**PowerShell**

Before you begin the tasks, review the following training materials and articles:

* [Getting Started with PowerShell](https://accenture.percipio.com/courses/e03ec747-ea51-11e6-965b-0242c0a80b04/videos/e03fffc6-ea51-11e6-965b-0242c0a80b04)
* [PowerShell Commands & Cmdlets](https://accenture.percipio.com/courses/e03fffc7-ea51-11e6-965b-0242c0a80b04/videos/e03fffc8-ea51-11e6-965b-0242c0a80b04)
* [PowerShell Pipeline, Objects, & Formatting](https://accenture.percipio.com/courses/e04074f5-ea51-11e6-965b-0242c0a80b04/videos/e04074f6-ea51-11e6-965b-0242c0a80b04)
* [Advanced Tools & Scripting with PowerShell 3.0 Jump Start](https://mva.microsoft.com/en-US/training-courses/advanced-tools-scripting-with-powershell-30-jump-start-8277?l=WOWaGUWy_8604984382)
* Docs about Select-Xml command in PowerShell
  + [Mastering everyday XML tasks in PowerShell](https://www.powershellmagazine.com/2013/08/19/mastering-everyday-xml-tasks-in-powershell/)
  + [PowerShell work with XML](https://blogs.technet.microsoft.com/heyscriptingguy/2012/03/25/the-scripting-wife-learns-to-use-powershell-to-work-with-xml/) and [Working with XML](https://blogs.technet.microsoft.com/heyscriptingguy/2013/04/01/working-with-xml/)
* [An introduction to JSON](https://www.digitalocean.com/community/tutorials/an-introduction-to-json)
* [PowerShell playing with JSON](https://blogs.technet.microsoft.com/heyscriptingguy/2015/10/08/playing-with-json-and-powershell/) and [save in JSON with PowerShell](http://neimke.blogspot.com/2015/11/using-powershell-to-work-with-json.html)

[PowerShell: Getting Started - PluralSight course](https://app.pluralsight.com/library/courses/powershell-getting-started/table-of-contents)

**NB! Create separate script files for each sub-task.**

* 1. Create XML file with the following content:

<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<body>

<h2>My CD Collection</h2>

<table border="1">

<tr bgcolor="#9acd32">

<th>Title</th>

<th>Artist</th>

</tr>

<tr>

<td>.</td>

<td>.</td>

</tr>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Create a PowerShell script that will create 10 copies of the XML file with a different Artist and Title (the value of the *td* tag)

1. Create JSON file with the following content:

{

"glossary": {

"title": "example glossary",

"GlossDiv": {

"title": "S",

"GlossList": {

"GlossEntry": {

"ID": "SGML",

"SortAs": "SGML",

"GlossTerm": "Standard Generalized Markup Language",

"Acronym": "SGML",

"Abbrev": "ISO 8879:1986",

"GlossDef": {

"para": "A meta-markup language, used to create markup languages such as DocBook.",

"GlossSeeAlso": [

"GML",

"XML"

]

},

"GlossSee": "markup"

}

}

}

}

}

Create a PowerShell script to update the “SortAs” value to “OMPL” in the JSON file. Check the file to make sure the value is updated.

1. Create JSON file with the following content:

[

{

"RgName": "DOTcom-dev-rg",

"Name": "dev",

"AppName": "devse-cd"

},

{

"RgName": "DOTcom-dev-rg",

"Name": "dev-cm",

"AppName": "devse-cm"

},

{

"RgName": "DOTcom-uat-rg",

"Name": "uat",

"AppName": "uatse-cd"

},

{

"RgName": "DOTcom-uat-rg",

"Name": "uat-cm",

"AppName": "uatse-cm"

}

]

In PowerShell script use a loop to iterate through JSON file and print out to console all keys and values equal to your defined parameter

1. Check the local file system via PowerShell script. If there is a file you expect, print something in the console, otherwise download any file from a trusted URL. Check both scenarios.
2. Create a PowerShell script that checks a variable. If the variable is *undefined* or *Null* or *Empty*, then, set a new value to that variable and print in the console, otherwise print in the console the original value of the variable.
3. Create a PowerShell script with the *switch* parameter. If the *switch* parameter is true, then execute Get-Disk cmdlet.
4. Create a main.ps1 and a secondary.ps1 PowerShell scripts. From the main.ps1 script call (execute) the secondary.ps1 script by passing two parameters. In the secondary.ps1 script execute some calculations using parameters passed from main.ps1 script.
5. Create PSM (PowerShell Module) that contains one simple function with parameters. Import this module with PowerShell script and execute the function (which is defined in PSM) by passing the parameter.
6. Create an array and fill it with values.
7. Create an object and fill it with values.