Advanced Scripting   
Arithmetic Operators

Last Updated: 4/19/2024 5:46 PM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

This exercise presents a few practice drills involving PowerShell operators.

# Setup

## Requirements

* PowerShell

# Task 1—Arithmetic Operators

## Steps

1. The **+** (addition/concatenation) operator:
   1. Defined for numbers, strings, and arrays. When both numbers and strings of differing data types are used, the right-hand operand will be converted to the left-hand type. Widening of the data type is performed if necessary. Try the following expressions, then record the returned value and type. If the expression produces an exception message, record the error.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expression** | **Value** | **Type** | **Error if any** |
| 1+2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 + '2' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 + 'two' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 + '2.5' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| '1'+ 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 'blue' + 'green' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1,2,3 + 4 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1,2 + 2,3 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |

1. The **\*** operator:
   1. Defined for numbers, strings, and arrays. For both numbers and strings, if differing data types are used, the right-hand operand will be converted to the left-hand type. Widening of the data type is performed if necessary. Try the following expressions, then record the returned value and type. If an exception message is produce, record the error.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expression** | **Value** | **Type** | **Error if any** |
| 1\*2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 \* '2' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 \* '2.5' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| '1' \* 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 \* 'two' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 'red' \* 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1,2,3 \* 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1,2 \* 2,3 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |

1. **–** (subtraction) **/** (division) and **%** (modulus).
   1. Defined only for numbers. If no number data types are used, PowerShell will attempt to convert to numbers. Widening of the data type is performed if necessary. Try the following expressions, then record the returned value and type. Record the error if one occurs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Expression** | **Value** | **Type** | **Error if any** |
| 1 - 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| 1 / '2' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| '10'-'5' | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |
| '5' % 2 | Click or tap here to enter text. | Click or tap here to enter text. | Click or tap here to enter text. |

# Task 2—Assignment Operators

The **=** (assignment) operator is used to assign a value, or the results of an expression or pipeline, to a variable. If a variable does not yet exist, the assignment operator creates it.

## Steps

1. Basic assignment. Enter:  
   $a=10
   1. Enter **$a**
   2. Enter **Get-Variable a**
   3. What value is in $a? Click or tap here to enter text.
2. Assignment as an expression result  
   $b=1gb/1mb
   1. What value is in $b? Click or tap here to enter text.
3. Assignment as the result of a pipeline expression  
   $c=gps|measure|select count
   1. What value is in $c? Click or tap here to enter text.
4. Multiple assignments in one expression  
   **$d = $d = $f = 100**
   1. What is in **$d, $e, $f**? Click or tap here to enter text.
   2. Another   
      $g = ($h = ($i = 10) + 4) \* 2
   3. What are the values of **$g, $h, $i**? Click or tap here to enter text.
5. You can combine an arithmetic operator to the assignment operator to perform a reassignment of the variable to its current contents with the operator and operand applied. Example: **$v+=1** is the same as **$v = $v + 1**. As you proceed, record each subsequent assigned result:

|  |  |
| --- | --- |
| **Enter** | **What is the value of $j ?** |
| $j=10 | Click or tap here to enter text. |
| $j\*=2 | Click or tap here to enter text. |
| $j-=20 | Click or tap here to enter text. |
| $j+=1000 | Click or tap here to enter text. |
| $j/=10 | Click or tap here to enter text. |
| $j%=10 | Click or tap here to enter text. |

1. You can also assign multiple variables to an array of values.  
   $k,$l,$m = 10,'red',(get-date)
   1. What is the value in $k? Click or tap here to enter text.
   2. What is the value in $l? Click or tap here to enter text.
   3. What is the value in $m? Click or tap here to enter text.
2. Easily swap values  
   $n='n'  
   $o='o'  
   $n,$o=$o,$n
   1. What is the value of $n? Click or tap here to enter text.
   2. What is the value of $o? Click or tap here to enter text.

# Task 3— Increment ++ and Decrement -- Unary Operators

The increment and decrement operators are used to add 1 or subtract 1 from a variable and store the results back in the original variable. They can be placed before or after the variable. If placed before the variable the adjustment is done prior to evaluating the expression. If placed after, it happens after evaluating the expression.

We call these *unary* operators because they only have one operand. (All the *binary infix* operators in previous tasks have two operands, one before and one after the operator symbol.)

## Steps

1. Start with numeric variables  
   $a,$b=100,200  
   $a++
   1. What was output?Click or tap here to enter text.
   2. What is the current value of **$a**? Click or tap here to enter text.
2. Pre-increment (increment before)  
   Write-Host (++$a)
   1. What was output?Click or tap here to enter text.
   2. What is the current value of **$a**? Click or tap here to enter text.
3. Post-decrement (decrement after)  
   Write-Host ($b--)
   1. What was output?Click or tap here to enter text.
   2. What is the current value of **$b**? Click or tap here to enter text.

# Task 4—+ and - Unary Operators

The **+** and **-** unary operators convert the operand to a number, if it is not a numeric type already. The **-** operator then multiplies the number by -1.

## Steps

1. The unary + operator is useful to convert a string to a number when you are not sure of the type you need.
   1. Try:  
      $a=+'123'
      1. What datatype is in **$a**?Click or tap here to enter text. *Hint*: **$a.GetType().Name**
      2. What is the current value of **$a**? Click or tap here to enter text.
   2. Try:  
      $b=+'123.5'
      1. What datatype is in **$b**?Click or tap here to enter text.
      2. What is the current value of **$b**? Click or tap here to enter text.
   3. Try:  
      $a=+'2345678901'
      1. What datatype is in $a?Click or tap here to enter text.
      2. What is the current value of $a? Click or tap here to enter text.
2. The **-** unary operator converts the operand to a number if necessary, then changes its sign by multiplying it by -1
   1. Try:  
      $a=-'-123.5'
      1. What datatype is in $a?Click or tap here to enter text.
      2. What is the current value of $a? Click or tap here to enter text.

# Task 5— Arrays and array operators: comma, range, -join, -split

An *array* aggregates a collection of objects together into one object.

## Steps

1. We use the syntax @() to create an empty array in PowerShell, and the comma operator , to create non-empty arrays.
   1. Create an empty array, save it in a variable, view it, and look at its data type:   
      $a = @()   
      $a   
      $a.GetType().Name
   2. All PowerShell arrays have a length property. Get the size of the empty array:   
      $a.length
   3. Prefix an object with a comma to create an array with just one element, view it, and get its size:   
      $a = ,7   
      $a   
      $a.length
   4. Put a comma between each element to create a multiple-element array:   
      $a = 7,3   
      $a   
      $a.length
   5. Make longer array:   
      $a = 7,3,8,9   
      $a   
      $a.length
2. The range operator .. (two dots) is a convenient shortcut for creating sequences of integers.
   1. Another array with exactly one element:   
      4 .. 4
   2. Another array with two elements:   
      4 .. 5
   3. A much longer array:   
      12..49
   4. A descending sequence:   
      5..-5
   5. How many elements are in that descending sequence?   
      (5..-5).length   
      Your result: Click or tap here to enter text.   
      You should get the same result from: **(5..-5).Count**
   6. If you’re using PowerShell Core, you can also create ranges of characters:   
      'Q' .. 'j'   
      "@" .. "'"
3. The -join operator transforms an array into a string. The left operand is the array, and the right operand is a delimiter string; -join interpolates the delimiter between each array element.
   1. Separate array elements with a comma:   
      1,2,3,4 -join ','
   2. Separate array elements with a space:   
       12..49 -join ' '
   3. Now use -join as a unary “prefix” operator to combine array elements without delimiters:   
      -join (4..0)   
      Your result: Click or tap here to enter text.
4. The **-split** operator breaks a string into an array of strings at the matched delimiter.
   1. Split on a space:   
      'I love cilantro' -split ' '
      1. What was returned? Click or tap here to enter text.
   2. Split on the letter i:   
      'I love cilantro' -split 'i'
      1. What was returned? Click or tap here to enter text.
   3. Use more than one character as a delimiting phrase:   
      'flea on a fly on a wart on a frog on a knot on a log' -split ' on a ' Join your result back together with commas. Your output: Click or tap here to enter text.
   4. Use a square-bracketed character set (a regular expression/wildcard) to specify delimiters:  
      'query=src&start=0+end=4+limit=9' -split '[&+]'   
      Join your result with semicolons. Your output: Click or tap here to enter text.
   5. Use **-split** as a unary operator to split on white space:   
      -split 'I am really bad at spacing !'

# Task 6—[void] operator

Because of the square brackets, it looks like an object class or data type, but isn’t. The **[void]** operator can be placed in front of any expression to have the results discarded.

## Steps

1. Enter  
   Write-Output 'hi'
   1. What was the output? Click or tap here to enter text.
2. Enter  
   [void](Write-Output 'hi')
   1. What was the output? Click or tap here to enter text.
3. Enter  
   $a = Write-Output 'hi'
   1. What is the current value of **$a**? Click or tap here to enter text.
4. Enter  
   $a = [void](Write-Output 'hi')
   1. What is the current value of **$a**? Click or tap here to enter text. (Hint: evaluate the expression **$null -eq $a**, then interpret your result.)
5. In a pipeline, you can also discard output you don’t need by piping into the **Out-Null** cmdlet:  
   Write-Output 'hi' | Out-Null

# Deliverable

Upload this document with completed answers to I-Learn Canvas.