Advanced Scripting   
Conditional Logic

Last Updated: 9/23/2021 9:10 AM Version 1  
Document Prepared for: CIT361 Student

# Name Zach Lucas ID 895649438

# Instructions

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

# Overview

This exercise is a quick overview of conditional logic in PowerShell. If you have programmed in any C based languages, this will be easy to pick up since the syntax is quite similar. The only real difference is that {} are always required.

# Requirements

# Setup

# Task 1—The if Statement

The most used conditional logic statement is the if statement. You will use it all the time.

## Steps

1. Simple **if**. The if statement will execute the block of code after the condition if the expression evaluates to true.
   1. Open Either VSCode or the ISE and enter the following script:

$yours=Read-Host "What is your favorite color?"

$mine='blue'

if($yours -eq $mine){

Write-Host 'Our favorite colors are the same!'

}

* 1. Run the script (F5) several times entering green, blue, and Blue for your favorite color. What did you learn about case sensitivity?Case sensitivity doesn’t matter in PowerShell

1. **Else**. In our previous example the script outputs only when the colors match. But it would be better if we told the user when they did not match as well. The else block will get executed only if the if block does not.
   1. Add the following lines to your script

else{

"We don't have the same favorite color."

}

* 1. Run the script again entering blue and green to view the results

1. **ElseIf**. Sometimes you want to check several conditions. The elseif keyword can be used to chain if statements together. If the first condition does not return true, the elseif condition is evaluated. If that returns true then it’s statement block is executed. You can have as many elseif statements as you need. The else block only gets executed if none of the if or elseif conditions are true.
   1. Enter the following code

$answer=get-random -Minimum 1 -Maximum 100

$guess=Read-Host "What is your guess?"

if($answer -gt $guess){

"Higher"

}elseif($answer -lt $guess){

"Lower"

}else{

"Correct"

}

* 1. Run the code a few times to see the results. If you want to cheat to know the correct answer use a breakpoint.

# Task 2—The Switch Statement

The switch statement is used when you want to test multiple conditions. Each pattern in the switch statement is tested and if it matches the statement block is executed. Multiple blocks may match. Use the break statement if you don’t want the rest of the tests performed. The default block is only executed if there are no matches.

## Steps

1. You will now make a mini menu system using the switch to act on the user input.
   1. Enter the following code in a new script:

$command=Read-Host @'

What would you like to do?

1. Start

2. Stop

3. Continue

5. Quit

Enter Choice

'@

switch ($command){

'1' {'Starting'}

'2' {'Stopping'}

'3' {'Continuing'}

'4' {'Quitting'}

default {'Invalid Command'}

}

* 1. Run the program a few times entering 1, 2, 3 or 4 to see the output.

1. Now you will allow the user to type the menu word as well as the number.
   1. Modify your switch statement to look like this:

switch ($command){

'1' {'Starting'}

'start' {'Starting'}

'2' {'Stopping'}

'stop' {'Stopping'}

'3' {'Continuing'}

'continue' {'Continuing'}

'4' {'Quitting'}

'quit' {'Quitting'}

default {'Invalid Command'}

}

* 1. Run your script enter 1, stop, etc. until you get a feel for how it works.

1. Typing *continue* is way to long, let’s fix it so you only need to type the first letter using pattern matches.
   1. Modify your code to look like this

switch -Wildcard ($command){

'1' {'Starting'}

's\*' {'Starting'}

'2' {'Stopping'}

's\*' {'Stopping'}

'3' {'Continuing'}

'c\*' {'Continuing'}

'4' {'Quitting'}

'q\*' {'Quitting'}

default {'Invalid Command'}

}

* 1. Run the code try 1, s, start, q etc. Record your observations? Everything seems to work except for start and stop because they are defined by the same wildcard character.

1. Since the switch statement executes all blocks that match the pattern, when you entered **‘s’** it ran both start and stop. To fix that you could just fix the matching patterns so the user has to enter enough of the word to uniquely identify the options.
   1. Fix it by modifying your code as follows:

switch -Wildcard ($command){

'1' {'Starting'}

'sta\*' {'Starting'}

'2' {'Stopping'}

'sto\*' {'Stopping'}

'3' {'Continuing'}

'c\*' {'Continuing'}

'4' {'Quitting'}

'q\*' {'Quitting'}

default {'Invalid Command'}

}

* 1. Try it again and observe the results.

1. You can replace the pattern with an expression. In the previous example we use two lines to provide two different pattern matches. You can create an expression to do the same thing in one line.
   1. The stop options could be rewritten as:  
      **{$\_ -eq '1' -or $\_ -like 'sta\*'} {'Starting'}**
   2. Rewrite the switch statement using similar conditions for all options. Enter your script here:
   3. switch -Wildcard ($command) {   
       {$\_ -eq '1' -or $\_ -like 'sta\*'}{'Starting'}  
       {$\_ -eq '2' -or $\_ -like 'sto\*'}{'Stopping'}  
       {$\_ -eq '3' -or $\_ -like 'c\*'}{'Continuing'}  
       {$\_ -eq '4' -or $\_ -like 'q\*'}{'Quitting'}  
       Default {'Invalid Command'}  
      }
2. Since the switch statement executes all matches if you only want the first match executed you must use the break statement.
   1. Try this in a new script:

switch (1){

1 {'this matches'}

2 {'this doesn''t'}

1 {'so does this'}

}

* 1. Run the script, notice both lines that have 1 are executed.
  2. The break statement is used to tell the switch operator to quit processing other matches. Add a break statement:

switch (1){

1 {

'this matches'

break

}

2 {'this doesn''t'}

1 {'so does this'}

}

* 1. Run the script again. Record the output: this matches

# Deliverable

Upload this document with completed answers to i-learn.