***Лабораторна робота №13***

***«Програмування сценаріїв Windows PowerShell. Керуючі конструкції мови PowerShell: оператори перевірки умов, цикли. Використання масивів»***

***(Термін виконання – 2 тижні!)***

***Змінні PowerShell***

Як ми вже знаємо, імена змінних **PowerShell** завжди починаються зі знака долара (***$***). Змінні **PowerShell** не потрібно попередньо оголошувати або описувати, вони створюються при першому присвоєнні змінній значення. Якщо спробувати звернутися до неіснуючої змінної, то система поверне значення ***$Null***.

***$Null***, ***$True*** та ***$False***, є спеціальними змінними, визначеними в системі. Змінити значення цих змінних не можна.

Призначена для користувача змінна створюється після першого присвоювання їй значення. Наприклад, створимо цілочисельну змінну ***$а***:

***PS С:\> $а=1***

***PS С:\> $а***

***1***

Дати користувачу можливість ввести інформацію з екрану та присвоїти результат введення у певну змінну дозволяє командлет ***Read-Host***.

***PS C:\> $name=Read-Host "Input your name"***

***Input your name: Gleb***

***PS C:\> $name***

***Gleb***

***Керуючі інструкції***

***Інструкція If …ElseIf … Else***

У загальному випадку синтаксис інструкції ***If*** має вигляд

***If (умова1)***

***{Блок\_коду1}***

***[ElseIf (умова2)]***

***{Блок\_коду2}]***

***[Else***

***{Блок\_коду3}]***

При виконанні інструкції ***If*** перевіряється істинність умовного виразу ***умова1***.

Якщо ***умова1*** має значення ***$True***, то виконується ***блок\_коду1***, після чого виконання інструкції ***if*** завершується. Якщо ***умова1*** має значення ***$False***, перевіряється істинність умовного виразу ***умова2***. Якщо ***умова2*** має значення ***$True***, то виконується ***блок\_коду2*** і виконання інструкції ***if*** завершується. Якщо і ***умова1***, і ***умова2*** мають значення ***$False***, то виконується ***блок\_коду3*** і виконання інструкції ***if*** завершується.

Приклад використання інструкції ***if*** в інтерактивному режимі роботи. Спочатку змінній ***$a*** дамо значення 10:

***PS C:\> $a = 10***

Потім порівняємо значення змінної з числом 15:

***PS C: \> If ($a -eq 15) {***

***>> 'Значення $ a дорівнює 15'***

***>>}***

***>> Else { 'Значення $ a не дорівнює 15'}***

***>>***

***Значення $ a не дорівнює 15***

***Цикли While и Do … While***

Найпростіший з циклів PS - цикл ***While***, в якому команди виконуються до тих пір, поки перевіряється умова має значення ***$True***. Інструкція ***While*** має наступний синтаксис:

***While (умова) {блок\_команд}***

Цикл ***Do ... While*** схожий на цикл ***While***, однак умова в ньому перевіряється не до блоку команд, а після: ***Do {блок\_команд} While (умова)***. Наприклад:

***PS C:\> $val=0***

***PS C:\>Do {$val++; $val} While ($val –ne 3)***

***1***

***2***

***3***

***Цикл For***

Звичайно цикл ***For*** застосовується для проходження по масиву і виконання певних дій з кожним із його елементів. Синтаксис інструкції ***For***:

***For (ініціалізація; умова; повторення) {блок\_команд}***

Приклад:

***PS C:\> For ($i=0; $i –lt 3; $i++) {$i }***

***0***

***1***

***2***

***Цикл ForEach***

Інструкція ***ForEach*** дозволяє послідовно перебирати елементи колекцій. Найпростіший тип колекції - масив. Особливість циклу ***ForEach*** полягає в тому, що його синтаксис і виконання залежать від того, де розташована інструкція ***ForEach***: поза конвеєром команд або всередині конвеєра.

***Інструкція ForEach поза конвеєром команд***

В цьому випадку синтаксис циклу ***ForEach*** має вигляд:

***ForEach ($елемент in $колекція) {блок\_команд}***

При виконанні циклу ***ForEach*** автоматично створюється змінна ***$елемент***. Перед кожною ітерацією в циклі цієї змінної присвоюється значення чергового елемента в колекції. У розділі ***блок\_команд*** містяться команди, які виконуються на кожному елементі колекції. Наведений нижче цикл ***ForEach*** відображає значення елементів масиву ***$lettArr***:

***PS C:\> $lettArr = “a”, “b”,”c”***

***PS C:\> ForEach ($lett in $lettArr) {Write-Host $lett}***

***a***

***b***

***c***

Інструкція ***ForEach*** може також використовуватися спільно з командлетами, які повертають колекції елементів. наприклад:

***PS C:\> $ln = 0; ForEach ($f in dir \*.txt) {$ln +=$f.length}***

У прикладі створюється і обнуляється змінна ***$ln***, потім в циклі ***ForEach*** за допомогою командлета ***dir*** формується колекція файлів з розширенням ***txt***, що знаходяться в поточному каталозі. Інструкція ***ForEach*** перебирає всі елементи цієї колекції, на кожному кроці до поточного файлу виконується звернення за допомогою змінної ***$f***. У блоці команд циклу ***ForEach*** до поточного значення змінної ***$ln*** додається значення властивості ***length*** (розмір файлу) змінної ***$f***. В результаті виконання циклу в змінній ***$ln*** буде отримано сумарний розмір файлів в поточному каталозі, які мають розширення ***txt***.

***Інструкція ForEach всередині конвеєра команд***

Якщо інструкція ***ForEach*** з'являється всередині конвеєра команд, то PS використовує псевдонім ***ForEach***, відповідний командлету ***ForEach-Object***. У цьому випадку фактично виконується командлет ***ForEach-Object*** і не потрібна частина інструкції ***($елемент in $колекція)***, так як елементи колекції блоку команд надає попередня команда конвеєра.

Синтаксис інструкції ***ForEach*** всередині конвеєра команд має вигляд:

***команда | ForEach {блок\_команд}***

Розглянутий вище приклад підрахунку сумарного розміру файлів з поточного каталогу для даного варіанту інструкції ***ForEach*** прийме наступний вигляд:

***PS C:\> $ln = 0; dir \*.txt | ForEach { $ln +=$\_.Length}***

У наведеному прикладі спеціальна змінна ***$\_*** використовується для звернення до поточного об'єкту конвеєра і вилучення його властивостей.

***Створення і використання масивів***

Для створення і ініціалізації масиву достатньо привласнити значення його елементів. Значення, що додаються в масив, розділяються комами і відокремлюються від імені масиву символом присвоювання. Наприклад, наступна команда створить масив ***$a*** з трьох елементів:

***PS C:\> $a=1,5,7***

***PS C:\>$a***

***1***

***5***

***7***

Можна створити і ініціалізувати масив, використовуючи оператор діапазону (..). Наприклад, команда:

***PS C:\> $b=10..15***

створює і ініціалізує масив ***$b***, що містить ***6*** значень ***10***, ***11***, ***12***, ***13***, ***14*** і ***15***.

Для створення масиву може використовуватися операція введення значень його елементів з текстового файлу:

***PS C:\> $f = Get-Content c:data\numb.txt –TotalCount 25***

***PS C:\>$f.length***

***25***

У наведеному прикладі результат виконання командлета ***Get-Content*** присвоюється масиву ***$f***. Необов'язковий параметр ***-TotalCount*** обмежує кількість прочитаних елементів величиною ***25***. Властивість об'єкта масив - ***length*** - має значення, що дорівнює кількості елементів масиву, в прикладі воно дорівнює ***25*** (передбачається, що в текстовому файлі ***munb.txt*** принаймні ***25*** рядків).

***Звернення до елементів масиву***

Довжина масиву (кількість елементів) зберігається в свойcтве ***Length***. Для звернення до певного елементу масиву потрібно вказати його індекс в квадратних дужках після імені змінної. Нумерація елементів масиву завжди починається з нуля. В якості індексу можна вказувати і негативні значення, відлік буде вестися з кінця масиву - індекс ***-1*** відповідає останньому елементу масиву.

***Операції з масивами***

За замовчуванням масиви **PowerShell** можуть містити елементи різних типів (цілі 32-х розрядні числа, рядки, дійсні та інші), тобто є поліморфними. Можна створити масив з жорстко заданим типом, що містить елементи тільки одного типу, вказавши потрібний тип в квадратних дужках перед ім'ям змінної. Наприклад, наступна команда створить масив 32-х розрядних цілих чисел:

***PS C:\> [int []] $a = 1,2,3***

Масиви **PowerShell** базуються на .NET-масивах, що мають фіксовану довжину, тому звернення до інших елементів масиву фіксується як помилка. Є спосіб збільшення первісно визначеної довжини масиву. Для цього можна скористатися оператором конкатенації ***+*** або ***+ =***. Наприклад, наступна команда додасть до масиву ***$a*** два нові елементи зі значеннями ***5*** і ***6***:

***PS C:\> $a***

***1***

***2***

***3***

***4***

***PS C:\>$a+=5,6***

***PS C:\>$a***

***1***

***2***

***3***

***4***

***5***

***6***

При виконанні оператора ***+=*** відбувається наступне:

* створюється новий масив, розмір якого достатній для приміщення в нього всіх елементів;
* початкове вміст масиву копіюється в новий масив;
* нові елементи копіюються в кінець нового масиву.

Таким чином, насправді створюється новий масив більшого розміру.

Можна об'єднати два масиви, наприклад ***$b*** і ***$c*** в один за допомогою операції конкатенації ***+***. Наприклад:

***PS C:\> $d=$b+$c***

***Функції у PowerShell***

Функція в **PowerShell** - це блок коду, який має назву і знаходиться в пам'яті до завершення поточного сеансу командної оболонки. Якщо функція визначається без формальних параметрів, то для її завдання досить вказати ключове слово **Function**, потім ім'я функції і список виразів, що складають тіло функції (даний список повинен бути заключений у фігурні дужки). Наприклад, створимо функцію ***MyFunс***:

***PS C:\> Function MyFunc {"Всім привіт!"}***

Для виклику цієї функції достатньо просто ввести її ім’я:

***PS C:\> MyFunc***

***Всім привіт!***

Функція в **PowerShell** має доступ до аргументів, з якими вона була запущена, навіть якщо при визначенні цієї функції не були задані формальні параметри. Всі аргументи, з якими була запущена функція, автоматично зберігаються в змінній ***$Args***. Іншими словами, в змінної ***$Args*** міститься масив, елементами якого є параметри функції. зазначені при її запуску. Дпя прикладу додамо змінну ***$Args*** в нашу функцію ***MyFunc***:

***PS C:\> Function MyFunc {"Привіт, $Args!"}***

Так як змінна ***$Args*** поміщена в рядок в подвійних лапках, то при запуску функції значення цієї змінної буде обчислене (розширене), і результат буде вставлений в рядок. Викличемо функцію ***MyFunc*** з трьома параметрами:

***PS C:\> MyFunc Андрій Сергій Іван***

***Привіт, Андрій Сергій Іван!***

У **PowerShell**, як і в більшості інших мов програмування, при описі функції можна задати список формальних параметрів, значення яких під час виконання функції будуть замінені значеннями фактично переданих аргументів.

Список формальних параметрів вказується в круглих дужках після імені функції. Визначимо, наприклад, функцію ***Subtract*** для знаходження різниці двох своїх аргументів (зменшуваному відповідає параметр ***$From***, від’ємнику - параметр ***$Count***):

***PS C:\> Function Substract($From, $Count) {$From-$Count}***

При виконанні функції ***Subtract*** її формальні параметри будуть замінені фактичними аргументами, які визначаються або за позиції в командному рядку, або по імені.

Наприклад:

***PS C:\> Substract 10 2***

***8***

В цьому випадку відповідність формальних параметрів фактично переданим аргументів визначається по позиції: замість першого параметра ***$From*** підставляється число ***10***, замість другого параметра ***$Count*** підставляється число ***2***.

При вказівці аргументів можна використовувати імена формальних параметрів (порядок вказівки аргументів при цьому стає несуттєвим), наприклад:

***PS C:\> Substract -From 10 -Count 2***

***8***

***PS C:\> Substract -Count 3 -From 5***

***2***

***Сценарії PowerShell***

Сценарії **PowerShell** є текстовими файли з розширенням ***ps1***, в яких записаний код (команди, оператори та інші конструкції) на мові **PowerShell**. На відміну від сценаріїв **WSH** і командних файлів інтерпретатора **cmd.exe**, сценарії **PowerShell** можна писати поетапно, безпосередньо в самій оболонці, переносячи потім готовий код у зовнішній текстовий файл. Такий підхід значно спрощує вивчення мови і налагодження сценаріїв, дозволяючи відразу бачити результат виконання окремих частин сценарію.

сценарії виконуються системою тільки в тому випадку, коли це дозволено поточною політикою виконання. За замовчуванням діє політика **Restricted**, яка повністю забороняє виконання сценаріїв **PowerShell**. Це зроблено з міркувань безпеки, так як в сценаріях може міститися шкідливий код, який може пошкодити систему або несанкціоновано скористатися одними даними.

Перевірити активну політику виконання за допомогою командлету ***Get-ExecutionPolicy***.

***PS C:\Script> Get-ExecutionPolicy***

***Restricted***

Якщо у системі діє більш сувора політика безпеки (***Restricted*** або ***AllSigned***), потрібно встановити політику ***RemoteSigned***, дозволяє виконувати непідписані локальні сценарії:

***PS C:\Script> Set-ExecutionPolicy RemoteSigned***

Даний командлет може не спрацювати, якщо запускати його не від імені адміністратора. Для того, щоб дозволити виконання сценаріїв для поточного користувача, необхідно додати параметр ***–Scope*** зі значенням ***CurrentUser***:

***PS C:\Script> Set-ExecutionPolicy -Scope CurrentUser RemoteSigned***

При запуску сценаріїв **PowerShell** шлях до файлу з кодом потрібно завжди вказувати явно, навіть якщо сценарій знаходиться в поточному каталозі, так як це запобігає можливому несанкціонованому запуску іншої виконуваної програми з аналогічним ім'ям, що знаходиться, наприклад, в системному каталозі. При цьому можна навіть не вказувати розширення ***.ps1***. А можна вказати і повний шлях до файлу.

Розбір і обробка аргументів, переданих в сценарії, проводиться практично так само, як і в функціях (взагалі, сценарій - це фактично функція, яка знаходиться не в оперативній пам'яті, а на диску).

Аргументи вказуються після імені сценарію і розділяються між собою проміжками. Змінна ***$Args*** всередині сценарію містить масив, елементами якого є аргументи функції, зазначені при її запуску. Для прикладу напишемо сценарій ***SumArgs.ps1***. який буде повідомляти кількість параметрів, з якими він запущений, і їх суму.

Файл **SumArgs.ps1**:

***"Кількість аргументів: $($Args.count)"***

***$n=0***

***for($i=0; $i -lt $Args.Count; $i++) { $n+=$Args[$i] }***

***"Сума аргументів: $n"***

***PS C:\Script> .\SumArgs.ps1 4 7 8***

***Кількість аргументів: 3***

***Сума аргументів: 19***

Як бачите, масив ***$Args*** в сценаріях має таке ж значення і обробляється так само, як у функціях.

У сценаріях можна визначати формальні параметри, замість яких під час виконання будуть підставлятися фактичні аргументи, передані в сценарій. У функціях формальні параметри перераховувалися в круглих дужках після імені, тобто поза тілом функції. У сценаріях так вчинити не можна, так як тут все вміст файлу є тілом сценарію, тому формальні параметри задаються за допомогою спеціальної інструкції ***Param***. Ця інструкція повинна бути найпершою командою в файлі, передувати їй можуть тільки порожні рядки і коментарі. Для прикладу напишемо сценарій ***Add.ps1*** з двома формальними параметрами, який буде виводити суму своїх аргументів.

Файл **Add.ps1**:

***Param($x=2,$y=3)***

***$x+$y***

Запустимо отриманий сценарій з аргументами і без них:

***PS C:\Script> .\Add 10 20***

***30***

***PS C:\Script> .\Add***

***5***

Bсe працює, як і очікувалося: якщо не вказано жодних аргументів, то всередині сценарію використовуються значення за замовчуванням.

***Завдання***

1. Написати сценарій **PowerShell**, який знаходить у заданому користувачем каталозі один файл найбільшого розміру та три файли найменшого розміру.
2. Написати сценарій **PowerShell** для перевірки наявності файлів документів (***\*.doc***, ***\*.txt***) у заданому користувачем каталозі. У разі позитивної відповіді -вивесті їх список.
3. Написати сценарій **PowerShell**, що знаходить копії файлів в двох заданих користувачем каталогах. Для кожної такої пари вивести на екран запит про те, яку копію видалити.

***Контрольні питання***

1. Який синтаксис має оператор ***if*** у мові **PowerShell**? Наведіть приклади.
2. Як описати змінну у **PowerShell** та вивести її значення?
3. Які оператори циклів існують у **PowerShell**? Наведіть приклади.
4. Як оголосити масив у **PowerShell**? Які способи проініціалізувати масив існують? Наведіть приклади.
5. У чому полягає особливість функцій у мові **PowerShell**? Яким чином оголосити функції?
6. Які способи передати параметри у функцію у мові **PowerShell**?
7. Що представляє собою сценарії **PowerShell**? Які проблеми можуть виникнути при його запуску?