PowerShell

Membres du groupe :

- Josselin (GMSI) ;
- Sylvain (GMSI) ;
- Florent (GMSI) ;
- Mathis (DI).

Question 1

Donnez 7 caractéristiques d'une variable en programmation (pas uniquement PowerShell, mais programmation en général).

1. Nom

```
 C'est le nom de la variable, il doit être unique dans sa portée, ne pas commencer par un chiffre, ne pas contenir d'espace et ne pas être un mot réservé du langage. En fonction du langage, il peut être sensible à la casse.
```

2. Type

```
 C'est le type de la variable, il peut être de type primitif (entier, flottant, booléen, caractère, chaîne de caractères) ou de type complexe (tableau, liste, dictionnaire, objet, etc.). En fonction du langage, il peut être implicite ou explicite.
```

3. Valeur

```
 C'est la valeur dans la case mémoire de la variable, elle peut être modifiée directement ou indirectement en fonction du langage.
```

4. Portée

```
 C'est la zone du programme dans laquelle la variable est accessible, elle peut être globale (accessible partout), locale (accessible uniquement dans une fonction, une classe ou une boucle). Dans le cas d'un langage orienté objet, la portée peut être privée (accessible uniquement dans la classe), protégée (accessible dans la classe et ses enfants) ou publique (accessible partout).
```

5. Durée de vie

> C'est la durée pendant laquelle la variable est accessible, elle peut être temporaire (accessible uniquement dans une fonction, une classe ou une boucle) ou permanente (accessible partout). Dans certains langages, la durée de vie peut être définie par l'utilisateur, tandis que dans d'autres, elle est définie par le langage, et celui-ci se charge de la gestion de la mémoire. En java, le garbage collector se charge de la gestion de la mémoire en supprimant les variables inutilisées.

6. Mutabilité

> C'est la possibilité de modifier la valeur de la variable, elle peut être mutable (modifiable) ou immuable (non modifiable). En fonction du langage, la mutabilité peut être définie par l'utilisateur ou par le langage. Une variable immuable est appelée constante et ne peut être modifiée après son initialisation.

7. Initialisation

> C'est la valeur initiale de la variable, elle peut être définie par l'utilisateur ou par le langage. En fonction du langage, la variable peut être initialisée à sa déclaration ou à un autre moment. Par exemple, en java, l'attribut d'une classe peut être initialisé dans le constructeur de la classe, mais déclaré en dehors.

Question 2

Goal : Afficher des variables

Script question 2

```
$nomUtilisateur = Read-Host -Prompt "Veuillez saisir votre prenom"

$ageUtilisateur = Read-Host -Prompt "Veuillez saisir votre age"

Write-Host "Bonjour $nomUtilisateur, vous avez $ageUtilisateur ans."
```

Question 3

Goal : Jeu du nombre secret

Script question 3

```powershell
$nombreSecret = Get-Random -Minimum 0 -Maximum 101

$nombreEssais = 0

function Get-Essai {
    return Read-Host "Devinez le nombre (entre 0 et 100)"
}

do {
    $essaiUtilisateur = Get-Essai
    $nombreEssais++

    if ($essaiUtilisateur -gt $nombreSecret) {
        Write-Host "Trop haut ! Mon nombre est plus bas." -ForegroundColor Red
    }
    elseif ($essaiUtilisateur -lt $nombreSecret) {
        Write-Host "Trop bas ! Mon nombre est superieur." -ForegroundColor Blue
    }
    else {
        Write-Host "Bravo, vous avez trouve en $nombreEssais essais !" -
ForegroundColor Green
    }
} while ($essaiUtilisateur -ne $nombreSecret)
```

Question 4

Goal : Lancer notepad si ce n'est pas déjà fait

Script question 4

```powershell
$notepadProcess = Get-Process -Name notepad -ErrorAction SilentlyContinue

if (-not $notepadProcess) {
    Start-Process notepad
    Write-Host "Notepad a ete lance."
} else {
    Write-Host "Notepad est deja en cours d'execution."
}
```

Question 5

Goal : Créer un utilisateur local administrateur et un utilisateur local non administrateur (puis les supprimer)

Script question 5

```powershell
function Test-AdminPrivileges {
    $currentUser = New-Object
```

```
Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurre
nt())
    $adminRole = [Security.Principal.WindowsBuiltInRole]::Administrator
    return $currentUser.IsInRole($adminRole)
}

if (-not (Test-AdminPrivileges)) {
    Write-Host "Le script doit etre execute en tant qu'administrateur. Veuillez
relancer le script en tant qu'administrateur."
    $null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
    Exit
}

function Write-LocaleUserAccount {
    param([string]$moment = "actuellement")
    $comptesLocaux = Get-WmiObject Win32_UserAccount | Where-Object {
$_.LocalAccount -eq $true }

    Write-Host "Comptes utilisateurs locaux $moment :"
    foreach ($compte in $comptesLocaux) {
        Write-Host $compte.Name
    }
}

function Get-SecurePassword {
    $securePassword = Read-Host -Prompt "Entrez le mot de passe" -AsSecureString
    return $securePassword
}

function Compare-Password {
    param([System.Security.SecureString]$password,
[System.Security.SecureString]$verifyPassword)
    return
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto([System.Runtime.InteropS
ervices.Marshal]::SecureStringToBSTR($password)) -eq
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto([System.Runtime.InteropS
ervices.Marshal]::SecureStringToBSTR($verifyPassword))
}

function New-Account {
    param([bool]$isAdmin = $false)
    $login = Read-Host -Prompt "Entrez le login du nouveau compte"
    $password = Get-SecurePassword
    $verifyPassword = $null
    do {
        $verifyPassword = Get-SecurePassword
        if (-not (Compare-Password -password $password -verifyPassword
$verifyPassword)) {
            Write-Host "Les mots de passe ne correspondent pas. Veuillez
reessayer."
        }
    } while (-not (Compare-Password -password $password -verifyPassword
$verifyPassword))
```

```
    New-LocalUser -Name $login -Password $password -PasswordNeverExpires
    if ($isAdmin) {
        Add-LocalGroupMember -Group "Administrateurs" -Member $login
    }
    Write-Host "Nouveau compte $('non ' * (-not $isAdmin))administrateur '$login'
cree."

    return $login
}

function Remove-Account {
    param([string]$login)
    Remove-LocalUser -Name $login
    Write-Host "Compte '$login' supprime."
}

Write-LocaleUserAccount
$loginAdmin = New-Account -isAdmin $true
Write-LocaleUserAccount "apres creation compte admin"
$loginNonAdmin = New-Account
Write-LocaleUserAccount "apres creation compte non admin"
Remove-Account -login $loginAdmin
Write-LocaleUserAccount "apres suppression compte admin"
Remove-Account -login $loginNonAdmin
Write-LocaleUserAccount "apres suppression compte non admin"
```

Question 6

Goal : Trier les fichiers d'un dossier dans des dossiers par extension de fichier

Script question 6

```
function Get-Folder {
    param([String]$relativePath = "")
    return (Join-Path -Path $PSScriptRoot -ChildPath $relativePath)
}

$folder = Get-Folder -relativePath "../files-to-sort"
$destination = Get-Folder -relativePath "../sorted-files"

$files = Get-ChildItem -Path $folder -File

foreach ($file in $files) {
    $fileExtension = $file.Extension
    if (-not (Test-Path -Path "$destination\$fileExtension")) {
        New-Item -Path "$destination\$fileExtension" -ItemType Directory
    }
    Copy-Item -Path $file.FullName -Destination "$destination\$fileExtension"
}

Write-Host "Copie terminee."
```

Question 7

Goal : Créer des utilisateurs de façon interactive

Script question 7

```
function Test-AdminPrivileges {
    $currentUser = New-Object
Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurre
nt())
    $adminRole = [Security.Principal.WindowsBuiltInRole]::Administrator
    return $currentUser.IsInRole($adminRole)
}

if (-not (Test-AdminPrivileges)) {
    Write-Host "Le script doit etre execute en tant qu'administrateur. Veuillez
relancer le script en tant qu'administrateur."
    $null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
    Exit
}

function Write-LocaleUserAccount {
    param([String]$moment = "actuellement")
    $comptesLocaux = Get-WmiObject Win32_UserAccount | Where-Object {
$_.LocalAccount -eq $true }

    Write-Host "Comptes utilisateurs locaux $($moment) :"
    foreach ($compte in $comptesLocaux) {
        Write-Host $compte.Name
    }
}

function Get-SecurePassword {
    param([String]$compte)
    $securePassword = Read-Host -Prompt "Entrez le mot de passe pour '$($compte)'"
-AsSecureString
    return $securePassword
}

function Compare-Password {
    param([System.Security.SecureString]$password,
[System.Security.SecureString]$verifyPassword)
    return
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto([System.Runtime.InteropS
ervices.Marshal]::SecureStringToBSTR($password)) -eq
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto([System.Runtime.InteropS
ervices.Marshal]::SecureStringToBSTR($verifyPassword))
}

function New-Account {
    param([bool]$isAdmin = $false)
```

```
    $login = Read-Host -Prompt "Entrez le login du nouveau compte $('non ' * (-not
$isAdmin))administrateur"
    $password = Get-SecurePassword $login
    $verifyPassword = $null
    do {
        $verifyPassword = Get-SecurePassword $login
        if (-not (Compare-Password -password $password -verifyPassword
$verifyPassword)) {
            Write-Host "Les mots de passe ne correspondent pas. Veuillez
reessayer."
        }
    } while (-not (Compare-Password -password $password -verifyPassword
$verifyPassword))

    New-LocalUser $login -password $password -PasswordNeverExpires
    if ($isAdmin) {
        Add-LocalGroupMember "Administrateurs" $login
    }
    Write-Host "Nouveau compte $('non ' * (-not $isAdmin))administrateur
'$($login)' cree."

    return $login
}

function Remove-Account {
    param([String]$login)
    Remove-LocalUser -Name $login.Split(" ")[0]
    Write-Host "Compte '$($login.Split(" ")[0])' supprime."
}

function Show-Menu {
    Write-Host "Bienvenue dans le gestionnaire de comptes utilisateurs."
    Write-Host "Veuillez choisir une action :"
    Write-Host "1. Creer un compte administrateur"
    Write-Host "2. Creer un compte utilisateur"
    Write-Host "3. Afficher les comptes utilisateurs locaux"
    Write-Host "4. Supprimer un compte utilisateur"
    Write-Host "5. Afficher ce menu d'aide"
    Write-Host "6. Quitter"
}

do {
    Show-Menu
    $userChoice = Read-Host -Prompt "Entrez votre choix"
    switch ($userChoice) {
        "1" {
            $loginAdmin = [String] (New-Account -isAdmin $true)
            Write-Host -login $loginAdmin
            Break
        }
        "2" {
            $loginNonAdmin = [String] (New-Account)
            Write-Host -login $loginNonAdmin
            Break
```

```
            }
            "3" {
                Write-LocaleUserAccount
                Break
            }
            "4" {
                $login = Read-Host -Prompt "Entrez le login du compte a supprimer"
                Remove-Account $login
                Break
            }
            "5" {
                Show-Menu
                Break
            }
            "6" {
                Write-Host "Au revoir."
                Exit
            }
            Default {
                Write-Host "Choix invalide. Veuillez reessayer."
            }
        }
    } while ($true)
```