GP Grundlagen Programmierung

PowerShell

Quelle: https://openclipart.org/detail/191890/powershell-icon

Kursunterlagen

Inhaltsverzeichnis

Identifikation und Änderungsgeschichte

Dokumenttitel: Kursunterlagen
Autor: Michael Graf
Dateiname: HandOut-GP-GrundlagenProgrammierung-v10.docx
Ablageort: https://rau8804-my.sharepoint.com/personal/michael_graf_r-au_ch/Documents/RAU/Reform_21 - Temp/1 Entwicklung Module/GP-GrundlagenProgrammierung/Lernende/HandOut-GP-GrundlagenProgrammierung-v10.docx

Version	Datum	Bemerkungen
1.0	19.10.2021	Initialversion / MG

1. Einleitung

Im folgenden Kapitel soll erläutert werden, welche Scriptsprachen verbreitet sind und was sie gemeinsam haben, welche Unterschiede zwischen Scriptsprachen und Programmiersprachen bestehen und welche Ziele in den nächsten Tagen verfolgt werden.

1.1 Ziele

Ziel dieser drei Tage sind ein erster Einblick in die Welt der Scriptsprachen. Das bedeutet, dass Sie zum Schluss dieser Ausbildungstage in der Lage sein sollen einfache Scripts selbst zu entwickeln. Das Entwickeln von Scripts gewährt einen ersten Einblick in die Entwicklung von Programmen und wie Aufgaben von Systemtechnikern später automatisiert werden können.

1.2 Was sind Scriptsprachen

Grundsätzlich können mit Scriptsprachen, wie oben erwähnt, Abläufe des Betriebssystems automatisiert oder eben ganze Programme realisiert werden. Wichtigster Unterschied zu anderen Programmiersprachen ist die Abhängigkeit von einem Interpreter. Während Sprachen wie C/C++ in Befehle umgewandelt werden, die von der CPU ausgeführt werden können, benötigt es für die Ausführung von Scriptsprachen einen Interpreter als Zwischenschicht. Dieser Interpreter liest die Befehle ein und führt sie aus.

Hier ein kurzer Überblick über verschiedene Scriptsprachen:

PowerShell	Befehle zur Administration von Windows-Systemen
Bash Script	Für die Automatisierung auf Linux-Systemen
Python	Sehr umfangreiche Sprache.
Perl	Sehr stark bei der Verarbeitung von Text
JavaScript	Hauptsächlich auf Webseiten verwendet (Client)
PHP	Hauptsächlich für Webseiten verwendet (Server)

1.3 Handlungsziele und Handlungsnotwendige Kenntnisse

Quelle: ICT-Berufsbildung Schweiz

Titel		**RAU: GP, PowerShell**
Bildungsplan		Applikationsentwicklung B3
Handlungs-kompetenz		B3.1: Entwickeln die Funktionalität benutzerfreundlich, z. B. löst die gleiche Funktion immer die gleiche Aktion aus, bei Blättern bleiben eingegebene Informationen erhalten usw.
Kompetenz		Kann die grundlegenden Elemente, die jeglicher Programmierung zugrunde liegen, in einem Programm umsetzen.
Objekt		Ausführbare Programme nach prozeduralen Ansätzen, nach Vorgaben erstellt.
Handlungsziele	3.	Ablaufstruktur und Daten mit einer Programmiersprache in ein Programm umsetzen.
	4.	Den Quellcode verständlich ausarbeiten (Variablennamen, Struktur, Kommentare), um die Nachvollziehbarkeit des Programms sicherzustellen.

Handlungsziel	**Handlungsnotwendige Kenntnisse (Ressourcen)**	
3.	1.	Kennt den grundlegenden Aufbau eines Programms (Positionierung von Deklaration und Verarbeitung usw.) und kann diesen Aufbau an einem Programm beispielhaft erläutern.
	2.	Kennt die grundlegenden Befehle einer Programmiersprache (Kontrollstrukturen, Operatoren) und kann aufzeigen, welche Verarbeitungsanweisungen damit realisiert werden können.
4.	1.	Kennt die wichtigsten Gliederungsmöglichkeiten (z.B. einrücken von geschachtelten Kontrollstrukturen) und kann erläutern, welchen Beitrag diese zur Lesbarkeit von Programmen leisten.
	2.	Kennt die wichtigsten Regeln für einen verständlichen Quellcode (sprechende Variablenbezeichnungen, geeignete Kommentare) und kann erläutern, welchen Beitrag diese Angaben zur Lesbarkeit von Programmen leisten.

Handlungsnotwendige Kenntnisse beschreiben Wissenselemente, die das Erreichen einzelner Handlungsziele eines Moduls unterstützen. Die Beschreibung dient zur Orientierung und hat empfehlenden Charakter. Die Konkretisierung der Lernziele und des Lernwegs für den Kompetenzerwerb sind Sache der Bildungsanbieter.

2. Die Entwicklungsumgebung ISE

ISE steht für Integrated Scripting Environment. Ziel dieser ISE ist es also, alle Elemente in einer Software zu sammeln. Bei anderen Programmiersprachen wird von einer IDE (Integrated Development Environment) gesprochen.

Die PowerShell ISE hat drei wesentliche Elemente:

Scriptbereich	Darin werden die Scripts geschrieben. Unterstützt durch Syntax Highlighting.
Konsolenbereich	Die Scripts werden ausgeführt und Befehle können ausprobiert werden.
Befehls-AddOn	Bietet Informationen zu den verfügbaren Befehlen.

Nun sollen Sie sich mit einigen kurzen Übungen mit den einzelnen Elementen vertraut machen.

	PowerShell	Erste Schritte, grundlegende Konzepte Was ist PowerShell?
	Lernvideo	Einführung ISE Integrated Scripting Environment

2.1 Konsolenbereich

Der Konsolenbereich erinnert stark an die klassische Kommandozeile und wird auch ähnlich genutzt. Im Gegensatz zur Kommandozeile kann hier jedoch mit PowerShell gearbeitet w erden, was wesentlich umfangreichere Möglichkeiten bietet.

A1	Zum Einstieg sollen Sie sehen, welche Befehle ihnen überhaupt zur Verfügung stehen. Benutzen Sie dazu den Befehl **Get-Command** und Sie erhalten eine lange Liste.
A2	In dieser Liste werden sie auch verschiedene Typen von Befehlen erkennen. Erklären Sie kurz wo die Unterschiede liegen.
A3	Nun sollen Sie einen Befehl herausnehmen, um etwas mehr darüber zu erfahren, z.B. wie man ihn einsetzt. Dazu können Sie den Befehl **Get-Help <Befehlsname>** nutzen. Nun sollte Ihnen angezeigt werden, wie Sie den Befehl aufrufen müssen und welche Informationen Sie übergeben können. Achten Sie dabei auf den Unterschied von Elementen in eckigen Klammern [] und spitzen Klammern <>. Elemente zwischen [] sind optional und müssen nicht zwingen angegeben werden.

Üben Sie nun den Umgang mit der Konsole in dem Sie verschiedene Files erstellen, bewegen und wieder löschen. Dazu benötigen Sie die folgenden Kommandos:

- **Get-ChildItem** (dir als Alias) Zeigt den Ordnerinhalt an.
- **Copy-Item** (cp als Alias) Kopiert ein File oder Ordner von einem Ort zum anderen.
- **New-Item** (ni als Alias) Erstellt ein File oder Ordner.
- **Set-Location** (cd als Alias) Wechselt in einen angegebenen Ordner.
- **Remove-Item** (rm als Alias) Löscht ein File oder Ordner.
- **Rename-Item** (ren als Alias) Zur Umbenennung eines Files oder eines Ordners.
- **Move-Item** (mv als Alias) Verschieben eines Files oder eines Ordners.

Zur Übung sollen Sie in Ihrem Benutzerverzeichnis die folgende Dateistruktur erstellen.

```
Set-Location C:\Users\<Benutzername>
```

A4	Erstellen Sie folgenden Baum:
	• INs
	  ○ Windows
	  ▪ Installationsprotokoll.docx
	  ▪ Lizenzen.docx
	  ○ Linux
	  ▪ syslog
	  ▪ .bashrc
	  ○ Netzwerk
	  ▪ Netzwerkplan_physisch.vsdx
	  ▪ Netzwerkplan_logisch.vsdx
	Notieren Sie sich, welche Befehle Sie in welcher Reihenfolge benutzt haben.

A5	Nun verschieben, löschen oder kopieren Sie die Files so, dass Sie folgenden Baum erhalten.
	• INs
	  ○ RAU-Informatik
	  ▪ Lizenzen.docx
	  ▪ syslog
	  ▪ .bashrc
	  ○ Linux
	  ▪ Shellscript.sh
	  ▪ Netzwerkplan_logisch_Kopie.vsdx
	  ○ Netzwerk
	  ▪ Netzwerkplan_physisch.vsdx
	  ▪ Netzwerkplan_logisch.vsdx
	Notieren Sie sich, welche Befehle Sie in welcher Reihenfolge benutzt haben.

A6	Benutzen Sie nun einen einzigen Befehl, um alle Files anzuzeigen.

2.1.1 Pipeing

Ein wichtiges Konzept bei der Arbeit mit PowerShell, aber auch in Shell Scripts unter Linux Systemen ist das Pipeing. Dabei geht es darum, die Ausgabe des einen Befehls, wie durch eine Röhre (Pipe) in den nächsten zu leiten.

```
Get-Service Spooler | Restart-Service
```

So können auch mehrere zurückgegebene Objekte behandelt werden. Z.B. für Get-ChildItem

```
Get-ChildItem | Get-ItemProperty -Name LastWriteTime
```

Eine wichtige Funktion ist das Filtern bzw. gezielte Ausgeben von Objekten und deren Eigenschaften. Mit Where-Object wird gefiltert, welche Objekte angezeigt werden und mit Select-Object wird definiert, welche Eigenschaften angezeigt werden sollen.

z.B. `Get-ChildItem | Where-Object {$_.Name -like "*.exe"} | Select-Object Name`

A7	Benutzen Sie den Befehl Get-ChildItem um den aktuellen Ordnerinhalt anzuzeigen. Vergleichen Sie anschliessend das Resultat mit und ohne Filter.

2.2 Scriptbereich

Nun da Sie die grundlegende Anwendung von PowerShell geübt haben, wollen wir versuchen diese Kenntnisse bei der Erstellung eines Scripts zu repetieren und zu vertiefen. Einfache Scripts können sehr schnell erstellt werden, in dem die genutzten Befehle nacheinander in die Script-Datei eingetragen werden. Dies soll nun mit je einem Script für die Aufgaben A4 und A5 ausprobiert werden.

A8	Erstellen Sie nun ein Script zu A4, welches den gesamten Verzeichnisbaum erstellt. Der Befehl **Get-History** kann dabei sehr hilfreich sein.
A9	Nun erstellen Sie ein Script zu A5, welches die Mutationen im Baum vornimmt.
A10	Bevor Sie diese Scripts ausführen können, müssen Sie die Ausführung von Scripts auf dem System freischalten. Welcher Befehl wird dazu genutzt und welche verschiedenen Optionen können gewählt werden? Schreiben Sie dazu eine kurze Anleitung inkl. der Bedeutung dieser verschiedenen Varianten.
A11	Führen Sie nun ihre Scripts nacheinander mit der Taste **F5** oder den dazu verwendeten Schaltflächen aus. 

3. Operationen und Variablen

Operatoren sind wohlbekannt aus der Mathematik. Natürlich wird bei der Mathematik hauptsächlich über die arithmetischen Operatoren gesprochen. In der Umgebung von Programmieren und Scripten gibt es jedoch noch weitere. Gestartet wird jedoch mit Bekanntem.

3.1 Arithmetische Operatoren

A12	Für den Einstieg in das Thema Operatoren, lösen Sie die untenstehenden Rechnungen mithilfe der PowerShell Konsole. 67 + 5 = 46 – 9 = 7 * 12 = 184 / 4 = 167 % 3 = Was bedeutet dieses Prozentzeichen?
A13	Interessant ist, dass nicht nur Zahlen „addiert“ werden können, sondern auch Buchstaben oder Wörter und hier ist nicht die Rede von Algebra. Versuchen Sie es aus und notieren, welche Ausgabe auf dem Bildschirm erscheint: “abc“ * 3 : „hallo „ + „welt“ :

3.2 Zuweisungsoperator und Variablen

Wie Sie oben vielleicht bereits bemerkt haben, ist es nicht nötig bzw. sogar falsch, wenn Sie das Gleichheitszeichen verwenden für Rechnungen. Ihr System erwartet mehr. Das Gleichheitszeichen besitzt eine besondere Funktion. Es ist dazu da Werte zuweisen zu können.

Diese errechneten oder erhaltenen Werte, werden einer Variable zugewiesen. Diese Variablen können verschiedenste Werte speichern, um sie später wieder abrufen zu können. In PowerShell macht es dabei keinen Unterschied, ob es kleine Zahlen, grosse Zahlen oder gar ganze Sätze sind. Es wird selbst erkannt, wie es mit den gespeicherten Informationen umzugehen hat oder gibt eine Fehlermeldung, wenn die gespeicherten Werte nicht zur Situation passen. Passen Sie aber auf, wenn Sie mit anderen Programmiersprachen entwickeln. Es gibt Sprachen, in denen Sie genau definieren müssen, welche Art von Daten eine Variable enthält.

Es folgt ein kleines Beispiel anhand dessen die Variablen und deren Nutzung erklärt werden soll:

```
$zahl1 = 3 # Der Wert 3 wird der Variabel $zahl1 zugewiesen
$zahl2 = 1+4 # Das Resultat von 1 + 4 wird der Variabel $zahl2 zugewiesen

$resultat = $zahl1 * $zahl2 # Die Variablen können nun gelesen werden und das Resultat
wird in eine weitere gespeichert.

Write-Output $resultat # Das Resultat wird auf die Konsole geschrieben
```

Wenn wir uns das Beispiel oben ansehen, sehen wir die Zuweisung von Werten zu ihren Variablen mithilfe des Gleichheitszeichens und wir sehen ebenfalls wie diese Werte wieder gelesen werden können. Dabei verhalten sich Variablen wie die darin gespeicherten Werte.

Auch wichtig sind Kommentare. Sie dienen zur Dokumentation innerhalb des Codes und unterstützen so Andere dabei, Ihren Code zu verstehen. Nicht zu vergessen ist auch, dass Sie Ihren Code nach längerer Zeit wieder verstehen möchten. Dabei helfen Kommentare ungemein. Kommentare werden mit # eingeleitet und gelten bis zum Ende der Zeile.

3.3 Inkrement und Dekrement

Für viele Operationen in Verbindung mit dem Zuweisungsoperator („=“) gibt es Abkürzungen und Vereinfachung.

A14	Was wird nach dem folgenden Codeausschnitt auf die Konsole geschrieben? Informieren Sie sich über die gesehenen Operatoren und versuchen Sie die Lösung zu finden.
	`$zahl1 = 25`
	`$zahl2 = 50`
	
	`$zahl2 -= 25`
	`$zahl2 = $zahl2 + $zahl1--`
	`$zahl2++`
	`$zahl2 += ++$zahl1`
	
	`Write-Output $zahl2 $zahl1`

A15	Was wird hier auf der Konsole ausgegeben?
	`$text1 = "Das ist "`
	`$text2 = "richtig!"`
	
	`$ausgabe = $text1`
	`$ausgabe += $text2`
	
	`Write-Output $ausgabe`

3.4 Vergleiche und deren Operatoren

Nun kommt ein weiterer Schritt zu einem echten Programm hinzu. In beinahe jedem Programm wird der weitere Verlauf durch die Werte von Variablen bestimmt. Sie sollen jetzt lernen, wie diese Werte überprüft werden und wie der weitere Verlauf damit kontrolliert werden kann.

Ein Vergleich gibt immer ein Resultat zurück, ob der Vergleich stimmt (True) oder nicht (False). Zudem können Vergleiche auch miteinander verknüpft werden, um gleich mehrere Vergleiche zu überprüfen. Es gibt sehr viele Vergleichsoperatoren, die folgend kurz ausgeführt werden:

-eq	Equals (statt dem in der Mathematik üblichen „=“)
-ne	Not Equal
-lt	Less Than
-le	Less or Equal
-gt	Greater Than
-ge	Greater or Equal

Für Wörter oder ganze Sätze, werden gerne auch andere Operatoren benutzt, wie sie unten kurz beschrieben werden. Der Grund dafür ist, dass bei –eq die Zeichenkette genau übereinstimmen muss und keine Wildcards unterstützt werden.

-like	Nutzt Wildcards wie * und ?
-notlike	
-match	Nutzt Regular Expressions
-notmatch	

Da Regular Expressions in sich schon diese drei Tage füllen würden, liegt die Aufmerksamkeit auf den sogenannten Wildcards. Wobei „*“ für eine beliebige Anzahl beliebiger Zeichen steht und „?“ steht für ein beliebiges Zeichen.

A16	Geben Sie für die folgenden Vergleiche der Zahlen an, ob diese stimmen oder nicht. Schreiben sie dazu das Resultat (True oder False) 7 –ne 7 : 5 –eq 5 : 4 –lt 8 : 9 –le 9 : 6 –gt 10 :
A17	Nun bestimmen Sie das Resultat bei Vergleichen von verschiedenen Zeichenketten. „RAU“ –like „?AU“ „AU“ –like „?AU“ „Dampfwalze“ –notlike „Dam*lze“

Diese Vergleiche können mit logischen Operatoren miteinander verknüpft werden, um komplexere Überprüfungen durchführen zu können.

-and	Beide Vergleiche müssen True ergeben
-or	Einer oder beide Vergleiche müssen True ergeben
-xor	Genau ein Vergleich ergibt True
-not (!)	Ändert Resultat eines Vergleichs

A18	Was ergeben nun also folgende verknüpften Vergleiche? $zahl = 13 $wort = „Informatiker“ $zahl –eq 4 –xor $zahl –gt 7 $zahl –le 13 –and $zahl –gt 3 $wort –like „Info“ –or $wort –notlike „*matiker“ !($wort –like „Info“) –and $zahl –ne 3

3.5 If-Statement

Wie können diese Vergleiche nun zur Kontrolle des Programm- bzw. Scriptablaufs verwendet werden? Dazu wird ein If-Statement verwendet. Beachten Sie dazu das kurze Beispiel und die enthaltenen Kommentare.

```
$zahl = 7                                       # Eine einfache Zahl wird zum Vergleich definiert

if ($zahl -eq 5){                                # Zuerst wird geprüft, ob die Zahl gleich 5 ist.
    Write-Output "Die Zahl ist 5."               # Ist dies der Fall (True) wird es auf die Konsole geschrieben
}elseif($zahl -gt 5){                            # elseif ermöglicht einen zweiten Vergleich
    Write-Output "Die Zahl ist grösser als 5."
}else{                                          # der Code für else wird aufgerufen, wenn der Vergleich False war
    Write-Output "Die Zahl ist kleiner als 5."
}
```

Bei diesem Beispiel gilt es zu beachten, dass elsif oder else nicht zwingend nötig sind. Die Einrückung innerhalb der geschweiften Klammern dient dazu, die Lesbarkeit des Programms zu verbessern.

A19	Übernehmen Sie das Beispiel in die PowerShell ISE und führen Sie es mit verschiedenen Werten für $zahl aus.
A20	Schreiben Sie ein Script, welches überprüft, ob der Computer den richtigen Namen trägt. Der Name ihres Computers sollte dem folgenden Muster entsprechen: ELBINXX. Dieser Name ist auf dem Gerät selbst festgehalten. Nun ist der Name nicht zwingend korrekt. Schreiben Sie eine kurze Ausgabe auf die Kommandozeile. Den Computernamen erhalten Sie mit dem Befehl hostname.

4. Benutzereingaben

Der nächste Schritt ist es die erstellten Scripts interaktiver zu gestalten und dem Benutzer der Scripts eine Einflussmöglichkeit zu geben. Informationen können auf verschiedene Arten vom Benutzer erhalten werden.

4.1 Parameter beim Scriptaufruf

Wie bei verschiedenen Befehlen bereits zu erkennen, können Informationen wie das zu kopierende File als Parameter übergeben werden. Diese Möglichkeit kann auch innerhalb eines Scripts genutzt werden, indem wir alle Eingaben zuerst als Variablen speichern. Hier muss jedoch bekannt sein, welche Art von Informationen gespeichert werden soll.

```
Param(
    [int]$zahl = 3 # [Typ]Name = Standardwert
)

if($zahl -eq 3){
    Write-Output "Richtig"
}else{
    Write-Output "Falsch"
}
```

In diesen Param Bereich können nun alle nötigen oder möglichen Parameter eingetragen werden. Wird kein Wert geliefert, wird ein optionaler Standardwert gewählt.

Für diese Parameter können verschiedene Typen gewählt werden. Die grundlegendsten sind sicher Zeichenketten, Zahlen und Switches.

[string]	Eine Kette von Zeichen, also Wörter, Sätze usw.
[int]	Integer oder Ganzzahl.
[float]	Kommazahlen
[switch]	Erhalten keinen Wert. Wird auf True gesetzt, wenn der Parameter genannt wird.

A21	Übernehmen Sie nun ihr Script aus A19 und ändern es so ab, dass die zu vergleichende Zahl beim Aufruf übermittelt wird.
A22	Ergänzen Sie das Script und geben Sie die Möglichkeit einen Namen zu übergeben. Die Ausgabe sollte dann etwa wie folgt aussehen: *Christian fragt, ob 1 gleich 5 ist.*
A23	Erstellen Sie einen Switch-Parameter *lang*. Nur wenn dieser angegeben wird, soll überprüft werden, ob die Zahl kleiner oder grösser ist. Wird *lang* nicht angegeben, soll lediglich überprüft werden, ob die Zahlen übereinstimmen.

4.2 Eingabe während Ausführung

Zusätzlich zu den Parametern können vom Benutzer gezielt Informationen erfragt werden, wenn das Script bereits ausgeführt wird. Dies könnte in etwa wie folgt aussehen.

```
$name = Read-Host -Prompt "Wie lautet ihr Name?"
Write-Output "Hallo $name"
```

A24	Ändern Sie das Script aus A21 so ab, dass die Zahl und der Name nacheinander erfragt werden und nicht mehr als Parameter übergeben werden.
A25	Schreiben Sie ein Script, bei dem zwei Benutzer mit ihrem Namen eine Zahl eingeben können. Danach wird überprüft, wer sich näher an einer Zahl zwischen 1 und 100 befindet. Für Zufallszahlen kann der Befehl Get-Random benutzt werden. Prüfen Sie die korrekte Anwendung zuerst mit Get-Help.

5. Schleifen

Oft müssen in Programmen dieselben Dinge wiederholt werden. Meist für verschiedene Elemente. Dies wird mit sogenannten Schleifen umgesetzt.

5.1 For

For Schleifen werden genutzt, um etwas während einer bestimmten Anzahl von Wiederholungen auszuführen. Z.B. um von 0 bis 10 zu zählen. Das For Statement besteht dabei aus drei Teilen. Einer Initialisierung (meistens die Zählvariable), einem Vergleich und einer Operation, die nach jedem Durchlauf ausgeführt wird.

```
for($i=0;$i -le 10; ++$i){
    Write-Output $i
}
```

5.2 While

While Schlaufen werden dazu genutzt etwas auszuführen bis ein bestimmter Zustand nicht mehr erfüllt ist. So kann von einer Datei so lange gelesen werden, solange es etwas zu lesen gibt.

```
$input = "weiter"
while($input -ne "exit"){   # Wiederholt sich, bis exit eingegeben wurde
    $input = Read-Host "Weiter oder exit?:"
}
```

5.2.1 Do While

Mit Do While kann sichergestellt werden, dass eine Schleife mindestens einmal durchlaufen wird.

```
do{ # Wiederholt sich, bis exit eingegeben wurde
    $input = Read-Host "Weiter oder exit?:"
}while($input -ne "exit")
```

5.2.2 Do Until

Do Until ist beinahe dasselbe. Es wird jedoch ausgeführt bis die Bedingung erfüllt ist. So muss in unserem Beispiel das –ne in ein –eq geändert werden.

```
do{ # Wiederholt sich, bis exit eingegeben wurde
    $input = Read-Host "Weiter oder exit?:"
}until($input -eq "exit")
```

5.3 ForEach

ForEach wird benötigt um für jedes Element z.B. in einem Array einen Durchlauf zu machen. Arrays sind spezielle Variablen, welche mehrere Werte beinhalten. So kann z.B. leicht für jedes File in einem Ordner eine Aktion ausgeführt werden.

```
$files = Get-ChildItem

foreach($file in $files){
    Write-Output "File gefunden $file"
}
```

5.4 Aufgaben

A26	Schreiben Sie ein Script, das den Parameter –prefix entgegennimmt und diesen Text vor jeden Filenamen im aktuellen Ordner hängt. Beispiel: *Zusammenfassung_PowerShell.docx* wird mit dem Präfix ***RAU-*** zu ***RAU-****Zusammenfassung_PowerShell.docx*
A27	In dieser Aufgabe soll man seinen eigenen praktischen Notenrechner zusammenbauen. Damit man selbst nicht mehr so mühsam rechnen muss, wird die Eingabe auf zwei Werte beschränkt: Die maximale Punktzahl, die in der Prüfung zu erreichen ist und die erreichte Punktzahl in der Prüfung. Im Hintergrund wird eine Rechnung aufgebaut, die die beiden Variablen zu einem Wert zusammenzählt und mit einer If-Abfrage das Resultat der Note ausgibt. Berechnung Note: $\frac{erreichte\ Punktezahl}{maximale\ Punktezahl} \cdot 5 + 1$ Beispiel: 45 erreichte Punkte und 60 maximale Punkte ergibt die Note 4.75. Man soll ebenfalls entscheiden können, ob die Rechnung auf die gerundete Note ausgeführt wird und auch ob man eine weitere Note berechnen möchte.

6. Funktionen

Funktionen werden genutzt, um Programmteile, die an verschiedenen Orten im Code genutzt werden, zu definieren. So kann die Funktion einmal geschrieben werden und überall im Code verwendet werden. Programmcode der in einer Schleife genutzt wird, wird oft ausgeführt aber nur einmal geschrieben. Funktionen eignen sich also nur bedingt um diese zu vereinfachen. Die Lesbarkeit und das Verständnis des Programms kann jedoch klar verbessert werden. Es geht auch darum, die Redundanz im Code zu minimieren, damit ein Fehler auch nur an einer Stelle behoben werden muss oder zukünftige Erweiterungen schnell umgesetzt werden können.

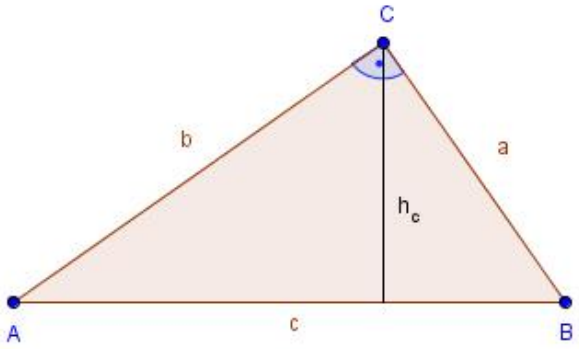
Funktionen übernehmen einen Übergabewert (Parameter) und geben falls nötig, einen Wert zurück. Ein einfaches Beispiel ist eine Funktion, welche eine Summe bildet.

```
function sum($zahl1, $zahl2){
    return $zahl1 + $zahl2
}

$ausgabe = sum 1 2

Write-Output $ausgabe
```

Eine Funktion muss immer oberhalb des Aufrufs definiert werden, damit die Funktion auch bekannt ist. In diesem Beispiel werden die zwei Zahlen übergeben und die Summe mit return zurückgegeben. Übergabewerte und Rückgabewert sind optional.

A28	Erstellen Sie ein Script, welches zur Berechnung von Dreiecken dient. Erstellen Sie in einer Schleife ein Menu, welches zur Auswahl stellt, ob die Fläche (A), Höhe (h_c) oder die Länge der Hypotenuse (c) berechnet werden soll. Gehen Sie davon aus, dass es sich jeweils um ein rechtwinkliges Dreieck handelt und der Benutzer die Längen der Seiten a und b übermittelt. Lagern Sie jede Berechnung als Funktion aus. Der Zeitpunkt und die Art, wie die Kantenlängen erfasst werden, ist Ihnen überlassen. In welchem Teil des Programms die Ausgabe ausgeführt wird, ist auch Ihnen überlassen. 
A29	Erstellen Sie ein ähnliches Programm. Dieses Mal haben Sie jedoch die Länge (a) und die Breite (b) eines Rechtecks. Ihr Programm soll Funktionen zur Berechnung der Fläche (A), der Diagonale (c) und des Umfangs (U) liefern.

A30	**Niveau 1:** Schreiben Sie ein Skript, welches den Benutzer eine Zahl erraten lässt. Nach jedem Versuch soll mitgeteilt werden, ob die gesuchte Zufallszahl höher oder tiefer ist. Sobald die geratene Zahl übereinstimmt, wird mit einer Gratulation beendet. Versuche sie die Aufgabe mithilfe einer Funktion umzusetzen. Eine Zahl zwischen 1 und 100 erraten.
A31	**Niveau 2:** Schreiben Sie ein Skript, welches den Benutzer eine Zahl erraten lässt. Nach jedem Versuch soll mitgeteilt werden, ob die gesuchte Zufallszahl höher oder tiefer ist. Sobald die geratene Zahl übereinstimmt, wird mit einer Gratulation beendet. Versuche sie die Aufgabe mithilfe einer Funktion umzusetzen. Im Skript soll zwischen drei verschiedenen Schwierigkeitslevel ausgewählt werden können. Ein Level von 1-10, 1-100 und von 1-1000.
A32	**Niveau 3:** Schreiben Sie ein Skript, welches den Benutzer eine Zahl erraten lässt. Nach jedem Versuch soll mitgeteilt werden, ob die gesuchte Zufallszahl höher oder tiefer ist. Sobald die geratene Zahl übereinstimmt, wird mit einer Gratulation beendet. Versuche sie die Aufgabe mithilfe einer Funktion umzusetzen. Im Skript soll zwischen drei verschiedenen Schwierigkeitslevel ausgewählt werden können. Ein Level von 1-10, 1-100 und von 1-1000. Dazu sollte in der Gratulation ersichtlich sein wie viele Versuche der Benutzer gebraucht hat.

7. Theorie Abschlussprojekt

7.1 Benutzer erstellen

Name	Erklärung
net user	Wenn dieser Befehl alleine steht, zeigt er an, welche Benutzer aktuell auf dem Computer vorhanden sind.
Benutzername	Dies ist der Name des Benutzers (bis zu 20 Zeichen lang) den man bearbeiten, löschen oder hinzufügen will.
Passwort	Um das Passwort festzulegen.
/add	Um den Benutzer hinzuzufügen.
/domain	Um den Benutzer zur Domain hinzuzufügen.
/delete	Um den Benutzer zu löschen.
/help	Um Details über diesen Befehl zu erhalten.

7.2 Netzlaufwerk hinzufügen

Mit dem Befehl "New-PSDrive" können Netzlaufwerke hinzugefügt werden.
New-PSDrive [-name**] string [**-root**] string [**-Persist**] [**-PSProvider**] string**

Name	Erklärung
-name	Definiert den Namen des Netzlaufwerks.
-root	Der Ort, wo das Netzlaufwerk herkommt.
-persist	Definiert, dass das Netzlaufwerk persistent ist.
-PSProvider	Der Name des PSProviders, kann FileSystem, Registry oder Certificate sein.

7.3 Out-GridView

Name	Erklärung
Out-Gridview	Erstellt ein Output-Fenster einer tabellarischen Ansicht der Daten, die damit abgerufen werden. Lässt sich in Piping verwenden.
-title «Name»	Gibt dem Fenster einen Titel.
-PassThru	Öffnet die Tabelle mit einer OK und Schliessen Schaltfläche. Die Auswahl wird durch OK an die Konsole zurückgegeben.
-wait	
-OutputMode	

8. Schlussprojekt

Nun haben Sie die wichtigsten Kenntnisse erlangt, um auch umfangreichere Scripts und Programme zu erstellen. Auf dem weiteren Weg geht es darum immer neue Cmdlets kennenzulernen, welche für die verschiedensten Aufgaben benötigt werden.
Als erste Aufgabe mit einem realen Nutzen erstellen Sie ein Script zur Konfiguration eines Betriebssystems. Dafür nehmen Sie die Vorgaben aus dem Modul 304.

Ziel ist es, dass das Script sämtliche Konfigurationen ohne Unterbruch selbständig ausführt.

8.1 Durchzuführende Konfigurationen Niveau 1

A33	Erstellen Sie mit dem Script folgende Benutzerkonten: Informatik Administrator Raurau_10 Elektronik Standard Elerau_10 Mechanik Standard Mecrau_10
A34	Verbinden Sie folgende Netzlaufwerke: J: \\192.168.75.10\Austausch\Informatik\Abgaben T: \\192.168.75.10\Austausch\Informatik\Vorgaben P: \\192.168.75.10\UserHome$\h_muster
A35	Fügen Sie Outlook dem Windows Autostart hinzu.

8.2 Durchzuführende Konfigurationen Niveau 2

A36	Ein Menü mit Grid-View erstellen. In diesem soll zwischen «Datei kopieren», «Datei verschieben» und «Datei löschen» ausgewählt werden können. Der neue Speicherort soll dann automatisch im Explorer geöffnet werden.

8.3 Durchzuführende Konfigurationen Niveau 3

A37	Erstellen Sie mit dem Script folgende Benutzerkonten: Informatik Administrator Raurau_10 Elektronik Standard Elerau_10 Mechanik Standard Mecrau_10
A38	Aktivieren Sie alle Desktopsymbole (Arbeitsplatz, Papierkorb, Benutzer, Systemsteuerung und Netzwerk).
A39	Definieren Sie die Anzeigeoptionen des Datei Explorer wie folgt. • Dateierweiterung auch bei bekannten Dateitypen anzeigen. • Geschützte Systemdateien anzeigen. • Versteckte Dateien anzeigen. • Vollständiger Pfad in der Adressliste anzeigen. • Detailansicht als Standard

A40	Verbinden Sie folgende Netzlaufwerke: J: \\192.168.75.10\Austausch\Informatik\Abgaben T: \\192.168.75.10\Austausch\Informatik\Vorgaben P: \\192.168.75.10\UserHome$\h_muster
A41	Fügen Sie die in A40 verbundenen Netzlaufwerke dem Schnellzugriff hinzu. Der Desktop- und Dokumentordner sollen ebenfalls hinzugefügt werden.
A42	Schalten Sie die Audioausgabe auf Stumm.
A43	Fügen Sie Outlook dem Windows Autostart hinzu.
A44	Stellen Sie sicher, dass www.r-au.ch als Startseite definiert ist.

8.4 Zusatzaufgabe: Konfigurationen Überprüfen

Während den Installationen im Modul 304 mussten Sie verschiedene Einstellungen vornehmen und verschiedene Programme zusätzlich installieren. Erstellen Sie nun ein Script, welches die getätigten Installationen und Konfigurationen überprüft. Gehen Sie dazu in zwei Schritten vor. Zuerst erarbeiten Sie, wie die nötigen Informationen in PowerShell gefunden werden können. In einem zweiten Schritt überprüfen Sie die gefundenen Informationen auf ihre Richtigkeit und stellen das Resultat dem Benutzer zur Verfügung.

A45	Überprüfen Sie, ob sich die Systempartition C: auf der SSD Disk befindet.
A46	Überprüfen Sie, ob das komplette Office Paket installiert wurde oder, ob einzelne Programme fehlen. In einer Ausgabe soll ebenfalls ersichtlich sein, welche Programme noch zu installieren sind.
A47	Kontrollieren Sie analog zu A46 folgende Programme: Visio Acrobat Reader GIMP Inkscape Notepad++ Visual Studio Eclipse Android Studio