Microsoft

Arrays

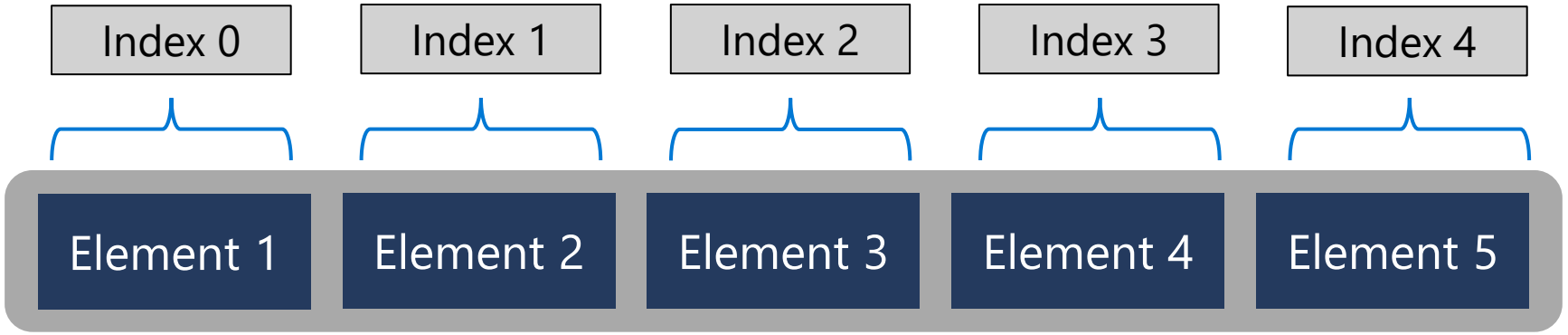
Arrays Overview

Array Overview

A data structure for storing a **collection** of objects, called **elements**

Can contain the **same or different** type of objects

Array elements are accessed using **index** positions, which **begin at zero**



Creating arrays

Output from cmdlets

- $Array = Get-ChildItem

Comma separated values

- $Array = 1, 5, 7, 8, “alpha”, “beta”

With zero or one elements

- $Array = @()
- $Array = @(“Element”)

```
PS> $Array = 1, 5, 8, "alpha", "beta"
PS> $Array

1
5
8
alpha
beta
```

```
PS> $Array = Get-ChildItem
PS> $Array

Mode                 LastWriteTime Name
----                 ------------- ----
d-----  6/12/2020   3:16 PM Directory1
-a----  6/12/2020   3:16 PM File1.txt
-a----  6/12/2020   3:16 PM Script1.ps1
```

Accessing Array Elements

$Array[0]	• Return element 1
$Array[3]	• Return element 4
$Array[0,3]	• Return elements 1 and 4
$Array[-1]	• Return the last element
$Array[-3]	• Return the 3rd to last element

```
PS> $Array = 1, 5, 8, "alpha", "beta"
PS> $Array[0]
1

PS> $Array[3]
alpha

PS> $Array[0,3]
1
alpha

PS> $Array[-1]
beta

PS> $Array[-3]
8
```

Working With Array Elements

An array element's value can be set by using the index number

- $Array[1] = "New value"

Array indexes can be a variable

- $Array[$i]
- $i is a common variable for indexes

Dot notation can still be used when calling specific array elements

- $Array[$i].property
- $Array[1].property

```
PS> $Array = Get-Service

PS> $Array[1].Name
AJRouter

PS> $i = 1
PS> $Array[$i]
Status   Name      DisplayName
------   ----      -----------
Stopped AJRouter AllJoyn Router Service

PS> $Array[1].Name = "NotAJRouter"
PS> $Array[1].Name
NotAJRouter

PS> $Array[$i].name
NotAJRouter
```

Demonstration

Creating and Accessing Arrays

Array Members

Be careful when returning array members

The pipeline will pass **one element** at a time to Get-Member

```
PS> $Array = 1, 2, "alpha", "beta"
PS> $Array | Get-Member
 TypeName: System.Int32

Name         MemberType
----         ----------
ToDouble     Method

 TypeName: System.String

Name               MemberType
----               ----------
ToLower            Method
```

Using –InputObject will pass the array as a **single object**

```
PS> $Array = 1, 2, "Alpha", "Beta"
PS> Get-Member -InputObject $Array
 TypeName: System.Object[]

Name             MemberType
----             ----------
Contains         Method
GetType          Method
Set              Method
Length           Property
IsReadOnly       Property
```

Length and Count

Array types have **2 properties** to identify how many elements they contain

- $Array.Length
- $Array.Count

.Count exists on all collection types

.Length is a property for other objects

```
PS> $Array = 1, 5, 8, "alpha", "beta"
PS> $Array.Length
5

PS> $Array.Count
5
```

```
   TypeName: System.Object[]

Name   MemberType    Definition
----   ----------    ----------
Length Property      int Length {get;}
Count  AliasProperty Count = Length
```

Adding Elements to an Array

Elements can be **added** to arrays by using the **+** operator

- $Array = $Array + 10
- $Array += 10

Arrays have an "Add" method, but it **does not add** elements to the array

- $Array.Add("Delta")

Arrays are a **fixed length** when created

- Adding a new element **creates a new array** with the new length

```
PS> $Array = 1, 5, 8, "alpha", "beta"
PS> $Array = $Array + 10
PS> $Array
1
5
8
alpha
beta
10

PS> $Array.Add("Delta")
PS> $Array
1
5
8
alpha
beta
10
```

Sorting an array

Only sorts the **output** of the array

- $Array | Sort-Object

Sorts the values of the array elements and **saves the order** to the array

- [array]::sort($Array)
- $Array = $Array | Sort-Object

```
PS> $Array = 8, 1, "beta", 5, "alpha"
PS> $Array | Sort-Object
1
5
8
alpha
beta
```

```
PS> $Array = 3, 1, 5, 8, 2
PS> [array]::sort($Array)
PS> $Array
1
2
3
5
8
```

Demonstration: Working with Arrays

Array Operators

Array Operators – 1

Operator	Syntax	Description
Range	1..5	Returns the values using the sequential integers
Format	“{1} {0} {2}” –f $Array	Inserts objects from an array into a string
Contains	$Array –Contains “value”	Searches an array for a specific value, returning true or false
In	“value” –in $Array	Searches for a specific value in an array, returning true or false
Split	"1-2-3-4-5" -split "-"	Turns a string into an array based on a delimiter
Join	$Array -join ":"	Turns a list of array items into a single string, can use a delimiter

```
PS> 1..5
1
2
3
4
5
```

```
PS> 2..-2
2
1
0
-1
-2
```

```
PS> $Array = 1,2,3,4,5,6
PS> $Array[1..4]
2
3
4
5
```

Array Operators – 2

Operator	Syntax	Description
Range	1..5	Returns the values using the sequential integers
Format	“{1} {0} {2}” –f $Array	Inserts objects from an array into a string
Contains	$Array –Contains “value”	Searches an array for a specific value, returning true or false
In	“value” –in $Array	Searches for a specific value in an array, returning true or false
Split	"1-2-3-4-5" -split "-"	Turns a string into an array based on a delimiter
Join	$Array -join ":"	Turns a list of array items into a single string, can use a delimiter

```
PS> $Array = "Doe", "John", .35, 1234567.89123
PS> "{1} {0}" –f $Array
John Doe

PS> "{2:p} ; {3:n} ; {3:n0} ; {3:c}" –f $Array
35.00% ; 1,234,567.89 ; 1,234,568 ; $1,234,567.89
```

Array Operators – 3

Operator	Syntax	Description
Range	1..5	Returns the values using the sequential integers
Format	“{1} {0} {2}” –f $Array	Inserts objects from an array into a string
Contains	$Array –Contains “value”	Searches an array for a specific value, returning true or false
In	“value” –in $Array	Searches for a specific value in an array, returning true or false
Split	"1-2-3-4-5" -split "-"	Turns a string into an array based on a delimiter
Join	$Array -join ":"	Turns a list of array items into a single string, can use a delimiter

```
PS> $Array = 2, 4, 6, 8, 10
PS> $Array –contains 4
True

PS> $Array –contains 3
False
```

Array Operators – 4

Operator	Syntax	Description
Range	1..5	Returns the values using the sequential integers
Format	“{1} {0} {2}” –f $Array	Inserts objects from an array into a string
Contains	$Array –Contains “value”	Searches an array for a specific value, returning true or false
In	“value” –in $Array	Searches for a specific value in an array, returning true or false
Split	"1-2-3-4-5" -split "-"	Turns a string into an array based on a delimiter
Join	$Array -join ":"	Turns a list of array items into a single string, can use a delimiter

```
PS> $Array = 2, 4, 6, 8, 10
PS> 4 –in $Array
True

PS> 3 –in $Array
False
```

Array Operators – 5

Operator	Syntax	Description
Range	1..5	Returns the values using the sequential integers
Format	“{1} {0} {2}” –f $Array	Inserts objects from an array into a string
Contains	$Array –Contains “value”	Searches an array for a specific value, returning true or false
In	“value” –in $Array	Searches for a specific value in an array, returning true or false
Split	"1-2-3-4-5" -split "-"	Turns a string into an array based on a delimiter
Join	$Array -join ":"	Turns a list of array items into a single string, can use a delimiter

```
PS> "1-2-3-4-5" -split "-"
1
2
3
4
5
```

Array Operators – 6

Operator	Syntax	Description
Range	1..5	Returns the values using the sequential integers
Format	“{1} {0} {2}” –f $Array	Inserts objects from an array into a string
Contains	$Array –Contains “value”	Searches an array for a specific value, returning true or false
In	“value” –in $Array	Searches for a specific value in an array, returning true or false
Split	"1-2-3-4-5" -split "-"	Turns a string into an array based on a delimiter
Join	$Array -join ":"	Turns a list of array items into a single string, can use a delimiter

```
PS> $Array = 2, 4, 6, 8, 10
PS> $Array –join ":"
2:4:6:8:10
```

Demonstration: Array Operators

Other Collections

Other Collections Examples

.Net has a lot of other collection types that can hold objects

Collections are typically optimized for specific goals

Improved search times in data	Growing in memory

Some common collection types include

ArrayLists	Hash Tables	Generic Lists	Stacks	Queues

ArrayLists

Size increases dynamically, which is **much** more efficient than arrays

Not created with PowerShell operators, like **@()** or **+=**

- Create with the **New-Object** cmdlet
- Add elements with the **.Add()** method

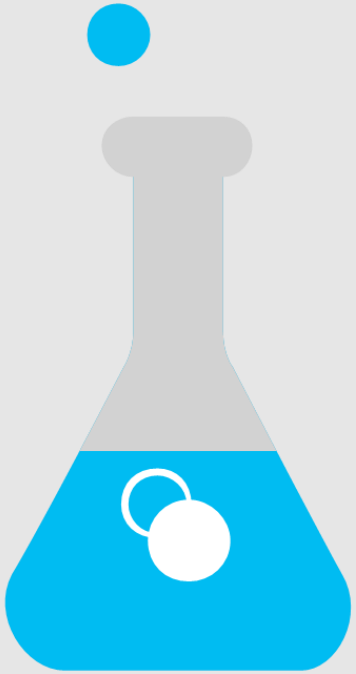
Elements are accessed with **index notation**, just like arrays

```
PS> $List = New-Object -TypeName System.Collections.ArrayList
PS> $List.Add("Hello") | Out-Null

PS> $List[0]
Hello

PS> $List[0].ToUpper()
HELLO
```

Demonstration: Other Collections

Lab 8:
Arrays

45 minutes



© Copyright Microsoft Corporation. All rights reserved.