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

Data presented in JSON and YAML

Exercise 4.3

# Instructions

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

# Overview

In the previous exercise you saw that XML, as a serialized object data format, enjoys first-class support in the .NET libraries and PowerShell. More recently, a couple of simpler data presentation formats have increased in popularity:

* Javascript Object Notation (JSON), and
* “YAML Ain’t Markup Language” (YAML).

PowerShell supports neither of these as richly as XML, but it nevertheless has grown to support each of them very well. Unlike XML, neither JSON nor YAML support *attributes* on nodes. However, every data structure and object can still be represented using either JSON or YAML, so it’s not surprising that technology professionals have increasingly opted to use them.

# Requirements

* Internet connection
* psfiles.zip

# Setup

If you haven’t already, download and extract the contents of **psfiles.zip**. (It has a custom XML file you will use in tasks 3 through 6.)

# Task 1—Convert CSV Data into JSON format

Make **psfiles/data** your current working directory.

## Steps

1. Import the Metals.csv spreadsheet data:   
   **$m=Import-Csv .\Metals.csv**
2. Due to the way CSV files are processed, the numeric values will be interpreted as strings. Let’s start by fixing the numeric data back to numbers:   
   **$m|%{$\_.SpecificGravity=+$\_.SpecificGravity;$\_.MeltingPoint=+$\_.MeltingPoint}**   
   (Note: this command is simpler than it looks. All we did was use unary **+** operators to convert the imported strings back into numbers.)
3. View and investigate the results.
   1. What data type is the SpecificGravity Property? Click or tap here to enter text.
   2. What data type is the MeltingPoint Property? Click or tap here to enter text. *(Hint: if you answered* **Object[]** *to either of these, keep digging until you figure out the correct answer.)*
4. Convert this data to JSON format:  
   **ConvertTo-Json -InputObject $m**
   1. Do it again, except this time instead of just outputting the resulting data to the terminal, save the ouptut to the file **Metals.json**:   
      **ConvertTo-Json -InputObject $m | Set-Content Metals.json**
5. View the resulting file’s contents again:   
   **Get-Content Metals.json**
   1. Look at how the JSON data structures are organized:
      1. Notice that the entire collection of metal objects is placed between **[** square brackets **]**, with commas between each item in the collection. (This happens to be how literal arrays are specified in the Javascript language.)
      2. Notice that property names and values are specified using *key-value pairs*, with a colon **:** character between each key and value.
      3. Notice that each metal item is placed between { curly braces}, with commas between each key-value pair. (This happens to be how literal objects are specified in Javascript.)
      4. I think you will agree that it’s a simpler data format, but notice that nowhere in the file is the word *Metal*! Unlike XML, JSON does NOT self-document the types of objects and collections that are represented. JSON technology practitioners’ best practice is to document that separately somehow.
   2. In the JSON formatted data, how are the **Symbol** and **Name** properties different than the **MeltingPoint** and **SpecificGravity** properties? Click or tap here to enter text.

# Task 2—Reading JSON Data

## Steps

1. Read the JSON data you just created into PowerShell, and convert them to PowerShell objects. The **ConvertFrom-Json** cmdlet expects a string, so you will need to provide a string. You might remember that the default behavior of the **Get-Content** cmdlet is to read the file into an array of strings, one element for each line of text in the file. You can make Get-Content import as one big string, using the **-raw** switch parameter:   
   **$jm=ConvertFrom-Json (Get-Content .\Metals.json -raw)**
2. View the data  
   **$jm**
3. For each property, list the datatype:
   1. Symbol: Click or tap here to enter text.
   2. Name: Click or tap here to enter text.
   3. MeltingPoint: Click or tap here to enter text.
   4. SpecificGravity: Click or tap here to enter text.

# Task 3—YAML

Unlike with JSON, PowerShell does not provide any native support for YAML. However, there are Libraries in the PowerShell Gallery that will. Let’s try one of them.

## Steps

1. Install the powershell-yaml module from the PowerShell Gallery:   
   Install-Module powershell-yaml -Scope CurrentUser   
   The module provides two new cmdlets, **ConvertFrom-Yaml** and **ConvertTo-Yaml**. They work pretty much the same way as the corresponding JSON cmdlets.
2. Look at a YAML file. Kubernetes uses YAML syntax to configure pods. View the sample file:   
   Get-Content .\kube.yml
   1. Study it. You should notice and recognize that YAML is a *whitespace-significant* data format. Exact indentation levels are used to directly represent an entire tree structure. (If you have experience with Python, you have already seen whitespace-significant indented code.)
   2. For a “leaf” (terminal) node in the tree, a key-value pair is represented using a colon between each key and value, similar to the way it’s done in JSON format.
   3. Parent-child relationships between nodes are treated as key-value pairs a little differently. A colon : follows the parent node, but then each child object is *indented*, carefully and equally, on subsequent lines.
   4. A hyphen – prefix marks every node that could become a *collection* of multiple elements, even where the collection currently consists of just one node.
3. Now load and parse the sample file:   
   $kube=ConvertFrom-Yaml (Get-Content .\kube.yml -raw)
4. Explore the object:  
   $kube   
   $kube.metadata.name   
   etc. Keep climbing down as far as you can into the tree’s branches. Each time you reach a “leaf” node, use the .GetType() method to get that property’s data type.
5. Write a PowerShell command line that accesses the value of the **containerPort** property.
   1. Enter your command line here. Click or tap here to enter text. (*Hint: it shouldn’t be too complicated. It should be just an expression that happens to evaluate to the number 80.*)

# Deliverable

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