```powershell
# Plik: 1_Demo_Podstawy.ps1

'Jestem Obiektem!'.Replace('Obiektem','Wspaniały')

Get-Date | select DayOfWeek, Year

Get-Service |
    where { $_.Status -eq 'Running' } |
    group { $_.Name.Substring(0,2) } |
    where { $_.Count -ge 2 }

Select-String -Pattern Add-Type -Path *.ps1
Select-String -Pattern Add-Type -Path *.ps1 | Get-Member

# Dostaję coś więcej niż to, co widzę...

Select-String -Pattern Add-Type -Path *.ps1 |
    select Line, LineNumber, Path

ls | where Extension -eq .txt |
    Rename-Item -NewName {
        $_ -replace 'Plik(Testowy)(?=[3-5])',
            'MojPlik$1'
    } -WhatIf


# Plik 2_Demo_Wbudowane.ps1
# uwaga: część składni wymaga PowerShella v3
# w przypadku poleceń where/ foreach w wersji drugiej:
# where { $_.Właściwość -operator 'Wartość' }

[AppDomain]::CurrentDomain.GetAssemblies() |
    where FullName -Match ^System\.Management\.Automation |
    foreach Location | clip

Import-Module .\DemoPowerShell.dll -Verbose

echo Alfa, Beta, Gamma | Get-MyObject

$zmienna = ls | Get-MyObject
$zmienna | where Index -gt 4

$zmienna[1].Value = Get-Date
$zmienna

New-Object DemoPowerShell.MyObject -Property @{
    Index = 3
    Value = 'Test'
}

Add-Type -Path .\WinSCP.dll -PassThru |
    where IsPublic

Add-Type -TypeDefinition @"
using System;

namespace Targi
{
    public enum Satysfakcja
    {
        Super = 1,
        Ujdzie,
```

```
        SrednioNaJeza,
        Kiepsko,
        Beznadziejnie = 0,
        PiekloNaZiemi = 666
    }
}
"@

[Enum]::GetValues(
    [Targi.Satysfakcja]
) | foreach {
    "{0,3} : {1}" -f @(
        $_.value__ ,
        $_
    )
}

# P/Invoke? Czemu nie!?
$signature = @'
[DllImport("kernel32.dll")]
public static extern uint GetPrivateProfileString(
    string lpAppName,
    string lpKeyName,
    string lpDefault,
    StringBuilder lpReturnedString,
    uint nSize,
    string lpFileName
);
'@
Add-Type -MemberDefinition $signature <#
#> -Name Win32Utils <#
#> -Namespace PInvokes <#
#> -Using System.Text

[PInvokes.Win32Utils]::GetPrivateProfileString

function Get-IniData {
    param (
        $Path,
        $Category,
        $Key
    )

    $Builder = New-Object Text.StringBuilder
    [PInvokes.Win32Utils]::GetPrivateProfileString(
        $Category,
        $Key,
        "",
        $Builder,
        $Builder.Capacity,
        $Path
    ) | Out-Null
    $Builder.ToString()
}

Get-IniData -Path C:\Windows\system.ini -Category Drivers -Key wave
Select-String -Path C:\Windows\system.ini -Pattern mmdrv\.dll -Context 1


[DateTime]::ParseExact(
    '9 Grudnia 2012',
    'd MMMM yyyy',
```

```powershell
    [cultureinfo]'pl-PL'
)

# Nietypowy format daty w nazwie pliku - żaden problem!
# PowerShell sam sobie nie poradzi... ;)
ls Plik-* | sort BaseName

# Ale zawsze możemy mu pomóc!
ls Plik-* | sort {
    $ParsedDate = [DateTime]::ParseExact(
        $_.BaseName,
        'Plik-dd~MM~yyyy',
        $null
    )
    Write-Host "File Name: " -NoNewline
    Write-Host ($_.Name) -NoNewline -ForegroundColor Cyan
    Write-Host " Date : " -NoNewline
    Write-Host $ParsedDate.ToString("dd MMMM yyyy") -ForegroundColor Yellow
    $ParsedDate
}
```

```
// Moduł z poleceniem/ klasą do skompilowania
// Gotowy plik wczytujemy przez Import-Module NazwaPliku.dll

using System;
using System.Management.Automation;

namespace DemoPowerShell
{
    // Tworzymy klasę, którą zwróci nasze "dzieło"... :)

    public class MyObject
    {
        public string Value
        {
            get;
            set;
        }
        public int Index
        {
            get;
            set;
        }
        public MyObject(string value, int index)
        {
            this.Index = index;
            this.Value = value;
        }
        public MyObject()
        {
            this.Index = 0;
            this.Value = "";
        }

    }

    // Nasz cmdlet...

    [Cmdlet(VerbsCommon.Get, "MyObject")]
    [OutputType("DemoPowerShell.MyObject")]
    public class MyCmdlet : PSCmdlet
```

```
    {
        [Parameter(
            Position = 0,
            Mandatory = true,
            ValueFromPipeline = true
        )]
        public string Bar
        {
            get { return bar; }
            set { bar = value; }
        }
        private string bar;

        private int count;

        protected override void BeginProcessing()
        {
            count = 0;
        }


        protected override void ProcessRecord()
        {
            this.WriteObject(
                new MyObject(bar, count)
            );
            count++;
        }
    }
}

# Plik Demo_WPF_XAML.ps1
# Wracamy do PowerShella...

Add-Type -AssemblyName PresentationCore
Add-Type -AssemblyName PresentationFramework
Add-Type -AssemblyName System.Windows.Forms

[xml]$XAML = @'
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Local Admins Tool" SizeToContent="WidthAndHeight" FontSize="14"
FontFamily="Consolas" Name="Window">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition MinWidth="100"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="30"/>
            <RowDefinition Height="30"/>
            <RowDefinition Height="30"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Label Content="Computer: " Grid.Column="0" />
        <TextBox Grid.Column="1" Name="TextBoxComputer"/>
        <Label Content="Domain: " Grid.Column="0" Grid.Row="1"/>
        <ComboBox Grid.Column="1" Grid.Row="1" Name="ComboDomain">
            <ComboBox.Items>
                <ComboBoxItem Content="AP"/>
                <ComboBoxItem Content="EU"/>
```

```
                <ComboBoxItem Content="MMS"/>
                <ComboBoxItem Content="NA"/>
                <ComboBoxItem Content="PII"/>
                <ComboBoxItem Content="ZA"/>
            </ComboBox.Items>
        </ComboBox>
        <StackPanel Grid.ColumnSpan="2" Grid.Row="2" Orientation="Horizontal">
            <Button Name="ButtonList" Content="Check" Width="100"/>
            <Button Name="ButtonAdd" Content="Add" Width="100"/>
            <Button Name="ButtonCancel" Content="Cancel" Width="100"/>
        </StackPanel>
        <ListBox Grid.ColumnSpan="3" Grid.Row="3" Name="AdminsList"
Visibility="Hidden"/>

    </Grid>
</Window>
'@

$Reader = New-Object System.Xml.XmlNodeReader $XAML
$Dialog = [Windows.Markup.XamlReader]::Load($Reader)
foreach ($Name in ($XAML | Select-Xml '//*/@Name' |
    foreach { $_.Node.Value})) {
        New-Variable -Name $Name -Value $Dialog.FindName($Name)
    }

$AddScriptBlock = {
    if ($_.Key) {
        if ($_.Key -ne 'Enter') {
            return
        }
    }

    $ComputerName = $TextBoxComputer.Text.Trim()

    if ($ComputerName -notmatch '^[a-z]{3}\d{6}$') {
        [Windows.Forms.MessageBox]::Show(
            "Wrong computer name: $ComputerName"
        )
        return
    }

    $DomainName = $ComboDomain.SelectedValue

    if (!$DomainName) {
        $DomainName = 'EU'
    }

    Invoke-Command -ComputerName VWAR -ConfigurationName $DomainName -ScriptBlock {
        Add-LocalAdmin -ComputerName $using:ComputerName -Domain $using:DomainName
    }
}

$TextBoxComputer.Add_TextChanged({
    switch -Regex ($this.Text.Trim()) {
        '^[a-z]{3}\d{6}$' {
            $this.Background = 'LawnGreen'
        }
        ^$ {
            $this.Background = 'White'
        }
        default {
            $this.Background = 'LightCoral'
```

```
        }
    }

})

$TextBoxComputer.Add_KeyDown($AddScriptBlock)
$ButtonAdd.Add_Click($AddScriptBlock)

$ButtonList.Add_Click({
    $DomainName = $ComboDomain.SelectedValue
    if (!$DomainName) {
        $DomainName = 'EU'
    }

    $ComputerName = $TextBoxComputer.Text
    if ($ComputerName -notmatch '^[a-z]{3}\d{6}$') {
        [Windows.Forms.MessageBox]::Show("Wrong computer name!")
        return
    }

    $AdminsRemote =
@(Invoke-Command -ComputerName VWAR -ConfigurationName $DomainName -ScriptBlock {
        Get-LocalAdmins -ComputerName $using:ComputerName
    })
    if ($AdminsRemote) {
        $AdminList.Items.Clear()
        foreach ($Admin in $AdminsRemote) {
            $AdminList.Items.Add($Admin)
        }
        $AdminsList.Visibility = 'Visible'
    } else {
        $AdminsList.Visibility = 'Hidden'
    }
})

$ButtonCancel.Add_Click({$window.Close()})
$Dialog.ShowDialog() | out-null

# Plik: 4_Demo_v3_Reflection.ps1
#requires -version 3

$Akceleratory =
    [PSObject].
    Assembly.
    GetType("System.Management.Automation.TypeAccelerators")

Add-Type -AssemblyName PresentationCore, PresentationFramework -PassThru |
    Where-Object IsPublic |
    ForEach-Object {
        $Class = $_
        try {
            $Akceleratory::Add($Class.Name,$Class)
        } catch {
            "Failed to add $($Class.Name) accelerator pointing to $($Class.FullName)"
        }
    }

[Window]@{
    OpacityMask = [DrawingBrush]@{
        Drawing = [DrawingGroup]@{
            Children = & {
                $Kolekcja = New-Object DrawingCollection
```

```
            $Kolekcja.Add([GeometryDrawing]@{
                Brush = 'Black'
                Geometry = [EllipseGeometry]@{
                    radiusX = 0.48
                    radiusY = 0.48
                    Center = '0.5,0.5'
                }
            })
            $Kolekcja.Add([GeometryDrawing]@{
                Brush = 'Transparent'
                Geometry = [RectangleGeometry]@{
                    Rect = '0,0,1,1'
                }
            })
            , $Kolekcja
        }
    }
}
Background = [LinearGradientBrush]@{
    Opacity = 0.5
    StartPoint = '0,0.5'
    Endpoint = '1,0.5'
    GradientStops = & {
        $Stopki = New-Object GradientStopCollection
        $Colors = 'Blue', 'Green'
            foreach ($i in 0..1) {
            $Stopki.Add(
                [GradientStop]@{
                    Color = $Colors[$i]
                    Offset = $i
                }
            )
        }
        , $Stopki
    }
}
Width = 800
Height = 400
WindowStyle = 'None'
AllowsTransparency = $true
Effect = [DropShadowEffect]@{
    BlurRadius = 10
}
TopMost = $true
Content = & {
    $Stos = [StackPanel]@{
        VerticalAlignment = 'Center'
        HorizontalAlignment = 'Center'
    }

    $Stos.AddChild(
        [Label]@{
            Content = 'PowerShell Rocks!'
            FontSize = 80
            FontFamily = 'Consolas'
            Foreground = 'White'
            Effect = [DropShadowEffect]@{
                BlurRadius = 5
            }
        }
    )
    , $Stos
```

```
    }
} | ForEach-Object {
    $_.Add_MouseLeftButtonDown({
        $this.DragMove()
    })
    $_.Add_MouseRightButtonDown({
        $this.Close()
    })
    $_.ShowDialog() | Out-Null
}
```