# Mod 10 PowerShell Arrays and Advanced Flow Control

SCRIPTING ESSENTIALS

DR. BURKMAN

## Functions

Preferably name your function in the Verb-Noun pattern of PowerShell

## Functions

Basic syntax:

```
function Test-My-Function ()
{
    Write-Host("Hello World")
}



Test-My-Function
```

## Functions

Passing a parameter:

```
function Test-My-Function ([string]$dog)
{

    Write-Host($dog)
}

Test-My-Function -dog 'hi'
```

You can skip the data type if PS can figure out what you want

```
function Test-My-Function ($dog)
{

    Write-Host($dog)
}

Test-My-Function -dog 'hi'
```

## Functions

Passing two parameters:

```
 function Test-My-Function ($dog, $cat)
{

    Write-Host($dog)
    Write-Host($cat *3)
}


Test-My-Function -dog 'hi' -cat 7
```

## Functions

A default parameter value:

```
function Test-My-Function ($dog = "hello", $cat)
{

    Write-Host($dog)
    Write-Host($cat *3)
}

Test-My-Function -cat 7 #-dog "hi"
```

## Functions

Returning a value:

```
function Test-My-Function ($dog = "hello", $cat)
{

    Write-Host($dog)
    Write-Host($cat *3)
    $mouse = "OSU"
    return $mouse
}


$bird = Test-My-Function -cat 7
Write-Host($bird)
```

## Arrays and Hash Tables

PowerShell doesn’t have lists but it does have arrays.

https://en.wikiversity.org/wiki/PowerShell/Arrays_and_Hash_Tables

## PowerShell Arrays

Just like with Python Lists, arrays are accessed by integer position, left to right, starting with zero and contain varying data types.  An out of bounds request does not generate an error condition.

```
 cls

$a = @(22,1,2,3,4,'a','b','c')

Write-Host $a[0]
Write-Host $a[-2]
Write-Host $a[4,5,6]
Write-Host $a[2..6]
```

## PowerShell Arrays

You can initialize an empty array then add values.  This actually copies the array then makes a new array.  Do not try myArray.add(), it will not work because an array is actually a fixed size.  We could steal the list from .net but let's stay with the local PowerScript functionality.

```
$a = @()

$a += 1,2,3
$a += "hi"

Write-Host $a
```

## PowerShell Arrays

Changing values is easy:

```
 $a=@()
$a += 1,2,3
$a += "hi"
write-host $a

$a[0] = "Bob"
Write-Host $a
```

## PowerShell Arrays

Deleting indexes is cumbersome.  You have to make a new array that contains the elements that you want from the old array:

```
$a = @()

$a += 1,2,3
$a += "hi"

$a[0] = "Bob"

Write-Host $a
Write-Host "---------"

$a = @($a[3], $a[0])

Write-Host $a
```

## PowerShell Arrays

Inserting values at a specific index is tricky.  You'll have to make a new array, then iterate through the elements

- See next slide

```
cls
    $myOldArray = @(9,7,5,3,1)
    $myNewArray = @()
    $indexPosition = 3  #change this to the desired index value
    if($indexPosition -eq 0)
    {
        $myNewArray += "mouse"
        for($i = 0;$i -lt $myOldArray.Length;$i++)
        {
            $myNewArray += $myOldArray[$i]
        }
    }
    elseif($indexPosition -ge $myOldArray.Length)
    {
        for($i = 0;$i -lt $myOldArray.Length;$i++)
        {
            $myNewArray += $myOldArray[$i]
        }
        $myNewArray += "mouse"
    }
    else
    {
        for($i = 0;$i -lt $indexPosition;$i++)
        {
            $myNewArray += $myOldArray[$i]
        }
        $myNewArray += "mouse"
        for($i = $indexPosition;$i -lt $myOldArray.Length;$i++)
        {
            $myNewArray += $myOldArray[$i]
        }
    }

    write-host($myNewArray)
```

# PowerShell Arrays

To find the index position of a value in an array, use $array.IndexOf(thing you are looking for):

- Note: -1 is returned if the value is not found

```
cls
    $myArray = @("V","T","U","O")
    write-host($myArray.IndexOf("U"))
```

## PowerShell Arrays

We can find the length of an array with .length

```
$a = @()

$a += 1,2,3
$a += "hi"

$a[0] = "Bob"

Write-Host $a
Write-Host "---------"

$a = @($a[3], $a[0])

Write-Host $a.Length
```

## PowerShell Arrays

You can have arrays of arrays:

```
 cls
$a = @(1,2,3)
$b = @("a", "b", "c")

$c = @($a,$b)

Write-host $c

$d = @(@(4,5,6),@("D","E","F"))

Write-host $d
```

## Arrays of Arrays

Arrays can have arrays but changing the underlying array doesn’t change the new array of arrays. You can directly change subarray elements though:

```
 $a= @(1,2,3)
$b = @("a","b","c")
$c = @(1.1, "house", 7)
$d = @($a, $b, $c)

Write-Host $d[1][1]

$a += 4

write-host $a
Write-Host $d
$d[0] += 4
write-host $d
```

# Copying Arrays

You cannot just say $a = $b.  That will just create two pointers ($a and $b) pointing at the same array.  To copy an array you must iterate through the original array and add each element to the new array.  Much like inserting an element into an existing array:

```
cls
$original_array=@(1,2,3,4,5,6,7,8,9,10)
$copied_array=@()

foreach ($i in $original_array)
{
    $copied_array += $i
}

write-host "original: $original_array"
Write-Host "copy:     $copied_array"

$copied_array[0]=0
Write-Host "______________________________"

write-host "original: $original_array"
Write-Host "copy:     $copied_array"
```

## PowerShell Arrays

Checking to see if an array has a certain value:

```
cls

$a = @(1,2,3,"a","b","c")

$dog = $a.Contains("a")

write-host $dog
```

```
cls

$a = @(1,2,3,"a","b","c")

$dog = $a -Contains "a"

write-host $dog
```

## PowerShell Arrays

Sorting needs piped and can use switches:

```
cls

$a = @("H",7,3,"a",1.44,"c", "A")

$a = $a | sort

write-host $a
```

```
cls

$a = @("H",7,3,"a",1.44,"c", "A")

$a = $a | sort -Descending -CaseSensitive

write-host $a
```

# PowerShell Arrays

Looping with foreach and with for:

```
$a = @(1,2,3,"a","b","c")

foreach($i in $a)
{
    Write-Host $i
}
```

```
$a = @(1,2,3,"a","b","c")

for ($i=0; $i -le $a.Length; $i++)
{
    Write-Host $a[$i]
}
```

## Here’s an example from Python, rewritten to PowerShell:

```
cls

$catNames = @()

while ($true)
{
    $catNumber = $catNames.Length + 1
    $name = Read-Host ("Enter the name of cat  $catNumber `(Or enter nothing to stop`)")
    if($name -eq "")
    {
        break
    }
    $catNames += $name
}

Write-Host("The cat names are:")
foreach($cat in $catNames)
{
  Write-Host($cat)
}
```

Escape with a backtick

Double quotes will unpack variables

## Another Example

```
cls

$messages = @("It is certain",
"It is decidedly so",
"Yes definitely",
"Reply hazy try again",
"Ask again later",
"Concentrate and ask again",
"My reply is no",
"Outlook not so good",
"Very doubtful")

$ranIndex = Get-Random -Minimum 0 -Maximum 9

Write-Host(Get-Random($messages))
Write-Host($messages[$ranIndex])
```

## Scoping Arrays and Functions

```
cls

$array = @(1,2,4,5)

function changeArray ($thingToAdd)
{
    $array += $thingToAdd
    write-host $array
}

changeArray -thingToAdd "dog"
write-host $array
```

```
dog
1 2 4 5
```

Using a global variable

```
cls

$array = @(1,2,4,5)

function changeArray ($thingToAdd)
{
    $Global:array += $thingToAdd
    write-host $array
}

changeArray -thingToAdd "dog"
write-host $array
```

```
1 2 4 5 dog
1 2 4 5 dog
```

## Scoping Arrays and Functions

```
cls

$array = @(1,2,4,5)

function changeArray2 ($array, $thingToAdd)
{
    $array += $thingToAdd
    Write-Host $array
    return $array
}

$array = changeArray2($array, "elephant")
write-host $array
```

Passing the array in, changing, then returning the array to itself (preferred)

## Tips

If array.length gives you any problems try array.count

- This is particularly true if you are trying to get the length of an array when there is just one element

Watch for $input/1 if doing math comparisons

We can use regular expressions!

- $dog -match "^[+]?[0-9]“
- $dog -nomatch "^[+]?[0-9]“

Checking multiple conditions in IF:

- If (this -or that -or what)