4.10 Les Fonctions

Avec PS, il est possible de définir des scripts paramétrés qui peuvent être passés à la ligne de commande. Par exemple, avec le script ci-dessous, un paramètre nommé `name` est défini et initialisé par défaut à `Bill`.

```
param($name = "Bill")
"Bonjour $name, Comment allez vous ?"
```

Exercice 48 Testez le script précédent sans donner de paramètres et en donnant un paramètre à la ligne de commande comme le montre l'exécution ci-dessous :

```
PS C:\Users\denis> scripts:\Hello_Param.ps1
Bonjour Bill, Comment allez vous ?
PS C:\Users\denis> scripts:\Hello_Param.ps1 Denis
Bonjour Denis, Comment allez vous ?
```

Réponse :

Par ailleurs, comme tout langage de haut niveau, vous souhaiteriez créer des sous-routines (fonction ou procédure) qui pourrait être appelées plusieurs fois dans votre script avec des paramètres différents. Ceci est possible en PS avec l'instruction `function` qui peut se présenter principalement de 2 façons :

Hello_Param.ps1

```
1  function Hello1 ([String]$name="Bill") {
2      "Bonjour $name, je m'appelle Hello1 ?"
3  }
5  function Hello2 {
6      param([String] $name="Bill")
7      "Bonjour $name, je m'appelle Hello2 ?"
8  }
```

Lorsque vous définissez une fonction, elle ne sera visible que dans le script que vous venez de créer à moins de modifier la portée de la fonction. Pour cela, il suffit de modifier l'entête de la fonction de la façon suivante : `function global:Hello1` et d'exécuter le script dans lequel la fonction a été déclarée. Ensuite, la fonction sera accessible via la ligne de commande. Vous pouvez également connaître la liste des fonctions connues et accessibles de votre session PS en listant le disque prédéfini `function` (`get-childitem function:`).

Exercice 49 Saisissez le script suivant et exécutez-le. Qu'en déduisez-vous ?

NbParam.ps1

```
1  function NbParam ($x,$y) {
2      "1st param = $x"
3      "2nd param = $y"
4      "other params = $args"
5  }
6
7  NbParam 1
8  NbParam 1 2
9  NbParam 1 2 3
10 NbParam 1 2 3 4
```

Réponse :

Exercice 50 Au même titre que `$args` dans l'exercice précédent, il existe une variable prédéfinie `$input` contenant l'ensemble des arguments transmis dans le pipe. Sachant cela, écrivez une fonction `Sum` qui calcule la somme des arguments récupérez dans le pipe. On pourra utiliser le mot clé `return` pour renvoyer une valeur dans la fonction. Voici un exemple d'exécution :

```
PS C:\Users\denis> 1..5 | sum
15
```

Réponse :

Exercice 51 Maintenant que vous êtes capable d'accepter en entrée de votre fonction des objets du pipe, vous souhaitez afficher quelque chose avant et après l'exécution de la fonction. Pour cela, modifiez votre fonction en affichant un message avant et après le code de votre fonction. Que constatez-vous ?

Réponse :

Afin de résoudre le problème posé dans l'exercice précédent, PS propose les instructions de bloc `Begin`, `Process` et `End` afin de regrouper les instructions qui devront être exécutées avant (`Begin`), pendant (`Process`) ou aprés (`End`). Le bloc `Process` est exécuté une fois pour chaque objet récupéré du pipe et est accessible par `$_`. Reportez vous à la rubrique `about_functions` de l'aide.

Exercice 52 Pour illustrer cette fonctionnalité, nous souhaitons écrire une fonction `ConvertUS2Metric` qui convertit un fichier de données (cf. un exemple ci-dessous) stocké en utilisant la métrique américaine en un autre ensemble de données utilisant la métrique européenne. Une fois le fichier traité, nous souhaitons connaître le nombre de lignes traitées par la fonction. Indices : °C= (°F-32) x 5/9 ; KmPH=MPH x 1.609344 ; mm=25.4 x inch. Utilisez un paramètre qui correspond au séparateur de champ (virgule, point-virgule, espace, tabulation...). Voici comment la fonction devrait être utilisée : `gc USData.csv | ConvertUS2Metric`.

USData.csv

```
Time,Degree (F),Size (inch),Speed (MPH)
2008-01-01,77,2.6,124.89
2010-12-31,86,4.7,66.777
2004-02-28,71.6,1.5,128.2
1900-01-01,99.5,4.9,156.67
1972-02-08,82.4,5.2,28.98
```

FRData.csv

```
Date , Degré (C) , Taille (mm) , Vitesse (KMH)
01/01/2008,25,66.04,200.99097216
31/12/2010,30,119.38,107.467164288
28/02/2004,22,38.1,206.3179008
01/01/1900,37.5,124.46,252.13592448
08/02/1972,28,132.08,46.63878912
```

Réponse :

Comme en java avec la commande `javadoc`, il est possible de saisir des commentaires utilisés par ailleurs. C'est pourquoi, le format donné à l'Exercice 42 peut être utilisé automatiquement par la commande `Get-help`. Pour avoir une idée plus précise comment cela fonctionne, saisissez l'instruction `help comment_Based_Help`.

Exercice 53 Modifiez le script précédent pour que l'aide que vous obtenez avec la commande Get-help soit la plus correcte possible.

Réponse :

Exercice 54 Maintenant que vous savez faire des scripts bien commentés, il se peut que le collaborateur pour qui vous développez le script ne maîtrise pas l'anglais (la preuve, vous êtes bien content de lire l'aide en français). PS permet l'internationalisation avec le principe de section DATA dans les fonctions/scripts. Utilisez l'aide sur `about_script_internationalization` et créez les fichiers d'aide en français et en anglais pour le script précédent.

Réponse :

Exercice 55 Nous souhaitons personnaliser PS-ISE en ajoutant dans le menu `Composants additionnels` une fonction qui va rechercher sur le net et sur le moteur de recherche Google de l'aide sur le texte actuellement sélectionné. Indice : il est fortement conseillé de regarder la cmdlet new-Object. Essayez ensuite avec Firefox s'il est installé.

Réponse :