uniquement, qui liste les machines présente sur une plage ip.
  Il devra afficher le Nom de la machine, son adresse IP, le nom de l'utilisateur actuellement connecté, pour toute machine allumée.
- **Bonus** : Est il possible d'optimiser son temps de traitement ?
- **Indication** : !!! Ne tester cela que sur votre propre réseau virtuel. !!!

```
# Fonction pour obtenir un chemin de répertoire valide
function Get-ValidPath {
    param([string]$prompt)
    do {
        $path = Read-Host $prompt
        if (-not (Test-Path $path -PathType Container)) {
            Write-Host "Chemin invalide. Veuillez réessayer."
        }
    } while (-not (Test-Path $path -PathType Container))
    return $path
}

# Demander le répertoire source
$sourceDir = Get-ValidPath "Entrez le chemin du répertoire source"

# Lister le contenu du répertoire source
Write-Host "Contenu du répertoire source :"
Get-ChildItem $sourceDir | Format-Table Name, LastWriteTime, Length

# Demander le répertoire de destination
$destDir = Get-ValidPath "Entrez le chemin du répertoire de destination"

# Demander le préfixe pour le renommage
$prefix = Read-Host "Entrez le préfixe pour le renommage (ex: Vacances_Ete_)"

# Copier et renommer les fichiers
$copiedFiles = @()
Get-ChildItem $sourceDir | ForEach-Object {
    $newName = "{0}{1:yyyyMMdd}_{2}" -f $prefix, $_.LastWriteTime, $_.Name
    $destPath = Join-Path $destDir $newName
    Copy-Item $_.FullName -Destination $destPath
    $copiedFiles += $destPath
}

# Vérifier que tous les fichiers ont été copiés
$allCopied = $true
foreach ($file in $copiedFiles) {
    if (-not (Test-Path $file)) {
```

```
        $allCopied = $false
        Write-Host "Erreur : Le fichier $file n'a pas été copié correctement."
    }
}

# Si tous les fichiers ont été copiés, proposer de supprimer les originaux
if ($allCopied) {
    $deleteOriginals = Read-Host "Tous les fichiers ont été copiés. Voulez-vous
supprimer les originaux ? (O/N)"
    if ($deleteOriginals -eq "O") {
        Get-ChildItem $sourceDir | Remove-Item -Force
        Write-Host "Les fichiers originaux ont été supprimés."
    }
}

Write-Host "Opération terminée."
```

2.11. Gestion Installation imprimante

- **Objectif** : Installation d'imprimante
- **Instruction** :
  1. Réalisez un script qui permet de faire l'installation d'une imprimante.
  2. Modifiez le script pour qu'il demande la saisie d'une adresse IP. Le script contiendra la liste des imprimantes et de leurs IP, ainsi que le chemin pour atteindre le driver a installer et procédera a l'installation de cette imprimante en particulier+
- **Bonus** : Faites en sorte que si une adresse réseau et non IP est fournit au script alors toutes les imprimantes de ce réseau seront installées sur le poste. Externalisez les informations relatives aux imprimantes dans un fichier CSV.

```
<#
Le CSV aurait ce format :
IP,Name,DriverPath
192.168.1.100,Imprimante1,C:\Chemin\vers\pilote1.inf
192.168.1.101,Imprimante2,C:\Chemin\vers\pilote2.inf
#>

# Fonction pour installer une imprimante
function Install-Printer {
    param(
        [string]$IP,
        [string]$Name,
        [string]$DriverPath
    )

    # Ajouter le port d'imprimante
    Add-PrinterPort -Name "IP_$IP" -PrinterHostAddress $IP
```

```
    # Installer le pilote
    # utilisation de pnputil car powershell ne sais pas ajouter un driver au magasin
de pilotes.
    # Add-PrinterDriver necessite que le pilote soit deja present dans le magasin.
    pnputil.exe /add-driver $DriverPath /install

    # Ajouter l'imprimante
    Add-Printer -DriverName (Get-PrinterDriver | Where-Object {$_.InfPath -eq
$DriverPath}).Name -Name $Name -PortName "IP_$IP"

    Write-Host "Imprimante $Name installée avec succès."
}

# Charger les informations des imprimantes depuis un fichier CSV
$printers = Import-Csv -Path "C:\Chemin\vers\imprimantes.csv"

# Demander la saisie de l'adresse IP ou réseau
$input = Read-Host "Entrez l'adresse IP de l'imprimante ou l'adresse réseau"

if ($input -match "^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$") {
    # Si c'est une adresse IP
    $printer = $printers | Where-Object {$_.IP -eq $input}
    if ($printer) {
        Install-Printer -IP $printer.IP -Name $printer.Name -DriverPath
$printer.DriverPath
    } else {
        Write-Host "Aucune imprimante trouvée avec cette adresse IP."
    }
} else {
    # Si c'est une adresse réseau
    $networkPrinters = $printers | Where-Object {$_.IP -like "$input*"}
    foreach ($printer in $networkPrinters) {
        Install-Printer -IP $printer.IP -Name $printer.Name -DriverPath
$printer.DriverPath
    }
}
```

2.12. Pendu

- **Objectif** : Jeu du pendu
- **Instruction** : Créez un jeu du pendu en PowerShell avec les caractéristiques suivantes :
  Le jeu doit choisir un mot aléatoire dans une liste prédéfinie de mots français.
  Le joueur a 6 essais pour deviner le mot.
  À chaque tour, affichez le mot partiellement deviné et les lettres déjà proposées.
  Le jeu doit gérer les entrées invalides (chiffres, plusieurs lettres, etc.).
  À la fin du jeu, affichez un message de victoire ou de défaite.
- **Bonus** : Avec de l'asciiart affichez l'état du jeu a chaque itération.

2.13. Motus

- **Objectif** : Jeu MOTUS
- **Instruction** : Créer une version simplifiée du jeu Motus avec les règles suivantes :
  Le jeu choisit un mot français aléatoire (6 lettres)
  La première lettre du mot est toujours révélée
  Le joueur dispose de 6 tentatives pour trouver le mot
  À chaque tentative :
  Les lettres correctement placées s'affichent en vert
  Les lettres présentes mais mal placées s'affichent en jaune
  Affichage des lettres déjà proposées
  Gestion des entrées invalides

2.14. QCM

- **Objectif** : Création d'un Quizz
- **Instruction** : Creer un script qui permet de jouer a un quizz.
  Il affichera la question.
  Il affichera une ou plusieurs proposition de réponses.
  Il affichera la réponse, si la réponse est correcte l'affichage sera en vert.
  Il comptera les points.
  A la fin de son exécution il affichera le nombre de questions répondues avec le nombre de bonne réponses.
- **Bonus** : Les questions auront 3 niveaux de difficultés.
  La banque de questions sera externalisé dans un fichier CSV.
  Faites en sortes qu'il permette a X joueurs de s'affronter.
  Faites en sortes qu'il affiche au debut le top 3 des meilleurs scores avec leurs pseudos et la date.
  Ce top sera maintenu a jour durablement et en cas d'égalité le score le plus récent prendra la place du plus ancien.

```
<#
.SYNOPSIS
Script Quizz
.DESCRIPTION
Auteur : Butch
Date creation : 18/10/24
Dernier modificateur : Butch
Date derniere modification :

legende :
# : ligne de commentaire
```

```
## : ligne de debug
.EXAMPLE
exemple1 : .\LeNomDuScript
.NOTES
Ceci est une note
#>
# Charger les questions depuis le fichier CSV
$questions = Import-Csv -Path "questions.csv"

function Show-Menu {
    Clear-Host
    Write-Host "=== Quiz PowerShell ==="
    Write-Host "1. Facile"
    Write-Host "2. Moyen"
    Write-Host "3. Difficile"
    Write-Host "Q. Quitter"
}

function Get-RandomQuestion($difficulty) {
    $filteredQuestions = $questions | Where-Object { $_.Difficulte -eq $difficulty }
    return $filteredQuestions | Get-Random
}

function Start-Quiz {
    $participants = @{}
    $nbParticipants = Read-Host "Combien de participants ?"
    for ($i = 1; $i -le $nbParticipants; $i++) {
        $name = Read-Host "Nom du participant $i"
        $participants[$name] = 0
    }

    Show-Menu
    $difficulty = switch (Read-Host "Choisissez la difficulté") {
        1 { "Facile" }
        2 { "Moyen" }
        3 { "Difficile" }
        Default { "Facile" }
    }

    $questionCount = 0
    $askedQuestions = @()

    do {
        $question = Get-RandomQuestion $difficulty
        while ($askedQuestions -contains $question.Question) {
            $question = Get-RandomQuestion $difficulty
        }
        $askedQuestions += $question.Question

        Write-Host "`n$($question.Question)"
        Write-Host "1. $($question.Reponse1)"
```

```
        Write-Host "2. $($question.Reponse2)"
        Write-Host "3. $($question.Reponse3)"

        foreach ($participant in $participants.Keys) {
            $answer = Read-Host "$participant, votre réponse (1-3)"
            if ($answer -eq $question.ReponseCorrecte) {
                $participants[$participant]++
                Write-Host "Correct !"
            } else {
                Write-Host "Incorrect. La bonne réponse était :
$($question."Reponse$($question.ReponseCorrecte)")"
            }
        }

        $questionCount++
        Write-Host "`nScores actuels :"
        $participants.GetEnumerator() | ForEach-Object {
            Write-Host "$($_.Key): $($_.Value)"
        }

        if ($questionCount -lt 20) {
            $continue = Read-Host "Continuer ? (O/N)"
            if ($continue -ne "O") { break }
        }
    } while ($questionCount -lt 20)

    Write-Host "`nQuiz terminé ! Scores finaux :"
    $winner = $participants.GetEnumerator() | Sort-Object Value -Descending | Select-
Object -First 1
    $participants.GetEnumerator() | ForEach-Object {
        Write-Host "$($_.Key): $($_.Value)"
    }
    Write-Host "`nLe vainqueur est : $($winner.Key) avec $($winner.Value) points !"
}

Start-Quiz
```

2.15. Mail to Prénom Nom

- **Objectif** : manipuler des chaines de caractères
- **Instruction** : Votre RH vous fournit une liste d'adresses emails (fichier mails.txt), vous avez besoin d'en extraire uniquement les prénoms et les noms.
  1. Le fichier source fournit mails.txt, devra garder son intégrité.
  2. Réalisez un script qui charge le contenu du fichier, puis suite au traitement, les prénoms et noms devront être stocké dans un fichier PrenomsNoms.txt.
- **Indication** : split, replace, match
- **Bonus** : le traitement doit se faire en une seule étape.

```
## recuperation des informations du fichier texte .\ indique le repertoire ou se
trouve le script
## le script et le fichier source doivent etre dans le meme repertoire pour que cela
fonctionne
$mails = get-content -Path ".\mails.txt"
##reinitialisation du fichier destination
clear-content .\stagiaires.txt
#write-host "$mail"
foreach ($email in $mails) {
    ## Extraire prénom et nom avec une seule regex
    ##^ : début de la ligne, ([a-z]+) : capture le prénom (lettres minuscules), \. :
le point séparateur, ([a-z]+) : capture le nom (lettres minuscules),
    ## \d+ : chiffres (promotion, année, etc.) ignorés, @ : l□arobase, marque la fin
de la partie à extraire
    ## dans le cadre de l'utilisattion de -match PowerShell remplit automatiquement la
variable spéciale $matches avec le résultat du dernier match réussi
    if ($email -match '^([a-z]+)\.([a-z]+)\d+@') {
        $prenom = $matches[1]
        $nom = $matches[2]
        ## autre option : write-ouput ici plutot que write-host car je ne veux pas
faire d'affichage ecran mais du retraitement
        ## Write-output "$prenom $nom" | out-file .\stagiaires.txt -append
        "$prenom $nom" | out-file .\PrenomsNoms.txt -append
    }
}
```

2.16. Random Stagiaires

- **Objectif** : Justice pour les stagiaires
- **Instruction** : Vous êtes un formateur vous avez besoin de pouvoir interroger équitablement vos stagiaires. Vous êtes donc en quête d'un outil permettant de répondre a ce besoin.
  1. Trouver la commande qui permet de faire un tirage aléatoire. Quelles sont ses options ?
  2. Créer un script pour un tirage aléatoire Vous décidez de donner un numéro a chaque stagiaire. Créer un script qui tire un chiffre aléatoirement dans la limite du nombre de stagiaire. Il affichera la phrase "le numéro du désigné volontaire d'office est : < numéro >".
  3. Vos stagiaires ne retiennent pas leurs numéros. Vous les trouvez susceptible mais décidez donc qu'il serait plus judicieux de pouvoir afficher leur Prénom et leur nom plutôt que des numéros, sous la forme : **le volontaire d'office est : < numéro › <Prénom Nom**.
  4. Vous n'êtes pas satisfait que les stagiaires puissent être tiré plusieurs fois et pas d'autres. Trouvez un moyen de corriger cette problématique.
  5. Vous n'êtes pas satisfait que le script se ferme une fois terminé Trouvez une solution pour qu'il demande a l'utilisateur du script si il souhaite procéder a un nouveau tirage.
  6. Vous vous rendez compte que lorsque vous relancer le script le lendemain matin il repart de zéro. Faites en sortes que ce ne soit pas le cas.

- **Indication** : Les tableaux possèdent des index. A l'inverse d'une variable, le contenu d'un fichier ne s'efface pas.

```
<#
.SYNOPSIS
Script de tirage aléatoire.

.DESCRIPTION
Auteur : Butch
Date création : 12/10/22
Dernière modification : 29/01/25
Ce script permet de tirer aléatoirement un numéro ou un élément dans une liste, tout
en supprimant les éléments déjà tirés.

.EXAMPLE
.\LeNomDuScript

.NOTES
- Le script utilise un fichier source au format texte contenant une liste d'éléments
(ex. noms).
- Il génère un fichier temporaire pour suivre les éléments restants.
- Des améliorations possibles incluent l'ajout d'une interface graphique.
#>

# Définition des fichiers source et temporaire
$FicSource = "stagiaires.txt"
$FicSourcepath = ".\$FicSource"
$FicName = "randomtemp.txt"
$FicTempPath = ".\$FicName"

# Vérification ou création du fichier temporaire
if (-not (Test-Path $FicTempPath)) {
    New-Item -ItemType File -Path $FicTempPath | Out-Null
    Write-Host "Fichier temporaire créé : $FicTempPath" -ForegroundColor Green
} else {
    Write-Host "Fichier temporaire détecté : $FicTempPath" -ForegroundColor Yellow
}

# Fonction pour charger une liste depuis un fichier
function Charger-Liste {
    param (
        [string]$path
    )
    if (Test-Path $path) {
        return [System.Collections.ArrayList](Get-Content -Path $path)
    } else {
        Write-Host "Erreur : Le fichier '$path' est introuvable." -ForegroundColor Red
        exit 1
    }
}
```

```
# Chargement initial de la liste
$comparaison = Get-Content $FicTempPath
if ($comparaison) {
    $reprendre = Read-Host "Voulez-vous reprendre le tirage en cours ? (O/N) [Défaut :
O]"
    switch ($reprendre.ToUpper()) {
        "N" {
            $listenum = Charger-Liste -path $FicSourcepath
            Write-Host "Liste réinitialisée depuis le fichier source."
-ForegroundColor Cyan
        }
        Default {
            $listenum = Charger-Liste -path $FicTempPath
            Write-Host "Reprise du tirage en cours." -ForegroundColor Cyan
        }
    }
} else {
    $listenum = Charger-Liste -path $FicSourcepath
    Write-Host "Liste chargée depuis le fichier source." -ForegroundColor Cyan
}

# Boucle principale pour les tirages successifs
do {
    # Sauvegarde initiale de la liste dans le fichier temporaire
    $listenum | Out-File -$FicTempPath $FicTempPath -Encoding UTF8

    # Tirage aléatoire jusqu'à épuisement ou arrêt par l'utilisateur
    do {
        if ($listenum.Count -eq 0) {
            Write-Host "Tous les éléments ont été tirés ! Fin du tirage."
-ForegroundColor Green
            break
        }

        # Tirage d'un élément aléatoire
        $valeur = $listenum | Get-Random

        # Affichage du résultat et suppression de l'élément tiré
        Write-Host "`nAnd the winner is: $valeur" -ForegroundColor Magenta
        $listenum.Remove($valeur)

        # Mise à jour du fichier temporaire après suppression
        $listenum | Out-File -$FicTempPath $FicTempPath -Encoding UTF8

        # Demande à l'utilisateur s'il souhaite continuer ou arrêter le tirage en
cours
        $saisie = Read-Host "Appuyez sur Entrée pour continuer ou tapez 'Q' pour
quitter"
    } until ($saisie.ToUpper() -eq "Q")
```

```
    # Demande si un nouveau tirage doit être lancé ou si le script doit se terminer
    Write-Host "`nVoulez-vous effectuer un nouveau tirage ? (Entrée pour continuer, Q
pour quitter)" -ForegroundColor DarkYellow
    $stopencore = Read-Host

} until ($stopencore.ToUpper() -eq "Q")

Write-Host "`nMerci d'avoir utilisé le script ! À bientôt." -ForegroundColor Green
```