Power Shell 101

Module 6 Hands-on Activity – PowerShell in VSCode

5/16/2019 Developed by Sion Yoon
Center for Information Assurance (CIAE) at City University of Seattle

Learning Outcomes

- Learn about quotes, strings, arrays, and variable provider
- Learn how to use the pipeline and operators to work with XML

Resources

- Quotes, strings, and arrays
  - https://www.itprotoday.com/powershell/single-quotes-vs-double-quotes-powershell-whats-difference
  - https://www.youtube.com/playlist?list=PL6D474E721138865A
- Variable Provider
  - https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_variable_provider?view=powershell-6
- Working with XML
  - https://github.com/eXist-db/demo-apps/tree/master/data
  - https://mva.microsoft.com/en-us/training-courses/getting-started-with-powershell-3-0-jump-start-8276?l=TCQ8JQWy_404984382

Activities

- Quotes
- Strings
- Arrays
- Variable Provider
- Working with XML
- Q&A

Quotes

1. Open PowerShell in VSCode.

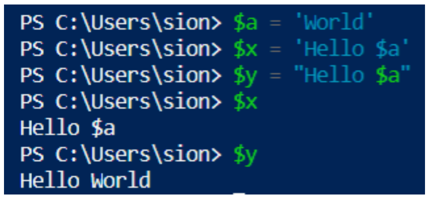
2. Last week, we learned about different operators and how to use variables to store objects. Let’s dive deeper into learning more about them this week. Create the following variables:

```
% $a = ‘World’
```

```
% $x = ‘Hello $a’
% $y = “Hello $a”
```

3. See the contents of $x and $y:

```
% $x

% $y
```

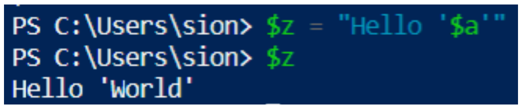


```
PS C:\Users\sion> $a = 'World'
PS C:\Users\sion> $x = 'Hello $a'
PS C:\Users\sion> $y = "Hello $a"
PS C:\Users\sion> $x
Hello $a
PS C:\Users\sion> $y
Hello World
```

You will see that using single quotes will take exactly what you typed and puts it into the variable as you see with $x. Using double quotes, on the other hand, PowerShell will look for the dollar sign character, and it assumes that any characters following the dollar sign up to the next whitespace are a variable name. It replaces the variable with its contents as you see with $y.

4. What if a variable is within a single quote that is within a double quote? For example, type the following:

```
% $z = “Hello‘$a’”

% $z
```

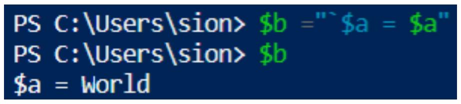


```
PS C:\Users\sion> $z = "Hello '$a'"
PS C:\Users\sion> $z
Hello 'World'
```

You will see that $a has been replaced with its content. The $a variable is within single quotes, but the outer quotations are what matters. In this case, PowerShell is not looking at the single quotes as quotation marks but literal characters.

5. Within double quotes, PowerShell also looks for escape characters. Type:

```
% $b = “`$a = $a”

% $b
```



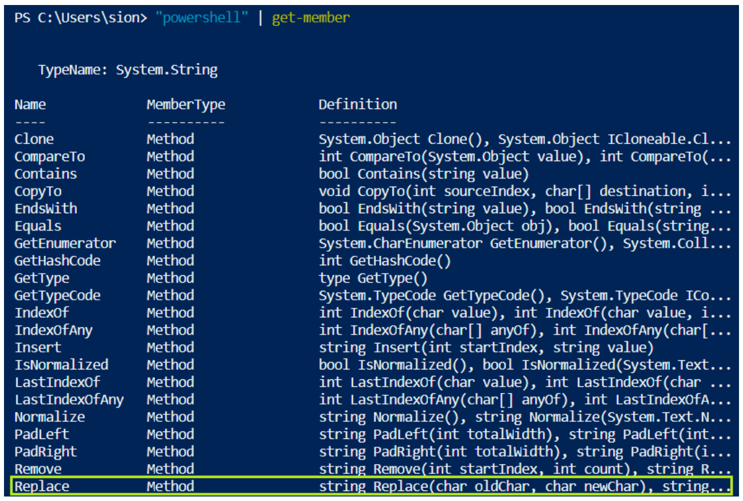
```
PS C:\Users\sion> $b ="`$a = $a"
PS C:\Users\sion> $b
$a = World
```

The escape character escapes the first dollar sign, so it no longer indicates a variable as shown in the example.

Strings

6. In PowerShell, a string value is also an object. If you pipe a string into get-member, you can see a list of different methods for the string value. Type:

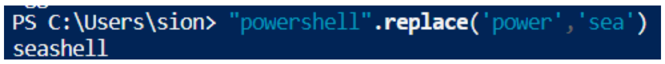
```
% "powershell" | get-member
```



```
PS C:\Users\sion> "powershell" | get-member


   TypeName: System.String

Name             MemberType            Definition
----             ----------            ----------
Clone            Method                System.Object Clone(), System.Object ICloneable.Cl...
CompareTo        Method                int CompareTo(System.Object value), int CompareTo(...
Contains         Method                bool Contains(string value)
CopyTo           Method                void CopyTo(int sourceIndex, char[] destination, i...
EndsWith         Method                bool EndsWith(string value), bool EndsWith(string ...
Equals           Method                bool Equals(System.Object obj), bool Equals(string...
GetEnumerator    Method                System.CharEnumerator GetEnumerator(), System.Coll...
GetHashCode      Method                int GetHashCode()
GetType          Method                type GetType()
GetTypeCode      Method                System.TypeCode GetTypeCode(), System.TypeCode ICo...
IndexOf          Method                int IndexOf(char value), int IndexOf(char value, i...
IndexOfAny       Method                int IndexOfAny(char[] anyOf), int IndexOfAny(char[...
Insert           Method                string Insert(int startIndex, string value)
IsNormalized     Method                bool IsNormalized(), bool IsNormalized(System.Text...
LastIndexOf      Method                int LastIndexOf(char value), int LastIndexOf(char ...
LastIndexOfAny   Method                int LastIndexOfAny(char[] anyOf), int LastIndexOfA...
Normalize        Method                string Normalize(), string Normalize(System.Text.N...
PadLeft          Method                string PadLeft(int totalWidth), string PadLeft(int...
PadRight         Method                string PadRight(int totalWidth), string PadRight(i...
Remove           Method                string Remove(int startIndex, int count), string R...
Replace          Method                string Replace(char oldChar, char newChar), string...
```

Let's try using the replace method. Using the definition provided, you can do the following:

```
% "powershell".replace('power','sea')
```



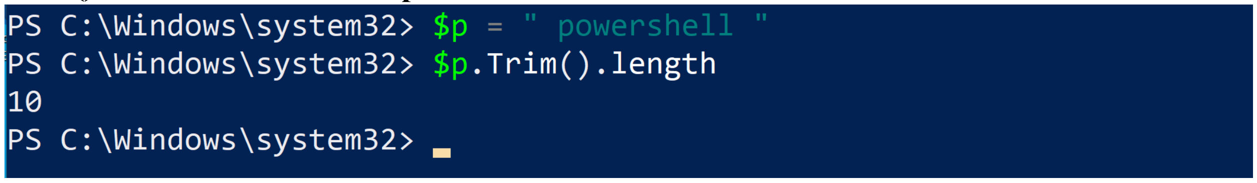
```
PS C:\Users\sion> "powershell".replace('power','sea')
seashell
```

Integers are also objects in PowerShell, and you can pipe it to get-member to see what methods you can use to manipulate the integer information.

Answer the following two questions by first creating a variable named "$p" that contains the string, " powershell ".

<u>Q&A #1</u>

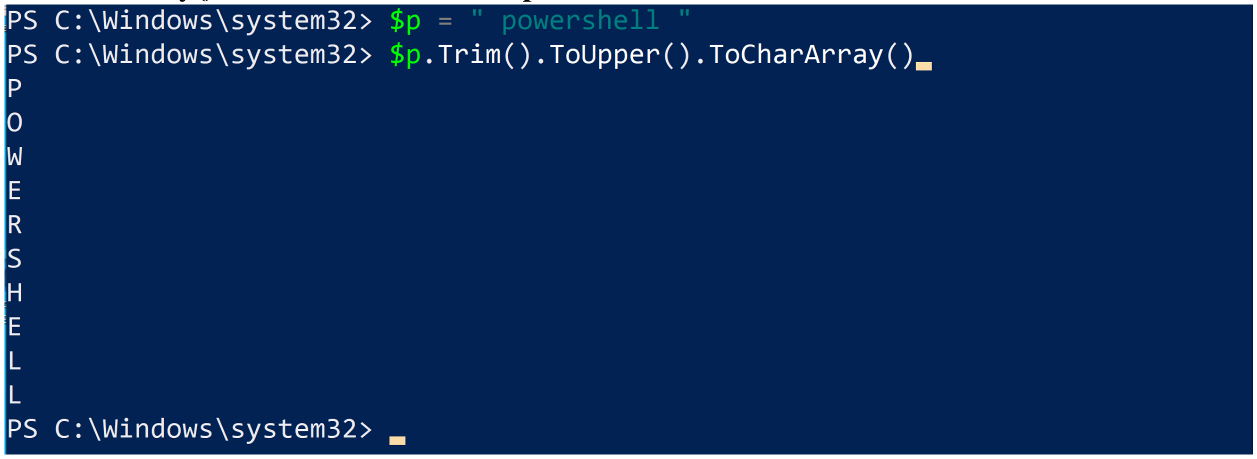
Using a single command, how would you remove all the leading and trailing whitespace characters and get the length of $p? Hint: one of the two methods you would need to use is Trim(). Provide the screen capture below:



```
PS C:\Windows\system32> $p = " powershell "
PS C:\Windows\system32> $p.Trim().length
10
PS C:\Windows\system32> _
```

<u>Q&A #2</u>

Using a single command, how would you remove all the leading and trailing whitespace characters, convert all the characters to uppercase, and convert the string value to an array of individual characters for $p? Hint: one of the three methods you would need to use is ToCharArray(). Provide the screen capture below:



```
PS C:\Windows\system32> $p = " powershell "
PS C:\Windows\system32> $p.Trim().ToUpper().ToCharArray()_
P
O
W
E
R
S
H
E
L
L
PS C:\Windows\system32> _
```

<u>Arrays</u>

7. To practice using arrays, let's first create a variable with get-service and look at the contents in the variable.

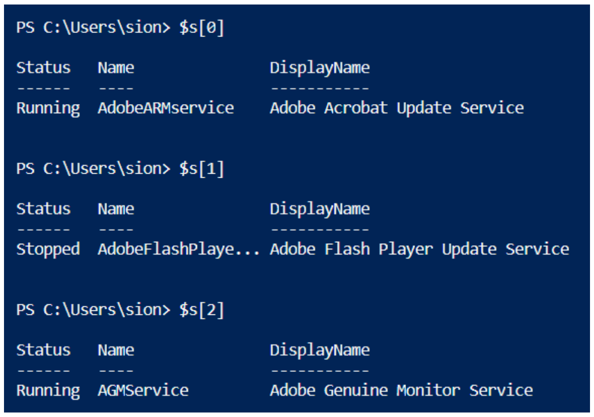
```
% $s = get-service

% $s
```

Using square brackets with an index, you can access the first, second, third, etc. item. Note that the very first character in a string is considered as index 0. Type:

% $s[0]

% $s[1]

% $s[2]

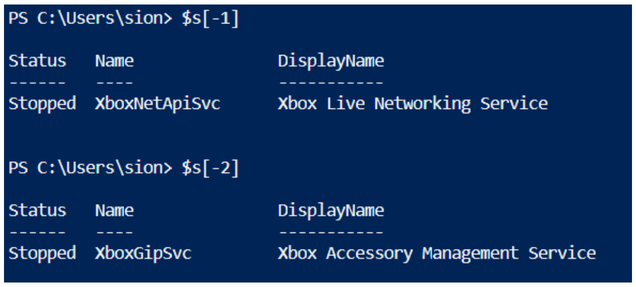


```
PS C:\Users\sion> $s[0]

Status   Name               DisplayName
------   ----               -----------
Running  AdobeARMservice    Adobe Acrobat Update Service


PS C:\Users\sion> $s[1]

Status   Name               DisplayName
------   ----               -----------
Stopped  AdobeFlashPlaye... Adobe Flash Player Update Service


PS C:\Users\sion> $s[2]

Status   Name               DisplayName
------   ----               -----------
Running  AGMService         Adobe Genuine Monitor Service
```

You can see that you were able to access the first, second, and third item from get-services. If you want to start from the end of the list, you would use negative numbers as follows:

% $s[-1]

% $s[-2]

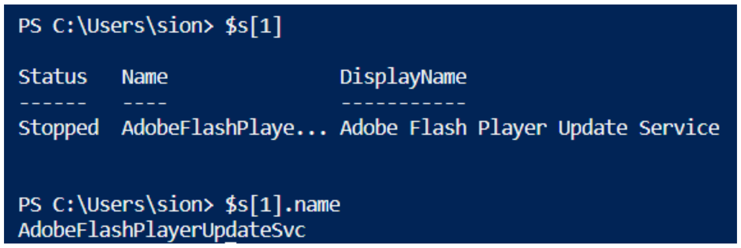


```
PS C:\Users\sion> $s[-1]

Status   Name               DisplayName
------   ----               -----------
Stopped  XboxNetApiSvc      Xbox Live Networking Service


PS C:\Users\sion> $s[-2]

Status   Name               DisplayName
------   ----               -----------
Stopped  XboxGipSvc         Xbox Accessory Management Service
```

Along with the squared brackets, a dot operator can be used to get an individual property or method for the item. Type:

% $s[1]

% $s[1].name



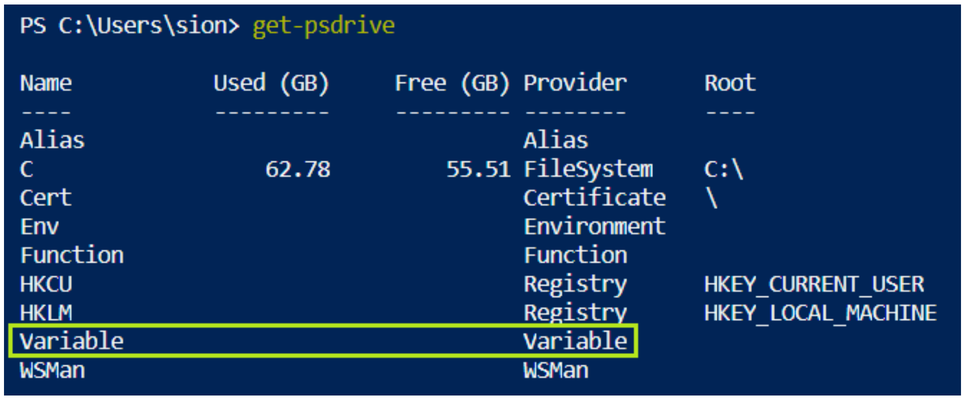
```
PS C:\Users\sion> $s[1]

Status   Name               DisplayName
------   ----               -----------
Stopped  AdobeFlashPlaye... Adobe Flash Player Update Service


PS C:\Users\sion> $s[1].name
AdobeFlashPlayerUpdateSvc
```

You will have more practice with this on the "Working with XML" section.

Variable Provider

8. Now, in one of our earlier hands-on-activities, do you remember when we learned about PSDrive? In case you do not remember, PowerShell providers connect different forms of storage to PowerShell and make the forms of storage look like a file system. If you look into PSDrive, you will see that there is a variable provider. The variable provider allows you to get, add, change, clear, and delete PowerShell variables in the current console. Type:

   % get-psdrive

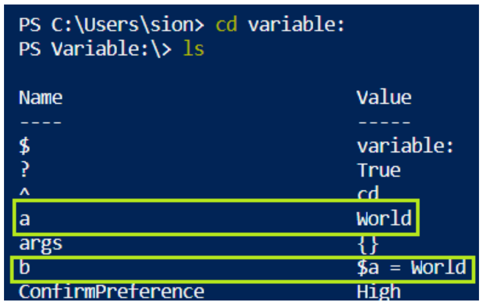


```
PS C:\Users\sion> get-psdrive

Name           Used (GB)     Free (GB) Provider      Root
----           ---------     --------- --------      ----
Alias                                  Alias
C                  62.78         55.51 FileSystem    C:\
Cert                                   Certificate   \
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
Variable                               Variable
WSMan                                  WSMan
```

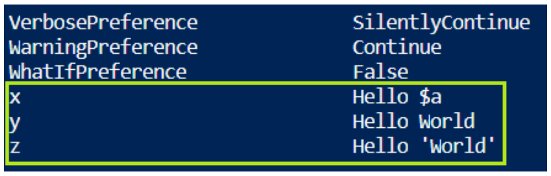
9. If you go into the variable drive, you will see all the variables that you just created as well as the existing variables. Type the following to go into the drive.

   % cd variable:

   % ls



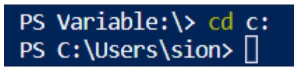
```
PS C:\Users\sion> cd variable:
PS Variable:\> ls

Name                           Value
----                           -----
$                              variable:
?                              True
^                              cd
a                              World
args                           {}
b                              $a = World
ConfirmPreference              High
```



```
VerbosePreference              SilentlyContinue
WarningPreference              Continue
WhatIfPreference               False
x                              Hello $a
y                              Hello World
z                              Hello 'World'
```

You can go back to the c: drive by typing:

```
% cd c:
```

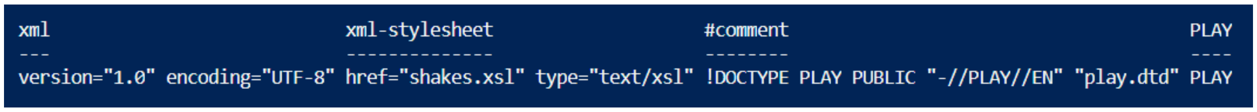


```
PS Variable:\> cd c:
PS C:\Users\sion> 
```

Working with XML

10. Download the zipped file that is attached (Module7_HandsOnActivity_XML). Save it into your username folder (ex: C:\Users\[username]>) and unzip the file. You will see an .XML file named "hamlet.xml". Open it with Notepad++ or on your browser (right-click anywhere on the content and click on "View source"). You will we the title, persona, act, scene, speech, etc. We will use PowerShell so that we can easily work with XML. First, let's cast the content as XML and store the object in a variable. Note that get-content gets the content of the item at the location specified by the path. Make sure you are in the directory where the hamlet.xml exists. Type:

```
% $h = [xml](get-content .\hamlet.xml)

% $h
```

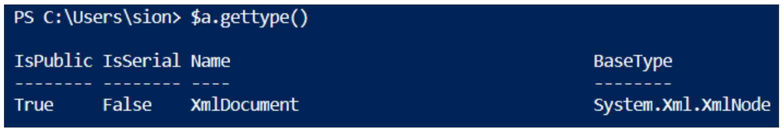


```
xml                               xml-stylesheet                      #comment                                          PLAY
---                               --------------                      --------                                          ----
version="1.0" encoding="UTF-8"  href="shakes.xsl" type="text/xsl"  !DOCTYPE PLAY PUBLIC "-//PLAY//EN" "play.dtd"  PLAY
```

You will now have an object that contains the XML.

11. Type the following to display the type of the object.

```
% $h.gettype()
```



```
PS C:\Users\sion> $a.gettype()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    XmlDocument                              System.Xml.XmlNode
```

12. You can confirm from this that it is an XML document. With this object, you can map through the contents easily by using the dot operator. Type:

    % $h.play

    % $h.play.act

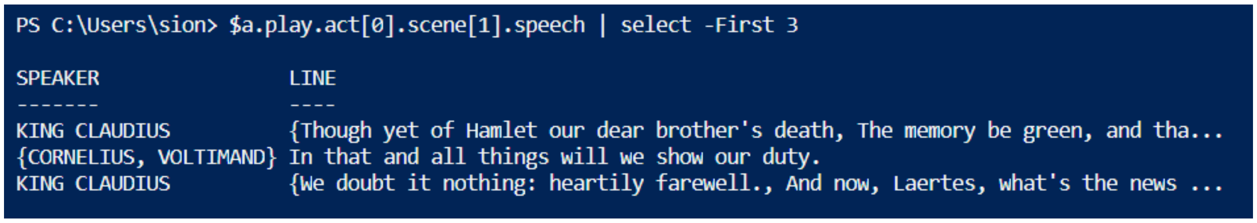
    Compare the results with the hamlet.xml file. It is hard to work with the .XML file but PowerShell allows you to get results of what you want to see, making is so much easier to work with.

13. Let's narrow the search more using the square brackets. What if you want to see all the speeches for act 1, scene 2. Type:

    % $h.play.act[0].scene[1].speech

14. You can also use the pipeline to select the first few speeches. Type:

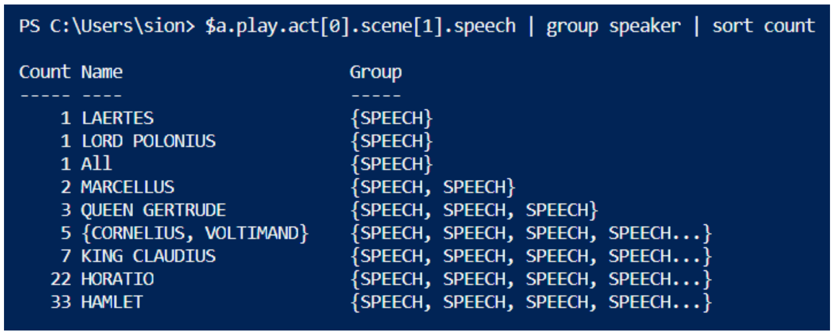
    % $h.play.act[0].scene[1].speech | select -First 3



```
PS C:\Users\sion> $a.play.act[0].scene[1].speech | select -First 3

SPEAKER               LINE
-------               ----
KING CLAUDIUS         {Though yet of Hamlet our dear brother's death, The memory be green, and tha...
{CORNELIUS, VOLTIMAND} In that and all things will we show our duty.
KING CLAUDIUS         {We doubt it nothing: heartily farewell., And now, Laertes, what's the news ...
```

15. Using the pipeline, you can also get a count of how many speeches each speaker has for the act and scene by grouping by speaker and sorting by count. Type:

    % $h.play.act[0].scene[1].speech | group speaker | sort count

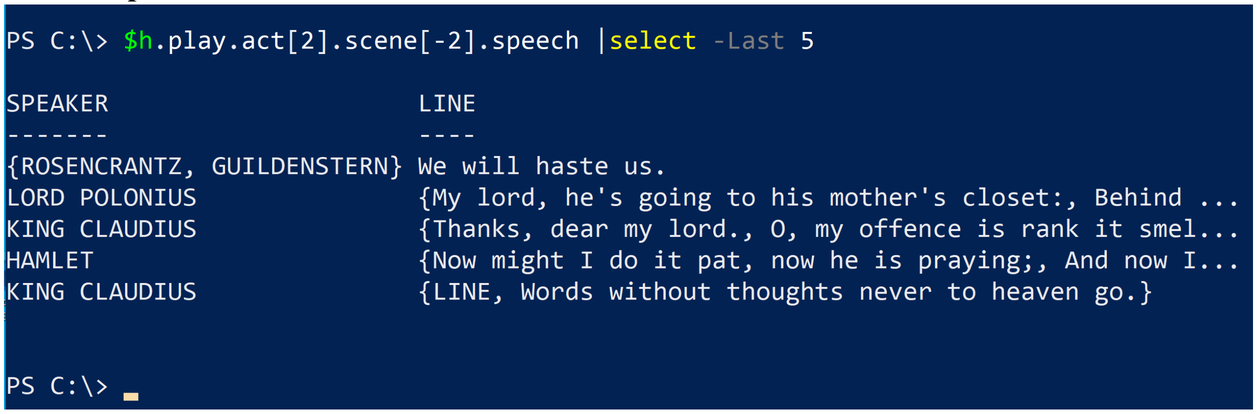


```
PS C:\Users\sion> $a.play.act[0].scene[1].speech | group speaker | sort count

Count Name                      Group
----- ----                      -----
    1 LAERTES                   {SPEECH}
    1 LORD POLONIUS             {SPEECH}
    1 All                       {SPEECH}
    2 MARCELLUS                 {SPEECH, SPEECH}
    3 QUEEN GERTRUDE            {SPEECH, SPEECH, SPEECH}
    5 {CORNELIUS, VOLTIMAND}    {SPEECH, SPEECH, SPEECH, SPEECH...}
    7 KING CLAUDIUS             {SPEECH, SPEECH, SPEECH, SPEECH...}
   22 HORATIO                   {SPEECH, SPEECH, SPEECH, SPEECH...}
   33 HAMLET                    {SPEECH, SPEECH, SPEECH, SPEECH...}
```

Use the $h variable that we created from the activity to answer the following questions.

Q&A #3

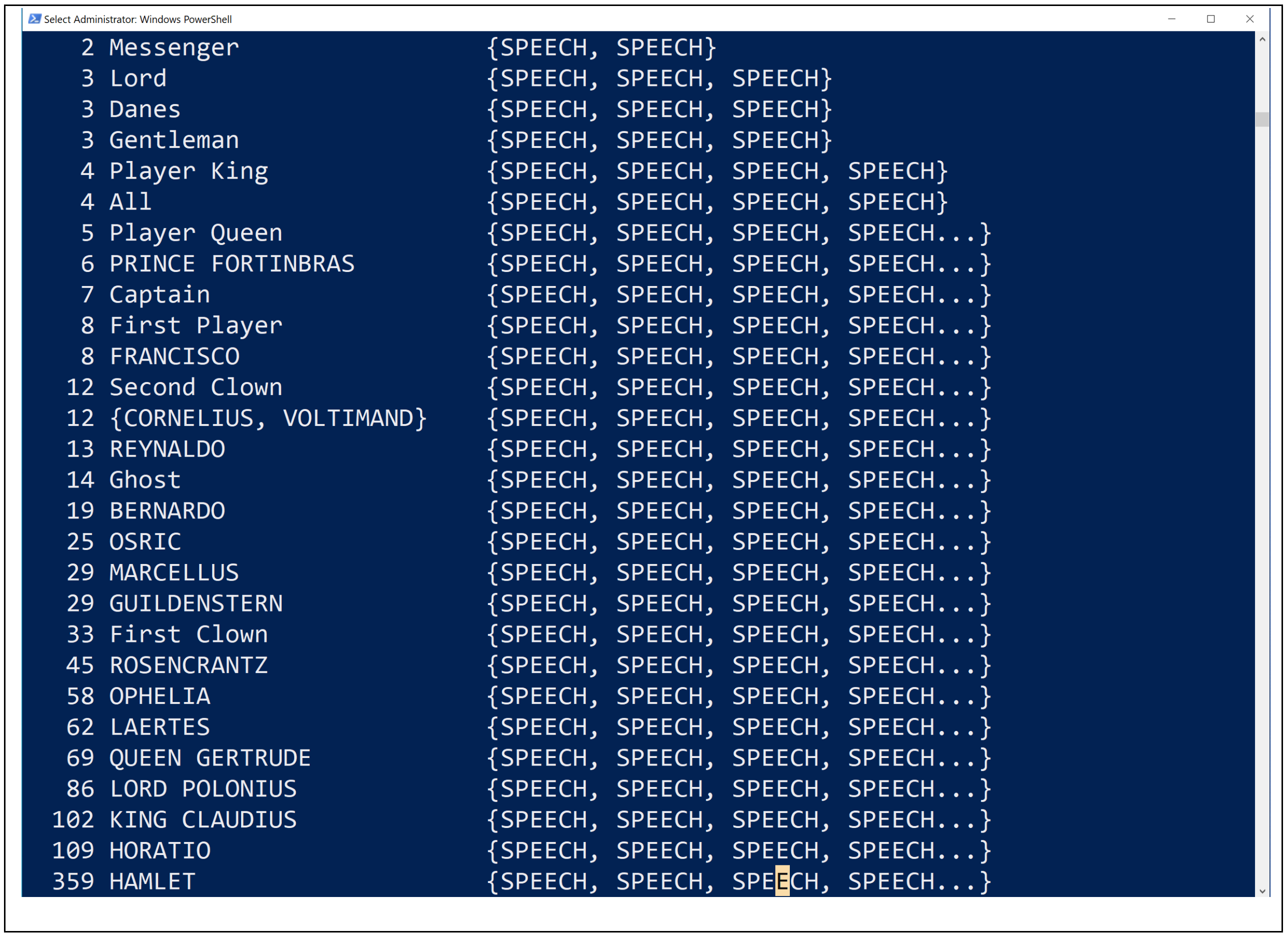
Write a command that executes the last five speeches for act 3 and 2nd to the last scene. Hint: you would need to use a negative number to get the 2nd to the last scene. Provide the screen capture below:



```
PS C:\> $h.play.act[2].scene[-2].speech |select -Last 5

SPEAKER                      LINE
-------                      ----
{ROSENCRANTZ, GUILDENSTERN} We will haste us.
LORD POLONIUS                {My lord, he's going to his mother's closet:, Behind ...
KING CLAUDIUS                {Thanks, dear my lord., O, my offence is rank it smel...
HAMLET                       {Now might I do it pat, now he is praying;, And now I...
KING CLAUDIUS                {LINE, Words without thoughts never to heaven go.}


PS C:\> _
```

Q&A #4

Get a count of how many speeches each speaker has for the entire play (all the acts and scenes). Provide the screen capture below:


Select Administrator: Windows PowerShell

```
   2 Messenger                  {SPEECH, SPEECH}
   3 Lord                       {SPEECH, SPEECH, SPEECH}
   3 Danes                      {SPEECH, SPEECH, SPEECH}
   3 Gentleman                  {SPEECH, SPEECH, SPEECH}
   4 Player King                {SPEECH, SPEECH, SPEECH, SPEECH}
   4 All                        {SPEECH, SPEECH, SPEECH, SPEECH}
   5 Player Queen               {SPEECH, SPEECH, SPEECH, SPEECH...}
   6 PRINCE FORTINBRAS          {SPEECH, SPEECH, SPEECH, SPEECH...}
   7 Captain                    {SPEECH, SPEECH, SPEECH, SPEECH...}
   8 First Player               {SPEECH, SPEECH, SPEECH, SPEECH...}
   8 FRANCISCO                  {SPEECH, SPEECH, SPEECH, SPEECH...}
  12 Second Clown               {SPEECH, SPEECH, SPEECH, SPEECH...}
  12 {CORNELIUS, VOLTIMAND}     {SPEECH, SPEECH, SPEECH, SPEECH...}
  13 REYNALDO                   {SPEECH, SPEECH, SPEECH, SPEECH...}
  14 Ghost                      {SPEECH, SPEECH, SPEECH, SPEECH...}
  19 BERNARDO                   {SPEECH, SPEECH, SPEECH, SPEECH...}
  25 OSRIC                      {SPEECH, SPEECH, SPEECH, SPEECH...}
  29 MARCELLUS                  {SPEECH, SPEECH, SPEECH, SPEECH...}
  29 GUILDENSTERN               {SPEECH, SPEECH, SPEECH, SPEECH...}
  33 First Clown                {SPEECH, SPEECH, SPEECH, SPEECH...}
  45 ROSENCRANTZ                {SPEECH, SPEECH, SPEECH, SPEECH...}
  58 OPHELIA                    {SPEECH, SPEECH, SPEECH, SPEECH...}
  62 LAERTES                    {SPEECH, SPEECH, SPEECH, SPEECH...}
  69 QUEEN GERTRUDE             {SPEECH, SPEECH, SPEECH, SPEECH...}
  86 LORD POLONIUS              {SPEECH, SPEECH, SPEECH, SPEECH...}
 102 KING CLAUDIUS              {SPEECH, SPEECH, SPEECH, SPEECH...}
 109 HORATIO                    {SPEECH, SPEECH, SPEECH, SPEECH...}
 359 HAMLET                     {SPEECH, SPEECH, SPEECH, SPEECH...}
```

PUSH YOUR WORK TO GITHUB

Once you completed the Hands-on practice, **export this document to PDF**, save in the same Module 6 folder you cloned, then do the following to push your work to GitHub

Open the terminal from the VSCode by hitting the control + ~ key, make sure you are in the right path, for example: KimNguyen/Desktop/ISEC505/HOP06-KimNguyenMai/Module 6

Type the following command:

>>> **git add .** (to copy all changes you have made)
>>> **git commit -m "Submission for Module 6 – YOUR GITHUB USERNAME"** (To add a message to your submission)
>>> **git push origin master** (to upload your work to Github)