10. óra

# PowerShell visszatekintés

## Változók

$név = érték;

## Tipizált változók, adattípusok

int, byte, boolean, char, string,datetime

## Objekt orientált

Nem szöveges, mint a UNIX

.NET framework teljes könyvtárrendszere áll mögötte.

## Konstans változók

Set-Variable -name alma -value "jonatán" -option constant

## Aritmetikai műveletek

Nincs külső parancs, a keretrendszer automatikusan el tudja végezni.

.NET keretrendszer minden művelete elérhető.

[math]::pi; [math]::sin(90);[math]::sqrt(4);

## Konverzió

[system.convert]::toint32("123456");

## Segítség, információ

Microsoft Developer Network (Fejlesztői könyvtár, dokumentáció)

Visual Studio 2010 súgója (lokális vagy online verziója is van)

Google

(Könyv)

# Programozás PowerShell-ben (utolsó előadás)

Jövő héttől megint Istenes Zoltán előadásai jönnek. (Hardver)

Microsofttól 500 oldalas PDF netről letölthető. (Magyar nyelven)

## Felülbírált típusmegadás

[int]$a="alma" //Természetesen hibát ad, az alma nem szám. De "5"-öt 5-té alakítja.

## Változóbehelyettesítés

$a="alma";"${a}fa" //="almafa"  
$afa //Hiba, nem lesz almafa.  
"piros$a" //=pirosalma, így működik.

## Escaping (karakter semlegesítés)

Ha $ jelet akarunk, `$ (nem \$, ahogy megszoktuk.)

## RegEx escaping

A megszokott \ jel működik.

## RegEx mintaillesztés

match

## Tömbök

### Megadása

$tömb="alma", "körte", "barack"; //automatikusan felismeri, hogy tömböt akarunk

$tömb=@("alma", "körte", "barack"); //teljes változat

### Indexelés 0-tól, résztömb kiválasztása

echo $tömb[0] //=alma

echo $tömb[1..2] //=körte barack

### Tömbök összefűzése

$tömb1+$tömb2;

### Tömbbe tömb

$tömb=1,2,3  
$tömb[2]=@(3, 4, 5, 6); echo $tömb[2][1] //=3

### Tömb is objektum

echo $tömb.length //=6

### Új elem hozzáadása (a végéhez)

$tömb+="szilva";

### Összes elem kiírása

$tömb //vagy echo $tömb

## Tömbműveletek

### Tartalmazás

1,2,3,4 -contains 3 //=$true

1,2,3,4 -notcontains 1 //=$false

### Elemek kiválasztása tulajdonság alapján

-eq -ne: 1,2,3,4 -ne 3 //=1,2,4

-lt -gt: 1,2,3,4 -lt 3 //= 1,2

-le -ge: egyenlőséget is megenged

-join -split -csplit: (…) A csplit = case split, vagyis a kis/nagy betűkre érzékeny

### Asszociatív tömbök

A tömb indexeit nem számmal határozom meg, hanem szóval, karaktersorozattal.

$at=@{"kulcs"="érték;...} //elemek között fontos a pontosvessző: ;

$at=@{a=4; b=5}

Elem elérése: $at.a vagy $at[a]

Új elem hozzáadása tömb összefűzéssel: $at+=@{c=6}

Kiírás, ahogy megszoktuk: $at vagy echo $at

## .NET framework tömbök: Collections

System.Collections

Ritkán van szükség rájuk, de jól jöhet.

$t = new-object system.collections.arraylist  
$t1=[system.collections.arraylist] (2,3,4)  
$t1.add(5)  
$t1.contains(3) //=$true  
$t1.instert(2,20) //2. elem után lesz a 20 hozzáadva  
$t1.sort()

## Elágazás (if, elseif, else, switch)

-eq -ne -gt -le -ge -not -and -or

-ceq -ieq (case sensitive/insensitive szövegekhez)

-like \*,?,[ab.] karakterek

-match regex

if (kif) {utasítások} elseif (kif) {utasítások} else {utasítások}

if($a -gt 2)  
 {echo nagyobb}  
elseif ($a -eq 2)  
 {echo egyenlő}  
else  
 {echo kisebb}

Switch a többirányú elágazás

switch(Read-Host -prompt "Kérek egy betűt")  
{  
"a" {a}  
"b" {b}  
default {nem a nem b}  
}

## Ciklus (for, foreach)

for($i=0;$i -lt 5; $i++){echo $i}

foreach($i in 1,2,3,4){$i}

## Parancsmegadás folytatása új sorban

` karakter a sor végére

(Több parancs egy sorban pedig ;-vel elválasztva.)

## Foreach-Object szűrő

get-process|foreach-object`  
-begin {echo kezdem}`  
-process {echo $\_.name}` //$\_ az aktuális cső-elem neve  
-end{echo kész}

Röviden: get-process|foreach-object -begin {echo kezdem} -process {echo $\_.name} -end{echo kész}

## Ciklus (while, do until)

while($a -gt 0){$a–} //nem feltétlen fut le legalább egyszer

do{$a++}while($a -lt 10) //legalább egyszer biztosan lefut

do{$a++}until($a -gt 3) //ez is lefut 1x min.

## Függvénydefiníció

function f($a){return $a+1}

f(5) //=6

$lastexitcode az utoljára végrehajtott külső program visszatérési értéke

## Függvény paraméterek

Alapértelmezett adatok megadhatóak (fent példa)

Kötelező paramétert így lehet megkövetelni: mandatory $param vagy $param=$(throw Kérek paramétert!)

Változószámú paraméteres függvény: $args tömb segítségével.

### Klasszikus paraméter megadás

function global:Where-BigProcess(  
[int] $meret = 200MB,  
[String] $property = "VirtualMemorySize",  
[String] $formatString = "Total {2} = {1}"  
){…}

function global:Where-BigProcess([int] $meret = 200MB, [String] $property = "VirtualMemorySize", [String] $formatString = "Total {2} = {1}"){…}

### PARAM blokk

function global:Where-BigProcess {  
PARAM(  
[int] $meret = 200MB,  
[String] $property = "VirtualMemorySize",  
[String] $formatString = "Total {2} = {1}"  
)  
…}

### Paraméter átdás szépsége

alma($név, $ár, $szín)

(…)

(Sorrendet felcserélhetjük, paraméterek nevét rövidíthetjük.)

## Változók láthatósága

-scope kapcsoló

Ha azonos nevű változót definiálunk, akkor alap esetben a lokálisat látjuk: get-variable i -scope 0

De get-variable i -scope 1 segítségével elérhetjük a kintit is.

## printf, sprintf TODO innen folytat (ellenőrzés)

"Az i értéke: {0}" -f $i

"Az i értéke dátum formátumban: {0:D}" -f $i

## Nevesített és normál paraméterek keverése

nevesített: PARAM(…)

normál: Amit nem kezel a PARAM, az bekerül az args tömbbe.

$args.length=$args.count

## Belső változók

$\_ – aktuális csővezeték objektum   
$? – előző parancs eredmény státusza (logikai)  
$home – felhasználó home könyvtára  
$$ – előző parancssor utolsó szava  
$^ – előző parancssor első szava  
$host – aktuális kiszolgáló (nem név)  
$pshome – PS install könyvtár  
$profile – felhasználó profile fájl neve  
$myinvocation futási (…)

## Források

Listázása a Get-PSProvider paranccsal.

## Felülírás, hozzáírás, olvasás

Set-Content, Get-Content, Add-Content

## Objektumok részletesebb kiírása

Hozzá pipe-oljuk valamelyiket ezek közül: out-string, out-csv, out-clixml

## Where-Object

Where-Object{!$\_.PSisContainer}

## Sort-Object

Magyar abc szerint is tud sorba rendezni: -culture kapcsolóval (Lásd: Get-Culture, Set-Culture)

dir|sort

## Select-Object

get-process|select-object processname,id

## Measure-Object

-line -word -sum -max, stb

## Get-Process

$p=Get-Process powershell

$p.kill()

## Start-Job, Get-Job, Stop-Job

Távoli gépen is.

## Get-Credential

Hitelesítés

## Snapin

Olyan modul, ami bináris formában van megírva

Get-PSSnapin //listázás

Add-PSSnapin //telepítés

Moduldefiníciós lehetőség, forráskódként is lehet.

## WMI

Windows Management Instrumentation (Infrastructure)

## TODO

–like és–match operatorpéldák

PS> gci -r | where-object { $\_.name -match "\.x[ml][ls]" }

... # reg. kif, azXML, XLS, XMS, XLL fájlok.

PS> get-process | where-object { $\_.name -match "ss$" }

... # összessystem service processz.

PS> $p = Get-Content planes.txt; `

$p -match "a[ie]ro?plane"

... # sorok az airplane, aeroplane, szavakkal.

PS> $p -like "\*plane\*"

... # sorokban a plane bent van.