Windows PowerShell Scripting and Toolmaking

Cursusduur: 5 Dagen **Cursuscode: M55039** **Trainingsmethode: Virtual Learning**

Beschrijving:

This instructor-led is intended for IT professionals who are interested in furthering their skills in Windows PowerShell and administrative automation. The course assumes a basic working knowledge of PowerShell as an interactive command-line shell, and teaches students the correct patterns and practices for building reusable, tightly scoped units of automation.

Virtueel en Klassikaal™

Virtueel en Klassikaal™ is een eenvoudig leerconcept en biedt een flexibele oplossing voor het volgen van een klassikale training. Met Virtueel en Klassikaal™ kunt u zelf beslissen of u een klassikale training virtueel (vanuit huis of kantoor)of fysiek op locatie wilt volgen. De keuze is aan u! Cursisten die virtueel deelnemen aan de training ontvangen voor aanvang van de training alle benodigde informatie om de training te kunnen volgen.

Doelgroep:

This course is intended for administrators in a Microsoft-centric environment who want to build reusable units of automation, automate business processes, and enable less-technical colleagues to accomplish administrative tasks.

Doelstelling:

- After completing this course, students will be able to:
- Describe the correct patterns for building modularized tools in Windows PowerShell
- Build highly modularized functions that comply with native PowerShell patterns
- Build controller scripts that expose user interfaces and automate business processes
- Manage data in a variety of formats
- Write automated tests for tools
- Debug tools

Vereiste kennis en vaardigheden:

Before attending this course, students must have:

- Experience at basic Windows administration
- Experience using Windows PowerShell to query and modify system information
- Experience using Windows PowerShell to discover commands and their usage
- Experience using WMI and/or CIM to query system information

Cursusinhoud:

Module 1: Tool Design

This module explains how to design tools and units of automation that comply with native PowerShell usage patterns.

- Tools do one thing
- Tools are flexible
- Tools look nativeLab : Designing a Tool
- Design a tool

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type

This module explains how to add comment-based help to tools.Lessons

- Where to put your help
- Getting started
- Going further with comment-based help
- Broken helpLab : Designing a Tool
- Comment-based help

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script

- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 2: Start with a Command

This module explains how to start the scripting process by beginning in the interactive shell console.

- Why start with a command?
- Discovery and experimentationLab : Designing a Tool
- Start with a command

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 9: Handling Errors

This module explains how to create tools that deal with anticipated errors.Lessons

- Understanding errors and exceptions
- Bad handling
- Two reasons for exception handling
- Handling exceptions in our tool
- Capturing the actual exception
- Handling exceptions for non-commands
- Going further with exception handling
- Deprecated exception handlingLab : Designing a Tool
- Handling errors

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 16: Publishing Your Tools

This module explains how to publish tools to public and private repositories.Lessons

- Begin with a manifest
- Publishing to PowerShell Gallery
- Publishing to private repositoriesLab : Designing a Tool
- Publishing your tools

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help

- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 3: Build a Basic Function and Module

This module explains how to build a basic function and module, using commands already experimented with in the shell.Lessons

- Start with a basic function
- Create a script module
- Check prerequisites
- Run the new commandLab : Designing a Tool
- Build a basic function and module

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Run a command and observe error handling behaviors

- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 17: Basic Controllers: Automation Scripts and Menus

This module explains how to create controller scripts that put tools to use.Lessons

- Building a menu
- Using UIChoice
- Writing a process controllerLab : Designing a Tool
- Basic controllers

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing

- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design

Module 10: Basic Debugging

This module explains how to use native PowerShell script debugging tools.Lessons

- Two kinds of bugs
- The ultimate goal of debugging
- Developing assumptions
- Write-Debug
- Set-PSBreakpoint
- The PowerShell ISELab : Designing a Tool
- Basic debugging

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options

- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell

patterns, from business requirements.

Module 4: Adding CmdletBinding and Parameterizing

This module explains how to extend the functionality of a tool, parameterize input values, and use CmdletBinding.Lessons

- About CmdletBinding and common parameters
- Accepting pipeline input
- Mandatory-ness
- Parameter validation
- Parmeter aliasesLab : Designing a Tool
- Adding CmdletBinding and Parameterizing

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 11: Going Deeper with Parameters

This module explains how to further define parameter attributes in a PowerShell command.Lessons

- Parameter positions
- Validation
- Multiple parameter sets
- Value from remaining arguments
- Help messages
- Aliases
- More CmdletBinding

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 18: Proxy Functions

This module explains how to create and use proxy functions.Lessons

- A proxy example
- Creating the proxy base
- Modifying the proxy
- Adding or removing parametersLab : Designing a Tool
- Proxy functions

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods

Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Define parameter validation

Module 5: Emitting Objects as Output

This module explains how to create tools that produce custom objects as output.Lessons

- Assembling information
- Constructing and emitting output
- Quick testsLab : Designing a Tool
- Emitting objects as output

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 19: Working with XML Data

This module explains how to work with XML data in PowerShell.Lessons

- Simple: CliXML
- Importing native XML
- ConvertTo-XML
- Creating native XML from scratchLab : Designing a Tool
- Working with XML

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline

requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 6: An Interlude: Changing Your Approach

This module explains how to re-think tool design, using concrete examples of how it's often done wrong.Lessons

- Examining a script
- Critiquing a script
- Revising the script

After completing this module, students will be able to:

Module 12: Writing Full Help

This module explains how to create external help for a command.Lessons

- External help
- Using PlatyPs
- Supporting online help
- "About" topics
- Making your help updatableLab : Designing a Tool
- Writing full help

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and

input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 20: Working with JSON Data

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 13: Unit Testing Your Code

This module explains how to use Pester to perform basic unit testing.Lessons

- Sketching out the test
- Making something to test
- Expanding the test
- Going further with PesterLab : Designing a Tool
- Unit testing your code

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a

This module explains how to using JSON data in PowerShell.Lessons

- Converting to JSON
- Converting from JSONLab : Designing a Tool
- Working with JSON data

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer

- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 7: Using Verbose, Warning, and Informational Output

This module explains how to use additional output pipelines for better script behaviors.Lessons

- Knowing the six channels
- Adding verbose and warning output
- Doing more with verbose output
- Informational outputLab : Designing a Tool
- Using Verbose, Warning, and Informational Output

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Write basic unit tests for PowerShell functions

Module 14: Extending Output Types

- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 21: Working with SQL Server Data

This module explains how to use SQL Server from within a PowerShell script.Lessons

- SQL Server terminology and facts
- Connecting to the server and database
- Writing a query
- Running a query
- Invoke-SqlCmd
- Thinking about tool design patterns

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the

- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Run commands with extra output enabled

Module 8: Comment-Based Help

This module explains how to extend objects with additional capabilities.Lessons

- Understanding types
- The Extensible Type System
- Extending an object
- Using Update-TypeData

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 22: Final Exam

This module provides a chance for students to use everything they have learned in this course within a practical example.Lessons

- Lab problem
- Break down the problem
- Do the design
- Test the commands
- Code the toolLab : Final Exam
- Lab oneLab : Final Exam

- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script
- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Module 15: Analyzing Your Script

This module explains how to use Script Analyzer to support best practices and prevent common problems.Lessons

- Performing a basic analysis
- Analyzing the analysisLab : Designing a Tool
- Analyzing your script
- Lab two

After completing this module, students will be able to:

- Describe the native shell patterns that a good tool design should exhibit
- Describe the benefits of discovery and experimentation in the console
- Discover and experiment with existing commands in the console
- Build a basic function
- Create a script module
- Run a command from a script module
- Describe the purpose of CmdletBinding and list common parameters
- Parameterize a script's input
- Define parameters as mandatory
- Define parameters as accepting pipeline input
- Describe the purpose of object-based output
- Create and output custom objects from a function
- Describe the native patterns that a good tool design should exhibit
- Redesign a script to meet business requirements and conform to native patterns
- Describe the six output channels in the shell
- Write commands that use verbose, warning, and informational output
- Describe the purpose and construction of comment-based help
- Add comment-based help to a function
- Identify causes of broken comment-based help
- Describe the native patterns for handling errors in a command
- Add error handling to a command
- Describe the tools used for debugging in PowerShell
- Debug a broken script
- Describe the use of positional parameters
- Describe additional parameter validation methods
- Describe how to define multiple parameter sets
- Describe other parameter definition options
- Describe the advantages of external help
- Create external help using PlatyPS and Markdown
- Describe the purpose of unit testing
- Describe the purpose of the ETS
- Extend an existing object type
- Describe the use of Script Analyzer
- Perform a basic script analysis
- Describe the tool publishing process and requirements
- Publish a tool to a repository
- Describe the purpose of basic controller scripts
- Write a simple controller script

- Describe the purpose of proxy functions
- Create a simple proxy function
- Describe the use of XML within PowerShell
- Use XML data within a PowerShell function
- Describe the use of JSON data within PowerShell
- Use JSON data within a PowerShell function
- Describe the use of SQL Server from within PowerShell
- Write and run SQL Server queries
- Design tools that use SQL Server for data storage
- Create PowerShell tools, using native design patterns, from business requirements.

Nadere informatie:

Neem voor nadere informatie of boekingen contact op met onze Customer Service Desk 030 - 60 89 444

info@globalknowledge.nl

www.globalknowledge.com/nl-nl/

Iepenhoeve 5, 3438 MR Nieuwegein