420-3T5-EM Automatisation de tâches

Fonctions, Pile d'appel, Portée

Rencontre #10

1

Au menu aujourd'hui

- Fonctions
- Pile d'appel *(call stack)*
- Portée *(scope)*
- Trucs avec le débogueur
- Exercices

2

Fonctions

3

Fonctions

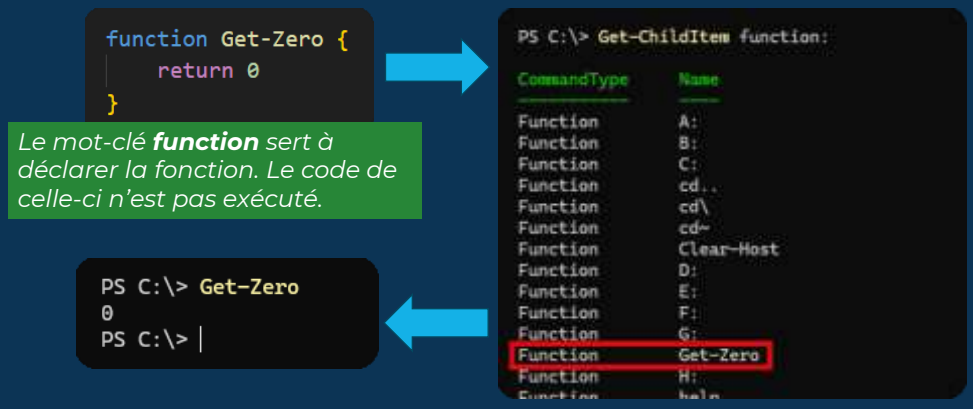
- Une fonction est essentiellement un **script enregistré** dans la session PowerShell
- Elle est définie par le mot-clé **function**
- Son code est compris entre les accolades **{ }**

```
function Get-Zero {
    return 0
}
```

4

Déclaration d'une fonction

Comme une variable, une fonction doit être **déclarée** dans la **session PowerShell en cours**.



*Le mot-clé **function** sert à déclarer la fonction. Le code de celle-ci n'est pas exécuté.*

Fonctions dans un script

La fonction est utile dans un script lorsque la même logique doit être **répétée à plusieurs endroits** du script, pour **alléger le code**, ou pour **réutiliser une logique commune** entre plusieurs scripts.

Par exemple:

- Écriture d'une entrée dans un fichier log
- Génération d'un mot de passe aléatoire
- Calcul complexe

Déclaration dans un script

- Une fonction doit obligatoirement être **déclarée avant** de pouvoir être appelée.
- La simple déclaration du bloc *function* n'a **pas pour effet d'exécuter son code**. Il faut l'appeler pour ça.

```powershell
function Get-Prenom {
    $prenom = Read-Host -Prompt "Veuillez saisir votre prénom"
    $prenom = (Get-Culture).TextInfo.ToTitleCase($prenom)
    Write-Output $prenom
}

Write-Host "Bonjour, $(Get-Prenom)"
```

7

7

Une fonction est un script

Une fonction se comporte comme un script et comme une commande.

- Ses **intrants** sont des **paramètres**, dont certains peuvent être mappés à l'entrée du pipeline
- Ses **extrants** sont des objets s'étant **accumulés à la sortie**, et seront retournés sur le **pipeline**.

8

8

Paramètres de fonction

On peut déclarer des paramètres dans une fonction de la même manière que dans un script.

```powershell
function Get-Prenom {
    param(
        [Parameter(Mandatory)] [string] $Prenom
    )

    $PrenomMajuscules = (Get-Culture).TextInfo.ToTitleCase($Prenom)
    Write-Output $PrenomMajuscules
}

Write-Host "Bonjour, $(Get-Prenom -Prenom "jEaN-pAUl")"
```

9

9

Paramètres de fonction (abrégé)

- On peut aussi définir les paramètres sous cette forme plus brève.

```powershell
function Get-Prenom ($Prenom) {
    $PrenomMajuscules = (Get-Culture).TextInfo.ToTitleCase($Prenom)
    Write-Output $PrenomMajuscules
}
```

- Ça ne nous permet toutefois pas de faire des validations, mais pour les fonctions simples ça peut faire l'affaire.

10

10

Objets en sortie

- Pour sortir un objet, il suffit de l'envoyer au bout du pipeline, pour qu'il s'accumule à la fin de la fonction.

```
function Get-Zero {
    0
}
```

- On peut aussi utiliser la commande **Write-Output**

```
function Get-Zero {
    Write-Output 0
}
```

11

11

Instruction « return »

L'instruction **return** envoie l'objet en sortie sur le pipeline, puis **met fin immédiatement** à l'exécution de la fonction.

```
function Get-Zero {
    return 0
    Write-Output "Cette ligne ne sera jamais écrite"
}
```

Tous les objets accumulés sortent de la fonction, avec celui spécifié avec **return**.

12

12

Pile d’appel

Call stack

13

13

Pile d’appel *(call stack)*

- La pile d’appel est une structure qui garde **la trace des appels** de fonctions, scripts ou autres blocs de code.
- Chaque fois qu’une fonction (ou un script) est appelée, une nouvelle entrée est **ajoutée à la pile**.
- Lorsqu’elle se termine, son entrée est **retirée de la pile** et le flux de contrôle **retourne à son appelant**.

14

14

Démo: pile d'appel

```powershell
function A {
    Write-Host "A appelle B"
    B
    Write-Host "A terminée"
}

function B {
    Write-Host "B appelle C"
    C
    Write-Host "B terminé"
}

function C {
    Write-Host "C terminé"
}

Write-Host "Script appelle A"
A
Write-Host "Script terminé"
```

Ligne	Effet	Pile d'appel

15

15

Débogage

La commande **Get-PSCallStack** permet d'obtenir l'état de la pile d'appel au moment de son exécution.

Ça peut être pratique pour déboguer.

```powershell
function FonctionA ([string]$Nom) {
    FonctionB -Nombre 42
}

function FonctionB ([int]$Nombre) {
    Get-PSCallStack
}

FonctionA -Nom "Minou"
```

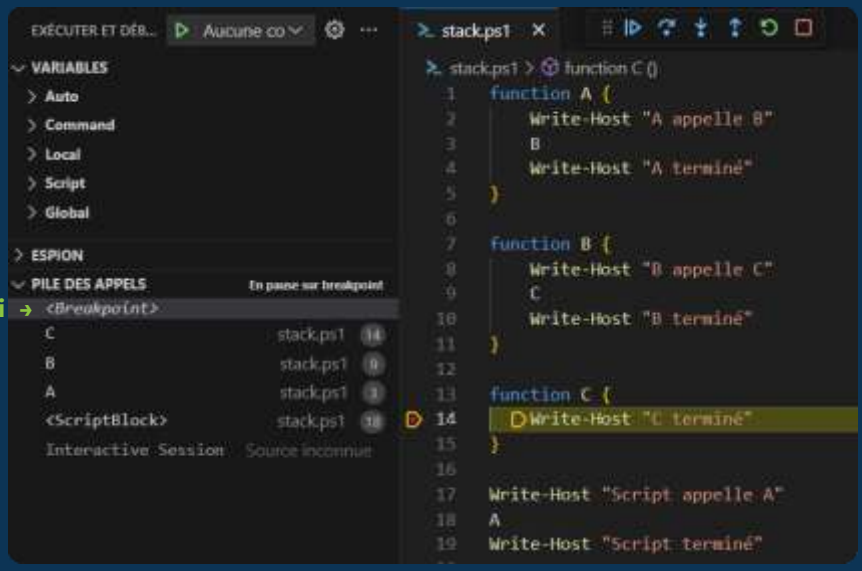
```
Command       Arguments Location
-------       --------- --------
FonctionB     {Nombre=42} script.ps1: line 6
FonctionA     {Nom=Minou} script.ps1: line 2
script.ps1    {}          script.ps1: line 9
<ScriptBlock> {}          <No file>
```

16

16

Débogage avec VS Code

Vous êtes ici →



17

17

Portée

Scope

18

18

Portée *(scope)*

Considérez le scénario suivant:

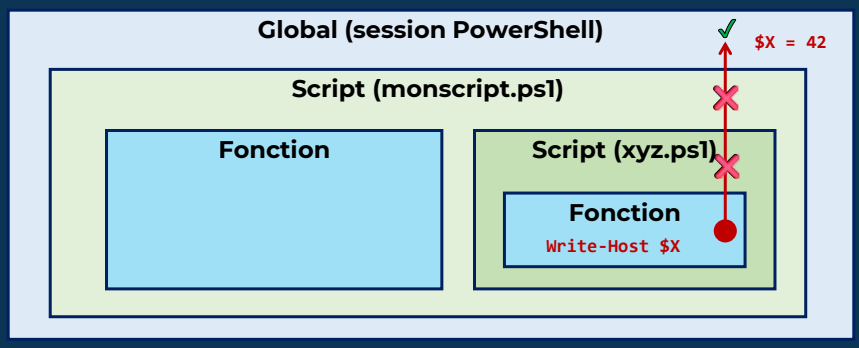
```powershell
function MaFonction {
    $MaVariable = "Wouf!"
}

$MaVariable = "Miaou!"

MaFonction

Write-Host $MaVariable
```

Qu'est-ce qui sera écrit?

19

Fonctionnement de la portée

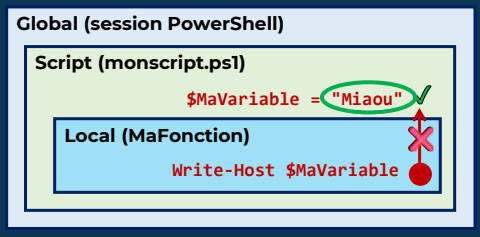
- L'accès à une variable (ou une fonction) est tenté en priorité **à son niveau de la pile d'appel**.
- Si non trouvé, **on remonte** dans la pile d'appel.



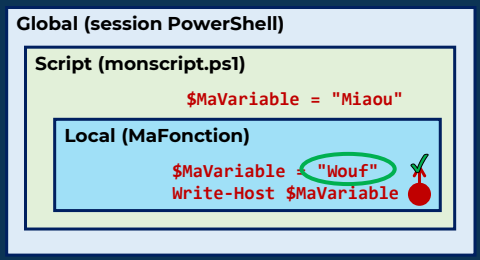
20

Exemples

```powershell
function MaFonction {
    Write-Host $MaVariable
}

$MaVariable = "Miaou"

MaFonction
```




```powershell
function MaFonction {
    $MaVariable = "Wouf"
    Write-Host $MaVariable
}

$MaVariable = "Miaou"

MaFonction
```



21

21

Portées en PowerShell

- **Local**
  - S'arrête au script ou à la fonction
  - Disponible plus haut dans la pile d'appel
- **Script**
  - S'étend sur l'ensemble du script
  - Disponible partout dans le script, y compris dans des fonctions
- **Global**
  - S'étend sur l'ensemble de la session PowerShell
  - Disponible partout sur la pile d'appel

22

22

Affectations

L'affectation d'une variable se fait dans la **portée locale** seulement.

```powershell
function MaFonction {
    $MaVariable = "Wouf"
    Write-Host $MaVariable
}

$MaVariable = "Miaou"

MaFonction

Write-Host $MaVariable
```

```
PS C:\Scripts> .\MonScript.ps1
Wouf
Miaou
```

23

23

Indicateurs de portée

- Les variables déclarées ont une **visibilité** et une **durée de vie** limitées à leur **contexte d'exécution**.
- On peut utiliser les **indicateurs de portée** pour contrôleur la visibilité et la durée de vie.
- Les indicateurs de portée sont:
  - $**global:**MaVariable: Définit $MaVariable dans toute la session
  - $**script:**MaVariable: Définit $MaVariable dans tout le script, peu importe le nombre de fonctions imbriquées
  - $**private:**MaVariable: Définit $MaVariable dans la portée locale seulement et la rend inaccessible de l'extérieur

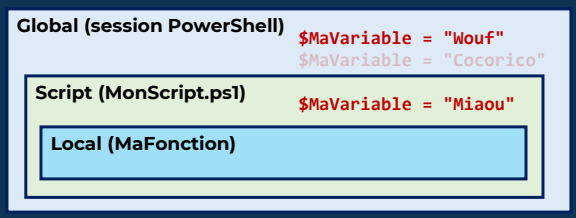
24

24

Indicateur de portée: Global

```powershell
function MaFonction {
    $global:MaVariable = "Wouf"
    Write-Host $MaVariable
}

Write-Host $MaVariable

$MaVariable = "Miaou"

MaFonction

Write-Host $MaVariable
```

```
PS C:\Scripts> $MaVariable = "Cocorico"

PS C:\Scripts> .\MonScript.ps1
Cocorico
Miaou
Miaou

PS C:\Scripts> $MaVariable
Wouf
```



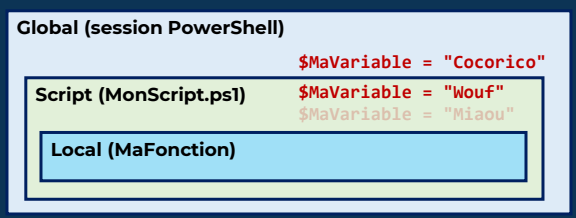
25

25

Indicateur de portée: Script

```powershell
function MaFonction {
    $script:MaVariable = "Wouf"
    Write-Host $MaVariable
}

Write-Host $MaVariable

$MaVariable = "Miaou"

MaFonction

Write-Host $MaVariable
```

```
PS C:\Scripts> $MaVariable = "Cocorico"

PS C:\Scripts> .\MonScript.ps1
Cocorico
Wouf
Wouf

PS C:\Scripts> $MaVariable
Cocorico
```



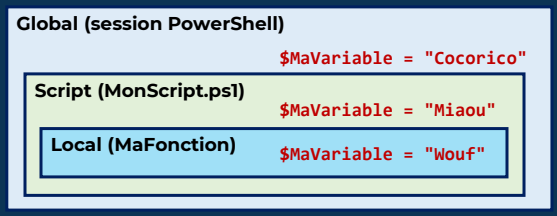
26

26

Contourner la portée locale

```powershell
function MaFonction {
    $MaVariable = "Wouf"
    Write-Host $MaVariable
    Write-Host $script:MaVariable
    Write-Host $global:MaVariable
}

$MaVariable = "Miaou"

MaFonction
```

```
PS C:\Scripts> $MaVariable = "Cocorico"

PS C:\Scripts> .\MonScript.ps1
Wouf
Miaou
Cocorico

PS C:\Scripts> $MaVariable
Cocorico
```



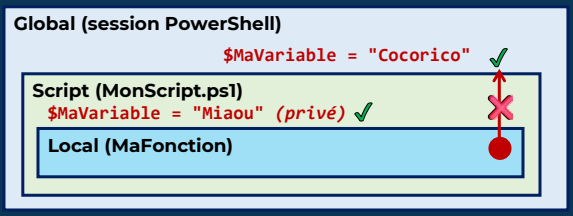
27

27

Indicateur de portée: Private

```powershell
function MaFonction {
    Write-Host $MaVariable
}

Write-Host $MaVariable

$private:MaVariable = "Miaou"

MaFonction

Write-Host $MaVariable
```

```
PS C:\Scripts> $MaVariable = "Cocorico"

PS C:\Scripts> .\MonScript.ps1
Cocorico
Cocorico
Miaou

PS C:\Scripts> $MaVariable
Cocorico
```



28

28

Dot-sourcing

Par défaut, tout ce qui est déclaré en portée locale dans un script sera automatiquement détruit à la fin de son exécution.

```powershell
function MaFonction {
    $MaVariable = "Wouf"
    Write-Host $MaVariable
}

$MaVariable = "Miaou"

MaFonction

Write-Host $MaVariable
```

```
PS C:\Scripts> $MaVariable = "Cocorico"

PS C:\Scripts> .\MonScript.ps1
Wouf
Miaou

PS C:\Scripts> $MaVariable
Cocorico

PS C:\Scripts> MaFonction
Erreur: Terme non reconnu
```

29

29

Dot-sourcing

En ajoutant un **« . »** avant l'appel du script, ce dernier est exécuté dans la portée globale. Les variables et les fonctions **persistent** après la fin du script

```powershell
function MaFonction {
    $MaVariable = "Wouf"
    Write-Host $MaVariable
}

$MaVariable = "Miaou"

MaFonction

Write-Host $MaVariable
```

```
PS C:\Scripts> $MaVariable = "Cocorico"

PS C:\Scripts>. .\MonScript.ps1
Wouf
Miaou

PS C:\Scripts> $MaVariable
Miaou

PS C:\Scripts> MaFonction
Wouf
```

30

30

Bonnes pratiques

- Déclarez votre ou vos fonctions de script **au début** de celui-ci, juste **après le bloc Param()**
- Nommez vos fonctions selon la norme **Verbe-Nom**
- **Initialisez** toujours vos variables localement pour éviter une confusion de portée
- Privilégiez la **portée locale**
- Utilisez les **intrants et extrants de fonctions** au lieu d'accéder à des variables des autres portées

31

31

D'autres trucs avec le débogueur

32

32

Débogage: types de pas à pas

Pas à pas principal (F10)
- Saute par-dessus les fonctions

Pas à pas détaillé (F11)
- Entre dans la fonction pour continuer le pas à pas

Pas à pas sortant (Maj+F11)
- Sort de la fonction en cours et retourne dans le script principal

33

33

Configuration du débogueur

Le débogueur est très utile, mais moins lorsqu'il s'agit de tester de passage d'argument au script.

```powershell
param(
    [ValidateNotNullOrEmpty()]
    [Parameter(Mandatory, ValueFromPipeline)]
    [String] $Message,

    [ValidateRange(1,[int]::MaxValue)]
    [int] $Repetitions = 3
)

foreach ($Iter in 1..$Repetitions) {
    Write-Host "$Iter : $Message."
}
```

Quand on fait F5, le script est dot-sourcé sans argument!

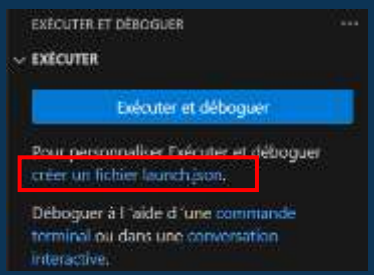
```
PS C:\> . 'C:\Scripts\MonScript.ps1'
```

34

34

Configuration du débogueur

Il faut créer un fichier **launch.json** et **ajouter une configuration à votre espace de travail**




```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "PowerShell: Launch Current File",
            "type": "PowerShell",
            "request": "launch",
            "script": "${file}",
            "args": []
        },
        {
            "name": "Miaou 5 fois",
            "type": "PowerShell",
            "request": "launch",
            "script": "${file}",
            "args": [
                "-Message 'Miaou'",
                "-Repetitions 5"
            ]
        }
    ]
}
```

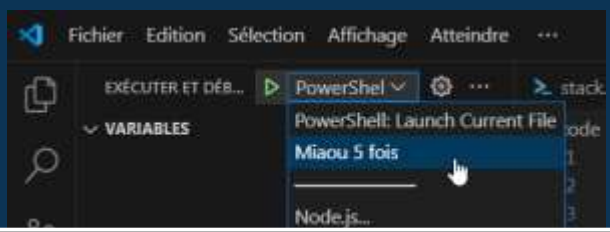
Guillemets simples!

35

35

Configuration du débogueur

Ensuite sauvegardez le fichier dans votre espace de travail. Sélectionnez votre configuration, puis faites F5. Les paramètres vont se passer automatiquement!




```
PS C:\> . 'C:\Scripts\MonScript.ps1' -Message 'Miaou' -Repetitions 5
```

36

36