validateset(„y“,“Y“,“n“,“N“)$response = Read-host „is it ok? (y/n)

# validace vstupu

validateLenght(5,100) $cislo = Read-host „zadej číslo od 5 do 100“

[ValidateRange(1000,100000)]$principle = [int32] (Read-host "Principle amount you are looking for? 1000-100000")  
[ValidateRange(2,5)]$time = [int32] (Read-host "Tenure?  2-5 years")

-lt less

-gt greater

Switch ($num)

{

10 {“tvoje cislo je 10”}

100 {“tvoje cislo je 100”}

1000 {“tvoje cislo je 1000”}

default {“tvoje číslo není ani 10,100,100}

}

# Array

$a = 10,20

$a = @(10,20)

$b = @(30,40)

c$ = $a + $b

c$.lenght

c$.Contains(10)

c$.Indexof(30)

c$.isfixedsize – má zafixovanou velikost (c$.Add (50) – nebude fungovat – protože je fixed size)

řešení tohohle je vytvořit arraylist

c$ = New-Object System.collections.arraylist

* Pak bude fungovat c$.add(50)
* c$.AddRange(60,70) – přidat více

# Hash table

$simple\_hash\_table =@{

„number one“ = „1“

„number two“ = „2“

„number three“ = „3“

}

$simple\_hash\_table.add(“number four” , “4”)

$simple\_hash\_table.remove(“number four”)

$simple\_hash\_table.count - počet

$simple\_hash\_table.Keys – vyhodí jen první sloupec

$simple\_hash\_table.Values – vyhodí druhý sloupec

$simple\_hash\_table = [ordered]@{

„number one“ = „1“

„number two“ = „2“

„number three“ = „3“

} – to srovná podle klíče (1. sloupce podle abecedy)

$simple\_hash\_table[“number three”] – vyvolá hodnotu

# While loop

$i = 1

While ($i -le 10){

Echo $i

$i++

}

for($i=0; $i – le 10; $i++){

echo $i

}

$arr = @(“Student1”,”student 2”,”student3”)

For ($i=0; $i – le $arr.Count ; $i++){echo $arr[i] }

* provede echo pro každý člen v array

# Funkce

funtion Say-Hello {

write-output „Hello“

}

Say-Hello \* tim jí zavolám

Function Add-Numbers {

$total = $args[0] + $args[1]

Write-output „Total: $total“

}

Add-Numbers 10 200 <#to dodá za ty čísla

Můžu dodat i proměnné a array #>

<#Můžu dodat i proměnné a array #>

Function Say-Hello ($names){

Foreach ($name in $names){

Echo “Hii All! This is $name“

}

}

# Funkce s parametry

Function My-introduction {

Param

(

[string]$Name,

[string]$profession,

#[string]$profession = “Student”, - pokud nezadám parameter, bude tohle jako výchozí parametr

[int]$age

)

Write-output “Hello, My name is $Name , Iam $age year old $profession“

}

<# v tuhle chvíli můžu zadat věci jako:

My-introduction -name “Simon” – profession “IT admin” -age 23

Navíc mi powershell nabídne pomoc, když zadám funkci a “-“ ukáže mi výpis definovaných parametrů pro funkci”

# Error handling

\*script\* -ErrorAction Stop - zastaví

\*script\* -ErrorAction silentlycontinue – pokračuje a nezobrazí chybu

\*script\* -ErrorAction continue – ukáže chybu ale pokračuje

\*script\* -ErrorAction Inquire – zeptá se usera

$errorActionPreference = ‘stop’ – nastaví preference pro celý skript

\*script\* -ErrorVariable „error\_var“ – uložíé jako proměnnou, kterou můžu použít na logy atd…

Try {

\*script\* kde může nastat error

} catch {

\*script, který proběhne, pokud v try bude nějaká chyba

} finally {

\*script\* který proběhne ať **try** a **catch** proběhne dobře, nebo ne

}

Throw „můj vlastní error“

\*catch je možné směřovat ke specifické chybě

Př:

Try {

Get-content $file

} catch [System.IO.FileNotFoundException]{

Echo “file not found at $file

}

# Užitečné

\*command\* -whatif #neudělá command, ale řekne, co tím udělá

Po rouře můžu udělat další řádek a bude se to pořád považovat za jeden command

Get-content \* -tail 2 (přečte jen první 2 řádky)

Get-content \* [2] (přečte jen druhý řádek)

Get-content \* [2..5] (přečte druhý až pátý řádek)

Out-file \*path\* -apend (připojí text do existujícího souboru)

# Formátování

\*info\* - property name, fullname … poskládám si output, jak potřebuju

\*info\* | convertto-html – poperty name, fullname | out-file \*cesta\*

Dokonce jde do toho html přidat soubor se styly

\*info\* | convertto-html – poperty name, fullname **-cssuri mystyle.css** | out-file \*cesta\*

Nebo to css dát do hlavičky (jako proměnnou string s css) a pak

\*info\* | convertto-html – poperty name, fullname **-head $head**| out-file \*cesta\*

Convertto- html, csv, json, xml

# Class and methods

Class je funkce, jak vytvořit předem definovaný objekt

Class Student {

[string]$name=’NA’

[int32]$age=’0’

}

$stud = New-Object Student

$stud.age = 100

$stud.name = Ajan

# Plánovač PS skritpu

Run powershell script – vytvoř baťák

Powershell.exe c:\script\do.ps1

# Jak zabezpečit skripty

Certifikátem

Set-ExecutionPolicy -ExecutionPolicy AllSigned

Získání přes new-selfsignedCertificate

$cert =(dir cer:currentuser\my\ -CodeSigningCert)

Set-autenticodeSignature ‘cesta k ceritifikátu’

Ještě je potřeba přidat si přes mmc do důvěryhodných certifikátů

Při změně skriptu, je potřeba znovu podepsat, jinak bude opět hlásit error

# PS snipplets

V PS\_ISE – pravym klik a start snipplets – základní operace tam jsou už (do-while…) stačí jen přepsat co potřebuji syntaxe je správná

Můžu si udělat vlastní:

New-IseSnippet – title “název” -description “popisek” -text @’

Cely kod formátovaná jako string’

Ulo6en0 to je v c:\users\\*\Documents\WindowsPowershell\Snippets

# Web Scrapping – data mining

$webrequest = invoke-WebRequest -Uri [www.seznam.cz](http://www.seznam.cz)

$status\_code = $webrequest.StatusCode

$status\_desc = $webrequest.StatusDescription

$webrequest.Link | select href | out-file \*file\*

$webrequest.ParsedHtml.GetElementsbytagName(„TABLE“)[0].outerHTML

Write-verbose „Status code returned : $status\_code“ -Verbose

# Invoke-expression

Invoke-expression $var

Provede příkaz, který je jako string – to znamená, že můžu načíst texťák a provést všechno co je v něm jako command, nebo github script přímo z webu

# EVENTlog

Eventlog jde nastavit tak, že pokud se nějaký event stane, tak udělá akci

Jde zapisovat do eventlogu, nebo udělat úplně nový log

New-eventlog -logname ‘Application’ -source “My script”

Write-eventlog -logname ‘Application’ -source “My script” -entrytype error -EventID 1 -message “This is a test message”

Úplně nový log

New-EventLog -LogName "MyApp\_Log" -Source "MyApp" -MessageResourceFile "C:\MyApp\_Log.dll"

Write-EventLog –LogName 'MyApp\_Log' –Source "MyApp" –EntryType Error –EventID 1 –Message "This is a test message"