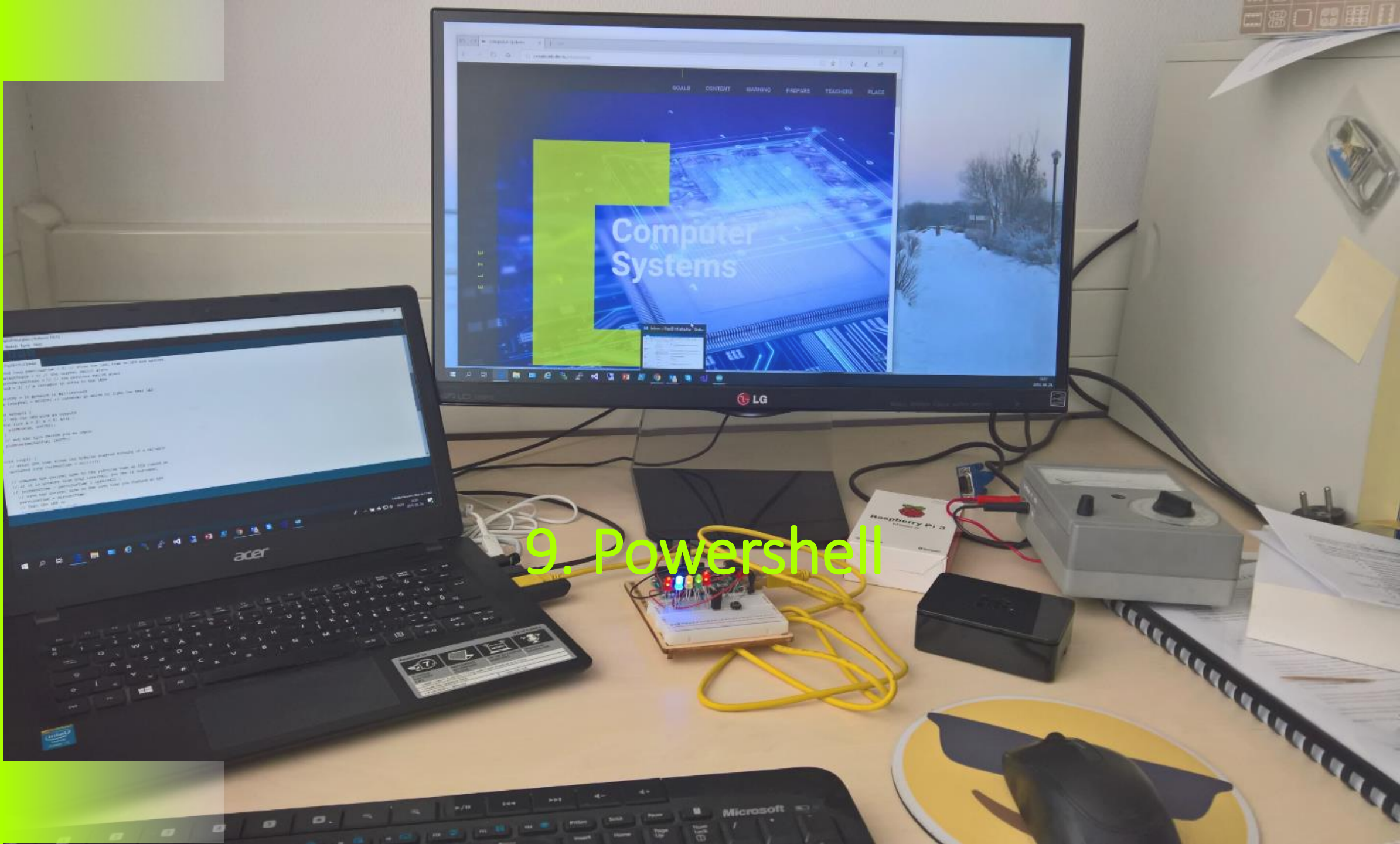
ELTE INFORMATIKA



9. Powershell

Visszatekintés

- Számítógépek, számábrázolás, kódolás,felépítés, fájlrendszerek
- Alapvető parancsok, folyamatok előtérben, háttérben
- I/O átirányítás, szűrők,reguláris kifejezések
- Változó, parancs behelyettesítés,aritmetikai, logikai kifejezések
- Script vezérlési szerkezetek,Sed,AWK
- Batch, WSH
- PS áttekintés

Mi jön ma?

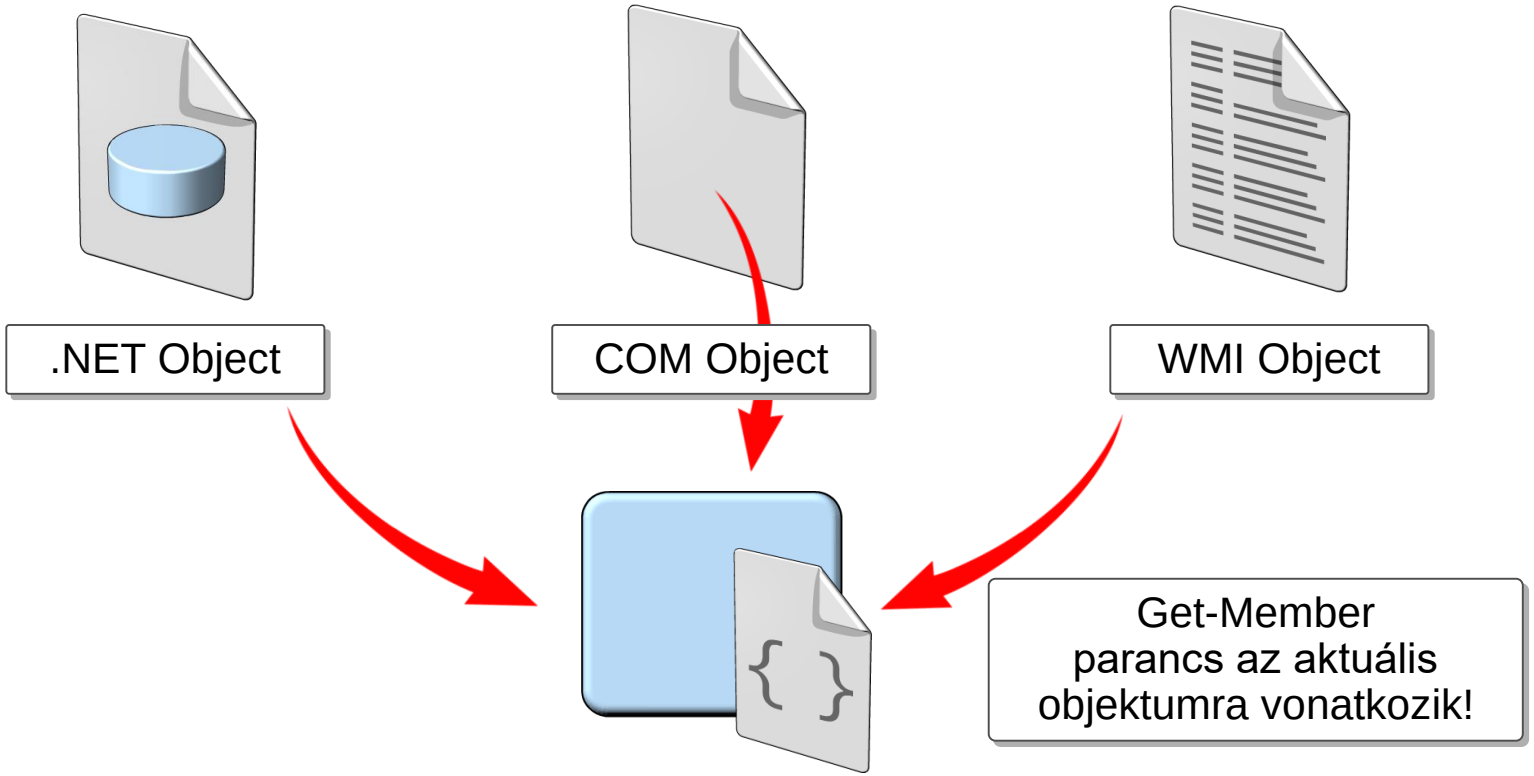
- PowerShell alapok
- PowerShell nyelvi elemek
- …

A PowerShell objektum orientált

- get-date – eredmény: 20xx. november y…
- „Cső, cső hátán”: get-date|get-member
  - A get-date objektum a get-member bemenetére kerül majd az objektum mezőit kapjuk.
  - -inputobject paraméter sok cmdlet-nél él
  - „fradi” | get-member
- get-date|get-member –membertype method
  - get-date metódusait kapjuk meg, hasonlóan property-t, tulajdonságokat is lekérhetünk.
- (get-date).month, day, …ticks

NET, COM, WMI Objektumok mint könyvtárak

- PowerShell a .NET-et nativ módon használja. A COM, WMI objektumokat is közvetlenül elér.



Alap PowerShell parancsok

- alias , kiírja a definiált rövidítéseket
  - Unix-hoz hasonló parancsok
- gcm – Get-Command, kiírja parancsokat
- echo – Write-Output, képernyőre írás, egyszerű, pipe használat
  - Write-Host – [Console]::WriteLine, előtér, háttérszín állítás
- Get-Help – rövid leírás parancsokról
  - Update-Help – letölti és installálja a helpet!
  - Get-Help –full Write-Host
  - set-alias gh get-help
- ps – Get-Process, futó processzek kiírása
  - Sleep – Start-Sleep, várakozás

Drive vs. Más adatforrás

- dir, ls – Get-ChildItem, könyvtár tartalom
  - Get-ChildItem c:\users\teszt*.* -include *.c,*.cpp # c,cpp kiterjesztés csak
- Get-PsDrive – Powershell adatforrások
- Cd könyvtár váltás – Set-Location parancs
  - Cd c:
  - Cd hklm:
  - Cd alias:
    - Dir- kilistáztatja az aktuális drive, adatforrás tartalmát
- Get-Location – pwd alias parancsa

Fontosabb fájl kezelő parancsok

- New-Item – fájl vagy könyvtár létrehozása
  - New-Item –itemtype directory almadir #mkdir dos parancs megfelelője!
- Copy-Item forrás cél [–recurse] # másolás (Alias: cp)
- Remove-Item – fájl, könyvtár törlése (Alias: rm, rmdir, del,…)
- Move-Item – fájl, könyvtár mozgatás (Alias: mv)
- Rename-Item – átnevezés (Alias: ren)
- Get-Item – fájl, könyvtár, reg.kulcs visszaadás
  - Get-item $(c:\users).LastWriteTime, Get-item hkcu:\software |get-member
- Test-Path fájl vagy könyvtár vagy reg.kulcs, #létezik-e?

PowerShell parancsok felépítése, paraméterek

- PowerShell parancs felépítés: Ige-főnév
  - PL: Get-Command
- Paraméterek megadása jellemzően: -név érték
  - Érték lehet: szám, szöveg, dátum
  - Pl: Get-Command –Verb write
- History – F7 előző parancsok
  - felfelenyíl, előző parancs
- Profile:Dokumentumok\WindowsPowerShell könyvtárban:
  - Microsoft.PowerShell_profile.ps1
  - profile.ps1 - Ezt csak az ISE hajtja végre!

PowerShell változók

- $név=érték, kötelező a $ jel a definiáláskor is
  - Pl: $f=„fradi”; echo $f
- egy sorba több parancs írható, ; az elválasztó
- Támogatott fontosabb alap típusok:

Adattípus	Értelmezése	Példa
[int]	Egész szám(32bit)	**-273, -1, 0, 10, 42**
[byte]	8-bit, bájt	**0, 1, …, 254, 255**
[boolean]	Logikai	**$false, $true**
[char]	Karakter(16 bit uni.)	**a, b, c, 1, 2, 3, !, #**
[string]	Szöveg	**“FTC”**
[single]	32 bites valós	**2.3e-1, 3.1415,…**
[datetime]	Idő	**April 1, 2008**

PowerShell változók használata

- Magunk is megadhatjuk (típuskényszerítés):
  - [int] $d=6.2e-4; echo $d # 0, $d egész lesz
  - $s= [string] 65; echo $s # 65 szövegként
  - $s1=[string] [char] 65; echo $s1 # A
  - $i=[int] ”65”; echo $i # 65 szövegként
- Ha nem jelölünk semmit, az értelmező eldönti a típusát
  - $d=6.2e-4; echo $d # 0,00062, valós lesz

PowerShell változók definiálása parancs segítségével

- Set-Variable –Name alma –value „jonatán” –option constant
  - Konstans definiálás
  - Egy leírás adható a –description paraméterrel
  - Get-Variable alma
- Clear-Variable alma # alma létezik, csak tartalma nincs.
- Remove-Variable alma # alma nem létezik

Változók láthatósága I.

- Egy környezetben definiált változókat, a környezetében használhatjuk, az ebből származó függvény, script látja!
  - Ha azonos nevűt definiálunk egy scriptben, függvényben, akkor alapból a lokálisat látjuk
- Ezen lehet segíteni a get-variable –scope paraméterrel.
  - Get-variable alma –scope 0 # aktuális környezet
  - Get-variable alma –scope 1 # szülő környezet
  - Get-variable alma –scope 2 # nagyszülő környezet
  - Stb.

Változók láthatósága II.

- Általános változó definíciós forma:
  - $[scope:]név vagy ${név}
  - Ha a scope elmarad, aktuális helyen (script, függvény) használható lesz a változó.
  - A scope lehet, global, local, private, script.
    - $global:x=5 # globális lesz az x változó, mindenhol látható
    - $script:y=6 # a teljes scriptben használható lesz
    - $local:z=„MTK” # lokálisan és a gyerek blokkokban látható
    - dir $env:ProgramFiles # az env drive eleme
      - A környezeti változók érhetők el ezen keresztül!
    - $env:Path += „;d:\tmp”

Aritmetikai műveletek PowerShell-ben

- +,-,*,/,% (maradék)- alapműveletek
  - Nem kell külön parancsot kiadni, mint pl. az expr!
  - $a= 32*3; echo $a # 96
  - $a=„alma”; $f=„fa”; $c=$a + $f; echo $c #almafa
  - $a= „125” + „2”; echo $a # 1252!
  - $a= 12 + „4”; echo $a # 16
    - automatikusan konvertálja a „4”-et
- Értékadások: =, +=, -=, *=, /=, %=
- Post növelés, csökkenés: $a++, $b—
- Bitműveletek: -band,-bor,-bxor,-bnot,-shl,-shr

Még több művelet

- A PowerShell mögött a .NET FrameWork áll.
  - Az összes típus, double, decimal stb. elérhető
  - Nem csak alaptípusok
  - Példa: [System.IO.DirectoryInfo]$home=Get-Item D:\home
- Bármely statikus tulajdonság, metódus elérés operátora a ::
  - [DateTime]::Now # aktuális dátum
- Teljes Math osztály is rendelkezésre áll
  - [math]::pi
  - [math]::sin(2), Stb.
- Konverzió: [system.convert]::toint32(„32”)
- Stb. , ….Net FrameWork teljes könyvtár használat

PowerShell változók összefoglalás

- $csapat=„Fradi”
- Automatikus típusmegadás, de felülbírálható
  - [int]$a=„alma” # ez hibát ad persze
- Minden alapművelet rendelkezésre áll! + .NET
- Érdekesség: $b=$csapat*5 #ez ok,”Fradi ötször”
  - $c=5*$csapat #hiba!!, „Fradi” nem lesz egész!!!
- Adatmegadás paranccsal
  - Set-Variable –name a –value „körte” –option constant
- Szöveges parancs végrehajtás operátora: &
  - $dir=„dir”; &$dir

Változó behelyettesítés

- $a=„alma”
- „$afa” # eredmény üres
- „${a}fa” # eredmény: almafa
- „piros$a” #pirosalma
- A $ karakter semlegesítése: `
  - „`$a változó értéke: $a”
  - Reguláris kifejezésben a \ karakter használandó semlegesítésre!
- Parancs behelyettesítés forma külön nincs!
  - $könyvtárlista=dir # Nincs szükség a `dir` formára!

Szövegek, behelyettesítés

- A „” között lévő szövegben lévő változó behelyettesítésre kerül!
- AZ ,’ közötti nem: ,Ez nincs behelyettesítve: $a’
- Hasonlóan Unix-hoz, lehet: ”echo ’$i vége’”
  - Itt (is) $i behelyettesítésre kerül!!!!
- Powershellben nincs input átirányítás (<,<<)
- Van helyette többsoros szöveg: @” …több sor is lehet… ”@
  - Közte a változók behelyettesítésre kerülnek!
  - @’ …. Több sor is lehet…. ’@ # nincs változó behelyettesítés

PowerShell tömbök I.

- Változók gyakori elnevezése: skalár, egy adatot tartalmaz, pl: $adat=„alma”
- Több adatot tartalmazó „skalár”: tömb
- Definiálás: $tömb="alma","körte","barack”
  - Teljes változat: $tömb=@(”alma”,”körte”,”barack”)
  - Elemek elérése a 0 indextől!
    - echo $tömb[1] # körte
    - echo $tömb[1..2] # körte barack
  - Egy elem nem csak egyszerű (skalár) lehet, hanem tömb is: $tömb[2]=@(2,3,4); echo $tömb[2][1] # 3

PowerShell tömbök II.

- A tömb is valójában objektum. A tömbök hossza a Length tulajdonsággal érhető el.
  - echo $tömb.Length
- Új elem hozzáadása: $tömb1+=6;
- Összes tömbelem kiírása: $tömb (azonos az echo $tömb utasítással)
- Tömböket összefűzhetünk: + jel segítségével
  - $tömb1=2,3,4,5
  - $tömb+=$tömb1
  - echo $tömb[3] # 2

PowerShell tömb műveletek

- -contains : Tartalmazás (-notcontains)
  - 1,2,3,4 –contains 3 # True
- -eq, -ne Eredmény az összes elem ami egyenlő, (nem egyenlő) adott értékkel
  - 1,2,3,4 –ne 3 # 1,2,4
- -lt, -gt Eredmény az összes elem ami kisebb, (nagyobb) adott értéknél
  - 1,2,3,4 –lt 3 # 12
- -le, -ge Kisebb, nagyobb vagy egyenlő
- -join, -split, -csplit (case split, kis-nagybetű)
- Stb.

PowerShell asszociatív tömbök

- $atömb=@{„kulcs”=„érték”; …}
  - $at=@{a=4;b=5} # Elemek között a ; !!!!!!
- Elem elérés: $at[a] vagy $at.a
- Elem értékadás: $at[a]=10
- Új elem hozzáadás: $at+=@{c=11}
- Asszociatív tömb kiírása: $at

```
PS C:\home\ps> $at
Name                           Value
----                           -----
a                              10
b                              6
c                              11
```

.NET FrameWork tömbök

- System.Collections a különböző adatszerkezetek névtere:
  - $t=new-object system.collections.arraylist
  - $t1= [system.collections.arraylist] (2,3,4)
  - $t1.add(10)
  - $t1.contains(3) # igaz
  - $t1.insert(2,20) # 3 után lesz a 20!
  - $t1.sort()
  - Stb.

Elágazás PowerShellben

- Összehasonlító operátorok, mint a tömböknél.
  - -eq, -ne,-gt,-lt, -le, -ge
  - -not, -and, -or,-xor logikai tagadás, és, vagy
  - Szövegnél: -ceq, Kis, nagybetű különböző, -ieq nem különböző,
  - -like *,?,[ab.] karakterek, -match reg. kif. használat
- If utasítás:
  - if (kif) {utasítás} [elseif (kif1)] else {utasítás}

```
$a=3
if ($a -gt 2)
 { Write-Host "A" $a "nagyobb mint 2." }
else
 { Write-Host "Nem nagyobb mint 2."}
```

Többirányú elágazás

- switch utasítás - .net nyelvekhez hasonló
  - alágakba nem kell break

```
# switch példa
#
# Beolvasás utasítás: read-host
$a=Read-Host -prompt "Írja be a kedvenc gyümölcsét "
switch ($a)
{
  "alma"          { "a értéke: "+$a }  # write-output  röv.
  "barack"        { "a értéke: "+$a }
  "szőlő"         { "a értéke: "+$a }
  "szilva"        { "a értéke: "+$a }
  "körte"         { "a értéke: "+$a }
  default         { "a ismeretlen számomra: "+$a}
}
```

PowerShell Ciklusok I

- for – gyakorlatilag mint C,...stb –ben
  - Mindig kell a ciklusmag köré: {}
  - Pl: for($i=0;$i –lt 5;$i++) {echo $i}
- foreach($i in tömb) {ciklusmag}

```
#
$t=2,3,4,5
foreach($i in $t)
{
 write-host $i
}
```

PowerShell Ciklusok II.

- foreach – Foreach-Object, szűrő
  - Egy sorba több parancs: ;
  - Több sorba egy parancs: `
  - AWK-hoz hasonló begin, end blokk

```
# egy sor folytatása új sorban: ` karakter, fontos!!!
#
get-process|foreach-object `
 -begin { Write-Host "Elkezdtem a Get-Process feldolgozást!" } `
 -process { write-host $_.name -foreground green } `
 -end { Write-Host "Befejeztem a Get-Process feldolgozást!" }
```

PowerShell Ciklusok III.

- While ciklus: mint C-ben, do…while kif.– amíg igaz
- Until ciklus: do …until kif.; Amíg a kifejezés hamis!

```
$a=5
while ($a -gt 0)
{
 write-host $a
 $a--
}
```

```
do
{
 "Egyszer belépek a ciklusba biztosan!"
 write-host $a
 $a--
} while ($a -gt 0)
```

```
$a=0
do {
 "Egyszer belépek a ciklusba biztosan!"
 write-host $a
 $a++
} until ($a -gt 3)
```

Script szerkezet – Foreach minta

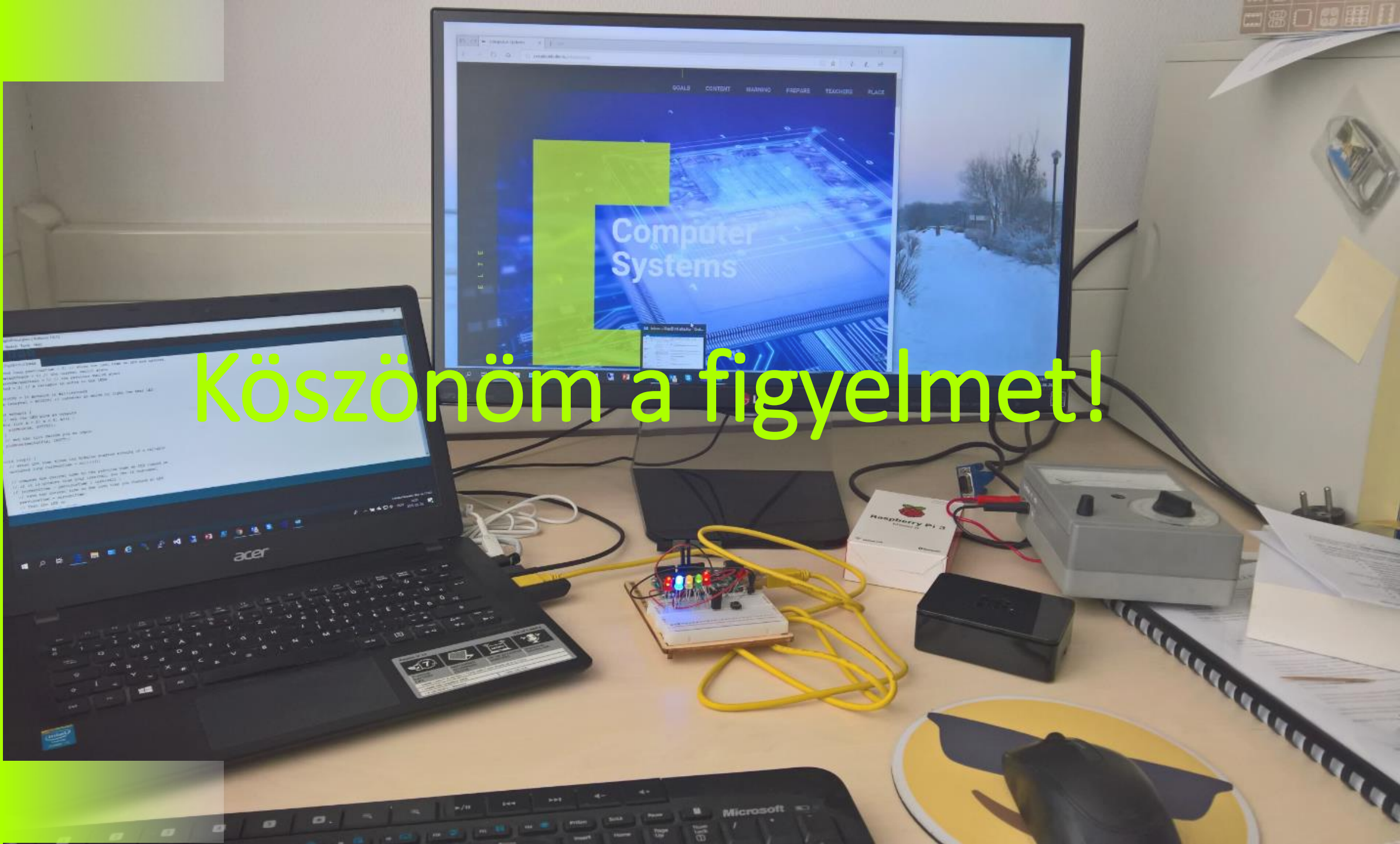
- Nem csak a foreach blokk állhat 3 blokkból!
- Egy script is három blokkot tartalmazhat:
  - Begin { utasítások} # A script elején egyszer végrehajtódik
  - Process { utasítások } # A script törzse, minden „csöves” objektumra végrehajtódik!
  - End { utasítások } # A script végén egyszer végrehajtódik
- A foreach utasítástól függetlenül a „csöves” kapcsolaton keresztül adatokat feldolgozó scriptek gyakori formája!
  - Példa: csoparam.ps1

Példa: csoparam.ps1

```
BEGIN {
 foreach( $i in $Args )
        { Write-Host "Begin blokk a paraméterek $i" -F red }
  Write-Host "Begin blokkban Pipeline: $_" -Fore red
}
PROCESS {
  foreach( $i in $Args )
        { Write-Host "Process blokk a paraméterek: $i" -F white }
  Write-Host "Process blokkban Pipeline: $_" -F white
  $_.GetType().Name # Objektum típusának neve
}
END  {
  foreach( $i in $Args )
        { Write-Host "End blokk a paraméterek: $i" -F green }
  Write-Host "End blokkban Pipeline: $_" -F green
}
```

```
PS C:\d\home\ps> 34,35|.\csoparam.ps1 3 4
Begin blokkban a paraméterek 3
Begin blokkban a paraméterek 4
Begin blokkban a paraméterek 5
Begin blokkban Pipeline:
Process blokkban a paraméterek: 3
Process blokkban a paraméterek: 4
Process blokkban a paraméterek: 5
Process blokkban Pipeline: 34
Int32
Process blokkban a paraméterek: 3
Process blokkban a paraméterek: 4
Process blokkban a paraméterek: 5
Process blokkban Pipeline: 35
Int32
End blokkban a paraméterek: 3
End blokkban a paraméterek: 4
End blokkban a paraméterek: 5
End blokkban Pipeline: 35

PS C:\d\home\ps>
```



ELTE INFORMATIKA