TP3 : PowerShell Boucles et fonctions

Sommaire

Table des matières

1. While 2
2. Do While 3
3. Do Until 4
4. For 5
5. For Each 5
6. If, Else If, Else 6
7. Switch 8
8. Fonctions 9
9. Fonctions avec Arguments 10
10. Fonctions avec Paramètres 11

1. While

Cette boucle continue à s'exécuter tant que la condition est vraie. Voici un exemple de cette boucle avec un script qui rajoute 1 à chaque passage dans la boucle jusqu'à atteindre 5. Une fois que 5 est atteint, la boucle ce ferme et ce termine avec un message indiquant que la boucle est terminée :

```powershell
# Initialisation de la variable compteur
$compteur = 0

# La boucle while s'exécute tant que le compteur est inférieur à 5
while ($compteur -lt 5) {
    Write-Output "Le compteur est à : $compteur"
    # Incrémentation : on ajoute 1 à la valeur actuelle de compteur
    $compteur = $compteur + 1
}

Write-Output "La condition n'est plus satisfaite, la boucle est terminée"

```

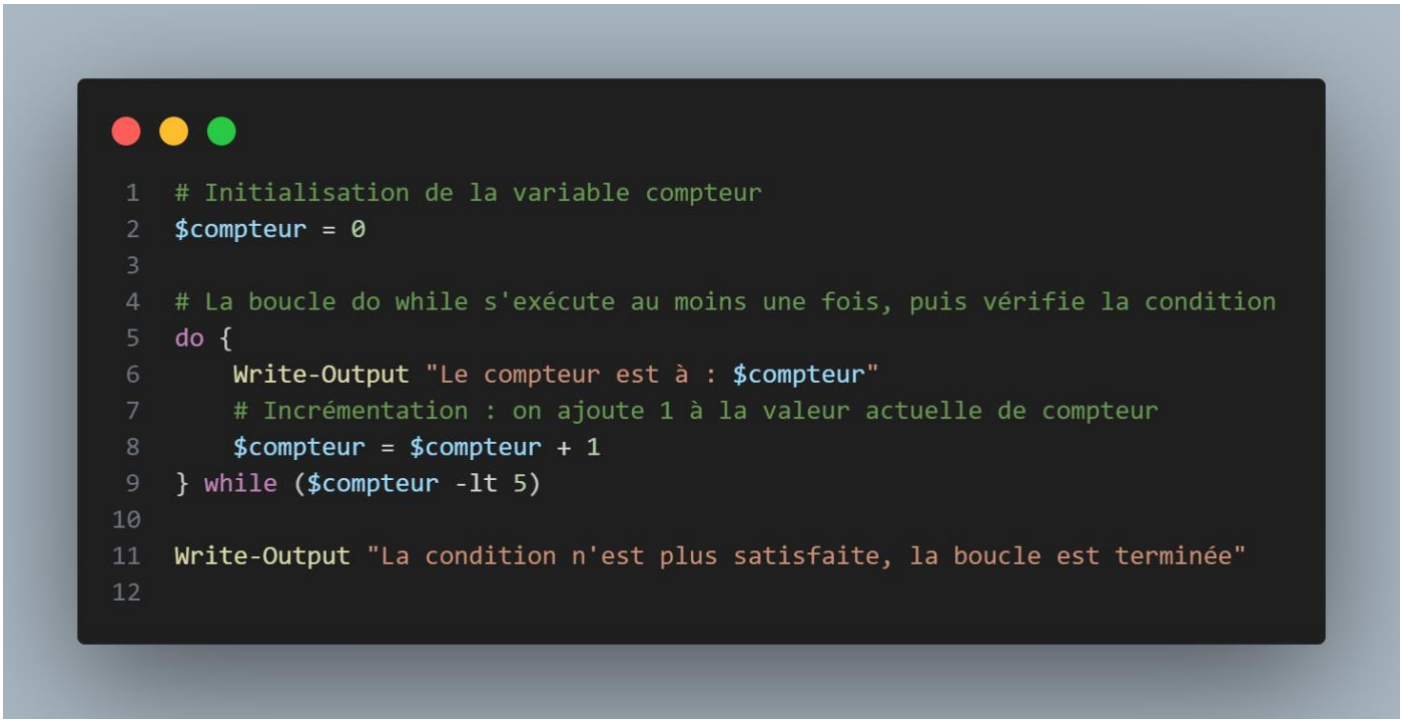
Dans ce script, on va ajouter +1 à la variable $compteur à chaque passage dans la boucle. La condition while ($compteur -lt 5) permet d'arrêter la boucle une fois que le nombre 5 est présent dans la variable $compteur.

```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Le compteur est à : 0
Le compteur est à : 1
Le compteur est à : 2
Le compteur est à : 3
Le compteur est à : 4
La condition n'est plus satisfaite, la boucle est terminée
PS C:\Users\rdubois>
```

On peut remarquer que la boucle se termine une fois qu'on arrive au nombre 5.

2. Do While

Cette boucle est assez similaire que la boucle while, mais contrairement à celle-ci, elle ne vérifie la condition qu'une fois que la commande a été exécuté. Ce qui signifie que la boucle commence par exécuter le bloc de code sans tester la condition au préalable. À la fin de chaque boucle, la condition ($compteur -lt 5) est vérifiée. Exemple avec un cas similaire du premier :

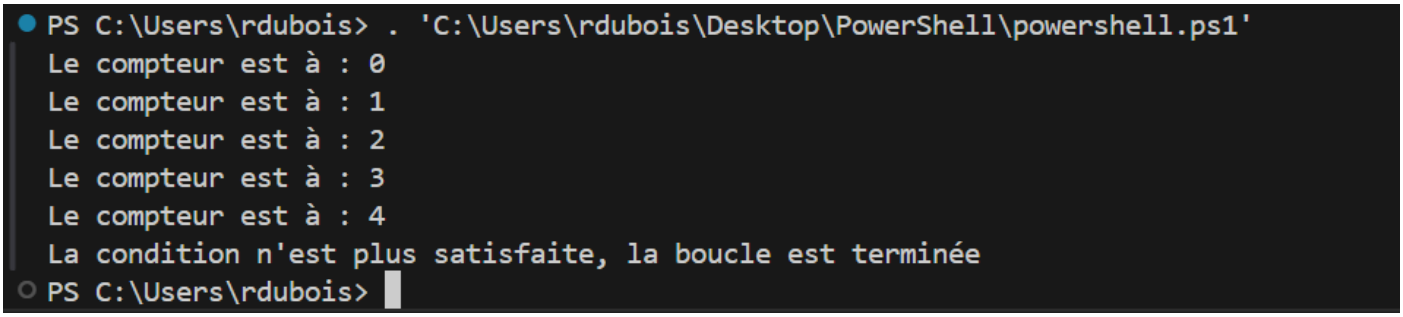


```powershell
# Initialisation de la variable compteur
$compteur = 0

# La boucle do while s'exécute au moins une fois, puis vérifie la condition
do {
    Write-Output "Le compteur est à : $compteur"
    # Incrémentation : on ajoute 1 à la valeur actuelle de compteur
    $compteur = $compteur + 1
} while ($compteur -lt 5)

Write-Output "La condition n'est plus satisfaite, la boucle est terminée"
```

On remarque bien que la condition while est placé à la fin du code et non au début comme précédemment avec le do au début du script. Ce qui nous donne le même résultat en sorti :



```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Le compteur est à : 0
Le compteur est à : 1
Le compteur est à : 2
Le compteur est à : 3
Le compteur est à : 4
La condition n'est plus satisfaite, la boucle est terminée
PS C:\Users\rdubois>
```

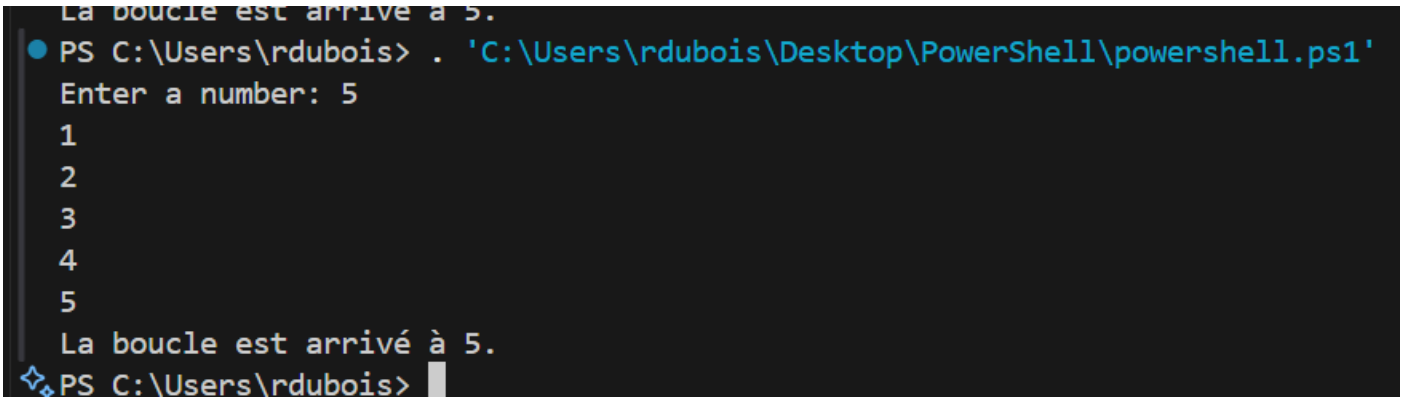
3. Do Until

La boucle s'exécute d'abord, puis la condition est testée à la fin. La boucle continue tant que la condition n'est pas arrivée au résultat. Ici, la boucle se répète jusqu'à ce que le nombre défini par l'utilisateur (variable $j) sois obtenu en partant de 0 et en rajoutant 1 à chaque passage dans la boucle à la variable $i. Les deux variables sont ensuite comparées dans la condition « until ($i -eq $j) ». Exemple :

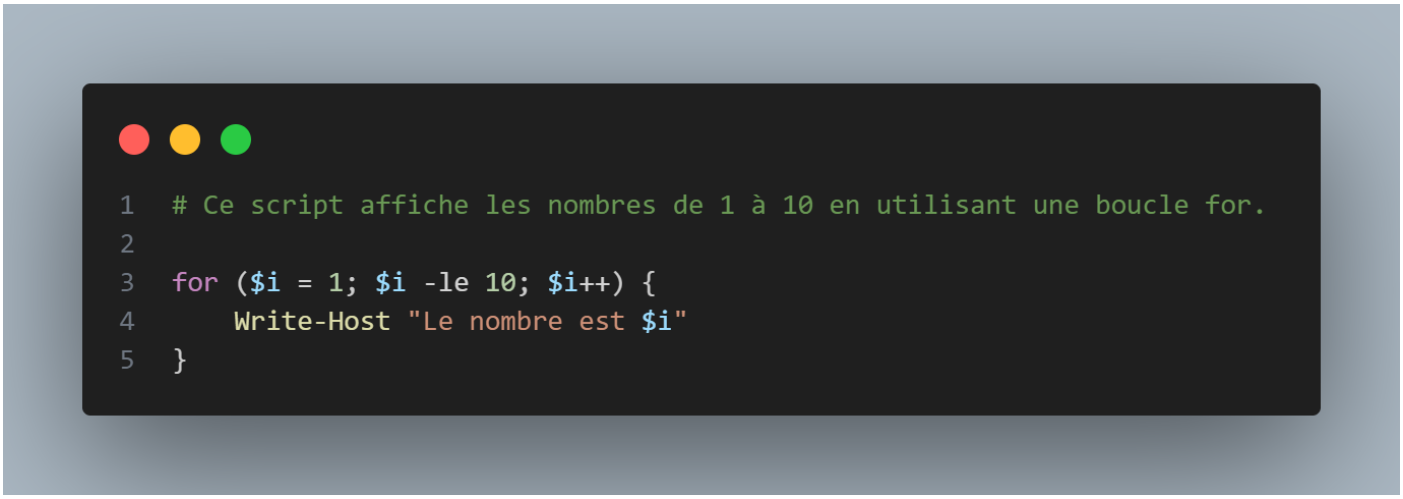
```powershell
$i = 0
$j = Read-Host -Prompt "Définir un nombre"

do {
    $i++
    Write-Host $i
} until ($i -eq $j)

Write-Host "La boucle est arrivé à $j."
```



4. For



```powershell
# Ce script affiche les nombres de 1 à 10 en utilisant une boucle for.

for ($i = 1; $i -le 10; $i++) {
    Write-Host "Le nombre est $i"
}
```

La condition « $i -le 10 » vérifie que $i (défini par défaut sur 1) est inférieur ou égal à 10. Tant que cette condition est vraie, le bloc de la boucle s'exécute. Exemple de sortie :



```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Le nombre est 1
Le nombre est 2
Le nombre est 3
Le nombre est 4
Le nombre est 5
Le nombre est 6
Le nombre est 7
Le nombre est 8
Le nombre est 9
Le nombre est 10
PS C:\Users\rdubois>
```

5. For Each

En PowerShell, la boucle ForEach permet de parcourir chaque élément d'une collection (tableau, liste, objet) sans avoir besoin d'un compteur, exemple de script avec une liste de nombre :

```powershell
# Ce script parcourt une liste de nombres et les affiche avec une boucle ForEach.

$listeNombres = 1..10  # Crée une liste de nombres de 1 à 10

foreach ($nombre in $listeNombres) {
    Write-Host "Le nombre est $nombre"
}
```

La boucle « foreach ($nombre in $listeNombres) » prend chaque élément de la liste et l'affecte à $nombre. Ce qui donne en sortie :

```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Le nombre est 1
Le nombre est 2
Le nombre est 3
Le nombre est 4
Le nombre est 5
Le nombre est 6
Le nombre est 7
Le nombre est 8
Le nombre est 9
Le nombre est 10
PS C:\Users\rdubois>
```

6. If, Else If, Else

En PowerShell, les structures If, ElseIf, et Else permettent d'exécuter différentes actions en fonction d'une condition, exemple avec un script permettant de vérifier si le nombre choisi est positif, négatif ou nul :

```powershell
# Ce script vérifie si un nombre est positif, négatif ou nul.

$nombre = Read-Host "Entrez un nombre"  # Demande à l'utilisateur de saisir un nombre
$nombre = [int]$nombre  # Convertit l'entrée en entier

if ($nombre -gt 0) {
    Write-Host "Le nombre est positif."
}
elseif ($nombre -lt 0) {
    Write-Host "Le nombre est négatif."
}
else {
    Write-Host "Le nombre est nul."
}
```

Read-Host demande un nombre à l'utilisateur.

[int]$nombre convertit cette entrée en nombre entier.

Condition If :

- Si $nombre est supérieur à 0, il est positif.
- Si $nombre est inférieur à 0, il est négatif.
- Sinon, il est nul.

Ce qui donne ceci en sortie avec les nombres -1, 0 et 1 :

```
PS C:\Users\rdubois>
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Entrez un nombre: -1
Le nombre est négatif.
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Entrez un nombre: 1
Le nombre est positif.
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Entrez un nombre: 0
Le nombre est nul.
PS C:\Users\rdubois>
```

7. Switch

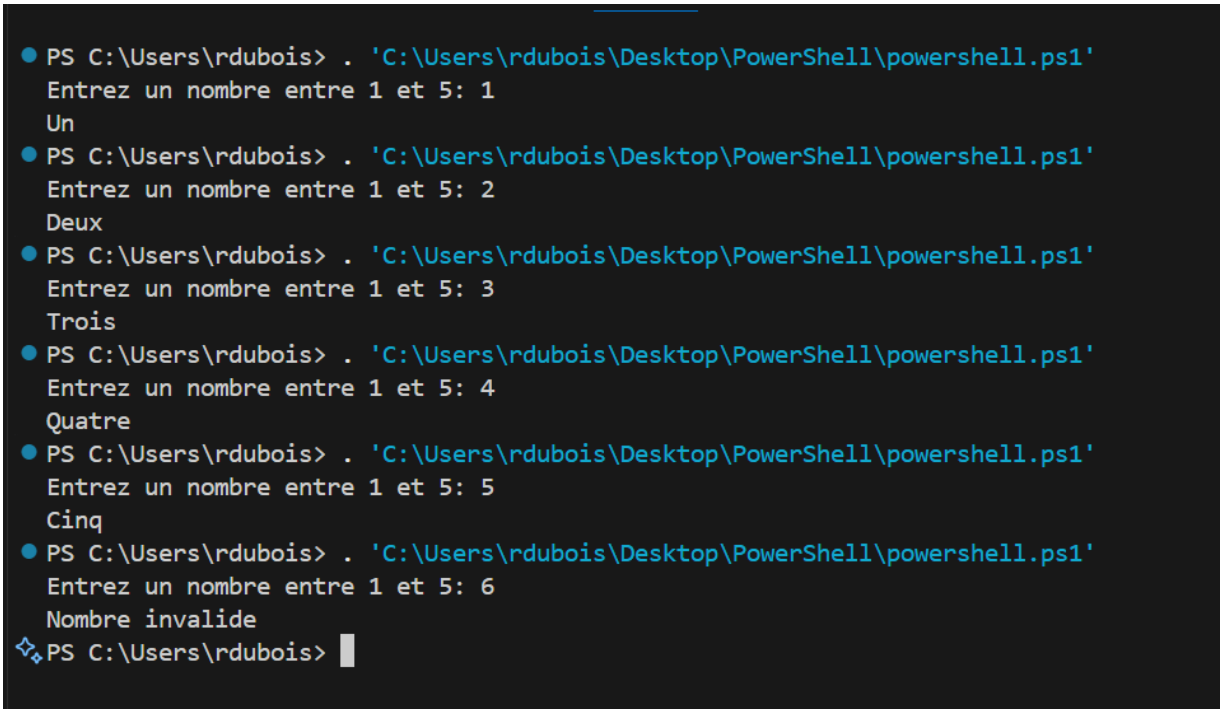
En PowerShell, la structure **Switch** permet d'exécuter différentes actions en fonction de la valeur d'une variable, sans avoir à utiliser plusieurs **If - Elseif**. Elle compare la valeur donnée avec plusieurs cas et exécute le bloc de code correspondant. Si aucun cas ne correspond, un bloc **default** peut être utilisé pour gérer les valeurs non prévues. Exemple de script demandant d'entrer un nombre entre 1 et 5, puis avec un switch ayant pour rôle de rediriger vers le bon bloc entré par l'utilisateur :



```powershell
$i = Read-Host -Prompt "Entrez un nombre entre 1 et 5"

switch ($i) {
    1 {Write-Host "Un"}
    2 {Write-Host "Deux"}
    3 {Write-Host "Trois"}
    4 {Write-Host "Quatre"}
    5 {Write-Host "Cinq"}
    default {Write-Host "Nombre invalide"}
}
```

Résultat en sortie pour chaque valeur de 1 à 6 :



On remarque bien que chaque valeur de 1 à 5 est bien reconnue par notre « switch » mais que lors qu'on dépasse 5, on obtient le résultat défini par défaut : « Nombre invalide ».

8. Fonctions

En PowerShell, une **fonction** permet de regrouper un ensemble d'instructions sous un seul nom, facilitant ainsi la réutilisation du code et l'organisation d'un script. Exemple d'une fonction écrivant « Hello World » avec la date d'aujourd'hui :

```powershell
function HelloWorld
{
    Write-Host "Hello World!"

    $date = Get-Date
    Write-Host "Current date: $date"
}

HelloWorld
```

```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Hello World!
Current date: 03/16/2025 18:04:02
PS C:\Users\rdubois>
```

9. Fonctions avec Arguments

En PowerShell, $args récupère tous les arguments passés à un script ou une fonction sans paramètres définis. Exemple avec notre script précèdent en rajoutant son prénom et nom en argument $args :

```powershell
function HelloWorld
{
    Write-Host "Hello World! $args"

    $date = Get-Date
    Write-Host "Current date: $date"
}

HelloWorld "Raphaël Dubois"
```

Résultat en sortie avec l'argument indiqué :

```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Hello World! Raphaël Dubois
Current date: 03/16/2025 18:13:45
PS C:\Users\rdubois>
```

10. Fonctions avec Paramètres

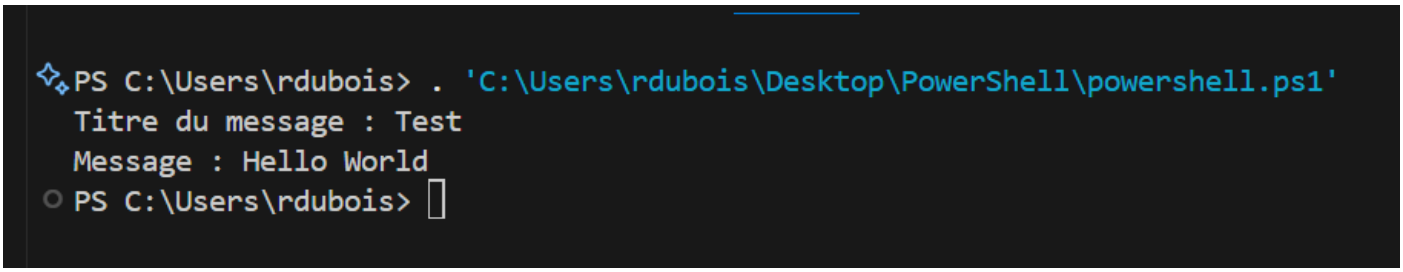
En PowerShell, les **paramètres** sont des variables définies dans une fonction ou un script pour recevoir des valeurs spécifiques lors de son appel. Contrairement aux arguments avec `$args`, les paramètres permettent de définir clairement les entrées attendues. Voici un exemple avec une fonction indiquant un titre et un message reçu lors de l'appel de la fonction avec les paramètres :



```powershell
function Show-Message {
    param(
        $message,
        $titre
    )
    Write-Host "Titre du message : $titre"
    Write-Host "Message : $message"
}

Show-Message -message "Hello World" -titre "Test"
```

Ce qui nous donnera en sortie, le texte que nous avons indiqué dans notre fonction et ce qu'on aura défini comme titre et message dans les paramètres de notre fonction (« Hello World » en message, et « Test » en titre) :



```
PS C:\Users\rdubois> . 'C:\Users\rdubois\Desktop\PowerShell\powershell.ps1'
Titre du message : Test
Message : Hello World
PS C:\Users\rdubois>
```