

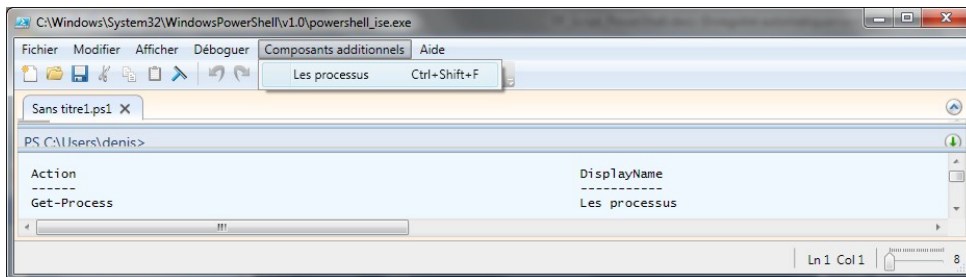
Exercice 13 Fermez PS. Relancer le et exécutez le dernier script créé. Que se passe-t-il ? Pourquoi ?

Exercice 14 Faites en sorte que le disque spécifique créé précédemment `scripts` soit connu à chaque lancement de PS (cf. ). Choisissez bien le chemin que vous allez utiliser car nous allons l'utiliser dans le reste du cours.

### 4.3 Ajouter ses propres fonctions dans le menu de PS-ISE

Une des caractéristiques les plus intéressantes de PS-ISE est de pouvoir le personnaliser avec des commandes que nous utilisons souvent pour l'administration comme nous l'avons fait plus haut pour la ligne de commande. Il est donc possible de personnaliser les menus de PS-ISE. Pour cela, il suffit d'écrire sa cmdlet ou fonction et d'appliquer la méthode `Add` sur le contenu de la variable `CurrentPowerShellTab.AddOnsMenu.Submenus` (qui est une collection) de l'objet `psISE`.

Exercice 15 Modifier le profile de PS-ISE en ajoutant un sous-menu personnalisé qui ajoute une commande qui affiche les processus courants. Pour cela, utilisez la méthode `Add("Nom du sous-menu", {votre-Commande}, "Raccourci-clavier")` sur la collection précédente afin d'obtenir le résultat ci-dessous :



## 4.4 Connexion de commandes : le pipe

Comme les autres shells, PS permet de connecter deux commandes à l'aide du pipe « | ». C'est simplement un moyen de transférer les informations résultantes de la première commande à la commande suivante. Il y a toutefois des différences fondamentales.

Exercice 16 Saisissez la suite d'instructions suivantes :

```
Get-Process | Export-Csv procs1.csv  
Notepad procs1.csv
```

Grâce au pipe, vous venez d'exporter les processus courants dans un fichier nommé `procs1.csv` au format CSV (Comma Separated Values) et de lancer la visualisation de ce fichier dans l'éditeur de texte. Comme vous venez de le voir, PS utilise le même principe de pipe que celui que vous pouvez connaître mais l'étend pour obtenir des effets plus importants. Il est plus riche et plus moderne car il travaille directement sur des objets comme nous le verrons plus tard. Par ailleurs, il existe d'autres formats d'export que CSV comme CliXML (`Export-Clixml`). L'intérêt d'exporter réside dans la possibilité de comparer ensuite les fichiers ou de les publier. Il est possible de traduire du contenu en CSV ou autre format comme de l'HTML sans forcément les sauvegarder sur disque en utilisant les commandes `ConvertTo-*`.

Exercice 17 Grâce à la cmdlet `Send-MailMessage`, envoyez votre fichier texte par mail à la personne qui est sur la machine d'à côté. Ensuite, identifiez les processus qui sont différents d'une machine à l'autre grâce à la commande `Compare-Object`. Indice : limitez vous à considérer toute la ligne pour l'instant.

Exercice 18 Afin de ne récupérer que le nom des processus qui sont différents sans s'intéresser à d'autres informations comme le % CPU ou autres, répondez à la question précédente en intégrant la commande `import-csv` ou `import-CliXML` en fonction de votre précédent export. On s'intéressera uniquement à la colonne `Name`.

Exercice 19 Par défaut, PS vous affiche les résultats de votre commande sous format texte. Il est possible de le récupérer sous un autre format. Testez la commande suivante : `Get-Process | Out-GridView`.

Exercice 20 Merci de réfléchir et de noter ce que pourrait faire cette commande SANS L'EXECUTER :

```
Get-Process | Stop-Process
```

Exercice 21 Vous souhaitez réellement savoir ce qui se passerait si vous exécutiez réellement cette commande mais vous avez peur de ce que le prof vous réserve ?! Dans ce cas, ayez confiance en lui en ajoutant l'option `-WhatIf` à la fin de la commande de l'exercice précédent. Alors, cela vous donne quoi ?

Exercice 22 Afin de bien comprendre la notion d'objets dans les scripts, saisissez l'instruction suivante :

```
Get-ChildItem C:\ | Format-Table -property Name,LastWriteTime
```

Cette commande récupère la liste des fichiers ou répertoires du chemin précisé et l'affiche en utilisant un format bien précis grâce au pipe. Néanmoins, PS diffère des autres shells dans la mesure où les informations qui transitent dans le pipe ne sont pas textuelles mais sont bien orientées objets. Pour bien comprendre cette différence, nous allons essayer d'ajouter une chaîne de caractères devant chaque ligne récupérées précédemment. Pour cela, l'instruction qui vous viendrait à l'idée serait :

```
"un peu de texte avant: " + (Get-ChildItem C:\ | Format-Table -property Name,LastWriteTime)
```

Vous obtenez un résultat complètement différent de celui que vous espériez. Ceci s'explique par le fait que vous essayez de combiner du texte avec une *collection d'objets* formatés ; ainsi la chaîne spécifiée s'affiche grâce à la méthode `ToString` qui est appelée pour chaque objet de la collection. Or, la méthode `ToString` appliquée à un objet renvoyé par `Format-Table` affiche tout simplement `Microsoft.PowerShell.Commands.Internal.Format.FormatEntryData`. Ainsi, PS affiche autant de fois cette chaîne qu'il y a d'objets dans la table concaténée avec « un peu de texte avant ».

Exercice 23 Proposez l'instruction qui permet de faire ce que nous voulions au départ. Pour cela, transformez la

collection d'objets formatés en chaîne de caractères à l'aide de la cmdlet `Out-String`.

Encore une autre instruction qui vous prouvera que PS est orienté objets, saisissez la commande suivante :

```
"un peu de texte" | Get-Member
```

Vous obtiendrez les informations sur l'objet précédé du pipe, c'est-à-dire de l'objet `System.String`.

**Exercice 24** En utilisant l'aide de PS, renseignez vous sur cette nouvelle commande `Get-Member` et sur une de ses options `-force`. Appliquez cette option sur la commande précédente et comparez. Identifiez les lignes supplémentaires. Cherchez à quoi elles correspondent grâce à l'aide ?

## 4.5 Les caractéristiques du langage

### 4.5.1 Les variables

Comme vous aurez pu le comprendre avec les instructions précédentes et/ou l'aide que vous avez consulté, les variables en PS sont définies à l'aide du caractère `$`. Voici un exemple :

```
$a= 1  
$b= "1"
```

**Exercice 25** Appliquez la méthode vue dans l'Exercice 24 sur chacune des variables `a` et `b` et comparez. Que donne les instructions suivantes : `$a*= 2.5` et `$b*=2.5` ? Affichez le contenu de `a` et `b` et regardez leur type.

Pour avoir plus d'informations sur les variables, lisez l'aide fournie par PS en utilisant l'instruction `help variable`. Remarquez que PS crée par défaut un disque contenant toutes les variables (`variable:`)

### 4.5.2 Les variables prédéfinies

En plus des variables définies par l'utilisateur, PS supporte des *variables automatiques* (constantes utilisées pour stocker des valeurs de l'état courant ; utilisez l'instruction `help about_Automatic`), des *variables d'environnement* (stocke des valeurs concernant l'environnement de travail de l'utilisateur courant et du système utilisé ; utilisez l'instruction `help about_Environment`), de *préférence* (stocke des valeurs modifiables de

configuration de PS courant ; utilisez l'instruction `help about_Preference`). Vous utiliserez très souvent les variables `$_`, `$Args`, `$Error`, `$Input`, and `$MyInvocation` mais voici les principales variables.

Exécuter l'instruction `Get-ChildItem variable:` pour avoir la liste des variables prédéfinies et leurs contenus.

Variables Automatiques	Description
<code>\$\$</code>	Dernier élément de la commande reçue par PS
<code>\$^</code>	Premier élément de la commande reçue par PS
<code>\$?</code>	Stocke le résultat d'exécution de la dernière commande (True ou False en fonction de la réussite de la commande)
<code>\$_</code>	Stocke l'objet courant dans l'ensemble des objets d'un pipe. Utiliser cette variable dans les commandes qui effectuent une action sur tous les objets d'un pipe.
<code>\$Args</code>	Représente un tableau de paramètres
<code>\$ConsoleFileName</code>	Stocke le plus récent chemin de fichier de console utilisé (.psc1)
<code>\$Error</code>	Stocke un tableau des dernières erreurs survenues dans la session courante. <code>\$Error[0]</code> représente l'erreur la plus récente
<code>\$ExecutionContext</code>	Stocke un objet <code>EngineIntrinsics</code> qui représente le contexte d'exécution de l'hôte Windows PS
<code>\$False / \$True</code>	Stocke les valeurs booléennes False et True
<code>\$ForEach</code>	Stocke l'énumérateur d'une boucle <code>ForEach-Object</code>
<code>\$Home</code>	Chemin complet du répertoire utilisateur
<code>\$Host</code>	Objet représentant l'application courante hôte
<code>\$Input</code>	Stocke les entrées qui sont passées à une fonction
<code>\$LastExitCode</code>	Code de sortie du dernier programme Windows qui a été exécuté
<code>\$Matches</code>	Stocke les chaînes de caractères résultats lors de l'utilisation de l'opérateur de comparaison <code>-match</code>
<code>\$MyInvocation</code>	Stocke un objet représentant des informations de la commande courante
<code>\$NestedPromptLevel</code>	Stocke le niveau d'imbrication de la ligne de commande. La valeur 0 correspond au niveau original
<code>\$NULL</code>	Stocke la valeur NULL ; peut être utilisée à la place de la chaîne « NULL »
<code>\$PID</code>	Numéro identifiant du processus représentant PS pour l'utilisateur courant
<code>\$Profile</code>	Chemin complet de PS pour l'utilisateur courant
<code>\$PSBoundParameters</code>	Table de hachage contenant les valeurs des paramètres actifs
<code>\$PsCulture</code>	Paramètres de culture du système courant (fr-FR)
<code>\$PsDebugContext</code>	Contient des informations sur l'environnement de débogage en cours (cf. §Le débogage (Dummies page 240))
<code>\$PsHome</code>	Chemin complet du répertoire d'installation de PS
<code>\$PsUICulture</code>	Informations de culture sur PS-ISE
<code>\$PSVersionTable</code>	Informations sur la version de PS
<code>\$Pwd</code>	Objet représentant le répertoire courant
<code>\$ShellID</code>	Chaîne représentant le nom du shell courant
<code>\$This</code>	Définit une propriété ou une méthode de script à l'intérieur d'un script

PS supporte les concepts de stockage comme le kilo-octet (KB), le méga-octet(MB), le giga-octet (GB), le téra-octet(TB), le peta-octet(PB). Il existe par ailleurs des instructions spécifiques aux variables comme `Get-Variable`, `Set-Variable`, `Remove-Variable` et `Clear-Variable`.

**Exercice 26** Saisissez le code ci-dessous. Qu'affiche t-il ? Inversez l'ordre des variables `MyString` et `MyDouble` lors de la définition de la variable `Outstring`. Que se passe t il ? Pour y remédier, essayez d'introduire `[Double]` devant la variable `MyDouble`. Afin de mieux écrire votre script, récrivez cette dernière version avec l'opérateur `-as`.

```
$MyString = " Windows PowerShell "
$MyDouble = 2.0
$outstring = $MyString + $MyDouble
```

```
write-output $outstring
```

### 4.5.3 Les opérateurs

Les opérateurs possibles sont :

- Les opérateurs *arithmétiques* : +, -, \*, /, % (modulo) ;
- Les opérateurs *d'affectations* : =, +=, -=, \*=, /=, %=, ++, -- ;
- L'opérateur \$ qui désigne une variable ;
- L'opérateur \$\_ représente l'instance de l'objet courant renvoyé par le pipe ;
- L'opérateur & pour invoquer un bloc d'instructions comme le montre l'exemple ci-dessous.

```
PS C:\Users\denis> $a=(Get-Process -id 0)
PS C:\Users\denis> &$a
```

Handles	NPM(K)	PM(K)	WS(K)	UM(M)	CPU(s)	Id	ProcessName
0	0	0	24	0		0	Idle

- Les opérateurs de *regroupement* : () ou {} qui regroupe un ensemble d'instructions pouvant être appelées dans une instruction, @() qui exécute plusieurs instructions et stocke chacun des résultats dans un tableau. Par exemple, l'instruction @( \$env:COMPUTERNAME; Get-Date; \$env:LOGONSERVER ) [1] renverra la date courante.
- Les opérateurs de *comparaisons* : -eq (=), -ne(≠), -lt(<), -gt(>), -ge(≥), -le(≤), -contains, -notcontains, -like, -notlike, -match, -notmatch
- L'opérateur de conversion -as qui convertit une entrée en un type spécifique à .NET ("3/31/10" -as [datetime] donnera "Wednesday, March 31, 2010 12:00:00 AM".
- L'opérateur de construction de tableaux : , (\$a = 1, 2, 4, 6, 4, 2)
- L'opérateur de construction d'intervalle : .. (\$a = 2..24)

**Exercice 27** Trouver l'instruction qui modifie le PATH courant en y ajoutant un nouveau répertoire comme c:\toto. Modifiez vos profils (PS & PS-ISE) pour ajouter le chemin qui permet d'accéder à vos scripts.

**Exercice 28** Trouvez l'instruction qui permet de définir la constante PI à 3.1415926. Puis supprimer la variable. Que constatez-vous ? Essayez avec l'option ReadOnly.

