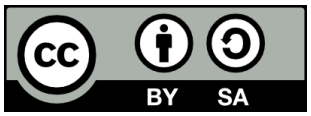
Les rudiments de la programmation en PowerShell



Contenu

- **Un script PowerShell, c’est quoi?**
- **Les concepts de base**
  - Les variables
  - Les constantes
  - Les commentaires
  - Les opérateurs
  - Les structures conditionnelles

Un script PowerShell, c’est quoi?

Photo par Max Duzij sur Unsplash

Un script PowerShell, c’est quoi?

- **Dans le cours de Linux, vous avez vu les scripts Bash**
  - Script Bash = un fichier texte qui contient des commandes Bash à exécuter
  - Peut aussi inclure des commentaires, utiliser des variables, des structures conditionnelles et des boucles, etc.
- **Un script PowerShell, c’est un peu la même chose**
  - Mais avec la syntaxe et les commandes de PowerShell au lieu de Bash

Extension pour les scripts PowerShell

- **Un fichier de script PowerShell prend l'extension « .ps1 »**

  (Et non, ça n'a aucun lien avec la Playstation 1!)

Photo par Waldemar Brandt sur Unsplash

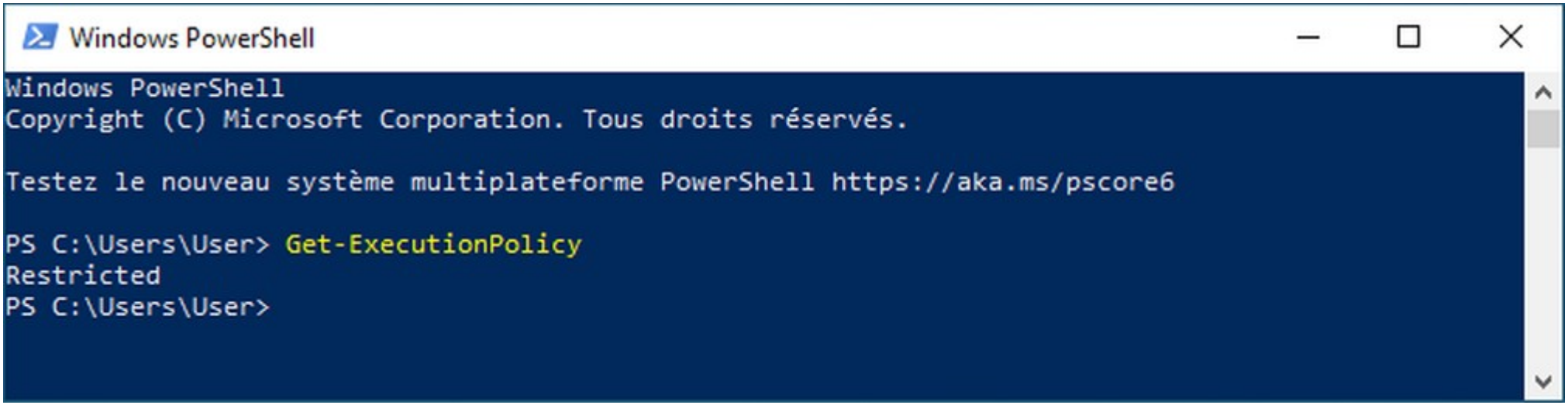
Exécuter un script PowerShell

- **Par défaut, PowerShell ne nous laisse pas exécuter de scripts!**
  - C'est une mesure de sécurité
  - Évite qu'un utilisateur un peu naïf double-clique sur un script envoyé par un méchant hacker et qu'il s'exécute automatiquement

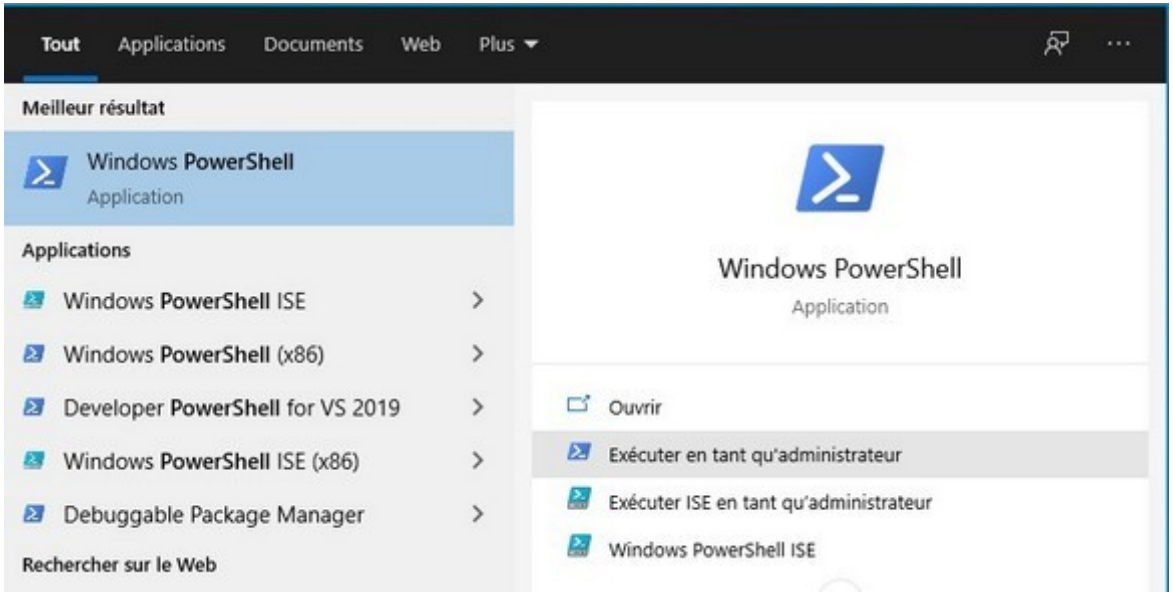
Politiques d’exécution de PowerShell

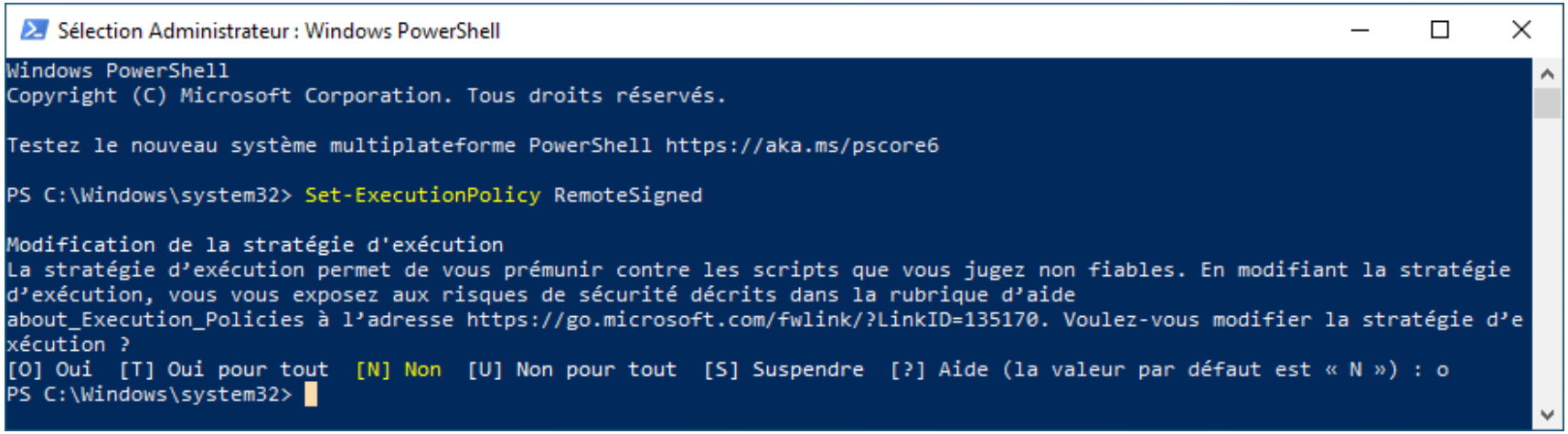
- **Restricted:** PowerShell n’exécute aucun script.
- **AllSigned:** PowerShell exécute seulement les scripts signés (nous reviendrons sur cette notion plus tard dans le cours)
- **RemoteSigned:** PowerShell va refuser d’exécuter un script provenant d’Internet s’il n’est pas signé, mais il va exécuter les autres scripts sans problème.
- **Unrestricted:** PowerShell va accepter d’exécuter n’importe quel script, mais va demander une confirmation avant d’exécuter un script provenant d’Internet.

Vérifier quelle politique d’exécution est active



Changer la politique d’exécution





Les variables

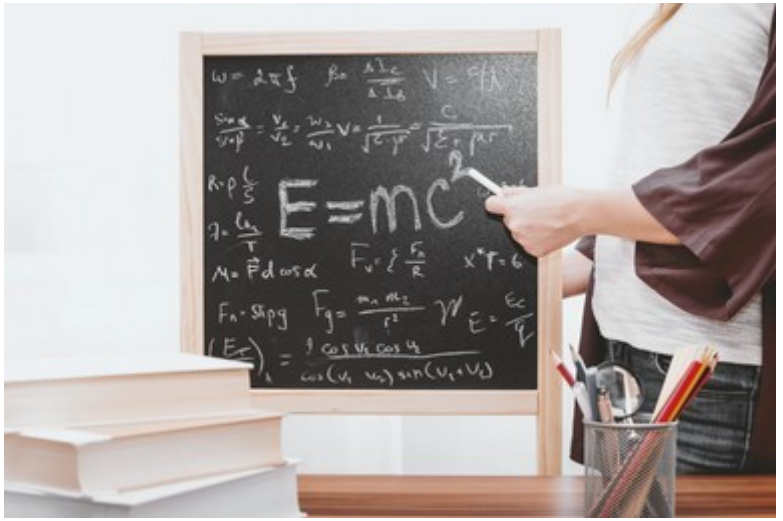


Photo provenant de JESHOOTS.com

Les variables en PowerShell

- **Le nom d'une variable commence par le symbole « $ » (ex: $maVariable)**
  - N'est pas sensible à la casse
  - Peut contenir des caractères spéciaux (non recommandé)
  - Peut même contenir des espaces (non recommandé *)
    - Dans ce cas, le nom de la variable doit être placé entre accolades (« { } »)
      - (ex: ${ma variable})

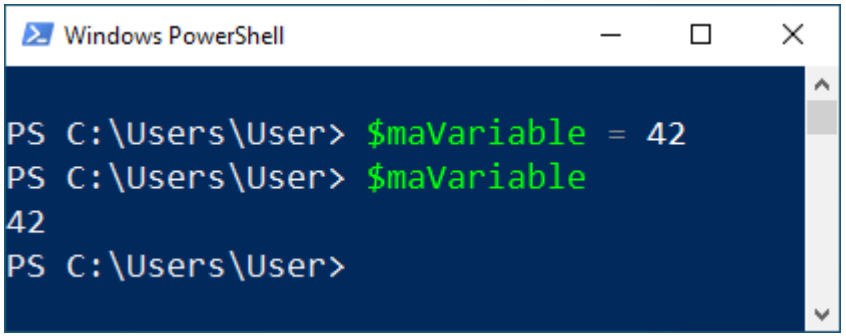
* *Réaction probable d'un programmeur à la vision d'un nom de variable comprenant des espaces*

Déclarer une variable

- **Pour déclarer une variable, il suffit de la nommer, suivie du symbole « = », puis de sa valeur**
  - $maVariable = 42

Afficher une variable

- **Pour afficher une variable, il suffit d'écrire son nom**
  - On peut aussi utiliser la commande « Write-Output » (echo)



Les types

- **La valeur d'une variable possède un type**
- **Le type détermine les opérations qui peuvent être effectuées sur la variable**
  - (ex: on peut multiplier deux nombres, mais on ne peut pas multiplier deux chaînes de caractères!)

Types courants

- **Types numériques**
  - Nombre entier (int ou System.Int32)
    - Ex: $maVariable = 42
  - Nombre entier long (long ou System.int64)
    - Pour les nombres entiers > 2^31 (un bit est réservé pour le signe)
  - Nombre à virgule flottante (double ou System.Double)
    - Ex: $maVariable = 3.1416

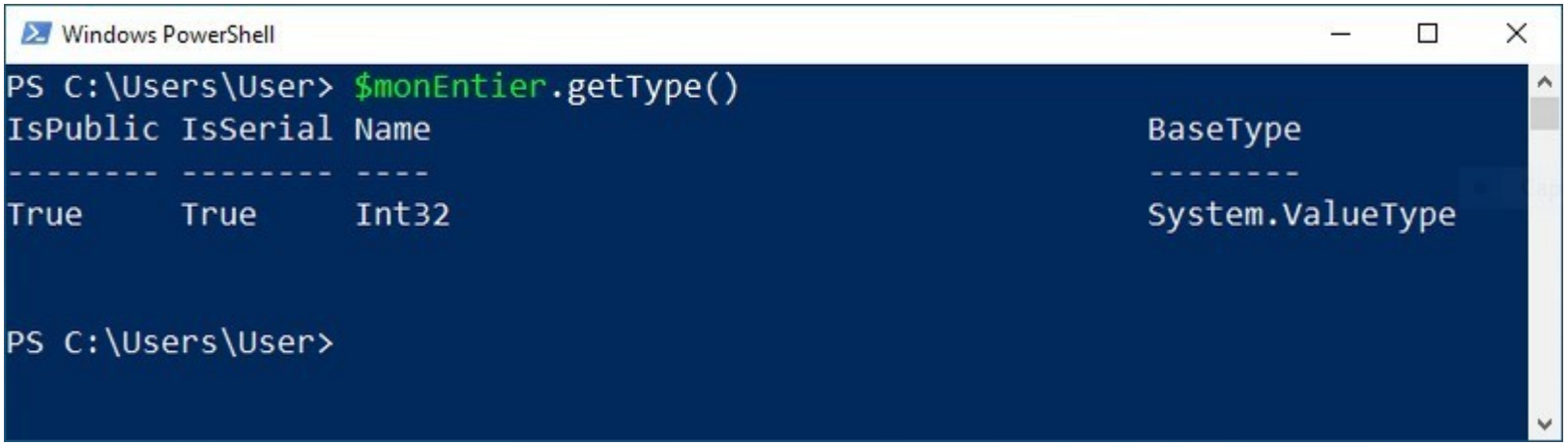
Types courants

- **Chaîne de caractères (string ou System.String)**
  - Ex:
    - $maVariable = "Bonjour le monde!"
    - $maVariable = 'Bonjour le monde!'
  - Lorsqu'un utilise les guillemets (""), on peut inclure d'autres variables dans notre chaîne
    - Ex:
      - $prenom = "Tancrède"
      - $salutation = "Bonjour $prenom!"
        - La valeur de « $salutation » est « Bonjour Tancrède! »

Types courants

- **Booléen**
  - Prend soit la valeur « $true » (pour « vrai ») ou la valeur « $false » (pour « faux »)
    - Ex:
      - $monBooleen = $true

Afficher le type d’une variable



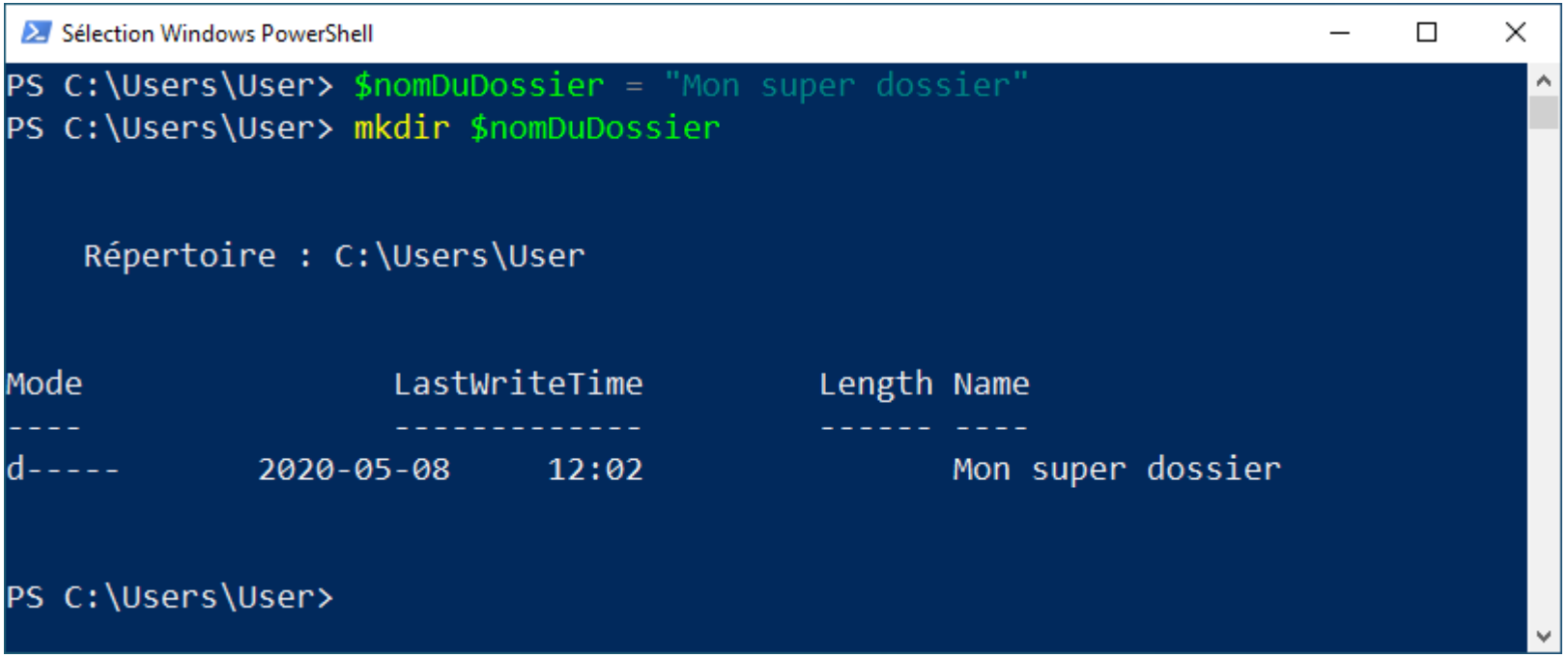
Conversion de types

- **On peut convertir une variable d’un type à l’autre**
  - $maVariable = "42"
    - Cette variable est de type string, même si la chaîne de caractères ne contient que des chiffres
  - $maVariableConvertie = [int]$maVariable
    - Nous avons converti la chaîne de caractères « "42" » vers le nombre entier « 42 »

Conversion de types

- **On peut aussi forcer une variable à toujours être d'un certain type**
  - [int]$maVariable = 42
  - $maVariable = "23"
    - La chaîne de caractères « "23" » est automatiquement convertie vers le nombre entier « 23 »

Utiliser une variable dans une commande



Les constantes

Photo par David Wilson sous licence CC BY 2.0

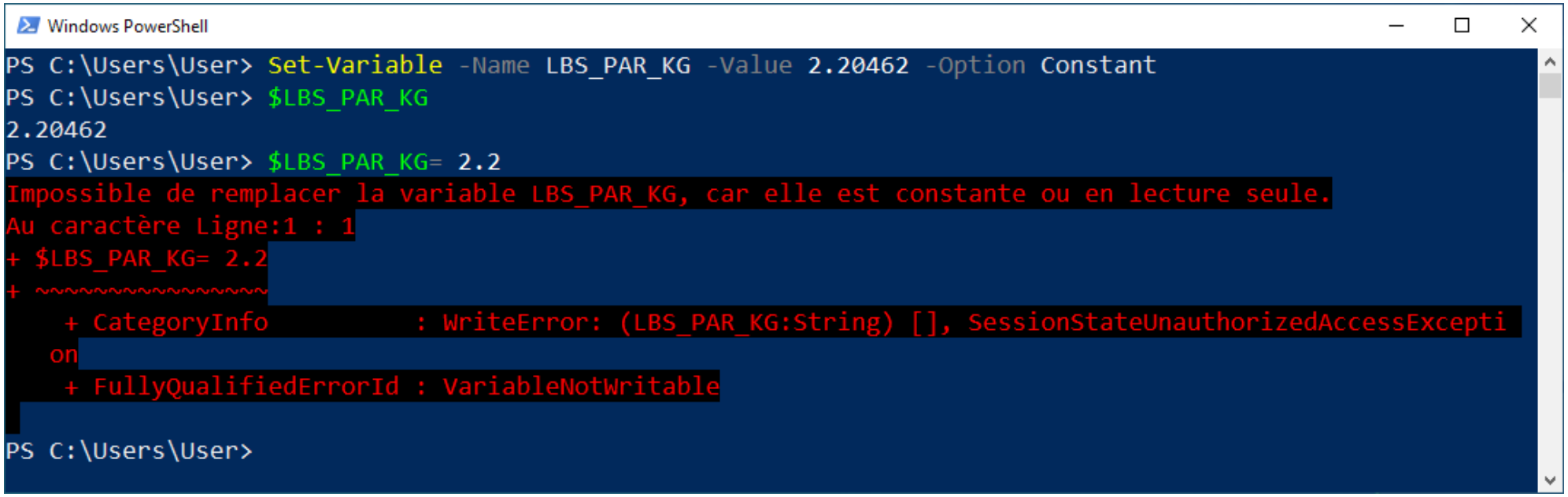
Les constantes

- **Une constante, c'est une variable qui ne change jamais de valeur**
  - Ex: la consante π en mathématiques
- **Utile pour...**
  - Éviter de répéter une valeur qu'on utilise à plusieurs endroits dans le code
  - Rendre le code plus facile à comprendre en évitant d'utiliser des valeurs non documentées (valeurs « *hardcodées* »)

Exemple d’utilisation d’une constante

- **Ligne de code avec valeur « hardcodée »**
  - $poidsEnKg = $poidsEnLbs * 2.20462
- **La même ligne de code avec une constante**
  - $poidsEnKg = $poidsEnLbs * $LBS_PAR_KG
    - Il est de pratique courante d’écrire les constantes toutes en majuscules avec des « _ » entre les mots

Créer une constante



Les commentaires

Les commentaires

- **Les commentaires permettent de documenter le code**

```
1   # Ceci est un commentaire sur une ligne
2
3   <#
4   Ceci
5   est
6   un
7   commentaire
8   sur
9   plusieurs
10  lignes
11  #>
12
```

```
1   # Demander à l'utilisateur d'entrer son prénom
2   $prenom = Read-Host -Prompt 'Entrer votre prénom'
3
4   # Afficher « Bonjour, $prénom »
5   echo "Bonjour, $prenom"
```

Attention: Mettre trop de commentaires peut être considéré une mauvaise pratique.

Votre code devrait être écrit de manière à être facilement compréhensible, par exemple en utilisant des noms de variables significatifs!

Les opérateurs

Photo provenant de JESHOOTS.com

Les opérateurs

- **Les opérateurs permettent d'effectuer des opérations sur des variables**
- **Types d'opérateur:**
  - Arithmétiques
  - D'affectation
  - Unaires
  - De comparaison
  - Logiques
  - De chaînes de caractères

Opérateurs arithmétiques

- **$a + $b :** Addition
- **$a - $b** : Soustraction
- **$a * $b** : Multiplication
- **$a / $b** : Division
- **$a % $b** : Modulo
  - Retourne le reste d'une division, ex: 5 % 3 = 2
  - Exemple d'utilisation: vérifier si un nombre est pair ou impair

Opérateurs d’affectation

- **$a = 3**
- **$a += 3** : incrémente la variable de 3
  - Ex:
    - $a = 5
    - $a += 3 # la valeur de $a est maintenant 8
- **$a -= 3**
- **$a *= 3**
- **$a /= 3**
- **$a %= $b**

Opérateurs unaires

- **$a++** : incrémente $a de 1
- **$a--** : décrémente $a de 1

Opérateurs de comparaison

- **$a -eq $b** : est égal à
  - retourne $true si $a est égal à $b
- **$a -ne $b** : est différent de
- **$a -lt $b** : est plus petit que
- **$a -gt $b** : est plus grand que
- **$a -le $b** : est plus petit ou égal à
- **$a -ge $b** : est plus grand ou égal à

Opérateurs logiques

- **$a -ge 5 -and $a -le 10**
  - $a est plus grand ou égal à 5 **et** $a est plus petit ou égal à 10
- **$a -lt 5 -or $gt 10**
  - $a est plus petit que 5 **ou** est plus grand que 10
  - Il ne s'agit pas d'un « ou exclusif »
- **-not ($a -lt $b)**
  - Opérateur de négation
  - Dans ce cas, équivalent à « $a -ge $b »
  - Peut être combiné avec des -and et des -or

Opérateurs de chaînes de caractères

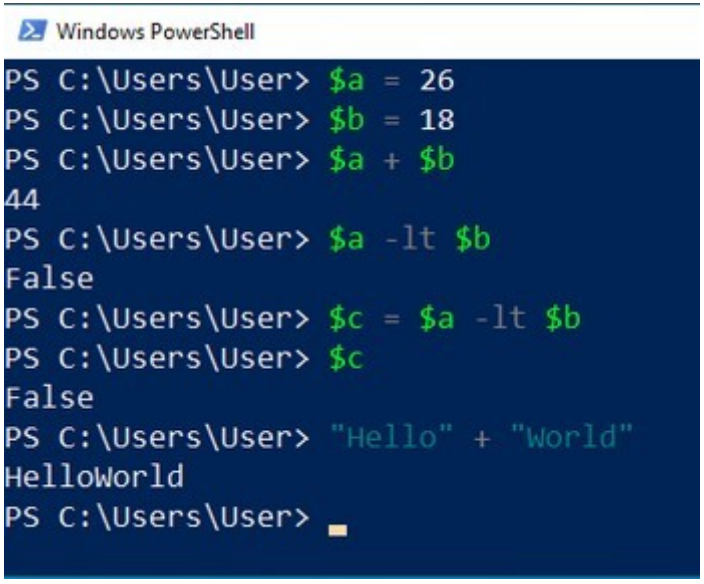
- **$a + $b : concatène $a et $b**
  - Ex:
    - $a = "Mon chat"
    - $b = " est très tannant"
    - $c = $a + $b
    - # La valeur de $c est "Mon chat est très tannant"

Arthur, chat TRÈS tannant

- **Les autres opérateurs de chaînes de caractères seront abordés plus tard dans le cours**

Les expressions

- **Une combinaison de variables, de valeurs et d'opérateurs se nomme une expression**
- **Les expressions peuvent être**
  - tapées directement dans la console pour être évaluées
  - utilisées dans des scripts



Les structures conditionnelles

Photo par Kelly Sikkema sur Unsplash

If / else

```
if (expression) {
  # code à exécuter si l'expression est vraie
} else {
  # code à exécuter si l'expression est fausse
}
```

If / elseif / else

```
if (expression1) {
  # code à exécuter si expression1 est vraie
} elseif (expression2)  {
  # code à exécuter si expression1 est fausse
  # et expression2 est vraie
} else {
  # code à exécuter si les deux expressions sont
  # fausses
}
```

Exemple

```powershell
[int]$nombre = Read-Host 'Entrer un nombre'
if ($nombre -lt 100) {
    echo "$nombre est plus petit que 100"
} elseif ($nombre -lt 1000) {
    echo "$nombre est plus petit que 1000"
} else {
    echo "$nombre est plus grand que 1000"
}
```

Fin de la présentation

Des questions?

Photo par Jon Tyson sur Unsplash