**Batch information:**

* **Batch Start Date: 2025-08-04**
* **Batch Name: WiproNGA\_DWS\_B5\_25VID2550**
* **First Name: Seetal**
* **Last Name: Biswal**
* **User Id: 34933**
* **Batch ID: 25VID2550**
* **Assignment Name- Objects, Arrays, Variables**

**Scope**

**More Operators**

**Scripting Constructs**

**Modularization**

**Understand basic platform value, concepts and usage**

**Objects, Arrays, Variables:**

**Objects:**

1. Properties and methods: Objects have properties (data) and methods (actions).

2. Types: Objects can be of various types, such as strings, integers, or custom objects.

3. Accessing properties: Use the dot notation (e.g., $object.Property) to access object properties.

**Arrays:**

1. Collections of objects: Arrays are collections of objects of the same or different types.

2. Indexing: Arrays are indexed, allowing you to access specific elements using the index (e.g., $array[0]).

3. Array operators: Use array operators like +, -, and .. to manipulate arrays.

**Variables:**

1. Storing data: Variables store data, such as strings, integers, or objects.

2. Variable names: Choose meaningful variable names to make your code readable.

3. Variable types: Variables can be of various types, such as strings, integers, or custom objects.

**Examples:**

1. Creating an object: $person = [PSCustomObject]@{Name="John"; Age=30}

2. Creating an array: $fruits = @("Apple", "Banana", "Orange")

3. Assigning a variable: $name = "John"

**Scope:**

In PowerShell, scope refers to the region of the script where a variable or function is defined and accessible. Understanding scope is crucial for writing effective and efficient PowerShell scripts.

**Types of Scopes:**

1. Global scope: Variables and functions defined in the global scope are accessible from anywhere in the script.

2. Script scope: Variables and functions defined in the script scope are accessible only within the script.

3. Local scope: Variables and functions defined in a local scope (e.g., within a function) are accessible only within that scope.

4. Private scope: Variables and functions defined with the private modifier are accessible only within the current scope.

**More Operators:**

PowerShell provides various operators for performing different operations. Here's an overview of some additional operators:

**Comparison Operators:**

1. -eq: Equal to

2. -ne: Not equal to

3. -gt: Greater than

4. -ge: Greater than or equal to

5. -lt: Less than

6. -le: Less than or equal to

**Logical Operators:**

1. -and: Logical and

2. -or: Logical or

3. -not: Logical not

4. -xor: Logical exclusive or

**String Operators:**

1. -like: String matching with wildcards

2. -notlike: String not matching with wildcards

3. -match: String matching with regular expressions

4. -notmatch: String not matching with regular expressions

5. -contains: String contains a value

6. -notcontains: String does not contain a value

**Array Operators:**

1. -contains: Array contains a value

2. -notcontains: Array does not contain a value

3. -in: Value is in an array

4. -notin: Value is not in an array

**Type Operators:**

1. -is: Object is of a specific type

2. -isnot: Object is not of a specific type

3. -as: Convert an object to a specific type

**Redirection Operators:**

1. >: Redirect output to a file

2. >>: Append output to a file

3. 2>: Redirect error output to a file

4. 2>>: Append error output to a file

**Scripting Constructs:**

PowerShell scripting constructs are the building blocks of scripts, enabling you to control the flow of execution, make decisions, and repeat tasks. Here's an overview of key scripting constructs:

**Conditional Statements:**

1. If: Used to execute code based on a condition.

2. Else: Used to specify alternative code to execute when the condition is false.

3. ElseIf: Used to check additional conditions.

Example: if ($age -gt 18) { "Adult" } else { "Minor" }

**Loops:**

1. Foreach: Used to iterate over a collection of objects.

2. While: Used to execute code while a condition is true.

3. Do-While: Used to execute code while a condition is true, with the condition checked after the code block.

4. Do-Until: Used to execute code until a condition is true.

Examples:

- foreach ($file in Get-ChildItem) { Write-Host $file.Name }

- while ($i -lt 10) { Write-Host $i; $i++ }

**Modularization:**

Modularization in PowerShell refers to breaking down large scripts into smaller, reusable, and maintainable modules. This approach has several benefits, including:

**Benefits of Modularization:**

1. Reusability: Modules can be reused across multiple scripts, reducing code duplication.

2. Maintainability: Modules are easier to maintain and update, as changes are isolated to a specific module.

3. Readability: Modular code is more readable, as each module has a specific purpose and functionality.

4. Testability: Modules can be tested independently, making it easier to identify and fix issues.

**Types of Modules:**

1. Script Modules: Modules that contain PowerShell scripts and functions.

2. Binary Modules: Modules that contain compiled .NET code.

**Understand basic platform value, concepts and usage:**

To understand the basic platform value, concepts, and usage, let's break it down:

**Platform Value:**

1. Integration: A platform integrates various components, tools, and services to provide a cohesive experience.

2. Scalability: A platform can scale to meet growing demands, ensuring performance and reliability.

3. Flexibility: A platform provides flexibility in terms of customization, configuration, and extension.

4. Innovation: A platform enables innovation by providing a foundation for new ideas, products, and services.

**Key Concepts:**

1. Architecture: The platform's architecture defines its structure, components, and interactions.

2. Components: Components are the building blocks of a platform, providing specific functionality and services.

3. Services: Services are the functional capabilities provided by a platform, such as data storage, processing, or analytics.

4. APIs: APIs (Application Programming Interfaces) enable interaction between components, services, and external applications.

**Usage:**

1. Development: Platforms provide a foundation for development, enabling developers to build applications, services, and integrations.

2. Deployment: Platforms provide a environment for deployment, ensuring scalability, security, and performance.

3. Management: Platforms provide tools and services for management, monitoring, and maintenance.

4. Integration: Platforms enable integration with other systems, services, and applications, providing a seamless experience.