Chapter 10 - Expert Programming 101

Start

When you want to prove to people that think LYING or having TERRIBLE LOGIC doesn't matter ... ?
Whip out the HOW TO PROGRAM toolbelt, to educate those people on HOW they're DUMBASSES.
And then, publicly humiliate them for thinking this way.

Overview (1)

(Originally written between 05/13/22 → 05/18/22)

Greetings (Reddit) PowerShell Community,

Over the last month and a half, I've been slowly working on a utility that implements a combination of PowerShell class structures, (XAML/Extensible Application Markup Language), Multithreading & PowerShell Runspaces that can automatically throttle itself and synchronize the window with the GUI dispatcher, but that is a rather complicated project to try and dive into without the utility being complete and testing well.

The link to it is listed in Overview (2).
I'll talk about it briefly and then begin this explanation into building class structures in PowerShell, and how to translate an existing series of variables into a class.

Last night I decided to take a break from the threading stuff, and dive into the topic of converting scripts to .Net class types. I then wrote this lesson plan

05/23/2022	Chapter 10 - Expert Programming 101	In the document, within the link below

But as I wrote this lesson plan, and then I realized that the TANGENTS I later go on in this lesson plan REQUIRED SOME SERIOUS EXPANSION.

Thus, I started to write this 685 page book ...

10/08/22 | Top Deck Awareness - Not News
Used to be news…? Now it's Not News. Not News. Part of the Not News Network
https://github.com/mcc85s/FightingEntropy/blob/main/Docs/2022_1008_TDA_Not_News.pdf

... and teach people WHY classes are really ... incredibly USEFUL and UNDERUSED.
If people only knew that I look for ways to turn C# code into PowerShell code ...

Sometimes it's possible.
Sometimes it's not, without a lot of ADDITIONAL code.

In (C#/CSharp), classes are definitely not underused at all. In PowerShell, they definitely are.

Now, I've tried to teach some people in the community before about how to write classes in PowerShell, as it isn't exactly an intuitive process. But, I figured this one HERE is unlike any other lesson plan I've done before.

Nah.
This one right here ... ?
It's pretty long and detailed, but I'm certain it'll be a fun read.

Anyway, this lesson plan starts from an idea like:

```
Get-WMIObject -ClassName Win32_Product
```

... and then building a custom class off of THAT.

Not just writing a class off of it either, but- if an issue crops up, then pivot to an alternate strategy, and continue on. Not saying "Wow. What I need to do is too much work ... "

Cause I'm here to take some COMPLICATED SUBJECTS, and make em' simple, writing a tutorial on how to build yourself a class that can produce OTHER class templates and stuff ... with some story elements thrown in along the way.

This one may not necessarily have an associated video, but- it will probably cause a few people to chuckle and even draw up comparisons between how they CURRENTLY write their code, and what the benefits may be to going all out with a full-blown, class structure approach that can be pipelined by functions, or CmdLetBindings.

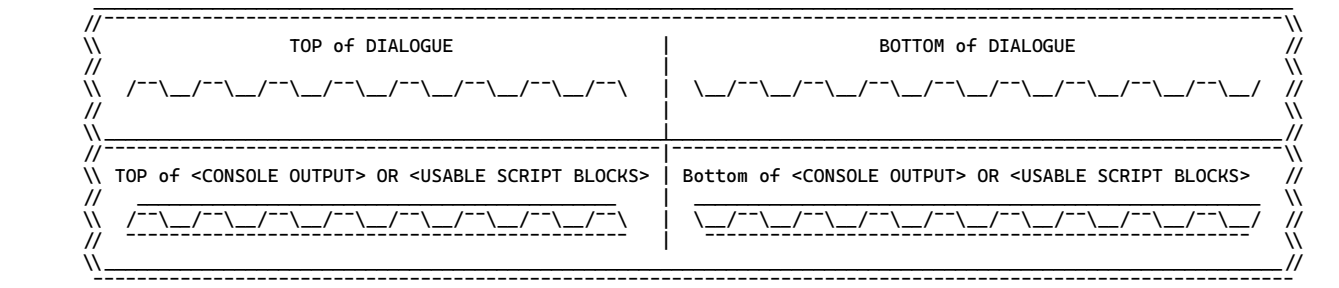
Anyway, the event log utility I've been working on, its no joke.
It does a lot more than just collect event logs- the utility has a fully featured graphical user interface that I teased about in the lesson plan I wrote about a month and a half ago ... but because this process is extremely lengthy (Not News is … hundreds of times longer).

I've had to adopt a methodology to implement a way to dynamically (allocate/adjust) running thread counts.
I call it AUTOTHREADING, but it's probably just a fancy word for automatic multithreading.

The lesson plan I wrote for it about a month and a half ago ... ? Could've been about anything.
But, as it took longer and longer to run through it, I realized that I had to find a way to implement something
VERY COMPLICATED into a utility that is dead simple.

So, I decided to hold off on completing and distributing that lesson plan, because I kept making changes to the code.
Just like SpaceX made so many changes to the BFR along the years ...

| I'm going to use INDICATORS throughout this entire chapter. Sometimes I will use a LABELED section, other times: |



Design Choices / Overview (1)

Musk : Guys, this isn't gonna work.
We need to throw ceramic tiles onto this thing ...
Engineer : Oh god.
That's gonna add a LOT of weight, and take a long time.
Musk : Yeah, but- we've burned through billions of dollars on all of these prototype SN#'s ...
Engineer : Maybe we'll get lucky next time, ya know?
SN 10 landed ...
Musk : *hard stare* Bro ...
brief pause It blew up AFTER landing ...
Engineer : *chuckles* Still landed in one piece, though ...
You had a RUD afterward ...
Musk : Look.
looks at watch I'm plenty patient.
But- Rapid Unscheduled Dissassemblies cost a LOT of money.
stern When choosing between patience, OR, a NASA CHALLENGER situation ...
Engineer : *stops chuckling* Obviously, patience.
A NASA CHALLENGER situation'd definitely be bad.
Musk : *raises one eyebrow* 7 people died that day buddy ...
You want that on your conscious ... ?
Engineer : *puts hand on back of neck* Nah, can't have (1) death on SpaceX's permanent record.
Musk : I mean, realistically ... ?
It's inevitably bound to happen at some point.
However-
Engineer : ... it's my job to make sure that doesn't happen anytime soon ...
Musk : You know it bro.
Get it done.
Engineer : Yes sir.

Overview (2) / Design Choices

I can imagine that when building PayPal, Elon didn't mess around.
You wanna pay a pal of yours over the internet ... ? Well, now you can.
Wanna know why you can pay a pal over the internet now ... ?
Cause some smart guys thought of a way to do it. Elon was one of those guys.

Prolly the main guy involved, in the inception of the world's first pal-paying-program, PayPal.

While what I'm doing isn't as COOL as PayPal, or mankinds' first reflyable 25-story building ...
... it uses SYNCHRONIZED HASHTABLES, CLASSES, SESSION STATE OBJECTS, FUNCTION/ASSEMBLY/VARIABLE entries, BOTH runspace factories, separate objects to control groups of threads, sends progress information to the GUI console, stopwatch, timing, reports, percentage of completion, passes stuff back and forth, handles hundreds of thousands of event logs, uses native System.IO.Compression classes, has a lot of error handling, and it really is all coordinated from the controller classes that drive the GUI.

Here's a link to the functions on my GitHub project ... https://github.com/mcc85s/Functions
I uploaded VARIOUS versions of the file/process, EventLog-Utility.ps1, or some variation of that.

This is the NEWEST VERSION of it ...
https://github.com/mcc85s/FightingEntropy/blob/main/Functions/EventLogs-UtilityRunspaces-v4.ps1

Various versions because I tried hundreds of ways to make the process FASTER or MORE EFFICIENT.
The tool is meant to essentially EXPORT the LOGS and SYSTEM DETAILS, and then SAVE to an ARCHIVE.

Then, the tool can LOAD one of those ARCHIVES and be able to REVIEW the EVENT LOGS from another system.

Ultimately, the program extracts all of the event logs from a system and automatically calculates how many threads it can use, and divides the work accordingly, to process everything in a PARALLEL manner, though it is NOT COMPLETE.

I suppose the most complicated thing about what I'm doing, is trying to do all of this from the GUI, without causing the dispatcher thread to lock up. The end result, looking sorta like Rufus.

But, strategically assigning properties and values at specific a (location/time), is stressful.
Like, "balancing a broom handle in a hurricane", probably took a while to figure out.
Probably didn't happen overnight. Wonder who would've said something that 'insane' ...

Yeah. Some people used to think that "balancing a broom handle in a hurricane" was impossible.
And, as people heard something that sounded impossible, being stated by a real smart bastard,
they would constantly confuse that person's intellect, with insanity.

They rarely ever realize that doing the same thing over and over again, expecting different
results, is the definition of insanity. The DIFFERENCE between the two, is "strategy".

As for the things I mentioned a few paragraphs up, there are some examples in the community that exhibit PORTIONS of what I'm attempting to do. Jim Moyle has a couple of good XAML/Runspace videos and his work is good, but it's a few years old now. I'd be interested in seeing Jim do more of those videos, to be perfectly honest.

But also, who would be surprised that I'd say Boe Prox (... in addition to 1RedOne).
1RedOne has a sample that looks very close to what I am attempting to do.

But, Jim's exhibition and Boe/1RedOne's have a similar issue.
They're calling a new runspace within a scripted runspace.
I'm doing something totally different.

Boe has plenty of work out there in the wild, such as PoshRSJobs (very useful module btw).
But I've run into many limitations with it, and had no idea how to troubleshoot them.

In reference to a veteran MVP's work like his ... ? I'm not gonna try to reinvent the wheel.
But, I did keep runspaces in the back of my mind as I kept thinking about reinventing the:

```
| WHEEL | AXLE | WAGON | CALCULATION of π |
```

Overview (2)

Can't just go reinventing stuff

It is one of the most challenging tasks I've ever decided to take on. The takeaway will be that I found a way to integrate the threading of multiple runspaces via many custom classes that I wrote to drive the backend of the utility. It is rather thorough in the information it collects, and it formats itself and saves a running system configuration and ALL event logs to a .zip file that the program never actually decompresses.

Sorta like JOHN CARMACK decided to do with Quake III Arena by using (*.PK3) files.

If you don't know who JOHN CARMACK is, or what a (*.PK3) file is, well ...
- JOHN CARMACK was the main guy at id Software for years, the guys that made Doom, Quake, etc.
- JOHN CARMACK is the guy who invented true 3D graphics in 1996 (pretty sure)
- JOHN CARMACK is the guy who basically kickstarted the modern hardware graphics industry
- (*.PK3) files are basically (PK) zip files, included w/ the 1999 release of Quake III Arena.
- I had a HOSTED SITE on PLANETQUAKE.COM, here's a link to a WAYBACK MACHINE entry ...
  ... to the many maps I made that used the PK3 format.
  https://web.archive.org/web/20220000000000*/planetquake.com/bfg20k
- Many of the maps I made are STILL AVAILABLE FOR DOWNLOAD on [.. ::LvL]

Then, the utility can import those files back into the GUI on another system if need be.
However, that utility isn't quite ready to show off yet.

```
| Objective  : Get a list of all the currently installed programs. If you don't? Well ...          |
|              You're not gonna hear "Computer Update", nor see the flashing icon that says "F1"  |
| Suggestion : Don't freak out. Just run the command Get-WMIObject Win32_Product in PowerShell ... |
```

<Editor> (1)

Can't just go reinventing stuff

```
$WMIList        = (Get-WMIObject Win32_Product)[0]
```

```
| or ... |
```

```
$WMIList        = Get-WMIObject Win32_Product | Select-Object -First 1
```

```
| ... does the same thing |
```

_____/
_____/ <Editor> (1)
  But Dad, you said ... /-----\
/----------

    Kid : ... but Dad, you said to run
    ____
    | Get-WMIObject Win32_Product |

    Me : Yep.
          I did.
    Kid : ... you used a [VARIABLE] followed by [EQUALS SIGN], and then added [PARENTHESIS], [SQUARE BRACKETS], and threw
          the [INDEX] number 0 in those brackets ...
          NONE of that is part of the command ...
    Me : That's right.
          Just, went right ahead, and started off with something ADVANCED, didn't I ... ?
    Kid : Yeah.
          This already looks pretty advanced to me ...
          I read through all of this stuff you wrote, and then FINALLY ...
          Here it is ...
           ... the moment I've been waiting for.
          Dad finally got to Line #1.
          Like a tidal wave of information.
          Can you explain all of this?
    Me : Gladly.

_____/
_____/ But Dad, you said ...
  Line #1: Breakdown /-----\
/----------

              _________________________3 6
    $WMIList = (Get-WMIObject Win32_Product)[0]
    -------1 2   -------------------------4 ----5

        Using a (1) VARIABLE,
                (2) EQUALS sign, and then a
                (4) EXECUTABLE/COMMAND/FUNCTION/TYPE/CLASS/METHOD/VARIABLE/PROPERTY/VALUE
                    returns its output to the VARIABLE.
        However,
                (3) PARENTHESIS content gets processed FIRST (See: Math/Order of Operations)
        And,   (5) SQUARE BRACKETS indicate to select a specific index/entry from an array[]
             ... (6) INDEX of the returned object,
                    All indexes start at number 0, not number 1
                     ... because a man named Paul Allen said so.

    Doing these things TOGETHER is how professionals and experts get the job done, quickly.
    Casting the VARIABLE in this manner ONLY captures the FIRST object returned from the operation.
    If the command returns nothing, $WMIList will just be $Null/<empty>
    If the command returns (1) or more, $WMIList will NOT be $Null/<empty>

    Here's some sample output that got returned by the console when I ran the above ...

_____/
_____/ Line #1: Breakdown
  <Console> (1) /-----\
/----------

    PS Prompt:\> $WMIList

    IdentifyingNumber : {2AF42320-5ECF-4BCA-B756-8F3677262D55}
    Name   : Branding64
    Vendor   : Advanced Micro Devices, Inc.
    Version   : 1.00.0009
    Caption   : Branding64

_____/
_____/ <Console> (1)
  Kid sort of gets it now /-----\
/----------

    Kid : Alright ... *adjusts his glasses* I sort of get it now.
    Me : Cool.
    Kid : How do you know that Paul Allen said that everything starts with the number 0, though?
    Me : Cause I imagine that Paul Allen and Bill Gates had many arguments when they were kids ...

        Bill : 0 isn't the first number, 1 is ...
        Paul : Is 0 a number?
        Bill : Yeah.
        Paul : So, 0 is the first number.
        Bill : No, 1 is the first number.
        Paul : 1 is the second number.
        Bill : ... well, that's just ...
        Paul : ... confusing?
        Bill : ... yeh.
               I would've thought that 1 was the first number.

Paul : Yeah, 9 is the tenth number.
There are ten numbers in base 10 for each digit.

Kid : Interesting ...
Me : I have no idea if they ever had that conversation or not, but nobody has ever told me otherwise.
Kid : Alright.
Cool.
Feel free to proceed.

_____/ Kid sort of gets it now

<Properties> (1) /-----\

/----------

Properties are in (EXECUTABLE/COMMAND/FUNCTION/TYPE/CLASS/METHOD/VARIABLE/PROPERTY/VALUE) list.
Because we'll refer to this list right here a lot, let's define a MENTAL VARIABLE named $List

```
| $List = (EXECUTABLE/COMMAND/FUNCTION/TYPE/CLASS/METHOD/VARIABLE/PROPERTY/VALUE) |
```

This is a series of properties for the object that came back as variable $WMIList

```
| IdentifyingNumber : {2AF42320-5ECF-4BCA-B756-8F3677262D55} |
| Name              : Branding64                             |
| Vendor            : Advanced Micro Devices, Inc.           |
| Version           : 1.00.0009                              |
| Caption           : Branding64                             |
```

I used to believe that VARIABLES, were the most powerful thing about programming ...
While they ARE powerful, VARIABLES are a piece of the puzzle and are actually PROPERTIES.

PROPERTIES are the most powerful thing about programming.
They are part of EVERY functional language, object-oriented language, mathematical formula,
algorithm, equation, theorem, story, character, operation, function, class, method, etc.

As they relate to programming, every object has properties that can, at any moment, change the
object they are part of *instantaneously*.

Before I can really dive deeper into properties though, I have to touch on the Lamda.

_____/ <Properties> (1)

<Lambda> /-----\

/----------

I'm going to shorthand all (3) of these, 1) Commands, 2) Functions and 3) Methods, as "Lamda"

A LAMBDA can either be:

```
| written OUTSIDE of the EXECUTION CONTEXT | a PROPERTY of an OBJECT | a METHOD | a TYPE/CLASS |
```

A LAMBDA is meant to be INVOKED, and when it is ... ? It will operate on:

```
| 0/1/+1 parameters                 |
|  other object(s) properties/values |
| parent object(s) properties/values |
```

A LAMBDA makes ASSUMPTIONS when not provided any PARAMETERS, based on its CONTEXT.
A LAMBDA is very SPECIAL, because its ROLE/DUTY when EXECUTED, MAY cause the OBJECT to:

```
| stay the SAME | become (less/more) COMPLEX | CHANGE |
```

_____/ <Lambda>

From Executables to Values (1) /-----\

/----------

A dude named KEVIN came along, and then suddenly, a LAMBDA was invoked ...
... now that dude's name is MARK.

You were expecting KEVIN.
But the reality is, you've been told that KEVIN no longer exists.

Now you're PISSED. KEVIN was a cool dude. Didn't deserve to be subjected to such a harsh Lambda.
New dude named MARK looks a hell of a lot like KEVIN did ... tries to tell ya that he WAS KEVIN ... now he's MARK.

That's impossible though, and you feel disrespected.

/--\

Mark : Hey dude.
You : Who the hell are YOU, dude ... ?
Mark : I'm MARK.
I was KEVIN.

```
       Now I'm MARK.
       KEVIN doesn't exist anymore.
You  : Yeah right buddy.
       KEVIN was twice the guy you'll EVER be ...
Mark : *scoffs* You're so predictable ...
You  : Sounds *exactly* like something KEVIN would've said, if he still existed.
       *you pay no mind to this fake-ass version of KEVIN that's somehow a LOT like KEVIN, but is named MARK*

\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/

Suddenly the lambda was invoked again, changing MARK's name back to KEVIN.
Then, you're told that KEVIN was brought back into existence.
You turn around, there he is. KEVIN's back, dude.
Bet you didn't see that comin' did ya ... ? It's almost like magic, but- even cooler than that.

In real life, this stuff isn't possible.
Relinquishing KEVIN's existence to bring MARK into existence ... ?
It's more than just unfair. It's practically unthinkable.
But, in the same token, relinquishing MARK's existence just to bring KEVIN back ... ?
It's also unthinkable. MARK didn't do anything to deserve it either. So, then what?

In math or programming, this stuff is definitely possible after all. That's how numbers work.
In reference to $List = (EXECUTABLE/COMMAND/FUNCTION/TYPE/CLASS/METHOD/VARIABLE/PROPERTY/VALUE),
what they each have in common is that they're all specific categories that can be acted upon by a
lambda, even though some of them are lamdas themselves.

I will use a Dr. Seuss-like explanation, in order to (discern/differentiate) one from another:

(1)      NOUNS are: [things]
         NOUNS are OBJECTS | KID, STORE, SODA

(2)      VERBS are: [activities]
         VERBS are OBJECTS | GO, SEE, HAVE, BUY, RETURN

(3)    ACTIONS are: [things doing activities]
       ACTIONS are OBJECTS | KID GOES to STORE and BUYS 1x SODA.

(4) CONDITIONS are: [probability that a thing is 1) doing an activity, or 2) has a required value]
    CONDITIONS are OBJECTS | STORE HAS (>/greater than) 1x SODA

(5) STATEMENTS are: [things that check a condition before doing an activity]
    STATEMENTS are OBJECTS | KID GOES to STORE, SEES 1x SODA, BUYS 1x SODA.

(6)    STORIES are: [parallel statements involving things, activities, conditions, and actions]
       STORIES are COMPLEX OBJECTS

                              KID   exists       | Noun, Verb, Action, Condition, Statement
                              SODA  exists       | Noun, Verb, Action, Condition, Statement
                              STORE exists       | Noun, Verb, Action, Condition, Statement
                              CLERK exists       | Noun, Verb, Action, Condition, Statement
                              KID   wants  SODA  | 2x Noun, Condition, Statement

                              (I'll simplify what all of these are now, but each of these are statements ... )
                              KID goes to STORE
                              KID enters STORE
                              STORE sees KID
                              STORE entered by kid
                              STORE has (>/greater than) 1x SODA
                              KID grabs SODA
                              CLERK asks KID, "MONEY ... ?"
                              KID says "NO MONEY, bro ... ".
                              CLERK says "GET MONEY".
                              KID gets SAD.
                              KID starts FAKE CRYING.
                              CLERK says "Awww ... GET MONEY."
                              KID stops FAKE CRYING.
                              KID leaves STORE.
                              KID gets MAD.

(7) INVOCATIONS/INSTANTIATIONS occur when a statement is validated, and this can result in:
    INVOCATIONS/INSTANTIATIONS are LAMBDAS/OBJECTS
    -------------------------     ---------------

    (0) object(s) being created, or nullified
                        -------     ---------

    (1) object(s) being created, or nullified
                        -------     ---------

    (+1) object(s) being created, or nullified
                         -------     ---------

These entries above ALL meet certain criteria that allow their context to be matched or defined.
We'll call these things PROPERTIES, since they ALL have the capability of being a PROPERTY.
Properties can be made up of LAMBDAS, NOUNS, VERBS, ACTIONS, CONDITIONS, STATEMENTS, and STORIES.

We cannot explicitly call a lambda the same thing as a verb, even though human intuition says they
```

both match the same criteria ... because a lambda MAY, or MAY NOT be a combination of these various elements.

This all touches on ENGLISH, or general (linguistics/language-speaking dynamics), but it also applies to math, and science, because ALL of those statements have to be considered by a processing unit that can logically deduce an outcome, and store the output in a VARIABLE, PROPERTY, FILE, or MEMORY ADDRESS.

(So if you have LAZY POLICE OFFICERS or just, general, all-around, lazy people that prefer to be INACCURATE all the time with EVERYTHING THEY SAY/DO ... then that is DANGEROUS.)

To refine our mental variable $List, lambdas DO occupy most of that list.
But really, LAMBDAS specifically are the MOST difficult to CLASSIFY, and thus other definitions need to be considered, in order to ACCURATELY classify their TYPES. (Not that different from INVESTIGATION)

(EXECUTABLE/COMMAND/FUNCTION/TYPE/CLASS/METHOD/VARIABLE/PROPERTY/VALUE)

The honest/direct truth here, is that ALL of these things fall within the definition of ...
... an EXECUTABLE.

Why ... ? They're all layers of something that can be acted upon.
If we called EVERYTHING "object", that would confuse people pretty easily ... even though they ARE all objects.
However, believe it or not, calling an EXECUTABLE the same thing as a VALUE can be true in a specific context.

Just like how a 10.7 GB FILE may be monstrous because of it's size, it is still a FILE.
There may be more CONTENT in a 10.7 GB .mp4 file, versus a 10kb .txt file ...
... but given enough thought ... ? A 10kb .txt file can actually reproduce a 10.7 GB .mp4 file.

The question is, will it be an exciting, nail-biting, thriller, if a 10kb .txt file has the source code to provide or BUILD that .mp4 file ... ? That really depends on what those instructions actually DO.
Because, it could literally just be the same exact character repeated ...

```
                                                                                                                  _____________________________/
\_____________________________________________________________________________________________/ From Executables to Values (1)
  <Console> (2) /-----------------------------------------------------------------------------------------------------------------\
/---------------
```

```
PS Prompt:\> $Int = [Int64](10.7 * 1GB)
PS Prompt:\> $Int

11489037517
```

```
                                                                                                                  _______________/
\_____________________________________________________________________________________________/ <Console> (2)
  From Executables to Values (2) /-----------------------------------------------------------------------------------------------\
/--------------------------------
```

11489037517 many times. Yeah. It won't be an exciting, action-packed, thriller of a movie.
If it is the same character repeated that many times, then that (*.mp4) file won't even open.
That's pretty boring. But, there COULD actually be something WAY more boring, than a file that claims to be an (*.mp4) file, won't open, and is 10.7 GB ... somewhere on the internet.

Though I can't imagine that anybody would really keep that on an accessible webserver.

Anyway, 10kb of information could ALSO have a link to resources that exist elsewhere.
So if you're like APT29, you could rick roll the US Department of Defense, make the next serious cyberattack against the United States look like a simple little 10kb txt file ... and just wait.

Pretty sure they'll be wicked pissed if that happens ... ?
But- it happened in December 2020.

```
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
```

```
APT29 #1 : Hey buddy ...
APT29 #2 : Yeh ... ?
APT29 #1 : I will BET you, like, millions of dollars, that if I put this 10KB text file somewhere ...
           ... some fuckin' moron will open it and allow us to commandeer the entire US Federal Government.
APT29 #2 : I'm not takin' that bet, I know they will.
APT29 #1 : Ehhhh ... how about for like $5 bucks then ... ?
APT29 #2 : So, I'm still not gonna bet against you because I know you're right ...
           Do you just want me to give you $5 bucks or something ... ?
APT29 #1 : Nah man ...
           You're really ripping the wind out of my sails, dude.
APT29 #2 : Dude ...
           I know how dumb Americans are though ...
           I'm like, your teammate.
           They're dumb enough to open a LOT of 10KB text files that they shouldn't be opening ...
APT29 #1 : Yeh, yeh ...
           *sighs*
           You think that APT29 #3 would take that bet though ... ?
APT29 #2 : *shrugs* I dunno, bro.
           Probably not.
APT29 #1 : This is just too easy, dude ...
```

```
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
```

Not all Americans are THAT dumb, but they're constantly DISTRACTED and VERY HARD TO INFORM of IMPORTANT STUFF.

Anyway, if a URL is combined with a function call like Invoke-RestMethod, that could turn APT29's latest and greatest threat to national security, into an object on a bunch of computers.

Then again, it could also be instructions for a video codec to create an 18 hour video of nothing but static. Most people that watch the 1997 movie Contact, directed by Robert Zemeckis, based on the 1985 novel with the same name by Carl Sagan, starring Jodie Foster …

… they really don't pay much attention to the line at the end of the movie …
"It recorded 18 hours of static".
But, James Woods character just wasn't super impressed …

Nah. Cause his character was so skeptical of the entire thing, that when he found out that device recorded 18 hours of static (because she was *actually gone* for about 18 hours, even though all of the cameras on Earth were able to record the thing drop straight down in several seconds) …

Obviously this went against everything that his character stood for. James Woods has always been a great actor, and performs roles where hes an asshole, very skillfully. Cause his character acted like a lot of people in real life do- resentful, miserable, looking for any single reason at all, to shake their finger in Jodie Foster's face.

The truth is, PERCEPTION and PERSPECTIVE mean quite a lot to CONTEXT/CONTENT.
That was the actual moral of the story of the 1997 movie CONTACT.

What do these things have to do with the properties of $WMIList, though … ?

Properties, Strings, Serialization, Deserialization / From Executables to Values (2)

Property	Value
IdentifyingNumber	{2AF42320-5ECF-4BCA-B756-8F3677262D55}
Name	Branding64
Vendor	Advanced Micro Devices, Inc.
Version	1.00.0009
Caption	Branding64

These PROPERTIES all happen to be STRINGS. Nothing wicked exciting about these particular values.
Except the vendor field is actually something to get excited about after all … in 2017, this company released a brand new (CPU/computer processing architecture) that took the industry to new heights.

This new architecture, believe it or not, was fabricated < 10 miles from where I live … side point.

In reference to these properties, each of them could in fact, be TRANSLATED, into something else, when a particular STRATEGY is seen. Doesn't necessarily have to be SEEN either. Because, there's a particular strategy that can be used, to detect someone elses' particular strategy, via a PATTERN.

Probably sounds like 10-dimensional chess or something, using a strategy to detect a pattern that detects someone elses' strategy … but, that's actually what computers are programmed to do.

The fact of the matter is, while they are *all* PROPERTIES, some are specific (types/classes), in addition to being a property and a value. For instance, IdentifyingNumber is a (GUID/globally-unique identifier).
The reason I know this, is because GUID's all follow a specific "convention" or "pattern".

When an object is serialized, it turns an object, its properties, and values, into a string.
Before a Lambda decided it was time, to tell this object "You're about to be serialized, bub" … ?

It was an OBJECT. Happy as could be, too.
Until the LAMBDA ordered it to become a STRING, and did what it was told …

Now, the OBJECT no longer exists, because it was forced to become a STRING.
While a STRING is still technically an OBJECT … ? It's not QUITE the same thing.

Not too dissimilar from MARK telling his buddy that he WAS KEVIN, but- now he's MARK …
Nobody believes MARK, that he WAS KEVIN. Could look virtually identical to KEVIN, too …
Might even sound like KEVIN when he talks. Might smell like KEVIN, eat the same food as KEVIN, and say the same things as KEVIN … ? But nobody will ever believe MARK, that he WAS KEVIN.
… now he's MARK.

Anyway, reinstantiation is a lot less dramatic, but it does involve patterns and properties to be specifically named. The correct lambda has to detect those properties, and test patterns, in order to reinstantiate the string back into an object.

If that object has any chance at all … of being willed back into existence?
A specific pattern could be the key ingredient in making all the difference in the world.
Suffice to say, strings can be (converted back into objects/deserialized), given enough context.

The IdentifyingNumber here can be reconstituted back into a GUID, but that is actually very tough.
Might need help from a magic lamp.

So, grab the closest magic lamp you can find.
Then, rub the lamp slightly, be sure not to upset the genie inside ...

We don't need a snarky, irritated genie ... do we? Of course not ...
When the big blue genie finally comes out?
That's the moment where you gotta wish for the string to become a GUID again.
Piece of cake, right?

I'm just kidding, it's actually easier than all of that. No lamps, no wishes.
Just a basic understanding of Regular Expressions, the most advanced mathematical probability matrix available to mankind ... where written languages and mathematics are merged into logical roadmaps ...

Logical roadmaps that make branch predictions ...
... branch predictions chiseling out the course of action that any modern microprocessor can be expected to take, at least, whenever they're programmed to perform specific lambdas in response to (parameters/input) it is provided.

That's the cinematic, long-winded explanation ... for patterns in strings of serialized text.

Regular Expressions (1) / Properties, Strings, Serialization, Deserialization

This specific (Regex/regular expression) pattern here ...
"[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}"
... can be used to match any GUID in existence, if the letters are lowercase. Not uppercase.
... if they're uppercase ... ? There's nothing anybody can do. It's over ... time to go home.

Just kiddin'. Even if they're uppercase, the pattern will probably still work, however-
nobody can say that with any certainty at all, without knowing the casing operand.
If you really want to be on the safe side, then the pattern should be written this way,
"[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}"

The reason why people don't really like that idea is because it made that pattern longer.
There are a number of ways to shorthand that pattern.
Here's a cool way to do it in PowerShell ...
(8,4,4,4,12 | % { "[0-9a-fA-F]{$_}" }) -join '-'

Anyway, these patterns represent a globally unique identifier, and they're useful in many ways.
Does the value for property "IdentifyingNumber" for variable $WMIList match the pattern ... ?

Let's find out ...

Regular Expressions (1) / <Console> (3)

| 1) Assign [String]"[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}" to var $Pattern |

```
PS Prompt:\> $Pattern = "[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}"
```

| 2) Now, check if the pattern matches $WMIList.IdentifyingNumber |

```
PS Prompt:\> $WMIList.IdentifyingNumber -cmatch $Pattern

$False
```

... it came back as $False, because the PATTERN is looking for LOWERCASE letters [a-f], and the input is using UPPERCASE letters [A-F].

Regular Expressions (2) / <Console> (3)

If we're being serious, there's PLENTY we can do here.
a) Drop 'c' from "-cmatch", or ...
b) change the pattern to consider uppercase letters.

However ...
IF   a) dropping the 'c' from "-cmatch" is out of your hands ... ?
AND   b) the pattern can't be changed
THEN c) no drama here, uppercase letters ARE impossible to filter out w/ this (condition/pattern) combo.

Though, there are PLENTY of WORKAROUNDS, one being that you could force the input string to use a method called .ToLowercase() on the string, probably plenty of other workarounds if you think of them.

But, typically people won't use "-cmatch", not unless they're being strict about CASING.

-cmatch means CASE   SENSITIVE
-imatch means CASE INSENSITIVE
-match means both will match

However, this whole entire document is meant to get people to think about the strategies involved on either end, being anywhere between:

1) a fairly innocent occasional script user who can enter their login password with one hand
2) a masterful developer/engineer who can crush any coworkers soul with one eyebrow
3) the godfather of the matrix ... knows everything. He's reading for fun.

If we need ABSOLUTE CERTAINTY that our PATTERN will match regardless, CHANGE the PATTERN.

Regular Expressions (2) <Console> (4)

| 1) Assign [String]"[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}" to var $Pattern |

```
PS Prompt:\> $Pattern = "[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}"
```

| 2) Now, check if the pattern matches $WMIList.IdentifyingNumber |

```
PS Prompt:\> $WMIList.IdentifyingNumber -cmatch $Pattern

$True
```

... it came back as $True, because the PATTERN matches, as the casing is ignored here.
Typically, people will use -match in their scripts far more often than -cmatch

Regular Expressions (3) <Console> (4)

The thing is, if we directly compare the Regex match pattern to the actual value ...

Pattern	[0-9A-F]{8}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{4}-[0-9A-F]{12}
Value	{2AF42320-5ECF-4BCA-B756-8F3677262D55}

... the pattern is a LOT longer than the actual string.

Suppose we had to comb through a billion of these GUIDs ... ?
It would require more data to FIND these patterns, than to simply interact with it ourselves.

What would help here, is assigning a STATIC METHOD that will remember the pattern.
That way, the pattern doesn't SPAM everybody with a CLONE of itself for EVERY ITERATION of a GUID out there.

I can tell you with sheer certainty, that if my old business partner decided that he HAD to hire an employee to specifically screen a lot of input strings, to determine by EYE, whether a GUID is bogus or not ... ? I'd tell him:

```
Me    : That's not a very good Computer Answer there, buddy ... ~!
Pavel : Nah ... ?
Me    : Nah ...
        It's a pretty LAME Computer Answer.
Pavel : *makes sad face*
```

Not JUST because that would be the most boring job in the world, but that's a pretty boring way to handle such a simple problem. Yeah. That employee he had to hire, they'll make many mistakes over the course of having to CONFIRM 8,4,4,4,12 base 16 digits, a billion times.

They'll ALSO take FAR longer to do the scope of work.
Whereas, a REGEX pattern costs a lot less time/money/resources AND saves someone from having to hire an employee to do (1) specific, and mundane job ... one that will expectedly bore that employee to sleep.

In other words, "ineffecient as hell". AKA, bad idea.
The pattern covers ALL of the mathematical probabilities that a GUID will match, using base16 for each character within the GUID string convention. 8, 4, 4, 4, 12 | where each number represents the number of consecutive base16

digits, and then joins each group with hyphens, and wraps it all with the curliest curly braces in existence.

I'll demonstrate a complicated way to create a class that generates a legitimate GUID.
This class will actually perform ADDITIONAL roles/duties/methods, that the base class [Guid] doesn't.

Regular Expressions (3) <Complicated>

<Complicated>

Kid : Dad, this sounds really complicated.
Me : Listen, Kid.
      The reason all of this SOUNDS incredibly complicated ... ?
      ... is because it IS incredibly complicated.
      I'm breaking it all down to be SO simple, even YOU could understand it.
Kid : *sigh* Is there a way around all of this ... ?
      I don't wanna reinvent the wheel, dad.
      That's just WAY too tough for me ...
Me : No way around it, I'm afraid.
      Not if you want to learn how to correctly build cool stuff, and be considered cool too.
Kid : *sigh* How cool ... ?
Me : REAL cool, kid.
Kid : That DOES sound COOL ... but this is too much for me ...
Me : So, if this already sounds like a task you're not ready for ... ?
      Then I guess it's time for you to act like mom, throw your hands up, and just, quit ...
      ... cause of how complicated this stuff is ...
      ... unless of course, you're ready to wear some big-boy pants.
Kid : Dad ... ?
Me : Big-boy pants.
      You wanna try some on ... right?
Kid : I already have big-boy pants on.
Me : *chuckles* Kid, those don't look like big-boy pants, at all.
Kid : Yeah, they are.
Me : Nah, those look more like pull-ups ...
      ... that's what young toddlers wear ...
      ... toddlers are less than half your age ...
      Those aren't big-boy pants ...
Kid : *trying not to grin* Will I be able to program my own game, if I listen to you ... ?
Me : That depends on you, ya friggen' pull-up wearin', kid-faced, boymeister.
Kid : *huge grin on his face* Dad ...
Me : What's up kid ... ?
      Ready to wear some big-boy pants ... ?
Kid : *makes a pretend clenched fist, still smiling, starts laughing ... * They're not pull-ups.
Me : Kid, those things are AT BEST, something that would be skin tight on a teenager.
Kid : That means they're big-boy pants after all.
Me : *scoffs* Kid, when I say big-boy pants ... ?
      I mean like, pants that an ADULT would wear.
Kid : *chuckling* Heh.
      Yeh dad ... ? Well ...
      chuckling How come you're not wearin' any?
Me : Ah, good question.
      You'll never believe this, kid ...
      But- I outgrew mine.
Kid : *rolls his eyes* Whatever ...
Me : Wait till you're ready to try some on, kid ... all this stuff'll help.

<Complicated> Overly-Complicated Class Definition that Generates GUID's

Using a (1) property named Guid, I want to (match/extend) the output for the [Guid] base class.
There are comments within the class here, feel free to study it, as I suggest doing so.

At least- if you're reading this, and you really like the idea of one day POSSIBLY becoming the godfather of the
matrix's right-hand man. Even if that never happens ... ? Maybe you'll impress this guy.

```
Class DemoGuid
{
    [String]   $Guid
    DemoGuid()
    {
        $This.Reset()
    }
    DemoGuid([Object]$Guid)
    {
        If ($Guid -match [DemoGuid]::Pattern())
        {
            $This.Guid = $Guid.ToLower()
            $This.Strip()
        }
        Else
        {
```

```
            [Console]::WriteLine("Exception [!] Entry is not a valid Guid")
        }
    }
    Static [UInt32[]] Digits() # // METHOD → Digit width block convention
    {
        Return [UInt32[]](8,4,4,4,12)
    }
    Static [Char[]] Chars()    # // METHOD → Available Base16 digits
    {
        Return [Char[]]"0123456789abcdef"
    }
    Static [String] Pattern()  # // METHOD → Strict Guid match pattern
    {
        Return ([DemoGuid]::Digits() | % { "[0-9a-fA-F]{$_}" }) -join "-"
    }
    Static [UInt32] Random()   # // METHOD → Generates random index between 0 and 15, base16
    {
        Return (Get-Random -Minimum 0 -Maximum 15)
    }
    Static [UInt32[]] Stage([UInt32]$X)
    {
        If ($X -notin 4, 8, 12)
        {
            Throw "Invalid Entry"
        }
        Return [UInt32[]](0..($X-1)| % { [DemoGuid]::Random() })
    }
    Reset() # // METHOD → Clears the output and (sets/resets) the GUID
    {
        # // SCRATCH VARIABLE → Hashtable to keep each section separated
        $Hash          = @{ }

        # // SCRATCH VARIABLE → Digit here decreases method calls → translates to increased performance
        $Digits        = [DemoGuid]::Digits()

        # // SCRATCH VARIABLE → Chars here decreases method calls → translates to increased performance
        $Chars         = [DemoGuid]::Chars()

        # // LOOP → Iterate through each digit width
        ForEach ($Digit in $Digits)
        {
            # // SCRATCH VARIABLE → Set collection Array
            $Collection = [DemoGuid]::Stage($Digit) | % { $Chars[$_] }

            # // SCRATCH VARIABLE HASHTABLE → Process each digit count into string
            $Hash.Add($Hash.Count,$Collection -join '')
        }
        # // PROPERTY ASSIGNMENT → Change the value of the property named Output,
        $This.Guid       = $Hash[0..4] -join "-"
    }
    Strip()
    {
        If ($This.Guid -match "(\{$([DemoGuid]::Pattern())\})" )
        {
            $This.Guid = $This.Guid -Replace "(\{|\})",""
        }
    }
    [String[]] Output()
    {
        $Return  = @( )
        $Split   = $This.Guid -Split "-"
        $Return += ""
        ForEach ($C in 0..4)
        {
            $Return += "Group[$($C+1)]: $($Split[$C])"
        }
        $Return += ""
        $Return += "Guid: $($This.Guid)"
        Return $Return
    }
    [String] ToString()
    {
        Return $This.Guid
    }
}
 ____________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ----------------------------------------------------------------------------------------------------
                                                                                                     /
\___________________________________________________/ Overly-Complicated Class Definition that Generates GUID's
  <Console> (5) /-----------------------------------------------------------------------------------------\
/---------------
```

We will use (2) manners of accessing the class I just wrote…

```
 _____________________________________________________________________________
| Manner 1) Direct value entry (unable to invoke methods within this class externally) |_____________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
-------------------------------------------------------------------------------------------------------------

PS Prompt:\> [DemoGuid]::New()

Guid
————
d3872299-caa8-e988-89c7-aa8e2597d796
 _____________________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
  -------------------------------------------------------------------------------------------------------------

That is the straight-up, 100% verbatim, output of the class/type I wrote above..
Actually, I'm adding the signs as I edit this document/script.
But the console output is genuine. Sorta looks like the default class here …
  ___________________________________________________________________________________________________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
-------------------------------------------------------------------------------------------------------------

PS Prompt:\> [Guid]::NewGuid()

Guid
————
2a329c56-9c30-43eb-9981-58726b740a41
 _____________________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
  -------------------------------------------------------------------------------------------------------------

The default method to create a new Guid with the default type, is right there, [Guid]::NewGuid()
The same exact thing pops out if you were to type this default command …
  ___________________________________________________________________________________________________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
-------------------------------------------------------------------------------------------------------------

PS Prompt:\> New-Guid

Guid
————
44a1fccf-f301-4a74-af1f-ffde7456f1fc
 _____________________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
  -------------------------------------------------------------------------------------------------------------

That's because the command is accessing that specific type via that function/cmdlet.
However, there are other methods within the class that I'd like to access, so it shows more details.
                                                                                                    _____________/
\_____________________________________________________________________________________________/ <Console> (5)
  Notes + Method Chaining /-------------------------------------------------------------------------------------------------------\
/-------------------------

    The other manner is to use a variable, which allows a user to have extended control over the methods within a
    class/type. Without casting it to a variable, there's no way to access those methods, unless you chain the method
    on at the end. If you use the variable assignment manner, you don't have to do that.
    ____________________________________________________________________
    | Method chaining is OK in some scenarios, but I don't like ever using it. |
    --------------------------------------------------------------------------

    In my opinion, method chaining has an extremely bad pitfall.

    Why … ?
    Well, if any single method in that method chain returns a /empty value … ?

    Now the object is null.
    It WASN'T null … at least, not up until it returned (1) null value.

    THEN, it went from NOT NULL at all … to DEFINITELY NULL.
    And, there's NO indication as to how that happened, either.

    Except, there is an indication, I'm telling you.
    (1) method in the chain returned NULL.

    It can only be (1) method that will do this, because once the object is null, then it can't process any
    additional methods. Cause the object is gone now.

    Look, you can spend entire days arguing with the computer, or me.
    It won't change anything.
    Just know … that I am absolutely correct here. Trust me on that.

    If billions of methods WERE already able to return output … ? They probably did.
    But, so long as (1) single null value got returned … ? Doesn't matter anymore.

    A value was returned in a method chain, it nullified the entire object, and that time was wasted.

    It'll FEEL like it's doing work.
    And, it's definitely doing work (though I don't know what the hell it is),
```

But- you won't SEE any output ...

It goes completely against all forms of logic.
WASN'T NULL, but then it got taken out of the game ... and now it's NULL.

It's like when an overexcited dog sees you come home.
Dog's wicked happy, and starts running around the house, galloping with full overexcited body movements,
practically jumping diagonally at walls and stuff ... ? Acting like a big idiot, cause you came home.

THEN, while he's hurling his whole body around practically giving himself whiplash ... ?
Well, when he's not lookin ... fake him out, and walk back outside.

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

You : Heh. That'll show ya, dog face ...

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Now, the dog is STILL, running around the house.
Cause, he definitely saw ya ...
You know he saw ya, and ... he knows that YOU know he saw ya ...
That's why he'll still be runnin' around the house like a big idiot ... ecstatic that you came home.
But, when the dog does this for a while, realizes you aren't in the house ... he'll be thinkin:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Dog : What the hell, dude ... ?
      Where'd ya go ... ?

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Then he'll be sad, staring at the door.
Tilting his head to the side, slightly.
Then, tilting it the other way ...

He's still thinkin:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Dog : What the hell?
      Thought he was ... home ... ?

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Same principle applies to method chaining, and you ... waiting for the output.
When you see absolutely no output? ... trust me.
Stay a few steps ahead of the dog here.

A single method is all it took for ALL of that hard work to return ... ... nothing.
That's because, it WAS doing work.
It WAS actually processing information.

Until it got nullified from existence ... just like how KEVIN was.

You COULD even be like me, and get pretty creative about what might've happened.
MAYBE, it grew a case of dementia. The object ITSELF doesn't even know where the hell it is.

The PowerShell host, too ... probably has no earthly idea where that data went.
If the PowerShell host, nor the object itself have a clue where the hell it is, or what's going on ...
What can anybody do ... ?

Nobody knows what happened. Nor what to do.

It's like someone having a cabin in the woods, and the owner doesn't know the address.
They'll GO there ... but, they'll get no mail. No phone calls either. Or visitors. Months later, they'll resurface.
Walking around in a grocery store naked, Like Walter White in his fugue state.

These are the dangers of method chaining, my friends.
Unless you do an amazing job of teeing the objects within the classes you write, it's a bad idea.

However, theres a cool way to avoid all of that.
If you DON'T use method chains. That's really, all that needs to be done, to avoid problems.

Methods, in a long continuous chain ... ? Separate them.
Then, if one of those methods returns null ... ? Then object won't be wiped from existence at all.
Nah. It'll still exist, the method just won't do anything to it. Then, you're good.

                                                                                                                    _____________________/
____/ Notes + Method Chaining
  <Console> (6) /---\
/---------------

    ____
    | 2) Variable usage (able to invoke methods within this class externally) |_____________________________________
    /--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

```
PS Prompt:\> $Guid = [DemoGuid]::New()
PS Prompt:\> $Guid

Guid
————
b91e367c-c092-1a24-bea0-c8565ce04098
```

Now, I wrote a custom method that returns output. I DID have it automatically write lines to the console, but it can cause some confusion when calling functions or methods externally.

```
PS Prompt:\> $Guid.Output()

Group[1]: b91e367c
Group[2]: c092
Group[3]: 1a24
Group[4]: bea0
Group[5]: c8565ce04098
Guid: b91e367c-c092-1a24-bea0-c8565ce04098
```

If I want to reset the Guid, I can do so with the variable I already declared, and add the method.
If building a class that MAY have to reset itself, it's good to have a separate method from the instantiation block, right … ? And in that instantiation block, call that method. Then, it'll work perfectly each time.
Review the class I wrote above to see what I mean.

```
PS Prompt:\> $Guid.Reset()
PS Prompt:\> $Guid

Guid
————
91e62a60-a3c9-55e1-9065-1c5842854ed5

PS Prompt:\> $Guid.Output()

Group[1]: 91e62a60
Group[2]: a3c9
Group[3]: 55e1
Group[4]: 9065
Group[5]: 1c5842854ed5
Guid: 91e62a60-a3c9-55e1-9065-1c5842854ed5
```

There's actually a third manner that I haven't mentioned, that I'll have to exhibit, but not yet.

That third manner involves writing a (function/type/class) and adding this particular child class to that parent class. Writing a method within a parent class which instantiates a child class, is a very useful way to build factory-like classes, and this can be implemented with functions too.

Parent class scopes provide a parent scope to access it's child class methods as well as their own, providing the parent a LOT more functionality … but with added functionality is added complexity, and the possibility that things get lost in translation without a reference.

Still, it is extremely useful, however, I won't cover that here.

<Console> (6)

<Properties> (2)

Let me return to the topic of properties. Below are the properties from the FIRST command we ran.

Property	Value
IdentifyingNumber	{2AF42320-5ECF-4BCA-B756-8F3677262D55}
Name	Branding64
Vendor	Advanced Micro Devices, Inc.
Version	1.00.0009
Caption	Branding64

To the untrained eye, it may appear as if I went on a lot of tangents and haven't quite made them seem very relevant … To the trained eye? I've opened up a lot of points of discussion that will make later comparisons a lot easier to understand. That's cause I haven't combined WHY they were each relevant and critical just yet, because if someone is going to teach people about classes, then basically every component of the class needs to be discussed.

Just like in the scene from Full Metal Jacket...

/--__/--\

Drill Sergeant : This is my rifle, this is my gun~!
Privates   : ...this is for fighting, this is for fun....

__/--__/

Truth be told, the privates all needed to know their gun, inside and out.
So far, we went over:

[X] The breakdown of Line #1 [X]

- Casting a variable to the output of a command wrapped in parenthesis AND is indexed

```
                _____________________________3  6
$WMIList = ( Get-WMIObject Win32_Product )[ 0 ]
-------1 2   --------------------------4  ----5
```

Using a (1) VARIABLE,
(2) EQUALS sign, and then a
(4) EXECUTABLE/COMMAND/FUNCTION/TYPE/CLASS/METHOD/VARIABLE/PROPERTY/VALUE
returns its output to the VARIABLE.
However,
(3) PARENTHESIS content gets processed FIRST (See: Math/Order of Operations)
And, (5) SQUARE BRACKETS indicate to select a specific index/entry from an array[]
...(6) INDEX of the returned object.

All indexes start at number 0, not number 1, because a man named PAUL ALLEN said so.

[X] The list of objects that can be operated against [X]

- Objects that are a) parenthesis, or are a b) variable, c) property, d) value, or is e) indexed

EXECUTABLES	COMMANDS	FUNCTIONS	TYPES	CLASSES	METHODS	VARIABLES	PROPERTIES	VALUES

[X] Properties in depth [X]

- ALL of those things above are properties in some way, shape, or form.
- Lambda is a property, or an object, which is ALSO a:

COMMAND	FUNCTION	TYPE/CLASS	METHOD

...and it can perate against:

itself	other objects	any input parameter(s) it is fed

- Nouns, Verbs, Actions, Conditions, Statements, Stories, and Invocations/Instantiations

1) Also properties, albeit ranging in complexity and changing in definition
2) Some of these properties can alter the host object itself

- Properties in relation to:

1) serialization from an object to a string
2) deserialization from a string back into an object/type

- Regular expressions and patterns
- Example class demonstration that expands the functionality of [Guid]

[X] Now we're here [X]

Now, we're gonna start inserting [Types/Classes] in places that are unquestionably convenient.

The point of elaborating on so many of these details, is so that now we can navigate strategies and start
throwing a lot of them together just like I did in Line #1. Because, Line #1 is just the tip of the iceberg.

_____/
_____/ <Properties> (2)
<Console> (7) /-----\
/---------------

Remember, $Pattern was set to "[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}"
1) Return $WMIList.IdentifyingNumber as a) [String], b) [String][GUID] and c) [String][DemoGuid]

/--__/--\

```
If ($WMIList.IdentifyingNumber -match $Pattern)
{
    [String]$WMIList.IdentifyingNumber.ToLower() -Replace "(\{|\})"
    [String][Guid]$WMIList.IdentifyingNumber
    [String][DemoGuid]$WMIList.IdentifyingNumber
}

2af42320-5ecf-4bca-b756-8f3677262d55 ← [String]$WMIList.IdentifyingNumber.
2af42320-5ecf-4bca-b756-8f3677262d55 ← [String][Guid]$WMIList.IdentifyingNumber
```

```
2af42320-5ecf-4bca-b756-8f3677262d55 ← [String][DemoGuid]$WMIList.IdentifyingNumber
```

With that overly-complicated class defininition I just demonstrated?

Well, I can use THAT, to turn MARK back into KEVIN again.
In other words, I can deserialize the string back into an object. Bringing KEVIN back to life.
All that talk about properties, values, regular expressions, conditions, and statements ... ?
It was headed somewhere useful after all.

```
[DemoGuid]$WmiList.IdentifyingNumber

Guid
————
2af42320-5ecf-4bca-b756-8f3677262d55
```

And, if I want the enhanced output ...

```
([DemoGuid]$WmiList.IdentifyingNumber).Output()

Group[1]: 2af42320
Group[2]: 5ecf
Group[3]: 4bca
Group[4]: b756
Group[5]: 8f3677262d55
Guid: 2af42320-5ecf-4bca-b756-8f3677262d55
```

There he is ... KEVIN's BACK in action, dude.

The type block in front of the $Variable.Property, converts that object into that type.
Sometimes the value will not be able to convert itself at all, then you'l get an error message.

However, when it works, it works beautifully, and it feels as if those objects were meant to be.
The class that I wrote ... ? Sure, it might've been overly-complicated to return that specific type ...
If that was ALL I was trying to do ... ?
I could've easily thrown the type in front of the Variable + Property "$WMIList.IdentifyingNumber", like so ...

```
[Guid]$WmiList.IdentifyingNumber / which would've given be this output ...

Guid
————
2af42320-5ecf-4bca-b756-8f3677262d55
```

However, I was doing more than that.
I've now demonstrated how classes are conceptualized, and written.

Dissemination / <Console> (7)

I was also able to integrate these various details back into the overarching lesson plan after all.
Now, it IS definitely debatable as to whether or not anybody should go ahead, and reinvent the wheel, or whatever pseudonym people want to use ... but, that's not what I did at all.

Every wheel has a process in it's construction.
Not every wheel manufacturer out there will be thrilled or ecstatic with the idea of somebody showing EVERYBODY around town, that *ANY* ordinary person who follows their step-by-step manufacturing process ... can build themselves one of the most kick-ass friggen wheels anyone ever made.

Cause at that point, they might get jealous when they see how COOL your wheel making process became.
Then what ... ?
They're gonna ask you to show 'em how it's done.

Cause. They're gonna wanna see this wheel making process of YOURS, since you went ahead and took things to a whole new level ... They may stand around the watercooler at work the next day, telling everyone there, that you were absolutely determined ...

Some of them might've been jealous, but, maybe it became envy after a while.
They saw determination they hadn't seen before ...

Now, you're one of them.
They made the best wheels in history, and then you just learned how to make the best wheels in history too ...

Now, are you really reinventing the wheel ... ? Or are you Bruce Wayne, training to be in the League of Shadows,
having a sword battle with Rahs Al Guhl ... The dude is insanely skilled, can't really leave much to chance ...
dude's been all around the world, being a total menace.

Not a phantom menace per se, but- a menace nonetheless.
Even if Batman gets any edge at all, Rahs Al Guhl is gonna try to slip him up so he loses his balance on the ice.

Batman may have barely won the sword fight on top of that frozen lake ...
But, even Batman has to be mindful of whenever he's on real thin ice ...
... otherwise, he might lose his footing, and drop into that really cold water.
(^ Extended metaphor)

In reality, I tore apart many components of what makes a class, a class, discussed what can be an object,
property, or value, explained what Regular Expressions can accomplish/do, and provided an in depth example
where I've added functionality to a standard base class ...

I'm going to continue with the properties here from $WMIList, as there is (1) other property here that can have
a [type] applied to it that isn't JUST a string ...

... however, it's worth noting that entire classes can be represented by a string, so sometimes the string output
isn't always telling of how complex an object truly is. Regardless ...

Here's the property box again ...

Property	Value
IdentifyingNumber	{2AF42320-5ECF-4BCA-B756-8F3677262D55}
Name	Branding64
Vendor	Advanced Micro Devices, Inc.
Version	1.00.0009
Caption	Branding64

... and that property I'm about to go over, is the Version property.


```
PS Prompt:\> $WMIList.Version

1.00.0009
```

1) Prefix the property $WMIList.Version with "[Version]"

```
PS Prompt:\> [Version]$WMIList.Version

Major  Minor  Build  Revision
-----  -----  -----  --------
1      0      9      -1
```


Now, this all started with some action jam packed into Line #1.
It needed a breakdown, but now, I'm going to finish what I started in reference to properties.

A VARIABLE, is an object that has a 1) name and a 2) value. Those are both properties.
A VARIABLE can ALSO be multiple objects that each have their own 1) name and 2) value.
A NAME is a PROPERTY.
A VALUE is also a PROPERTY.
A PROPERTY is consisted of a 1) NAME and a 2) VALUE.

But, how can a PROPERTY be consisted of it's own PROPERTIES ... ?
... that's just how the game is played, dudemeister.

OBJECTS are made up of PROPERTIES, and VALUES ... and when PROPERTIES and VALUES are accumulated ...
... then you have COLLECTIONS of PROPERTIES.
PROPERTIES definitely qualify as VARIABLES.

But, VARIABLES themselves aren't necessarily PROPERTIES, not unless they are attached to a specific OBJECT ...

<Console> (9) <Properties> (3)

In PowerShell, if a VARIABLE IS NOT attached to an [Object], then the VARIABLE is a property of the current (PowerShell host/execution context). THAT is the [Object] (that/those) VARIABLE(S) are properties of.

If you don't believe me, try this ...

| (1) Open a PowerShell console, and then assign [string]'cool stuff' to a variable named $Stuff |

```
$Stuff = 'cool stuff'
```

Did that?
Alright, cool stuff.

| (2) Now, type $Stuff into the console, and press enter |

```
PS Prompt:\> $Stuff

cool stuff
```

The variable is assigned on your end, right?
Alright, cool stuff. Let's check one more spot.

| (3) Now type in "Get ChildItem Variable:\Stuff" and press enter |

```
PS Prompt:\> Get-ChildItem Variable:\Stuff

Name                           Value
----                           -----
Stuff                          cool stuff
```

So, if you followed this guide correctly, the above information just came up in your console.
See how the name says "stuff", and the value says "cool stuff" ... ?
That's cause thats a straight-up, accurate representation, of a PROPERTY.

There's no question, that is literally, 100% an actual, factual, property class right there.
I mean, looks can be deceiving, actually ...

```
PS Prompt:\> (Get-ChildItem Variable:\Stuff).GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     False    PSVariable                               System.Object
```

Even though it does say PSVariable, fact of the matter is, it has the tell tale signs ...
... that it's basically the same thing as a property. One that has a 1) name, and 2) value.
Anyway ... still don't believe me? Here's where the magic happens.

| (4) Alright, so now, type "exit" |
| (5) The console you had open just closed, that's what 'exit' does. |
| (6) Now open ANOTHER PowerShell console |
| (7) Now, type $Stuff in the console, like you did last time. |

```
PS Prompt:\> $Stuff
PS Prompt:\>
```

See how the variable is gone ... ? That's because it was a PROPERTY of the PREVIOUS PowerShell Host that you exited.
Now, PROPERTIES can be written to disk, via files, or to memory pointers or even the registry ...
But, short of using those manners to EXPORT the PROPERTIES ... ?
VARIABLES are PROPERTIES of the PowerShell Host. So, if you CLOSE it ... ?
Those PROPERTIES are erased.

Hence, why you've seen the VARIABLE go back to "".

Convinced yet ... ?

_____/ <Console> (9)
<Properties> (4) The final stanza, Tony Danza /----------\
/----------

I'm going to restate some things I just stated above slightly differently.
If I know my stuff, what you just did ... ? It returned nothing.

That's because the variable above "$Stuff" was a property of the host object in the session you just closed.
Then you just went ahead and closed that host ... and opened a new one. Now, the property isn't there anymore.

The variable has to be declared again, if you want to continue using that variable.
I realize it's a rather anticlimactic way to prove my point, but-
... that process just proves the point, that variables, if they don't belong to an object?

They're properties of the current PowerShell Host.
So to compare, a square and a rhombus aren't the same thing ...
... not unless a specific rhombus meets the (1) condition where all of its angles are 90 degrees.

When that condition is true, then that rhombus is most definitely, ALSO a square.
Otherwise, a rhombus which doesn't fit that condition, is only a rhombus, not both.

Many of the components I mentioned like nouns, verbs, actions, conditions, statements, stories, and invocations/instantiations, they all have a similar situation where under certain conditions any given sentence may meet MULTIPLE CRITERIA.

The rhombus having 4 90 degree angles is pretty rare. Most of the time, it could be a diamond shaped rhombus.

It won't be a parallelogram, and it might ALMOST look like a square ... ? But, only almost.
In the story I wrote way farther up, a kid wanted a soda, went to a store, grabbed the soda, clerk said "money", kid said "no money", clerk insisted "money", kid started fake crying ... clerk said "money", kid stopped fake crying, kid got mad ...

Well, truth be told, that is a STORY, but it is ALSO basically a PROGRAM.

The kid was a character, the store was a setting, the clerk was another character, the conflict was that the kid wanted the soda and didn't have money. All of those things were a character in the story, the perspective from one to another allowed each of them to be a class with certain definitions.

The point is, all of these things I talk about can be different things, simultaneously.
Objects can be just an object, but that's pretty doubtful ... because they can ALSO be a string at a bare minimum.

Anyway, objects become strings when serialized, and sometimes they have an issue being deserialized, and this can cause problems with object reinstantiation. However, patterns and conditions being met allow deserialization to be more consistent and exact.

Also, sometimes the quantity of properties, or a number of matched existing property names and/or values can ALSO help determine a specific type of object from another. We're just about done talking about properties.
The key thing to remember about all of this, is that:

PROPERTIES	can either be a NULL/VOID OBJECT, SINGLE [OBJECT] or MULTIPLE [OBJECTS[]]
VALUES	can ALSO either be a NULL/VOID OBJECT, SINGLE [OBJECT], or MULTIPLE [OBJECTS[]]
VARIABLES	are technically PROPERTIES of the PowerShell Host/console
TYPES/CLASSES	are consisted of [PROPERTIES], [VALUES], and [METHODS] (also, [CONSTRUCTORS])
COMMANDS	are nearly identical to a [FUNCTION], and they both accept PARAMETERS
METHODS	are a lot like a FUNCTION, but it is a PROPERTY of a TYPE/CLASS/FUNCTION.

[COMMANDS], [FUNCTIONS], [TYPES/CLASSES], and [METHODS] are all LAMBDAS, and they can be a PROPERTY of an OBJECT.

[FUNCTIONS] are a specific calculation, or cluster of calculations that MAY or MAY NOT accept PARAMETERS.

[ACTIONS], [CONDITIONS], [STATEMENTS], and [STORIES] are consisted of all the above, and while all of these things are an [OBJECT] ... ?

They all have varying [SCOPES], [DEFINITIONS], [PROPERTIES] and [VALUES]. Does this feel Dr. Suess-like yet?

Cause, I've barely cracked the surface of how complex logic can get, though this is a SOLID foundation.

To keep this as concise as possible, everything that I just listed can be represented by a [BOOLEAN] when testing conditions. That's just how it works. Even with Regex.

An object's existence can be created or nullified at any moment when a condition is tested, the condition may be consisted of any assortment of things up above, but typically a condition is when a noun does a verb and a count of something matches or exceeds the minimum condition, or likewise in the opposite direction, where a count is equal to or less than the maximum allowable count.

When that condition passes or fails, The switch is activated and that branch is entered.
There are some things that the computer just can't do ahead of time, given the numerous matematical possibilities that exist in the universe ... But, when it joins the branch, it can get more detailed about what to look for.

Hence, why properties are so important.
The property rundown is finito. Now, you just gotta ask yourself one question ... Maybe 3.

Who's the boss, NOW, buddy? Is it you? Or Tony Danza? (Or, is it Angela ... ?)

<Properties> (4) The final stanza, Tony Danza

Back to Work

It's time to go all out beast mode with the $WMIList Object.
Now we're going to start dynamically building a class definition, for WMIObject.

Some people will probably be pretty amazed by this process here.
To get all of the property 1) names, 2) types, and 3) values, use the property:

$WMIList.PSObject.Properties ⇒ (produces a pretty big chart)

We can use techniques similar to Line #1, by using the pipeline.
Pipeline operations can use direct property names to select or query internal property values.

Assigning the variable $Prop to the entire operation, will store the output from the operation to the variable.

```
                The REGEX to the left, is filtering out WMI+PS classes while making the assignment.
_______________ The Regex probably needs a little explanation.
|-------------| The Regex pattern is wrapped with parenthesis, indicating a capture group
| "(^_|^PS)" | The pipeline symbol in the middle is an OR symbol.
|_____________| The Caret indicates the START of a string
--------------  The underscore matches an underscore, and the PS matches the letters PS.
```

If you're hesitant to run the operation that isn't commented out below, feel free to run the line right underneath the dashed line below, it won't actually assign the output to anything. The one below that which says Format-Table, will show you the table output, not the list.

Back to Work

Work for Real

We want to format the class with the right spacing between elements, cause I'm a perfectionist AND precise.
All of the type names and property names will line up if we collect em all to find the length of the longest type.

Assign the variable $Types to the following loop ONE-LINER.
The entire ONE-LINER BELOW is a COLLECTION of STRATEGIES thrown into a single line, actually.
Some of my Voodoo 3 5000 magic ...

To explain, it's a multifaceted ONE-LINER involving:
- ForEach-Object haphazardly piping itself into an array ...
- $_ token with the property length being greater than 0 in square brackets acts as a switch
- $False selects slot 0 in the array returning the string "String", cause that's binary for ya.
- $True selects slot 1 in the array returning ($_ -Replace "System\.","")

Work for Real

<Commands> (1)

```
_____________________________________________________________________________
| Assign variable $TypesMaxLength to the operation Types. Or, highlight them both and press F8  |_____________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-------------------------------------------------------------------------------------------------------------

$Types           = $Prop.TypeNameOfValue | % { @("String",$_ -Replace "System\.","")[$_.Length -gt 0]}
$TypesMaxLength  = ($Types | Sort-Object Length)[-1].Length
_____________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 -------------------------------------------------------------------------------------------------------------

_____________________________________________________________________
| Now, get the names, sort by length, and then grab the last name length since it has the longest string |_____
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\

$Names           = $Prop.Name
$NamesMaxLength  = ($Names | Sort-Object Length )[-1].Length
_____________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 -------------------------------------------------------------------------------------------------------------
```

Everybody reading this survived all of that somehow, right ... ? Nothin' exploded ... ? No demons teleporting into the world around ya ... ? No Strogg armies invading Earth ... ? No alarms went off ... ? We're all good here, right ... ?

Alright ... just checkin'.

```
_____________________________________________________________
| Create a variable named $Definition, as a hash table with these properties: |
| [String]   ClassName   |-----------------------------------------------------
| [Object[]] Property    |
| [String]   Param1Type  |
| [String]   Param1Value |
| [Object[]] Constructor |_____________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-------------------------------------------------------------------------------------------------------------

$Definition      = @{
```

```
    ClassName    = "Win32_Product"
    Property     = @( )
    Param1Type   = "[Object]"
    Param1Value  = "`$WMIObject"
    Constructor  = @( )
}
```

```
| Run through all:                                                                                  |
| 1) property types                                                                                 |
| 2) property names                                                                                 |
| 3) set the property values to the corresponding property value of the input parameter             |
```

```
| Add each TYPE/NAME to $Definition.Type array                                                      |
| Add each $Name in $Names with spacing to the $Definition.Constructor array                        |
```

```
ForEach ($X in 0..($Names.Count-1))
{
    $TypeBuffer  = " " * ($TypesMaxLength - $Types[$X].Length + 1)
    $NameBuffer  = " " * ($NamesMaxLength - $Names[$X].Length + 1)
    $Definition.Property      += "    [{0}]{1}{2}`${3}" -f $Types[$X] ,
    $TypeBuffer, $NameBuffer, $Names[$X]
    $Definition.Constructor  += "        `$This.{0}{1} = {2}.{0}" -f $Names[$X],
    $NameBuffer, $Definition.Param1Value
}
```

```
| Now, we can write all of the information we collected, to a class definition, and either:         |
| 1) copy it to the clipboard                                                                       |
| 2) write it to the console to then copy paste it that way back into the editor                    |
| 3) hold off ... cause it's a third option and it says "recommended"                               |
```

```
$ClassDefinition = @("Class $($Definition.ClassName)",
"{",
($Definition.Property -join "`n"),
"    $($Definition.ClassName)($($Definition.Param1Type)$($Definition.Param1Value))",
"    {",
($Definition.Constructor -join "`n"),
"    }",
"}") -join "`n"
```

```
<Class Definiton> /<Commands> (1)
```

```
| Here is the OUTPUT of ALL that stuff we just did ...  |
```

```
PS Prompt:\> $ClassDefinition

Class Win32_Product
{
    [UInt16]                                $AssignmentType
    [String]                                      $Caption
    [String]                                   $Description
    [String]                                   $ElementName
    [String]                                      $HelpLink
    [String]                                 $HelpTelephone
    [String]                               $IdentifyingNumber
    [String]                                    $InstallDate
    [String]                                   $InstallDate2
    [String]                                 $InstallLocation
    [String]                                   $InstallSource
    [Int16]                                     $InstallState
    [String]                                      $InstanceID
    [String]                                         $Language
    [String]                                   $LocalPackage
    [String]                                           $Name
    [String]                                   $PackageCache
    [String]                                    $PackageCode
    [String]                                    $PackageName
    [String]                                      $ProductID
    [String]                                    $RegCompany
    [String]                                      $RegOwner
```

```
    [String]                                  $SKUNumber
    [String]                                 $Transforms
    [String]                               $URLInfoAbout
    [String]                              $URLUpdateInfo
    [String]                                     $Vendor
    [String]                                    $Version
    [UInt32]                           $WarrantyDuration
    [String]                          $WarrantyStartDate
    [UInt32]                                  $WordCount
    [Management.ManagementScope]                  $Scope
    [Management.ManagementPath]                    $Path
    [Management.ObjectGetOptions]               $Options
    [Management.ManagementPath]               $ClassPath
    [Management.PropertyDataCollection]      $Properties
    [Management.PropertyDataCollection]    $SystemProperties
    [Management.QualifierDataCollection]     $Qualifiers
    [ComponentModel.ISite]                         $Site
    [ComponentModel.IContainer]               $Container
    Win32_Product([Object]$WMIObject)
    {
        $This.AssignmentType     = $WMIObject.AssignmentType
        $This.Caption            = $WMIObject.Caption
        $This.Description        = $WMIObject.Description
        $This.ElementName        = $WMIObject.ElementName
        $This.HelpLink           = $WMIObject.HelpLink
        $This.HelpTelephone      = $WMIObject.HelpTelephone
        $This.IdentifyingNumber  = $WMIObject.IdentifyingNumber
        $This.InstallDate        = $WMIObject.InstallDate
        $This.InstallDate2       = $WMIObject.InstallDate2
        $This.InstallLocation    = $WMIObject.InstallLocation
        $This.InstallSource      = $WMIObject.InstallSource
        $This.InstallState       = $WMIObject.InstallState
        $This.InstanceID         = $WMIObject.InstanceID
        $This.Language           = $WMIObject.Language
        $This.LocalPackage       = $WMIObject.LocalPackage
        $This.Name               = $WMIObject.Name
        $This.PackageCache       = $WMIObject.PackageCache
        $This.PackageCode        = $WMIObject.PackageCode
        $This.PackageName        = $WMIObject.PackageName
        $This.ProductID          = $WMIObject.ProductID
        $This.RegCompany         = $WMIObject.RegCompany
        $This.RegOwner           = $WMIObject.RegOwner
        $This.SKUNumber          = $WMIObject.SKUNumber
        $This.Transforms         = $WMIObject.Transforms
        $This.URLInfoAbout       = $WMIObject.URLInfoAbout
        $This.URLUpdateInfo      = $WMIObject.URLUpdateInfo
        $This.Vendor             = $WMIObject.Vendor
        $This.Version            = $WMIObject.Version
        $This.WarrantyDuration   = $WMIObject.WarrantyDuration
        $This.WarrantyStartDate  = $WMIObject.WarrantyStartDate
        $This.WordCount          = $WMIObject.WordCount
        $This.Scope              = $WMIObject.Scope
        $This.Path               = $WMIObject.Path
        $This.Options            = $WMIObject.Options
        $This.ClassPath          = $WMIObject.ClassPath
        $This.Properties         = $WMIObject.Properties
        $This.SystemProperties   = $WMIObject.SystemProperties
        $This.Qualifiers         = $WMIObject.Qualifiers
        $This.Site               = $WMIObject.Site
        $This.Container          = $WMIObject.Container
    }
}
```

```
    ________________________________________________________________________________________________________________
    \__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
     ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
                                                                                                      ___________________/
\____________________________________________________________________________________________________/ <Class Definiton>
  <Class Definition> Summary /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Above, the newly constructed class has been cleanly formatted, and procedurally spaced out.
But, here's why it's not all that COOL or USEFUL... yet.

If the constructed class is NO different than selecting ALL of the properties on the default command output, then... there's basically no point in doing all of this work. If you're adding methods, or other properties, THEN, there is. If you're NOT, though? There isn't much of a reason... regardless, it's a good way to see how it is CONSTRUCTED.

But in practice, unless you're planning on ADDING custom methods or properties...?
Then, there's an easier way to do all of that, and it's as simple as running the command, and capturing the output to a variable.

Cause that's how you COULD get the same output from the default command with far less work.

```
Then again, there could be other reasons, maybe some properties are hidden and you don't LIKE when properties
are hidin' out on ya … Well, it's hard for me to imagine that someone would do all of this just to spite some
hidden properties … but maybe somebody would … ? Idk.

However, there was a point to doing this after all.

Instantiate the variable, class definition below.
This enables the class to be used without having to explicitly write the
$Classdefinition output to the clipboard or the editor.
______________________________________________________________________________________________________
| (3) You just highlight the phrase, press F8, and let it load into memory. No file, No copy paste. |
------------------------------------------------------------------------------------------------------

No muss, no fuss, and, no guys named Gus …

/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\

Gus : *brief pause* … what?
Me  : You heard me …
Gus : *pause* … cause my name is Gus … ?
Me  : Yeah.
Gus : You know, that is extremely obnoxious, I'm not allowed around here, cause my name is Gus?
Me  :  … I never said that.
Gus : Oh. Well, what ARE you saying then … ?
Me  : Uh … *pause* Just sayin' that no guys named Gus were part of the script.
Gus : Ohhhhh …
Me  : Yeah, what'd you think I meant, dude?
Gus : I … I dunno man. Like I wasn't good enough or somethin' cause my name is Gus …
Me  : Nah, I never meant it like that, I was just sayin, no muss, no fuss-
Gus :  … no guys named Gus.
Me  : Yeah, I didn't mean like, you're not ALLOWED, to hang out … or whatever.
Gus : I've heard that phrase all my life man.
      No muss, no fuss this … no muss, no fuss that …
      No muss, no fuss, *everybody looks at me* there's Gus.
      Then everybody just goes completely silent, staring at me …
      It's unreal, dude.
Me  : Wow.
      Sorry for being that jerk that just had to make your life slightly more miserable …
       … for trying to make some corny rhymes …
Gus : I ain't mad at ya man …
      I just, felt like I was part of the team~!
Me  : You are part of the team, dude.
      I take it back.
Gus : Really … ?
      You mean that … ?
Me  : Yeh man.
      You don't even know bro, I have no reason not to respect the name Gus …
      Not one.
      If anything, plenty of reasons to respect it after all … cause.
      There's a little hot dog shack down in Watervliet, Gus' Hot Dogs.
Gus :  … are they good … ?
Me  : Are they GOOD … ?
      Are you kiddin' me?
      Buddy, that place is so many people go, when they are hungry, and they want some really good hot dogs.
      I can't really think of a better place, than Gus' Hot Dogs … in terms of the best hot dogs money can buy.
      Now I'm hungry and I wanna go there.
Gus : Wow.
      I … that made me feel WAY better already man …
Me  : Yeah man, look.
      No muss, no fuss, be sure to invite a guy named Gus.
      That'll be the motto.
Gus : Hey man … I appreciate that.
Me  : No problem dude.
      Hey- *points at Gus*
Gus : *both eyebrows up, slight head nod* Sup buddy … ?
Me  : I'm gonna get back to the lesson plan now, alright?
Gus : Oh yeah, not a problem buddy. Glad to be part of the team. Later.
Me  : You got it Gus. Later.

\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/

Man. Guess I shouldn't haphazardly try to rhyme everything that comes to mind.
That dude, sounds like he's been through the ringer … *shakes head*.
                                                                                                  ____________________________/
\________________________________________________________________________________________________/ <Class Definition> Summary
  Invoke $ClassDefinition /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

Anyway, yeah. If you want to invoke the class that was just created with the last several breaks of code,
then here is where that code gets turned into a class definition that can be invoked.
________________________________________________________________
| Go ahead and highlight the expression below, and press F8.   |
| If you don't, then the script won't work after this … So …  |____________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
```

```
---------------------------------------------------------------------------------------------------------------------
Invoke-Expression $ClassDefinition

_____________________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ---------------------------------------------------------------------------------------------------------------------

Now, to collect ALL of those WMI objects, this MAY or MAY NOT take more time than the first time,
that (Get-WMIObject Win32_Product) was accessed above ... Like the Lotto? Hey. You never know.
_____________________________________________________________________________
| Manner (1) | $Collect       = Get-WMIObject Win32_Product | % { [Win32_Product]$_ } |
-----------------------------------------------------------------------------

or ...
_____________________________________________________________________
| Manner (2) | $Collect       = [Win32_Product[]]@(Get-WMIObject Win32_Product) |
---------------------------------------------------------------------

They do the same thing.

I actually would like to discuss the difference between these two manners right here.

Manner (1) says that the command gets to ForEach-Object itself, into a new instantiaion of the class we just
generated, and then invoked into memory, above using the $ClassDefinition variable. For each object it finds ... ?
Then, a new object is instantiated, even if it ONLY returns a SINGLE OBJECT.
Then, the output is saved to the variable $Collect.

Manner (2) literally does the same exact thing, but since it is written differently, it's worth explaining that
they may not ALWAYS result in the same output ...

The [Win32_Product[]] is an object array, you can tell with the "[]" inside the square brackets.
The @ symbol is actually denoting that the content of those parenthesis is EXPECTING MULTIPLE OBJECTS,
So, using that technique with a FUNCTION in the PARENTHESIS, allows EVERY object that the function returns,
as THAT specific type/class, X number of times- even if it ONLY returns a SINGLE OBJECT.

Maybe people don't realize that a type and a class are actually the same thing (usually), and I should've said
that at some point in the previous 1500+ lines.

Well ... a type is a class, and a class is a type ... the terms, I'm fairly certain, are literally interchangeable.
                                                                                                    _____________________/
\___________________________________________________________________________________/ Invoke $ClassDefinition
  Computer Updated - Press F1 /-------------------------------------------------------------------------------------\
/-----------------------------

Uh-oh ... Matthew Caldwell didn't do any Google searches BEFORE having me image a thousand machines in the
warehouse ... Now I gotta do ALL that work all over again, because of this following article ...

https://xkln.net/blog/please-stop-using-win32product-to-find-installed-software-alternatives-inside
-----------------------------------------------------------------------------------------------------

Now, this is where our development process got thrown on it's head.
Why ... ? Oh, I'll tell ya ...

These guys say that the manner I've been using to collect a list of installed applications, is the way that LAME
people do it. Yep. Me ... ? Doing something the lame way. As usual. Couldn't believe that these guys knew me so well.

Still. When I found out? I swear, I didn't cry. Kept my head up. Eyes forward. Just ... kept on goin'.

Eventually ... I realized that they don't actually say anything about ANYBODY being lame. Nah.
I just interpreted it that way ...

They were just saying that WMIObject Win32_Product is SORTA broken (because legacy WMIC is rather complicated)

I felt better after I realized that ...
Can't believe I actually *allowed* myself to be so distraught by words they never even said ...
Jeez ... That's when I took a deep breath, said to myself "it'll be okay" ... and then, read the entire thing
they said.

I'm just kidding about being upset, I already have a function that does what this article suggests, because I
figured out how the experts look for installed programs way back in 2019 after I started following a team of
awesome guys' GitHub project ...

Come to find that they used to be some heavy hitters at Microsoft.
These guys wrote a book that I read.
That book became an odyssey into the programming world ...

Anyway, Johan Arwidmark and Mykael Nystrom wrote that book.
What I didn't know back then, and what I happen to know NOW, is that they literally wrote the friggen book,
about how professionals get stuff done. Not only did they have so much helpful information about how the
professionals get stuff done ... they also have a portion of their books and projects reserved for experts
that know what they're doing.

They have to at least TRY to keep this information low-key, because of how USEFUL it is. (Kidding)

One of those suggestions, is using the registry to look for installed program and stuff.
For years they had an edge over everybody with this WAY cooler, more efficient way to get stuff done ...
I realize, they WILL probably tell people that they didn't write a special series of chapters and entries for
```

the most highly seasoned experts out there in the wild... But obviously, they're not gonna tell people about secret chapters they wrote and authored...

Some people really should consider... not all heroes wear capes.
These guys know what the hell they're doin'.

Anyway, the registry.
It's not a thing people should play games with, but, sometimes an expert gets called in, to take a look around and perform a highly skilled site survey. Sometimes that happens to include taking a pretty quick look at it, especially if you're looking for programs to install or uninstall, and you need those programs to be a specific version, or higher.

Guess who I learned from on how to do those things? Heh. One of the most highly seasoned experts at Microsoft.
Following the link above they say: Don't use Get-WMIObject Win32_Product anymore...

It's slow, incomplete, problematic, not optimized, it's just a bad way to go about getting that info anyway, since Microsoft LITERALLY invested a LOT of time and resources in CimV2.

Fine I won't use WMIC. However, it means I have to reproduce the class stuff I already did...
Look. If you wanna go pro...? Use the REGISTRY, and look in the following keys/paths:

64-bit	HKLM:\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Uninstall
32-bit	HKLM:\Software\Microsoft\Windows\CurrentVersion\Uninstall

```
                                                                                    ____________________________/
\__________________________________________________________________________________/ Computer Updated - Press F1
  <Console> (10) /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Checks both (32-bit/64-bit) registry paths | No games being played here...

```
 ____________________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

PS Prompt:\> $Apps = ForEach ($Item in "\Wow6432Node","")
{
    "HKLM:\Software$Item\Microsoft\Windows\CurrentVersion\Uninstall\*" | Get-ItemProperty
}

PS Prompt:\> $Apps | ? DisplayName -match "Visual Studio"

DisplayName     : Visual Studio Community 2022
InstallDate     : 20220402
InstallLocation : C:\Program Files\Microsoft Visual Studio\2022\Community
DisplayVersion  : 17.1.6
Publisher       : Microsoft Corporation
DisplayIcon     : C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE\devenv.exe
UninstallString : "C:\Program Files (x86)\Microsoft Visual Studio\Installer\setup.exe" uninstall..
ModifyPath      : "C:\Program Files (x86)\Microsoft Visual Studio\Installer\setup.exe" modify --..
RepairPath      : "C:\Program Files (x86)\Microsoft Visual Studio\Installer\setup.exe" repair --..
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Wow6432Node\Mi..
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Wow6432Node\Mi..
PSChildName     : 889e9450
PSProvider      : Microsoft.PowerShell.Core\Registry
DisplayName     : CCleaner
UninstallString : "C:\Program Files\CCleaner\uninst.exe"
Publisher       : Piriform
InstallLocation : C:\Program Files\CCleaner
VersionMajor    : 6
VersionMinor    : 0
DisplayVersion  : 6.00
DisplayIcon     : C:\Program Files\CCleaner\CCleaner64.exe
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Wind..
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Microsoft\Wind..
PSChildName     : CCleaner
PSProvider      : Microsoft.PowerShell.Core\Registry

______________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
                                                                                                 ________________/
\_______________________________________________________________________________________________/ <Console> (10)
  Back to the Drawing Board /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

The problem is, now the class definition is out of date!
We don't need the old one anymore, cause it doesn't have what we need.

Time to write a totally new one.
However, what complicates matters here, is that each of these registry keys all have differing properties/entries in the registry - not only are they VERY different from Get-WMIObject Win32_Product, but they also contain inconsistent properties and values.

While SOME of the values are consistent... some of em aren't. So, this is a bit of a dilemma...

We might actually have to whip out the magic lamp again and make a new wish ...

How long should a person with a magic lamp wait, before making another wish ... ?
Cause, what if the genie is taking a nap, we try to get him to come out ... but ...
He was getting his genie beauty sleep. Then you get a real snippy genie with an attitude ... ?

/--__/--\

```
Genie : YEAH ... ? SUP ... ?
You   : Hey, sorry for waking you ...
Genie : Oh you better be, pal ... I was gettin' my beauty sleep in there, and you just had to go wakin' me up ...
You   : Sorry ...
        I just, I need to make another wish.
Genie : Oh you do, do you ... ?
        Interesting ...
You   : Yeah.
        I need to write a new class that can accommodate a bunch of registry keys on Windows.
Genie : Oh *scoffs*
        Obviously, that is worthy of another wish ... that ... is one of the toughest things a person can even do.
You   : Alright Genie, you don't have to patronize me, I get it ...
        I know I'm no Albert Einstein, but this really is hard to do right ...
Genie : Well ... you're not wrong. It is hard to do right. Plenty of ways to do it wrong.
You   : Yeah. Well ...
        *suddenly, you remember that you read a tutorial once upon a time*
        *a guy wrote a lesson plan that might help ya out ... *
You   : Ya know what, genie ... ?
Genie : Yeah ... ?
You   : I think I'm all set for right now, I'll save my wish for later ...
Genie : What ... ?
You   : Nah, I just remembered something ...
        I'm gonna give it a shot.
Genie : But, you can't just say that you wanna make a wish, and then not make one ...
You   : Why not?
Genie : Well ... *scoffs* just cause ...
        That's rude.
        Obviously ...
You   : Hey look Genie, I think you're being incredibly disrespectful ...
         ... and I don't even know if I really want the wishes anymore.
        You could totally have em back ...
Genie : No, no, no, no ...
        Alright, you can make this wish some other time.
You   :  ... really?
Genie : Yeh.
        Sorry boss man.
You   : Strange.
        Alright, talk to you later.
```

__/--__/

Strange. The genie came at ya all sideways at first, but when told that he could even have the wishes back ... ?
He changed his tune~!

Maybe the reason, is cause when people would rather do something tough on their own, especially if they can make
any wish they want ... ? Genies know that means they won't be needed ... Then they gotta wait around for like,
10 thousand years all over again ... They probably get bored waiting around all that time. So.

There's no need for a magic lamp, nor any wishes here.
We are just gonna use a strategy, in order to: 1) conceptualize, and 2) build a class factory

One that handles the properties of this specific situation involving registry keys, for programs.

So, defining a class may be difficult to do, UNLESS theres a way to determine which TYPE of registry entry each
of them may be, or we could even pull a template object, and use that draft the class ...
If we did that, then, we could provide an abstract way to force all of the items in $Apps to fit within the
same class for every entry.

To do this, lets start with a common application that was installed via MSI, since that has standard options
and the most consistency (so, like Microsoft Edge, or Google Chrome ...)

```
                                                                                                                  _____________________________/
\_____________________________________________________________________________________________________________/ Back to the Drawing Board
  <Template> /-------------------------------------------------------------------------------------------------------------------------------\
/------------
```

I currently have Microsoft Edge installed on this machine, it is hands down, the best there is.

```
 ____________________________________________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-------------------------------------------------------------------------------------------------------------------------------------

PS Prompt:\> $Edge            = $Apps | ? DisplayName -match "(^Microsoft Edge$)"
PS Prompt:\> $Edge

DisplayName     : Microsoft Edge
DisplayVersion  : 101.0.1210.53
Version         : 101.0.1210.53
```

```
    NoRemove        : 1
    ModifyPath      : "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /instal..
    UninstallString : "C:\Program Files (x86)\Microsoft\Edge\Application\101.0.1210.53\Installer\se..
    InstallLocation : C:\Program Files (x86)\Microsoft\Edge\Application
    DisplayIcon     : C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe,0
    NoRepair        : 1
    Publisher       : Microsoft Corporation
    InstallDate     : 20220521
    VersionMajor    : 1210
    VersionMinor    : 53
    PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Wow6432Node\M..
    PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_LOCAL_MACHINE\Software\Wow6432Node\M..
    PSChildName     : Microsoft Edge
    PSDrive         : HKLM
    PSProvider      : Microsoft.PowerShell.Core\Registry
    ____________________________________________________________________________________________________
    \__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
     ----------------------------------------------------------------------------------------------------

                                                                                                  ___________/
\_____________________________________________________________________________________________/ <Template>
  Non-Default Registry Keys /-------------------------------------------------------------------------------------------------\
/--------------------------
```

This time, we want to add a property for NON-DEFAULT registry keys in this uninstall path folder.
[Analogy]: Entries with NON-DEFAULT registry keys are similar to when somebody doesn't fit into
the typical mold that most people in society fit into since they're different than everyone else.
They have WAY different properties, and, they're hard to expect. People might get upset and say:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Default   : Hey buddy ...
              Why you always gotta be WAY different than everybody else?
Non-Default : Cause, I don't fit into the typical mold most people in society fit into ...
Default   : *Everybody shakes their head* That's obnoxious.
Non-Default : No way buddy, *points back at em* YOU'RE the obnoxious one, pal ...
              Askin' me why I gotta be different ... ?
              That's about the most obnoxious thing anybody could even SAY or DO ...
Default   : I could literally ask all of the people here in this room, what their properties are.
              They're gonna tell me that they have the same properties that I have....
Non-Default : Not me though.
Default   : Yeah I know.
              That's why we all collectively sigh at you for being WAY different.
Non-Default : It's cause I'm advanced ...
              …you can't even expect what properties I contain.
Default   : Yeah.
              That's ...
              …why we can't stand ya sometimes, pal.
Non-Default : Just cause I'm advanced, doesn't mean I'm a bad person, dude ...
Default   : Well, nobody ever said that you were a bad person ...
Non-Default : Sounded like it though ...
Default   : On any given day, your name could change to Kevin, and you'll be older or heavier ... ?
              You're like a shape shifter ...
Non-Default : ... it's cause I'm advanced. I have more properties than you.
Default   : Yeah buddy ...
              Sure.
              It's cause you're "advanced".
              Next week your name'll be Jeff.
Non-Default : Yeah ... ?
              Maybe it WILL be Jeff ...
              You act like it's such a heavy burden.
Default   : It is! *shakes head*
              This dude really is unbelievable! *everyone collectively agrees*

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Now, NON-DEFAULT registry keys in these uninstall paths (64-bit/32-bit) may contain some very old school entries
that aren't updated anymore. So, whether those entries are ADVANCED, or just OLD.

Don't get down on registry keys in this folder if their properties aren't the same as everybody elses.
Because, we can actually implement a DESIGN CHANGE to the CLASS that ACCOMMODATES any *unexpected properties*.
They won't be a part of the BASE class properties, but they'll still be visible and accessible.
So, everybody actually wins.

There's a few ways we could do this, we could ADD a member to each class for EACH property.

However, that isn't a great idea.
That will make many of them TOO DIFFERENT, and not adhere to a TABLE, as they'll be TOO unique in some cases.

Then, everybody will have to collectively sigh at the "advanced" guy's properties.
So, the best option would be to create a property that can host those "extraneous" properties, and that
property's value will be an object that can host an array of ADDITIONAL properties & values that are NON-DEFAULT.

That is the best way to go, actually.

Now, lets choose a property name like "EntryUnique". If we add this property BEFORE we access the PSObject.Properties, the addition will form to the format of the previously written stuff above.

We'll also need to ADD a few METHODS to this class (methods are nested functions), so it can refer to itself, provide self referencing brevity, and convert each individual NON-DEFAULT property, into an object array for each property. Then, we want to have a method that can format all of those objects and properties and write the output.

Non-Default Registry Keys

<Commands> (2)

We'll run through the script again ... but with a few added steps and more explanations.
Get $Edge.PSObject.Properties where the name doesn't start with either "_" or "PS"

```
PS Prompt:\> $Prop      = $Edge.PSObject.Properties | ? Name -notmatch "(^_|^PS)"
```

For the script that applies spacing/formatting to include the new property, and not need an added script at the end, we can inject the new property into the variable $Prop. But FIRST, we need to understand what TYPE of object it is, to instantiate that TYPE.

What does this variable $Prop, get us back in the console ... ?

```
PS Prompt:\> $Prop

Value                                MemberType   IsSettable IsGettable TypeNameOfValue Name
-----                                ----------   ---------- ---------- --------------- ----
Microsoft Edge                       NoteProperty       True       True System.String   Display..
101.0.1210.39                        NoteProperty       True       True System.String   Display..
101.0.1210.39                        NoteProperty       True       True System.String   Version..
1                                    NoteProperty       True       True System.Int32    NoRemov..
"C:\Program Files (x86)\Microsoft\E." NoteProperty       True       True System.String   ModifyP..
"C:\Program Files (x86)\Microsoft\E." NoteProperty       True       True System.String   Uninsta..
"C:\Program Files (x86)\Microsoft\E." NoteProperty       True       True System.String   Install..
"C:\Program Files (x86)\Microsoft\E." NoteProperty       True       True System.String   Display..
1                                    NoteProperty       True       True System.Int32    NoRepai..
Microsoft Corporation                NoteProperty       True       True System.String   Publish..
20220507                             NoteProperty       True       True System.String   Install..
1210                                 NoteProperty       True       True System.Int32    Version..
39                                   NoteProperty       True       True System.Int32    Version..
{}                                   NoteProperty       True       True System.Object[] EntryUn..
```

Looks like a standard, ordinary, run-of-the-mill, object collection table.
In order to INSERT a NEW PROPERTY to this list of properties, we have to figure out what each of these objects actually are. Now you could also use the Add-Member cmdlet, but there's another way.

The $Prop.GetType() method will return the "Object Type" object as seen below.

```
PS Prompt:\> $Prop.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     Object[]                                 System.Array
```

So, it's an object array, which is easy to spot because of the ol' double square brackets there.
We sorta knew this from the object collection table up above.

But, in order to ADD a new object to it, we have to determine what TYPE these objects in the array actually are, and then ... instantiate that TYPE. Select the first item of the array to determine what type of object array it is.

```
PS Prompt:\> $Prop[0].GetType()

IsPublic IsSerial Name                 BaseType
-------- -------- ----                 --------
True     False    PSNoteProperty       System.Management.Automation.PSPropertyInfo
```

Alright, so it's a PSNoteProperty array.

We knew that from the object table, but now we aren't making assumptions.

We can attempt to directly access the underlying base type.
Now, is PSNoteProperty an object that anybody could instantiate in PowerShell, without calling an assembly or adding a type definition ... ?

```
PS Prompt:\> New-Object PSNoteProperty

New-Object: A constructor was not found. Cannot find an appropriate constructor for type PSNoteP..
```

Apparently it is, because it wouldn't have come back with a SPECIFIC error message that says to add a constructor.
Otherwise, it would've said:

```
New-Object: Cannot find type [PSNoteProperty]: verify that the assembly containing this type is ...
```

Since the cmdlet New-Object PSNoteProperty doesn't provide an idea for the PARAMETERS we need to feed it without help, lets call the .NET base type, via [PSNoteProperty]::New but, with a twist.
BTW: "[PSNoteProperty]::New()" literally does the same thing as "New-Object PSNoteProperty"

_____/ <Commands> (2)

[CLR/.Net Tricks Explained]: Auto Completion, and Overload Definitions

Trick #1 : Auto Completion

```
[PSNoteProperty]::      ← Press CTRL+SPACE here in the [Console], to show default static methods

PS Prompt:\> [PSNoteProperty]::new(
Equals           new              ReferenceEquals
```

Trick #2 : Overload Definitions

```
[PSNoteProperty]::New   ← Press ENTER here (NO parenthesis/params) to show overload definitions

OverloadDefinitions
-------------------
PSNoteProperty New(String Name, System.Object Value)
```

[Parallel Study]: Looks like some standard-issue CSharp up above. Convert to PowerShell like so ...

[CSharp Notation (w/ Haskell Casing)]

```
PSNoteProperty New(String Name, System.Object Value)
```

[PowerShell Notation]

```
[PSNoteProperty]::New($Name,$Value)        ← $Name and $Value each need to be defined, for this.
[PSNoteProperty]::New("EntryUnique",@( ))  ← Direct value entry, no predefined variables needed.
```

[Compare/Contrast]

Q: Why's PowerShell similar to CSharp ... ?
A: Cause they're both made by Microsoft, and well made, that's why.

1: Class/Type Name, 2: Static Method/Function, 3: Param Type, 4: Param Variable

```
(1) CSharp     Ex #1 | PSNoteProperty New( String Name, System.Object Value )
    ------     -----   -------------1 --2  -----3 ---4  ------------3 ----4
(2) CSharp     Ex #2 | PSNoteProperty Variable = New PSNoteProperty( Name, Value )
    ------     -----   -------------1 -------X   --2 -------------1  ---4  ----4
```

```
| (3) PowerShell Ex #1 | [PSNoteProperty]::New( [String] $Name, [Object] $Value )    |
|     ---------- -----   ---------------1  --2  -------3 -----4 -------3 -----4      |
| (4) PowerShell Ex #2 | New-Object PSNoteProperty -ArgumentList $Name, $Value       |
|     ---------- -----   ---------2 -------------1 -----------X ----4  -----4        |
| (5) PowerShell Ex #3 | New-Object PSNoteProperty [String] $Name, [Object] $Value   |
|     ---------- ------  ---------2 -------------1 -------3 ----4  -------5 -----6   |
-------------------------------------------------------------------------------------
```

Each entry is Haskell-Cased. That means they all look <Proper with Capitalized Letters>.
Not every entry on the list will actually WORK if you go to use it as is.

Because, they are each atypically split to provide a comparison chart.
Some of these WILL work though, but testing them all, and knowing WHY which ones work...
...and which ones don't, is an important skill to have. Hence, why I made the chart.

While there ARE very subtle differences and variations here, they ALL draw some parallel structure AND more than just abstract similarity to one another. Can you spot the similarities?

```
                                              ______________________________________________________________________/
\____________________________________________/ [CLR/.Net Tricks Explained]: Auto Completion, and Overload Definitions
  PowerShell Class/Type Engine Assumptions /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Not all of the labels appear in each instance.
With PowerShell, the console is pretty good about ASSUMING (classes/types) for each variable.
That's because of the PowerShell Class/Type engine.

CSharp has no such 'making assumptions about any given variable' functionality, because it requires some hefty-handed SPECIFICITY and NEEDS to be strongly typed... Otherwise, expect many failures.

In PowerShell, the engine was made to make a lot of assumptions, and it was designed SO well, that it gets it right quite consistently. It's easy to take what it does, for granted.

```
__________________________________________________________________________
| (1) CSharp #1 → PSNoteProperty New( String Name, System.Object Value ) |
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

CSharp #1 has spacing that would cause compilation failures right off the bat (I believe), they were added to examine the components of each line. Typically each CSharp entry needs a semicolon at the (end of line/EOL), but not AFTER method invocation. In PowerShell, EOL semicolons are NOT necessary.

```
_______________________________________________________________________________
| (2) CSharp #2 → PSNoteProperty Variable = New PSNoteProperty( Name, Value ) |
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

I don't work with CSharp often enough to remember the specifics of calling types for each variable name and etc...
But- I still read it every day, as I implement a lot of it in PowerShell.

I DO know that CSharp is a *very* strongly typed language.
CSharp #2 is technically invalid, and won't work as is.
The parameters within the parenthesis do not have the types before them.

However, it CAN work, as long as it is part of a code block where those variables are already declared with types.
In other words, by itself it would fail.
But, if those variables just so happened to be declared already in a larger block, it would work.

```
________________________________________________________________________________
| (3) PowerShell #1 → [PSNoteProperty]::New( [String] $Name, [Object] $Value ) |
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

PowerShell Example #1 shows you that if you use the .Net Class/Type [SquareBrackets]::New() instantiation approach, you'll need to wrap them in square brackets and call a static method with the double colon.

Translating static methods from CSharp to PowerShell requires a little finagling, because in C#, the methods are called with "::". Another difference between CSharp and PS, are [SquareBrackets] around [Types] or [Classes].

```
___________
| Example |____________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

 System.Security.Principal.WindowsPrincipal  ← This is a class/type. That fails in the console.
[System.Security.Principal.WindowsPrincipal] ← That is ALSO a class/type. That succeeds.
______________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Most of the time, using the double colon after the square brackets will cause the engine to query the object with Intellisense or AutoComplete... which brings up its method suggestions.

```
____________________________________________________________________________
| Object Instantiation: Consider replacing <classname> with an actual class |_________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

PS Prompt:\> New-Object <classname>  ← (CmdLet/Function) invocation approach
PS Prompt:\> [<classname>]::New()    ← (Type/Class) instantiation approach
______________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Now, these altering variations are nearly identical to double clicking on a shortcut to launch an app.
You're typically not starting a program with these functions/classes, but you really could.

New-Object -ComObject Shell.Application ← Literally opens a COM object, which is what Windows itself is.

This parallel is suggesting that object instantiation is like launching an executable.

```
 ____________________________________________________________________________
| (4) PowerShell #2 → New-Object PSNoteProperty -ArgumentList $Name, $Value |
 ----------------------------------------------------------------------------
 ____________________________________________________________________________________
| Combine the previous block info with the notion that PowerShell example #2 uses:  |__________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
----------------------------------------------------------------------------------------------------------------

New-Object <classname> -ArgumentList $Param1, $Param2
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
| ... you can actually EXCLUDE the -ArgumentList and the CmdLet will |
| assume that the following entries are ArgumentList parameters ...  |
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

New-Object <classname> $Param1, $Param2
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 --------------------------------------------------------------------------------------------------------------

 ... although, not all cmdlets allow you to omit the -Parameter, all cmdlets are different, too.
In THIS case above, -ArgumentList is a <Parameter> that has an apparent position of 0.
That's how it can make assumptions.
At some point I will describe how to write MULTIPLE ParameterSets to a function/cmdlet.
 ________________________________________________________________________________
| (5) PowerShell #3 → New-Object PSNoteProperty [String] $Name, [Object] $Value |
 --------------------------------------------------------------------------------

PowerShell Example #3 will not work without wrapping [String]$Name in ([String]$Name)

That's because assumed parameters work a little differently.
The Class/Type engine is awesome, but it's not Superman, bro ...  it can't do everything.

As such, wrapping parenthesis around ([String]$Name) causes the engine to resolve the entry in the
parenthesis as it relates to the Order of Operations in mathematics.

If using [Type]$Variable as parameter input, they're absolutely necessary.
 _______________________________________________
| Using non-quoted strings as parameter input |
 -----------------------------------------------

A worthy thing to note here, is that using non-quoted strings as command parameters may not need quotes, at least
if there are no spaces in the string. For example ...
 ______________________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
--------------------------------------------------------------------------------------------------------------

PS Prompt:\> New-Object PSNoteProperty Michael, Awesome

Value           : Awesome
MemberType      : NoteProperty
IsSettable      : True
IsGettable      : True
TypeNameOfValue : System.String
Name            : Michael
IsInstance      : True
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 --------------------------------------------------------------------------------------------------------------

 ... but that only works for strings that have NO spaces.
Still, any tip or trick that can whittle away at character counts actually aids in making scripts easier to read,
clearer, and more concise. Plus, knowing these tricks can help you during the conceptualization/design process.

Like for instance, if you wanted to name something "Cool Name", you could cast a variable, and that may keep the
space, and you use that as a parameter. But, lets say you run into a different scope, as that is often the case
with class structures. Man. The name "Cool Name" may cause you to have to write a fair amount MORE code, than not
using the space at all.

Also, on another note, this trick also works as pipeline property input, like so:
 ______________________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
--------------------------------------------------------------------------------------------------------------

PS Prompt:\> Get-ChildItem "$Env:SystemDrive\" | ? Name -match Windows      ← See, no quotes
PS Prompt:\> Get-ChildItem "$Env:SystemDrive\" | ? Name -match ^\w+         ← Works w/ Regex
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 --------------------------------------------------------------------------------------------------------------

The reason why these work, is because the pipeline is accessing properties in a similar manner to how
(Get-Item $Path | Select-Object Name, Fullname, <etc>) works, so long as the pipeline is not within a literal
scriptblock or curly braces where the ($_ / Null) variable works, then this naked string input will work just
fine ...  as long as you're not using exotic characterss, or spaces.
                                                                               ________________________________________/
\_____________________________________________________________________________/ PowerShell Class/Type Engine Assumptions
  Comparing CSharp & PowerShell (1) /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/-----------------------------------

    The take away from all of this explanation, is that PowerShell is LITERALLY the same exact thing as CSharp under
    the skin, though I'm sure someone from Microsoft will pop out, and say:
 _________________
| MS Guy : Nope. |
```

Fine. There are obviously many caveats with them being basically identical.

| MS Guy : Yup. |

So, in order to adhere to the holy sacred art, of knowing what the hell I'm talking about, then calling it IDENTICAL to CSharp is incorrect. That said, the fact of the matter is that PowerShell is basically written with CSharp constructs, and has the same Common Language Runtime/CLR Framework.

So, it stands to reason they use many (if not all) of the same components.

| MS Guy: Maybe ... |

So, I have to expect that this Microsoft Guy is gonna watch my words like a HAWK ...
Because, after so many years of snappin' necks and cashin' checks ... ? He's used to all of that strongly written type ... and he's not the type of dude who plays games.

| MS Guy: Nope. I'm not. |

So, specificity matters to him.

| MS Guy: Yup. It does. |

Well, alright MS Guy ...
Maybe PowerShell and CSharp aren't exactly the same.

| MS Guy: They're not. |

But, if they use the same components, then they have similarities that aren't readily apparent.

| MS Guy: I can agree with that ... |

So, if that's the case, some EXPERIMENTATION and EXPANDING upon those OBSERVATIONS/SIMILARITIES, will inevitably evoke (strengths/weaknesses) of one over the other, as they all find their way into processes that people hadn't used before ...

| MS Guy: Interesting. |

At least, not until some dude with his obnoxiously written section headers had his sights on a checkered flag somewhere in the future. Somewhere. Cause, so is the guy from Microsoft. The both of these two dudes, day in, day out ... constantly looking for an edge in performance. Waiting for the day that they'd be first person to blast past that checkered flag.

| MS Guy: That's right, dude. |

Anyway, now the new guy came along, looking for ways to break Microsoft's most sophisticated language ...

| MS Guy: I too ... do this. |

But, maybe break is the wrong term here ...

Comparing CSharp & PowerShell (1)

Embrace, Extend, and Enhance

When is the last time you've ever heard anybody say:

| Someone: Hey, buddy. You better not go make something even cooler than it already was ... ~! |

Probably never ... right?
But, people do it all the time.
I could provide many, many examples, but I'm going to whip out one pretty specific example ...
When Microsoft combined Edge and Chrome.

Google originally made Google Chrome.
Microsoft originally made Internet Explorer.
Commodore originally made the Commodore64.
Paul Allen and Bill Gates made BASIC for the Altair 8800.

The point, a lot of people have made stuff ... and when everyone kept saying "This is the best",
Well, somebody said "I can do better than that. Check this out ... " Boom. New version of Basic.

Paul Allen and Bill Gates spent a fortune renting access to the supercomputers back in the 70's.
Look what they started ... ?

Microsoft has a very detailed history of "showing people how it's done ... "
Sometimes they'll roll up their sleeves when they say this.
Other times, they're done before they even have the chance to roll up their sleeves.

So, they don't even say anything ... Job's done.

Anyway, this is what they did to Google Chrome ...

They went ahead and improved Google Chrome.

Just like they do to everything they decide to do.
Not really sure why people are shocked by it, but they have a really long history of just going right ahead,
and making something even better.

Some people might say "How DO they manage to do that?"
Well, the answer is because they have the best software engineers in the world, and ...
... the world's best software is engineered at One Microsoft Way, Redmond WA 98052.
Always has been.

That's cause they invented the idea of software. So, when they decide to do something ... ?
They don't play games. One day they got together and said to Google ...

Embrace, Extend, and Enhance

Internet Explorer vs. Chrome vs. Edge

Microsoft : Hey Google.
Let us show you how to build a better web browser.
Google : Chrome is the best, pal.
Microsoft : No, we know that it WAS at one point.
But, we made Internet Explorer.
That was the best at one point in time too.
You built Google Chrome off of Internet Explorer.
Google : *scoffs* That's absurd.
We didn't build it off of Internet Explorer ...
ActiveX?
chuckles Lol.
Microsoft : Nah.
Not you, right?
You couldn't have done that ... could you have ... ?
Google : I mean ... we might've taken a few notes from Internet Explorer ... ActiveX was cool.
Microsoft : Prolly more than a few notes ...
Netscape Navigator and Mozilla Firefox too.
Google : Maybe we did, maybe we didn't.
You'll never know ...
Microsoft : Well, we made an ACTUAL operating system called Windows.
So ...
Google : *scoffs* Yeah, well, we made Android and smartphones.
Microsoft : Android is Chrome ...
And, Apple made the smartphone first. (← Ballmer also made smartphones, Windows Phone was well built)
Google : Yeah, well most people in the world use Android.
Microsoft : Yeah, well, most businesses that (generate PROFIT/spend MONEY) ... use Windows.
So ...
Google : Whatever bro.
We made Google Chrome.
And, Chrome OS.
Microsoft : Chrome OS isn't an OS, it's an offline web browser.
Besides, we know ...
We had to wait for you to show us a thing or two.
Google : And, that's what we did.
Microsoft : Yeah, but we already DID those things.
Showing somebody a 'thing or two', who already made those things a long time ago ... ?
It's rather anticlimactic, to say the least.
Google : Yeah, well ... everybody uses Google to search the web.
Microsoft : Cool story bro.
Look, we have so many complicated things we've already built ... ?
We had to give Google Chrome a touch that only the experts could give.
Google : Bro, we ARE experts.
Microsoft : I mean ... are you though?
Not from our angle ...
Google : Gonna pretend I didn't hear that.
Besides, nobody who's ANYBODY, can just go ahead,and build a WAY better version of Chrome than us.
Not without our say.
Microsoft : We actually went ahead and did that though ...
Google : *gulp* Yeh ... ?
Well ... it's still based on Chrome ...
Microsoft : It is.
But- we did a lot more than just give it a facelift ...

Internet Explorer vs. Chrome vs. Edge

Dreaming Big, Building Bigger

The truth is, when Microsoft really wants to do something ... ?
They will actually go right ahead, assign a whole football team worth of people that just so happen to be very
bright individuals, and they'll spend all day and night conceptualizing about the thing they set out to do.

Burning the midnight oil, making pot after pot of coffee, day in, day out ...
Some of them might even run alongside a black guy on a bicycle, like in the game Mike Tyson's Punch Out on the NES.

Just like Rocky Balboa, eating raw eggs ... ? Push ups ... ? Minimal sleep ... ?

All the while, having dreams about the code.
When they do this, they dream big.

Until the day their big break finally comes, like a Rubiks cube that's just about to be solved...

The new kid comes in after this dream where he finally finds the answer to the task.

/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯\

New Kid : Hey boss man... had a dream last night.
           Think I might've figured it out.
Boss Man : *long hard intense stare* Oh yeah?
New Kid : Yeh.
Boss Man : *Looks at the new kid suspensefully* Tell me about this dream you had, kid.
           Sounds important.
New Kid : It was.
           It changed everything.

__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/

So then the kid whips out the napkin that he drew the idea onto.
The napkin's all crumpled up, but that's ok...
New kid was able to open the napkin, remember the idea, and then he races over to the whiteboard.
All these other ideas are on the whiteboard, but he knows he's got the answer they've been looking for...
So he erases everything and starts from scratch.

Some of his teammates start to get WICKED PISSED, but the boss man interjects... tells em all to shush.
New kid starts drawing it all up.
The boss man intently watches, so does everybody else.

Everybody on this team of people is on the edge of their seats.
After a moment, the boss man starts to see this mathematical probability matrix taking shape...
...and immediately, he knows the kids idea is gonna work, at least theoretically.

Boss man's been around for a while.
He knows when the math works out...
So, boss man tells everybody....

/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯\

Boss Man : This is it, guys.
           snaps his fingers, and points at the new kid Kid's got the killer plan...
           Person1, I want you to do <this stuff>
Person 1 : Got it.
Boss Man : Person2, I want you do <that stuff>
Person 2 : That's...
           A lotta stuff, Boss man...
Boss Man : *gives Person 2 a hard stare*
Person 2 : ...alright, I got ya.
Boss Man : Person 3, I want you to keep that damn coffee pot stocked and topped for the next couple days.
           None of us are leavin' till this is done.
Person 3 : You got it boss man.

__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/

The boss man continues down the list of assistants, and shakes his head and stares at the whiteboard.
The new kid comes back up to the boss man.

/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯\

New Kid : You really think this is gonna work, huh?
Boss Man : Kid...
           I can feel it in my bones.
           I think you just about licked the problem.
New Kid : Can't believe it all just came to me, in a dream like that...
Boss Man : I can, kid.
           That's how it works.

__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/

So, now it's time to write the code, and test it. For the next 2 days, the entire team pounds away at their keyboards... all because of an idea that the new kid had.

They knew that the math worked out before even writing a single line of code.
They knew it could achieve some mysterious objective where it outperformed all of the ideas on their idea board.

After 2 days go by, they finally have it ready to compile, so that their oceans of virtual machine hive clusters could all concurrently test this piece of code. The results came pouring in, and all of the tests and checks were positive, and nominal.

| Headline: Microsoft does the impossible- AGAIN. Edge beats out Google Chrome... Typical Microsoft behavior. |

You wouldn't think that this story was really all that realistic, but... it probably is.

The end result, is that Microsoft was able to build a version of Chrome that uses less resources, is snappier,
uses all of the same extensions Chrome does whereby making Edge have backward compatibility with other extensions,
and now they've incorporated what made Edge v1 actually pretty cool, useful, and performant.

Google made the mistake they always make, they thought that Microsoft was just old news ...

But, they thought wrong.
Now, Google Chrome has all the cool additions of Edge from Windows 10.

To be perfectly fair, it was a battle that both companies fought for many years until the new kid with the idea
came along, and changed the entire way that games were played between these two titans of technology.

A masterpiece, unfolded.
Written on a napkin.
Decades in the making ...

Some may say that PowerShell is a functional language, but it really is a lot more than that.
I suppose for the people that only know how to use commands that come with the operating system, then it's pretty
easy to think PowerShell is just a functional language.

However, PowerShell is an Object-Oriented language, just like CSharp is ...

At least, whenever you do more than just use the default commands and functions.
Bash or (tsch/TCShell), are more FUNCTIONAL based, or even heavily STRING based.
---- ---- -------
Mainly because the output is a "stream" of individual bytes masked with characters ...
... characters marked with encoding ...
... encoding indexed with integers ...
... collections of integers being thrown into floating point calculations ...
... rounding the floating point calculations into percentages ...
... then doubles getting calculated by multi-tiered hex values ...

Now, all of these things happen in CSharp and PowerShell too ... but in PS?
EVERYTHING is either a single [OBJECT], or multiple [OBJECTS[]].

If it's a single [OBJECT]?
Well, it could definitely be VERY detailed.
But, it could also be a list or a collection of objects.

In that collection, each of those objects has an entire collection of properties.
Properties may contain single values, or multiple values.
Values can actually be subproperties, keys, or additional values.
Sometimes there may even be multiple objects that just so happen to be marked as values ...
... within a single property.

Then, you'll have to face yourself in the mirror. "They could go on infinitely inward, huh?"

Yeah.
They could.
Because, if there ARE values that represent objects, then ... there may be some recursive action going on,
almost like Horton Hears a Who.

Then what ... ?
You might have an entire arsenal of nested objects each with THEIR own properties.
Each of those objects might have properties which hold a value of an additional object ...

Probably sounds confusing ...
Objects, properties, values ...

But, values can be nested objects.
Those nested objects might even have many properties with single values, or multiple values.

Now you probably have no idea if I'm stating things metaphorically, or specifically ... do you?

The truth is, I could be stating one of those things.
Or, the other.

However, what could ALSO be the truth, is that it really could be BOTH things simultaneously.
Or, neither of em.

That's how quantum physics, entanglement, and superposition explain reality.
If that idea stretches your imagination a little too far ... ? Heh.

PowerShell can keep on going buddy.
Now, wrap your head around THAT.

Whether NESTED OBJECTS contain VALUES that just so happen to be MORE NESTED OBJECTS ... ?
... each with their own PROPERTIES and VALUES ... the truth is, you really COULD go on for seemingly eternity,

Nesting objects within values ...
... values within properties ...
... properties within an object ...

Before you know it ... you'll realize that basically PowerShell has a lot more control than string based languages.

That's why recursion is ... mind boggling.
CSharp has this control too, but ... kinda needs to be planned out, to control it.

PowerShell is built around the concept of DYNAMIC, and thus, has a LOT more FLEXIBILITY.
It does have it's own limitations as well, but there's a happy medium ...
Combining the strengths of CSharp and PowerShell is a process I continually conceptualize and expand upon.

Also, string based languages aren't capturing objects with their properties and values recursively inward, not unless a program is running and happens to be doing that, but chances are, that the program is outputting some text based derivative, like (stdout/Standard Out).

Whether that is/isn't the case, a master programmer has to build a construct that manages to do all of that, without breaking, or throwing an error. So, a master programmer has to be extremely considerate, and build the proper classes, set the proper constants, intialize the correct keys, and use mathematical models to build all of it ... Or, they could just hope they get lucky.

They have to keep in mind how many nested objects, properties, and values might be in there, lurking beneath the surface of it all. Because if they don't ... ? Well, the whole program will inevitably break apart, crash, or shut down. Then it won't be a program anymore ... It'll be an error. Or, a long list of errors.

Can't exactly call it a program if it doesn't work ...

Suffice to say, this programmer, this keyboard warrior, whatever you want to call them ...
... they have to find a way to perform such an impossible task, one that can contain every class, object, property, and value ... and if it goes in additional levels ... well, what then, buddy ... ?

These are all of the things a developer has to keep in mind ... all so that the the normal every day person can keep their sanity and wits. Probably doesn't sound fair, does it?

Them ... ? Thinking about how to reliably count the number of stars in the universe, and not break.
You ... ? Probably just getting a coffee at Starbucks to start your day.

_____/
_____/ Quantum Physics, Entanglement, Superposition
Doing the Impossible /-----\
/-----/

If that just blew your mind ... ? Sorry.

It's easy to get carried away with how much control there is, especially when just casually stating these words in a single sentence: "objects, properties, values, recursively, eternity" causes the speaker to start thinking about fractals, Rosen-Einstein bridges/condensates, Mandlebrot sets, and this guy:

| 01/27-31/20 | Dyan Beattie - The Art of Code | https://www.youtube.com/watch?v=6avJHaC3C2U |

None of these things are really REQUIRED to build a program, or understand (PowerShell/any other language) but, they definitely help. Before I started getting carried away in explaining how PowerShell is FAR MORE than just a functional language ... ?

Well, what I was alluding to is that a FUNCTIONAL language isn't exactly DESCRIPT about CONTROL over OBJECTS, PROPERTIES, and VALUES ... An OBJECT-ORIENTED language IS, as every component may need to be able to be calculated, translated, and duplicated with precision.

The possibilities don't stop there, either. But, Allow me to return to the subject of:

_____/
_____/ Doing the Impossible
Language Construct <Object-Oriented, Functional, String-based> (2) /-----\
/-----/

String based languages like tsch ARE reliable, however ...
It is very old, as it originates from (UNICS/Uniplexed Information and Computing System)
Which, eventually dropped the CS and traded it with an X, whereby changing UNICS to UNIX.
Only reason I know that it was ORIGINALLY named UNICS is because I read a portion of this guy's book ...

| 11/24/95 | The Road Ahead by Bill Gates | https://en.wikipedia.org/wiki/The_Road_Ahead_%28Gates_book%29 |

You would not believe how many things this smart bastard, BILL GATES predicted WAAAAY back in 1995 ...
Not only is he a smart bastard, but he's also a BILLionaire. One of the coolest ones out there, too. Side point.

Cisco routers & switches (or really, ANY good brand), use tsch.
Shocker, right ... ? Cisco networking devices using an operating system like tsch ... ?
No way ...
That's impossible ...

I'm just kidding. It is possible. It's what DELL, HP, even UBIQUITY, maybe RUCKUS, or thousands of other
brands actually use to FLASH the FIRMWARE on those NETWORK DEVICES. Tsch.
USER CONFIGURATION MODE and then GLOBAL/ADMINISTRATION MODE ...
... is like comparing NON-WHEEL to WHEEL in Bash/Linux.

While BASH isn't nearly as old as tsch, it takes a lot of inspiration from it.
Bash comes from the very widely authored realm of Unix/Linux/BSD.

And, not all Bash shells are created equal ...
Many old (DOS/CMD/COM) based components are ALSO string based.

COM objects ARE CMD components, but- with the added kick of having some new functionality, since some wizards had
the wireframe schematics, they chiseled in new accessiblility for PowerShell.

Java and Python are BOTH string based, AND class/object based. Java/CSharp are VIRTUALLY IDENTICAL ...

... though, I mean that in terms of how the SYNTAX, as well as how code is constructed.
Most programming languages have SIMILAR SYNTAX, but making comparisons between them CAN be TIME CONSUMING.

Saying programming languages are IDENTICAL or VIRTUALLY IDENTICAL comes with some CAVEATS.
Typically, they're definitely NOT EXACTLY the same.
Because, saying that would be ridiculous.

But, in appearance, logical construction, and format ... it's pretty easy to see how SIMILAR they are.
Python, is quite a different beast than anything else, but- it is very similar to standard C.

Standard C has not a single (+/plus), nor two of them (++), nor does it have a (#/sharp).
That's what Python is, except it can do a lot more cool stuff than Standard C can.

Python is ALSO a very good language, very extendable, flexible, powerful and capable, but it is
not even remotely as flexible, powerful or capable as PowerShell ... because it requires setup,
multiple components to install, and its console experience is rather detached from the script
editing. Unless you have a compiled executable. But, if it's an executable, then it doesn't
matter if you have the tools or devkits installed to build with it ... it's ready to run.

If we're gonna grasp straws, PowerShell Core requires all of that on Linux/MacOSX/BSD too.
So, if you're using Linux or Mac OSX, THEN ... Python is a GREAT choice. But, so is PowerShell.
On Windows, you have access to PowerShell Desktop AND Core. Then, it isn't much of a question.

```
                                                                                                    /
\_____________________________________________/ Language Construct <Object-Oriented, Functional, String-based> (2)
  Should I install Python ... ? PowerShell is just WAY too convenient ... /-------------------------------------------------\
/-------------------------------------------------------------------------
```

Guy : Should I use PowerShell (since it's already included on Windows) or, install Python ... ?
Decisions, decisions ...
Do I use Python version 3.0.0, 3.1.0, or heck, maybe 3.9.4 is the way to go ... ?
Maybe they're up to v4.0.0 now ...
Jeez.
So many versions.
What ever will I do ... ?
I want my CODE to work across MULTIPLE VERSIONS ...
I mean, I could totally goof up and install a version that has a VULNERABILITY ...
... included in the setup package ...
With PowerShell ... ?
Well, Microsoft updates that constantly, so no Log4J type action.
Man.
What should I do ... ?
You know, if I *really* wanted to, I'll bet ya that I could *totally* start working *right now*,
and not have to install a SINGLE COMPONENT at all ... ?
But, that would be just WAY too easy, AND convenient.
Not quite my style.
I feel like I NEED to install at least a DOZEN components FIRST, before I even begin ...
Sorry PowerShell, you're WAY too convenient.
Dudes at Microsoft being SUPER considerate all the time.
What the heck.
It's REALLY annoying how CONSIDERATE they are ...
AND consistent ...
And, just, very skilled at making software ...
What a bunch of jerks~!

```
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
```

Yeah, nobody has that conversation with themselves, do they ... ?

It's almost like the world's best software engineers knew what the hell they were doing when they built it ...
That's what *I* think when I hear PowerShell.

All jokes aside, Python IS extremely powerful.
It's nothing to sneeze at.
It can outperform PowerShell AND CSharp in many instances ...

But- does Microsoft use it more than CSharp or PowerShell?
Pretty sure that's a NEGATIVE.

If they're not using CSharp, they may use C++ if anything.
That doesn't mean they WON'T use Python ... ?
Cause they will.

| 07/24/17 | James Powell: Wanna be a Python Expert ... ? Hm ... ? | https://www.youtube.com/watch?v=cKPlPJyQrt4 |

I've gotten my hands dirty with Python, IF THAT ISN'T RATHER OBVIOUS ...
But- translating how I program in PowerShell to Python requires me to have to SPEND MORE TIME FIGURING THINGS OUT ... And that's the CLINCHER HERE. Time. Investment in TIME, matters in EVERY REAL WORLD APPLICATION.

Someday I will use Python interchangably with PowerShell AND CSharp ... ?
But- they're not gonna be as flexible as PowerShell (in my opinion).

As for the folks at Microsoft ... ?
Perhaps some of them occasionally dabble and moonlight with some Python script kiddies, as it has seen a lot of increased popularity recently, but- so has PowerShell.

All things considered, Python has some strengths that aren't anything to joke about at all.

But- it is:
- very taxing to keep up to date unless you're very involved in maintenance
- has security issues quite often as it is open source
- very susceptible to CVE's after each version has been distributed

I know this because I use a derivative of FreeBSD for Security Gateway appliances, like pfSense/OPNsense.
I thoroughly enjoy the CHALLENGE of using these platforms to provide ENTERPRISE GRADE NETWORK SECURITY.
It's much more DIFFICULT than using something that COSTS A LOT OF MONEY and is DEVELOPED BY CISCO.

If I had mountains of cash to burn ... ?
That is the equipment I would purchase and use ... because CISCO IS THE BEST THERE IS.
SIDE POINT.

Python is the most targeted codebase by far. Maybe I'm wrong, ECMA/JavaScript, PHP and OpenSSL get hit hard too.

But, I see basically every version of Python getting hit with CVE's, more than anything else.
It's harder to maintain cross-compatibility AND security, if you release an update for Python ...
... and within 1 hour of that update being distributed, there's a brand new CVE.

Then the next version will need a few days at a bare minimum ... to bug test and write.
While Python is GOOD in many cases, I think it is basically DOS on steroids.
I'm sure that's an OVERSIMPLIFICATION, but Python causes me to FEEL as if, it is the same thing that COM/CMD is.

Now, bringing it all back to comparing and contrasting with CSharp, since that's where this started.

Should I install Python ... ? PowerShell is just WAY too convenient ...

Comparing CSharp & PowerShell (2)

CSharp isn't your standard issue, do-it-in-a-jiffy type of programming language ... Nah.
It requires a LOT more meticulousness than PowerShell, as it is specifically, and strongly typed.

Whereas PowerShell allows the common lad to relax a bit. Cause ... PowerShell knows just how high CSharp sets its requirements. Consistently high too.
That's how I know PowerShell was designed from the ground up by some guys at Microsoft who looked at each other one day, and said:

/--__/--\

MS Guy1 : Hey, you know what bro ... ?
MS Guy2 : Sup bro?
MS Guy1 : I feel like CSharp needs to relax a little bit ...
MS Guy2 : Bro.
I've been thinking the same thing for a while now.
MS Guy1 : It's like ... don't get me wrong.
I love writing in C#.
But-
MS Guy2 : ... feels like somebody's breathing down your neck all the time, doesn't it?
MS Guy1 : Yep.
Besides ... we could totally make things easier on ourselves.
MS Guy2 : Well ... Some guy named Jeffrey Snover told Red Rover to move over ...
MS Guy1 : Oh yeh ... ?
MS Guy2 : Yeh.
Now we have PowerShell.
MS Guy1 : Is that the super-command prompt thing called Monad ... ?
MS Guy2 : That's ...
... the code name, yep.

MS Guy1 : I'm already impressed by the way its name SOUNDS ...
          Sounds powerful.
MS Guy2 : I heard that it is.
MS Guy1 : Very impressive.
MS Guy2 : Dude.
          We could TOTALLY help make this thing do some ninja-grade CSharp type stuff.
MS Guy1 : ... really?
          Because, that's ...
           ... something I could REALLY get behind.
MS Guy2 : Yeh, man.
          This thing's got potential.
          CmdLets, functions, variables ...
MS Guy1 : It's almost like somebody COOL has been up to something that CAN'T BE STOPPED ...
MS Guy2 : Not even almost.
          That's definitely what's happenin', I can feel it.
MS Guy1 : I'd be willing to bet that Google and Oracle will use it, too.
MS Guy2 : Don't say that bro ...
           ... you'll jinks it.
MS Guy1 : *gulp* Dude.
          I ... didn't mean to say it like that ...
MS Guy2 : *shakes head* Great, dude ...
          What if this thing becomes way TOO POWERFUL and USEFUL now, huh ... ?
          And then, Google and Oracle just, non-stop ...
           ... wind up using it more than WE will.
          Then what ... ?
MS Guy1 : I don't know what I was thinking, to be honest ...
MS Guy2 : Yeah, well ...
          If it DOES happen to become *shakes head* TOO powerful ... ?
          Well, I'm blamin' *points* you.
MS Guy1 : Ah man, don't to that ...
MS Guy2 : Oh, I'll tell everybody ... OooOOOoHHhhH MS Guy#1 just had to go jinxing PowerShell YEARS ago ...
MS Guy1 : Yeah, but this is where the worlds best software engineering takes place ...
MS Guy2 : Obviously.
          Let's get it done.
          Fist bump on it?
MS Guy1 : Oh, you know it bro ... *fist bump*

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Then, they got to work.
They (compared/contrasted) (C#/PowerShell), to build a new language in a way where dreams could become reality,
visions could become artwork, and the sculptors of tomorrow ... ?

Even Michael T. Niehaus was involved in some of it being APPLIED to SERIOUS BUSINESS CENTRIC APPLICATIONS ...
Hence, SYSTEM CENTER CONFIGURATION MANAGER, having a really "totally able to be controlled" aspect ...
 ... from PowerShell. Just like Hyper-V.

Which, to be FAIR ... ?

| 12/04/21 | [FightingEntropy(π)] FEInfrastructure Preview/Demo | https://youtu.be/6yQr06_rA4I |

 ... Michael T. Niehaus' work with the MICROSOFT DEPLOYMENT TOOLKIT is what got me to START PROGRAMMING.
Side point ...

As for how these engineers who had this conversation up above ... ?
They spent every day at work, building and pounding away at this idea in the future.
They were considerate enough to know that EVENTUALLY, everybody would want IN on this COOL thing, PowerShell.

So naturally ... they'd be doing their work within a Visual Studio like no other.
When other companies asked:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Companies : Hey Microsoft, could we like, BUY PowerShell from you ... ?
            It's really cool.
Microsoft : *chuckles* Yeah, I'm sure you and everyone else would ...
            But- too bad, it's not for SALE ...
             ... it's ours.
Companies : C'mon dude.
            Don't be so sheisty ...
Microsoft : *sigh* We knew people would say that, but nah.
            It ain't happenin', cap'n.
Companies : Well, Microsoft ... ?
            pouts Guess we'll just have to use Bash or Python ...
Microsoft : *scoffs* Fine.
            Have at it.
            We use those too.
Companies : Yeah, but you have PowerShell ...
Microsoft : Obviously.
            Python and Bash are fine.
Companies : But, we REALLY wanna use PowerShell, cause we like it a lot ...
Microsoft : Cool.
            You can USE it, but it's not for sale.

```
            It's our pride and joy, 20 years of work.
Companies : It didn't take you guys 20 years ...
Microsoft : Look, all you other companies couldn't comprehend how much thought went into it.
Companies : Well ...
             ... if we can USE it, then ...
             ... I GUESS that's good enough ...
Microsoft : Fine.
            Go ahead.
            Thought you wanted us to just flat out sell the platform ...
Companies : I mean ...
             ... is it for sale ... ?
Microsoft : *sigh* No.
Companies : Will you ever let us know if it ever DOES become something you'd sell ... ?
Microsoft : Surely, you can't be serious ...
Companies : Oh, I AM serious ...
            And, don't call me Shirly. That's rude.
Microsoft : OoOOooOOhHhhhHh, alright there Mr. Naked Gun 33 1/3.
Companies : Airplane.
Microsoft : What ... ?
Companies : That quote is from Airplane, not Naked Gun 33 1/3 ...
Microsoft : Dude.
            We're Microsoft.
            We obviously know that quote is from Airplane ...
            Didn't even need to "Google search" it, either.
Companies : Nah ... ?
Microsoft : Nah.
Companies : *hangs head in shame* Alright, Microsoft ...
            You win.

\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/

Now, I am pretty sure the above conversations have never actually happened, but I do know that the (men/women) at:

 ________________________________________
| One Microsoft Way, Redmond WA 98052 | ... ? They were under a LOT of pressure, to make this thing perfect.
 ----------------------------------------

Basically, they saw how Americas founding fathers tried to make the Constitition perfect ... ?
And then they worked AS HARD AS THEY DID ...
 ... to make PowerShell perfect, too. Or, at least to TRY to come somewhat close.
That's something that WILLIAM GATES and STEVEN BALLMER may even nod their heads in agreement ...

 ______________________________
| Yep. That's ...  what we did. |
 ------------------------------

Now, what'd they already have on hand, that was FIERCE, VERSATILE, POWERFUL, and sorta FLEXIBLE ... ?
Standard-issue C-Sharp.

The thing that C++ has so many things in common with.

Anyway, (comparing/contrasting) (C#/PowerShell) actually put Microsoft in a position, where they really can't
even sell it. Why ... ? Cause it'd be like asking someone to sell their kids.

/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\

Guy    : Hey.
Parent : Sup ... ?
Guy    : I see you've got MULTIPLE kids ...
         You should SELL me one of em.
Parent :  ... what ... ?
Guy    : SELL me one of your kids, I'll pay top dollar.
Parent : Uh, dude, you better get on outta here, I'd NEVER sell MY kids ...
Guy    : I only want (1) of em, not MULTPLE kids, dude ...
         Whaddya think I am, some kind of monster ... ?
Parent : You just asked me to SELL one of my KIDS ...
         So, yeah, I think you ARE a monster, to ask somethin' like that ...
Guy    : C'mon buddy ...
         Don't be so DRAMATIC ...
         It's just (1) kid ...
Parent : *pulls out his service pistol* You know, I was in the United States Marine Corps, right ... ?
         *cocks and points it at the guy*
         Now I'm gonna tell ya again ...
         You get on outta here and NEVER come back.
Guy    : WHOA, WHOA, BUDDY ...  ALRIGHT ...  JEEZ.

\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/

I would say it is sort of akin to asking General Motors if they know how to make a GOOD vehicle that doesn't
actually use any gasoline ...  The answer is that yes, General Motors DOES know how to make a good electric vehicle.

The problem is that they will never actually do that.

So, Microsoft selling PowerShell is a lot like General Motors making a good electric vehicle.
It's unfathomable.

The truth is, General Motors has made FAR MORE electric vehicles ...  than any other car company on the planet.
```

They just don't have any incentive to make them all that *well*.

I'm sure if they tried, they could definitely make these things as well as Tesla does ... ?
But, I might be fooling myself OR OTHERS, by making a statement like that ...

The problem is, finding any reason at all, to convince them to do such a thing.

I'm sure if you twisted Microsoft's arm, they could sell PowerShell and then build something to replace it.
But, they're not gonna do that, just like General Motors will never make a good electric vehicle that uses
NO GASOLINE, whatsoever. It actually cannot be expected, at any time.

GM has ALMOST made a good electric vehicle many, many times ... but each time those "vehicles" ALMOST made it
to market ... they somehow *vanished*.

The Terrible Magician / Comparing CSharp & PowerShell (2)

This is a bit of a tangent and this is gonna be a bit DETAILED and LONG WINDED, but I promise you ... ?
The tangent will coalesce back into the narrative of PROGRAMMING and the MORAL of the story will make itself
PERFECTLY CLEAR and RELEVANT toward the end.

In reference to electric vehicles, GM makes a better magician, or story teller, than a vehicle manufacturer.

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

GM   : Here kids, watch this highly rated electric vehicle get 400 miles per charge ...
       Kids wait around for like a year
Kids : Hey, GM ... where's that highly rated electric vehicle that gets 400 miles per charge ... ?
GM   : It'll be out next year.
       Had a few issues.
Kids : What kind of issues ... ?
GM   : Burning through R&D cash reserves, researching how to make good batteries n stuff.
Kids : But, Tesla has been doing it for a while now ...
GM   : That's Tesla though.
Kids : But, you guys get so many awards for the best vehicles in the industry ...
GM   : Gasoline powered.
       That's why.
Kids : Thought you guys could make electric vehicles that were competetive ...
GM   : Hell no, kids.
       We just SAY stuff like that to SOUND like we're experts.
Kids : That's ... messed up.
       So, does the vehicle which gets 400 miles per charge exist ... ?
GM   : On paper it most certainly does.
Kids : On paper ... ?
       What does that even mean?
GM   : It means that we printed a piece of paper with the facts and figures of this thing.
       It definitely exists alright.
       We just ran out of EV R&D money and built more gas cars.
Kids : So, why even tell people you're building an electric vehicle at all ... ?
GM   : Look kids.
       We would really prefer to build gasoline cars.
       That's what we do best.
Kids : But, what will you do when all of the oil on the planet is gone ... ?
GM   : We'll worry about it then, alright ... ?
       Now, scram.
       I need to go take a nap.
Kids : Wow.
       I've never met such an obnoxious magician in my entire life.
GM   : Look, I feel bad.
       Here's some pictures of the 2022 Chevy Corvette.
Kids : It does LOOK cool, but-
GM   : No, it's not electric powered.
Kids : Then, why would anyone want this thing ... ?
GM   : WHAT DO YOU MEAN, why would anyone want this thing ... ?
       It's got a lot of horsepower ...
Kids : Does it have as much horsepower as the Tesla Model S Plaid ... ?
GM   : ... no.
Kids : How much does it cost ... ?
GM   : Like, $120K.
Kids : I could buy a Tesla Model S Plaid for that much.
GM   : LISTEN KIDS ...
       All you guys wanna do, is talk about the Model S Plaid, don't ya ... ?
Kids : Yeah.
       It's cooler looking, and WAY faster than this *hand wavin' around* 2022 Chevy Corvette ...
GM   : But, ACTUAL experts designed this thing ...
       We invested a lot of R&D money into it.
Kids : Weird.
       Actual experts made something that uses gasoline and is still slower than the Tesla Model S Plaid ... ?
       Doesn't make a lot of sense there Mr. Magician ...
       Probably could've spent that money making an electric car.
       But, a GOOD one.
GM   : Could've, should've, would've.

So what ... ?
Kids : Maybe you guys just aren't very good at making cars anymore ...
GM : *scoffs* We'll ALWAYS be good at making cars there kids.
Don't be ridiculous ...
Kids : C'mon guys, lets go.
President Obama might have to give this guy another bailout check.

__/--__/

Now, would GM ever openly state these things ... ?
No, because that'd be WAY too honest ...

Besides, they could've used that bailout check that they got, to finally do right by the American people, in the form of making GOOD vehicles, right here in America.

But, *checks watch* they, still haven't gotten around to it.
I wonder why that is ...

Much like how TUCKER CARLSON and SEAN HANNITY have NEVER been interesting to watch ... ?
GM hasn't been an interesting vehicle manufacturer, since ... like, WAAAAY back in the 80's, or the 70's.
Basically BEFORE BOB LUTZ began to run General Motors, that's when General Motors was actually respectable.

Microsoft's Golden Standards / The Terrible Magician

Microsoft doesn't have this issue where they perform magic tricks or tell stories ...
Well, let me rephrase that ... They perform magic tricks that are SKILLFULLY executed, not TERRIBLY executed.
Not only do THEY know how real climate change is, but they never sound moronic, like, ever.

Also..?
If they say they're gonna do something, or they say they know HOW to do something, then ... they *definitely* keep to their word. No "vaporware" like GM and their electric cars.

When General Motors and AIG needed a bailout check after Bear Sterns got wiped out of existence ... ?
Microsoft didn't need one at all.
Might be because Steven Ballmer never put the company in a position to ever NEED one ...

Ballmer worked his ass off to make so many good things and ideas.

Slate PC	Windows 8	Windows Phone	Zune	AI Chat bot Tae

I don't know why ... but a lot of those ideas just landed on the market, terribly.
Tae had some SERIOUS issues with using racial slurs, after it was launched.
Then it had to be taken offline ...
But the rest of these things were WELL DEVELOPED.

Ballmer had A LOT of STIFF COMPETITION as the CEO of Microsoft ...
... none of those products were even bad.

Even STILL ... Ballmer never asked Obama for a bailout check. And, that says a million words.

The truth is, in a similar way to how GM would never be caught dead making a good electric car, these experts at Microsoft would never even think about selling their 1) kids, nor 2) PowerShell.

I won't lie, I haven't ALWAYS had a consistently good opinion in reference to Mr. Ballmer's engagement policies.
However- give credit where credit is due. The dude did a pretty amazing job of running Microsoft.

However, it's in Satya Nadella's hands now, and this dude was born for it, so go ahead and:

09/26/17	Hit Refresh	https://news.microsoft.com/hitrefresh	https://en.wikipedia.org/wiki/Hit_Refresh

Now, I haven't read Mr. Nadella's book, to be clear.
I will though.
I already know it's probably very good based on it's reviews, and the number of videos I've seen referencing it.

Already read a good portion of "The Road Ahead", and that was ALSO a good book ...
So, the tale of rediscovering one's self or sense of purpose ... it's like a similar mission among experts everywhere.

Did BOB LUTZ write a book or win a Pulitzer prize for running General Motors for 50 years ... ? Nah.

So, the comparison with General Motors doesn't even really compare ...
BOB LUTZ would have to be humbled enough and finally ask Elon Musk for advice on:
"how to make a good vehicle that doesn't use any gasoline".

I know that General Motors has SOME EV's out there now that SUPPOSEDLY keep up with Teslas ... ?
But I think the FACTS and FIGURES are ARTIFICIALLY INFLATED ...
Not to mention, it took Tesla reaching MASS PRODUCTION before General Motors said to themselves ...

Alright, so we REALLY need to make some decent electric vehicles now, ffs. ← LAZINESS in American auto industry

I'd be happy with Lutz even learning how to make a GOOD gasoline powered car, where the PARTS aren't made with INTENTIONAL DESIGN FLAWS. I mean, let's face it ...

Anyone could buy a European car, and that will automatically be worth more than any American car. Right off the lot.
It doesn't LOSE as much value as an American car. Why?

Because them Germans and Italians are a LOT better at making cars, than Americans (except for Tesla). Seriously.
Even the Koreans, Japanese, and Chinese folks can make HIGHER QUALITY VEHICLES for LESS MONEY.

What that actually means, is that people in America don't really care if they do a good job. Some Americans
take offense to this ... "Yeah, America sucks at making quality vehicles.", and I don't know why - it's TRUE.
They SHOULD be getting offended by people that don't pay any attention to the mistakes they make, because
they ultimately cost everybody even more in the long run.

I really shouldn't say anything negative about Ford or even Daimler-Chrysler, cause it's not exactly their
fault that they don't make MOST American cars. The ones they DO make, are typically better than anything GM
makes, hands down.

Here's what GM does very well ... Selling things that break.
Selling things that BREAK is DEFINITELY a LUCRATIVE business- GM is the best there is, at THAT.

That really isn't something you really want people to consider your company to be the BEST at ...
... because it means YOUR company is the BEST at making something TERRIBLY DESIGNED.

That's the beginning and the end of that conversation.
But, maybe that's just my opinion.

They found success selling convoys of vehicles that get thrown together with parts that someone intentionally
made poorly. Because, if it breaks a lot? That generates more profit, and to them, if it generates more profit
because it broke more often, then that is actually a genius plan.

So, their eating-leaded-paint-chips mentality, where they made more money, caused them to believe they did an
AMAZING job. But- stupidity will do that to ya ...

Those automatic window controllers need to be replaced every other winter on one of the doors.
Doesn't matter if you had the driver rear door fixed last year, what matters is that this year, the OTHER one
will ALSO suddenly break. You might feel like the boogeyman is just swinging around from one part to the next,
randomly breaking these things.

Heh. That's what they're very good at making people think.
Now that the OTHER rear window motor is broken ... ?
Each time you drive it in the winter, you'll feel like your air conditioner is on.
But, that's mainly cause the rear window motor is broke, and the window is wide open.

So, that real familiar blast of arctic air you feel ... ? It's genuine.
Because ... you're like everyone else who owns a GM vehicle.
You: "There's no way that they make this car to break like this ... "

Yeah there is, actually.
They don't actually care how ignorant the manner of generating more profit is, what matters is whether the
garbage they made generates a net profit at any cost whatsoever. Even the Germans would never do something this
moronic to something they put a lot of time, and effort into.

GM just does not care about the customer's opinion at all ...
Even if you are really cool or, important.
Why ... ?

As soon as the really important person says something to their buddy, someone at GM will wait for that person
to leave, and the moment they shut the door, they'll say:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

GM : Yeah right pal.
     Stop making terrible parts that break a lot ... ?
     chuckles Heh.
     YEAH RIGHT, DUDE ...
     Take a hike.
     There's just no real money in making cars that don't constantly break down.
     My grandfather tried that for like 50 years ... he was constantly broke.
     Now look at him.
     That dude may have built this company with his bare hands ... ?
     But- if he had only figured out how to source consistently lower quality parts that break more often ... ?
     He would've been a billionaire.

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

And now that person or those people happen to be living quite comfortably.
They make damn certain that THEIR vehicles aren't equipped with the defective products. Nah.
Not theirs.
Microsoft would never, ever, not in their entire existence, implement intentional design flaws.

```
                                                                                                            ______________________________/
\__________________________________________________________________________________________________________/ Microsoft's Golden Standards
  Insulting General Motors and Big Oil /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

Seriously. That's because they have ACTUAL gold standards... not "quote unquote gold standards".
When the FBI guys drive around in the State issued GMC Yukon Denalis, they may say:

```
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\

FBI Guy : Man, why does this thing always smell like burning oil...?
Me      : Well, it could be because, that's what it's doing.
FBI Guy : There's no way, dude...
          This is a deluxe edition Yukon Denali, brand new, and it has like 10K miles on it.
Me      : But- it was made by General Motors...
          Right...?
FBI Guy : Yeah.
Me      : WELL, if GM made it, then there IS a way that it's burning oil, because...
           ...EVERY GM vehicle does that.

\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
```

So, someone who thinks they're driving a good vehicle, they'll give the manufacturer the benefit of a doubt because of how much money they spent on it. With a name like General Motors, the more money people spend, the more innocent they look. But, the oil filters will ALWAYS have a slight groove on the thread where the filter screws into place and that's right where the oil starts to leak/burn, so the truth is:

```
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\

Me : Nah, it's not burning oil...
     It's just burning AND leaking oil.
     Big difference.
     When it DOESN'T leak...?
     It burns.
     When it DOESN'T burn...?
     It'll just leak.
     Can't have one without the OTHER...
     And that's the best of qualty General Motors craftsmanship.
     Straight-up BOB LUTZ 101.

\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
```

The reason why it sounds ridiculous, is because sometimes the truth is stranger than fiction.
There's a combination of leakage AND burning occurring simultaneously.
Maybe it only happens on (7/8) GM vehicles, and the (1/8) vehicles that doesn't leak...?
Well... it just BURNS the oil FASTER than it can leak.
So, someone might try to tell me that makes them an exception.

But, it doesn't.
Literally 100% of their vehicles do this.
I KNOW WHAT I'M TALKING ABOUT BECAUSE I'VE OWNED LIKE A DOZEN VEHICLES MADE BY GENERAL MOTORS.
THEY ARE ALL POORLY MADE <ON PURPOSE>.

They don't make a single vehicle that doesn't burn oil, or doesn't leak.
Wanna know why...?
Because then people BUY MORE OIL.

Suppose someone actually asked GM executives,
"Hey, do all GM vehicles leak/burn oil...?"

Cause I'll tell you right now, they will say no, and if you ask them to PROVE it...?
They will say: "We're too busy."

That's cause they're not busy at all, and they just lied about whether their vehicles all burn/leak oil.
That's how they operate. The truth is, it doesn't matter WHO spends WHAT amount. If a GM vehicle is involved when someone smells oil or something burning, ...?

That's because the engine of that vehicle is doing just that.
Believe it or not, but a Chevrolet Impala, uses the same exact chassis as a Cadillac CTS.

Even if the Cadillac has leather seats and a much more powerful engine... after 20K miles?
WELL, the transmission will OCCASIONALLY start SLIPPING into 3rd gear.

Why? Uh, intentional design flaw, helps to make certain that the transmission is on track to fall out RIGHT AFTER the warranty expires. If you think I'm kidding, I'm not. I might sound like I'm making some of this stuff up, but the only thing I am making up are these various conditions...

Every vehicle has these problems INTENTIONALLY implemented.
My 2004 Pontiac Grand Prix had a 3.8L V6, bumper to bumper warranty, and when driving the vehicle, my car would make these POPPING noises every time I would hit the accelerator, and then again when I'd hit the brakes.

That's because the dealership I bought the vehicle from, Bill Cass' Northstar Chevrolet...?

They had ASE certified technicians for sure.
They would CLAIM to have done work on my vehicle.

BUT- when I'd leave, the NOISE was actually STILL there.
They would give me an EXTREMELY DETAILED LOADOUT of ALL the "work" they SUPPOSEDLY did ... right ... ?
But I would still hear the god damn noise that prompted me into bringing the vehicle in for service.

So, I kept saying to myself

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Me : What if they're not actually doing any work on my vehicle ... ?
     Every single time I bring it in for warranty work ...
      ... it has the same exact problems it did before I dropped it off.
     Virtually identical ...

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Then when you ACCUSE them of not doing the work, they'll try to make you feel bad like:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Alex : Oh, I thought you and I were cool, buddy ...
       Can't believe you're actin' all psycho now ...

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

That person is trying to blame YOU for being RIGHTFULLY pissed off at THEM, for lying to you.
I was just TOO YOUNG to realize that I should CAPITALIZE on my GUT INSTINCTS like that, because so many people
make a living off of being COMPLETELY DISHONEST. This would include the POLICE, LAWYERS, JUDGES, and DOCTORS.

That's because I was charged for a warranty that they made no effort to provide.
They SAID that they did all of this work (allegedly), but if the vehicle still makes the SAME NOISES before it
got dropped off, and continues to make those same exact noises AFTER it's picked up ... ?

WELL ...

```
_____________________________________________________________
| Occams Razor - the SIMPLEST EXPLANATION is most likely correct |
-------------------------------------------------------------
```

Someone expected that their personality APPEARED to be trustworthy LOOKING, and that was ...
Bill Cass' Northstar Chevrolet
I mean even ITT Technical Institute got involved in this charade too ...

Don't get me wrong here, it's not like ALL of the people at GENERAL MOTORS are involved.
Most of their employees aren't intelligent enough to know that the people who OWN the company, they do this.
Same goes with many of the employees at ITT Technical Institute, they may not have been intelligent enough
to know that their COMMERCIALS, and STUDENT LOAN PROGRAMS were basically committing PREDATORY LENDING PRACTICES.

That's just how the game called "AMERICA" is played, my friends.

Yeah. That's right. Some people don't care if you catch them lying to you, as long as they LOOK
like they are a trustworthy person, they can 1) say, or 2) do whatever.

As for GENERAL MOTORS, my engine needed to be replaced at 107K miles, just 7K miles outside of the warranty.
Wanna know how much that cost me ... ? About $6,000. I paid for that, too.
THEN, at 130K miles ... ? Transmission fell out. Another $6,000 right there. I paid for that too.

So it just goes to show that DEALERSHIPS and GENERAL MOTORS are LYING SCUMBAGS sometimes.

I sorta told these useless bastards at Northstar that the vehicle had some problems. One guy once said to me:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Alex : Sometimes a vehicle just, it gets in an accident or something ... ?
       And, it never drives right after that.
(that dude may as well have said ...)
Alex : It's not actually gay if you're like me, and you put another mans' penis in your mouth ...

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

... and it would've been JUST AS CREDIBLE of a statement. To be fair ... ? I don't think the dude did that.
I'm just saying, if that dude wasn't such a lazy, lying cocksucker ... I would've paid off the vehicle and gotten
maybe 280K miles on the vehicle BEFORE it needed a BRAND NEW ENGINE OR TRANSMISSION ...

But, because he WAS a lazy, lying cocksucker ... ? Then I had to SHELL OUT AN ADDITIONAL $12K.
For a vehicle that originally cost $17K.
Almost like it was a TOTAL WASTE OF MY MONEY.

It's not like I have something against a dude who prefers to do that, sucking some other dude's dick ...
... but it's too much information for the given situation ... Ya know ... ?

I told this dude that my engine AND transmission kept making strange vibrating noises, and he's trying to tell
me about his sexual fantasies ... Doesn't make a whole lot of sense, right ... ?

An ASE certified technician from Northstar Chevrolet, named Alex ... ?
But, surprisingly, everyone thought he was cool as all hell.
I thought he was an idiot that PRETENDED to do actual work, and my god, I was correct after all.
Northstar Chevrolet has been LONG GONE ... for a pretty critical reason.

Anyway, had this feeling for a loooong time, that the engine and the transmission had serious issues.
I felt the transmission making some strange vibrations when it had like, 29K on it.
All the way til 130K when the transmission died on me too. Still had another year of payments to make on it.

But guess what ... ?
GM sucks at making cars.
They make ok trucks, but even they break down a lot.

My friend had a 2005 Chevy Silverado, guess what failed in it about a year before he was done paying it off ... ?
... the engine. Yeah. I don't know why people say they make good stuff, especially when their AWESOME DEPENDABLE VEHICLE just so happens to ALSO break down frequently.

So, not only do they suck at making cars, but they also suck at providing necessary services for the warranties.
Almost as if they're a TOTAL WASTE OF MONEY.

Because if you're like me, you'll notice that they don't actually do any work.
I actually feel bad for the dude Andy, because I actually believed him, but sometimes even with him,
I'd get the damn car back and magically, I would realize that if they did something at all,
it was barely noticeable.

They probably did do PORTIONS of the things they told me they did, but I spent an additional $12K on:

| Replacing the ENGINE and the TRANSMISSION before the vehicle was PAID FOR. So, they were definitely lazy |

Each time, I had a feeling that they literally charged me a LOT more money to do NO ADDITIONAL SERVICE to my vehicle. I financed a $17K car, and wound up paying about $35K when all was said and done with replacing the engine 3.5 years in, and then the transmission 6 months later.

Still owed another 18 months worth of payments when I had to shell out $5K for a new engine.
Then 12 months left when the transmission failed, that was an additional $6K.
I chose to buy this vehicle because I had hoped that buying a new car from a dealership, was a good idea.
Turns out that it was one of the WORST ideas I've ever had ...

So, I realize people might say "Why do you seem to think BOB LUTZ is the problem ... ?"

It's literally cause he was the director of GM for about 50 years, and ALL of my GM vehicles have had similar problems. Something ridiculous ... ? BOB LUTZ probably had something to do with the LINE being drawn through the ENGINEERING BLUEPRINTS somewhere.

Sometimes the high beam switches are like $400 bucks, and they'll break after like the first 15K miles.
BOB LUTZ probably told everybody that makes them,

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Lutz : Hey, use cheap plastic, but ... keep charging for like gold plated ones.
That'll show everybody ...

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

What I've realized, is that in order to purchase a GOOD vehicle that GM claims to have made, it has to have been made in some totally different continent, for instance, Australia. The Pontiac G8 was a really sweet car. But, BOB LUTZ had NOTHING TO DO with THAT vehicle being made. Nah.

HOLDEN, in AUSTRALIA, actually made it.
GM took credit for building it, cause why not ... ?
The 2003+ Pontiac GTO and the Pontiac G8 were both made in Australia.
Not America.
That's the ONLY reason why they were both REALLY GOOD cars.

Now, had they been made here in the United States ... ?
Then, they would've never been as well made.

You gotta figure, even vehicles that wind up getting fitted with bulletproof glass to protect former-governor Cuomo, the former governor of New York that somehow had 11 women accuse him of something he supposedly would never do ...

If GM made that thing ... ?
It's got many design flaws too.
Maybe if you throw a tennis ball at Cuomo's Yukon Denali, it could cause the transmission to fall out.

So, throwing it into 4-wheel drive, and going off roading with it, it's a pretty extreme activity ...
... it'll be engineered WELL for a task just like that ...
But- a properly thrown tennis ball that hits the right spot, will cause that transmission to immediately fail.

Then what ... ?
Dude was just going off-roading and won a championship amongst hundreds of other dudes off-roading in THEIR trucks ... But as soon as the trophy girl placed the FIRST PLACE TROPHY on the HOOD of the truck ... ? Ah man ...

The truck was revving it's engine alright, but now the transmission appears to have fallen out ... it's not goin' into gear anymore.

A lot of people will overlook stuff like that, a (little tennis ball/trophy) versus a $12K transmission ...
... the (tennis ball/trophy) somehow wins. At that point it's MORE than just dumb luck.

Maybe the hood won't stay up, the hydraulic pumps that are supposed to automatically hold the hood up, they have a seal that breaks because of a design flaw. They'll work for like 18 months reliably, and then magically one day, the seals met a condition where they matched equilibrium with the environment.

After that point, it's gonna hit ya in the head. So, if you want the hood to stay up without the pole ... ?

/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯\

GM : That'll be $500 bucks.
You : I don't wanna pay that.
GM : Don't wanna pay us $500 ... ?
      Oh.
      Looks like you'll need that pole then, chump ...

__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/¯¯__/

Even if people get them fixed, that's another $250 each, and they may last 18 months, if you're lucky.
But I have a WAY better idea ... What if the people who made them, actually built them to last a lot longer than that ... ? Ya know ... ? I would've thought that they could last like 10 years. Or more.

Sorta like parts on a German car, they like, last a LOT longer for some STRANGE reason.

On a GM though ... ? Oh nos.
You get like 6 months max before something else falls apart.

Sometimes they design brake rotors and calipers so that even a brand new set of brake pads will sound like the pads are scraping the rotor, but that's because the calipers have a slight bend in the hydraulic line which is meant to cause the brake pads to pinch in at a slight angle, and that causes the pads and rotors to get malformed and start warping right off the bat, causing the vehicle to feel 'dog legged' when coming to a stop ...

You'll say "I literally just changed the god damn brakes on this ... "
But, they don't care. You need to buy more already ...

Or, else you're not cool in their book.
They were installed correctly, and the calipers are good.
The problem is, they make these things poorly, rather consistently.

While they really could make things easier on everybody ... ?
Well, making all of these things with no flaws implemented off the rip, puts less money in their pockets ...
... they don't like THAT ... nobody does.

So, as soon as they hear something like LESS MONEY ... ?
Their voice starts TRAILING OFF.

Cause you said something like LESS MONEY FOR THEM, and now they're trying to stay positive, drowning you out.
LESS MONEY for them sounds like a traumatic experience.
They may even bawl at the eyes, just like Prince Andrew does when he sees the picture of him and Ms. Guiffre.
Or Andrew Cuomo does when he sees the picture of him with Brittany Commisso. Ya know ... ?

If their business isn't generating enough profit from parts made for that vehicle, then to them ... ?
That means it was actually poorly designed.

If it gets better gas mileage, sustains an impact in an accident, way better than their typical vehicle,
AND it has NO recalls or defective parts ... ?
They will actually consider that a failure on their part.

Yeah. 5-star crash safety rating ... ?
That's bad ...

These factors just don't generate the type of profit they would PREFER to see ...
We're talkin', they look for any stupid flaw that seems to happen a lot ...
... and when they notice this problem being super widespread across their entire fleet of vehicles ... ?

They will actually track down that manufacturer and they'll offer to award them with a contract in order to mass produce those defective parts.

In order for a product to be successful in GM's eyes ... ?
It can only be considered successful if it breaks a lot.
If a car has a LOT of these parts, then they will HEAVILY promote that vehicle.
A lot of ridiculous things that suddenly go bad for next to no reason at all, they all get their hopes up.
Mainly because they have eyes for this kind of thing, BOB LUTZ was the best in the industry at this task.

So, now they know a poorly developed part when they see one, and when OTHER car companies avoided using these parts ... ? Well, that's pretty dumb for them to do. Because, General Motors will shake their heads and double down.

Their eyes light up with delight.
They move heaven and earth to convince the part manufacturers, to "live a little".

It has to be fairly believable when these parts "randomly" break, that it's no coincidence at all.
Nah, you're just some psychotic individual to suggest something like that … that's why they're industry leaders.

The sad truth is, that is why BOB LUTZ was the director of GM for over 50 years.
He was very gifted at tracking down these terribly made parts, and then flooding every vehicle he could with THOSE consistently terrible parts … It might sound ridiculous, but this dude actually has a lot more experience than Elon Musk or Tesla, at making electric vehicles AND terrible cars.

Even with Elon Musk AND Tesla BOTH having a LOT less experience than GM, Ford, or Daimler-Chrysler …
… all of that is irrelevant. Why … ? American car corporations worry more about profit, and less about the quality of their products. With other corporations and industries, you get a lot less of an EVIL BASTARD vibe.

Too worried about BURNING and LEAKING oil … not about GAS MILEAGE or SUSTAINABILITY.
The MORE MONEY they can CONVINCE people to WASTE … ? The BETTER.

```
                                                                                  ______________________________________/
\________________________________________________________________________________/ Insulting General Motors and Big Oil
  Climate Change /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

That means they have MORE to give to RUPERT MURDOCH, SEAN HANNITY, or TUCKER CARLSON, which works great for them.
To them … ?
They don't care about wildfires.
Nor the plastic in the oceans.
Nor the cancer people contract from exhaust and smog.
Nor the many tornado outbreaks.
Nor the flash floods.
Nor the hurricanes.
Nor the ocean levels rising.

Nah.
All of that stuff is a lot less important to them.
Actually, all of those things are A LOT LESS IMPORTANT, than these people, in their eyes.
So, if many animals and plant species keep going extinct … ?

They obviously rationalize with themselves, that those animals or plants probably didn't have what it takes to exist in THEIR world. Because they figure, if those animal or plant species were worth a damn, they'd still be around … and that's what they say to themselves.
Keeps em optimistic about the future …

I realize, some people might think it's alarming that the car companies that keep building gas powered cars think all of these things, but what's worse is how they say nothing or even deny it, when their ACTIONS CONSISTENTLY PROVE OTHERWISE.

To them, they just don't have what it takes to do what Elon Musk is doing with Tesla, it's called TALENT.
What Musk/Tesla is doing primarily considers that oil will eventually all run out.

If that happened TODAY, how screwed would we be? Very.
If we ran out of oil, right now?
A lot of people would die, very quickly.

So, if the world ran out of oil TODAY, the question then becomes, are we currently prepared for that much loss of life … ? Well, no. The Doomsday clock has been ticking the whole entire time, and people just have no earthly idea how close we are to an extinction level event. We cannot moderate ourselves, even the smartest people on the planet have nothing at their disposal to restore order.

So, then the question should really be, will GM still try to sell these 2022 Chevy Corvettes when there's no oil or gas left … ? Yep. They sure as hell will. They will know that there's no way to enjoy or do anything with a vehicle that has a 20th century power source, gasoline.

So these idiots just keep making/selling the 2022 Chevy Corvette, even when ALL of the gas is gone.
Why … ? I think it is because they're MENTALLY HANDICAPPED, because $`$$$$.
It's a PERFECT explanation … isn't it … ?

If they AREN'T MENTALLY HANDICAPPED, then my next best suggestion is that they should consider BUYING the vehicle, instead of SELLING it … As the effort might allow them to catch on to how moronic the idea is.
Ya know … ?

```
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
```

Them : Hey, wanna buy a 2022 Chevy Corvette … ?
It looks insanely cool.
Oh wait, I gotta pretend to be the guy buying it …
Who's gonna sell it to me though … ?
Damnit.
Maybe this was actually a really stupid idea, to try and sell this car now …
OoOhh, just sell myself a 2022 Chevy Corvette when there's no more gas left on the planet.
How dumb am I … ?
My god.
I can only imagine how many other people felt when I asked em …
They probably thought. "Wow. This dude is a total moron … isn't he?"

```
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
```

Maybe everyone on the planet should consider that these people lie to themselves every single day.
Standard issue TUCKER CARLSON and SEAN HANNITY.
They might say climate change isn't real, NOW, but later on when they're all alone …
… and they realize that the weather is DRASTICALLY DIFFERENT than it used to be …
they'll have this startling idea …

/--__/--\

Them : You know … climate change probably WAS real that whole entire time and I was just some overpaid shill
whining on national television about everything …
Ah, god.
That IS what I did.
Wow.
I feel like a complete idiot …

__/--__/

For the longest time, they invested so much energy into lying to themselves, now they're telling themselves the truth, and it sounds so foreign. They didn't realize just how moronic they sounded to so many people … Not only do they lie to themselves, but they lie to everyone else too.

That's because they lack the ability to tell a single truth about anything at all.
If they COULD tell anybody the truth, they would've all decided to help Elon Musk build Tesla, and the batteries.
Why not sooner …? Just cause. Gasoline used to be extremely abundant. And then one day, it'll ALL be gone.

Why don't they do that, help an actual professional get the job done … ya know …?
Well … it'd be wishful thinking, that's why …
They're MENTALLY HANDICAPPED because of the `$$$$$, that's why.
These people are in absolute denial to think OTHERWISE.

Just like TUCKER CARLSON and SEAN HANNITY say "climate change isn't real …"
I'm not going to assume that they know any better, because they might not at all.
You could SHOW THEM …? But- it's like trying to tell some MENTALLY HANDICAPPED person a joke.

Sometimes you have to masterfully explain to some dude that gets paid millons of dollars every month, that all the money they keep getting paid won't do them any good if the shit hits the fan, and they declare martial law …
Or, if they get themselves killed.
Or, even both.

Trying to make SUGGESTIONS to either of them, is bound to cause some confusion.
As they may think someone is stating intent, rather than through the same sort of rhetoric they EACH use to sound increasingly more moronic, each additional appearance they make … someone should tell them, it is a true statement … doesn't matter who says it about anybody.

It's still true.
From what I know about death, nobody has ever been able to live comfortably, and maintain their finances really well, if at some point, they became a corpse. Ya know …?
Corpses are NOT HISTORICALLY KNOWN, for making FINANCIALLY SOUND INVESTMENTS …
If anything, they're a real DRAG to be around, might smell …?
Might not look like they're doin' too well …?
But that's because CORPSES are typically not alive to maintain their APPEARANCE or REPUTATION …

Sorta puts a damper on being able to live comfortably … or manage finances. Or really, anything.
But you know, it can't hurt for someone to use some poetry and descriptive scenes, in order to comfront their own mortality from time to time. In reality, Steve Jobs did that quite a lot.

He was a better presenter and public speaker, than these Fox boys ever were … and could phrase an entire life story, as rhetoric … When he told people that he didn't want to be the richest guy in the graveyard, I'm pretty sure he put a lot of thought into the phrase before he ever said those words out loud.

So, even though the dudes on Fox News constantly spam people with THEIR rhetoric, they may not get it if someone else does that same exact thing in return.

_____/
_____/ Climate Change
Steve Jobs: 'You've baked a lovely cake …' /---\
/---

What if everybody just showed up on their set, and just started phrasing rhetoric too.

/--__/--\

Guy : Steve Jobs "You've baked a very lovely cake, but you've used dog shit for frosting."
Carlson : Well, why would anybody do that …?
Guy : I don't know, I think it's a metaphor.
Carlson : What do you think it's a metaphor for …?
Guy : I could ask you the same question … you know?
Carlson : Well …
… to me …?
Sorta sounds like maybe people shouldn't shit where they eat …
Guy : Hey~!
That's- quite the interpretation~!
Carlson : Yeh.

chuckles Heh.
Pretty good one eh?
Guy : *chuckles* Lol.
Nah, not at all.
Carlson : But it makes perfect sense to me ...
Guy : I suppose people shouldn't shit where they eat, but that's not what it's about at all.
Carlson : Well ...
I tried pal.
I don't know what it is supposed to mean.
Guy : It means that if you're gonna do something phenomenal, go the full measure.
Otherwise, people are gonna detect such a really poor choice when they (taste/smell) the dog shit
Carlson : Alright.
I can see how that makes sense.
Guy : Yeah, but like usual, you needed to have it explained to you to have an opinion about something.
Carlson : ... I have my own opinions sometimes ...
Guy : Yeah.
You thought the metaphor was for not shitting where you eat ...
Carlson : It's a good bit of advice ...
Guy : Why don't you follow something like that ... ?
Ya know ... ?
Carlson : I do not shit where I eat ...
Guy : Could've fooled me, buddy.

__/--__/

Steve Jobs was a smart bastard for a number of reasons, not just cause he started Apple and invented the iPod, iPhone, and iPad ... though they definitely helped.

He just truly didn't understand the sense of hoarding everything he could, and just keepin' all that stuff to himself. Nah. Not him. He didn't care to be the richest man man in the graveyard ...

But- the quote really means that people can't use money if they get themselves killed.
So, when someone does something for years in exchange for millions of dollars, and the shit his the fan ... ?
Then, people may no longer be able to purchase amenities, gas, supplies, or food.

Cause it's all gone ...
Then whoever didn't prepare for the situation ... ?
They're all screwed.

_____/ Steve Jobs: 'You've baked a lovely cake ... '

Martial Law /---\
/-------------

It stands to reason, that this idiot might actually be surprised when he tries to go to the store and use some of those millions, when the world is in a state of martial law.

/--__/--\

Carlson : Ah, man.
Didn't realize just how moronic it would be ...
... to assume that the shit would never ACTUALLY hit the fan ...
Jeez.
I really am not a smart dude, am I ... ?
Soldier1 : *pointing rifle at Carlson* Are you TUCKER CARLSON or some shit ... ?
Carlson : Yeah, that's me.
Soldier1 : What the hell are you doin' here ... ?
It ain't safe buddy.
Carlson : I ...
... forgot I can't spend any money now.
Soldier1 : That's ...
... gotta be the dumbest thing I've ever heard a man say.
Carlson : You know, how was I supposed to know ahead of time, that this was gonna happen ... ?
Soldier1 : *stops pointing the rifle at this guy* You really are a helpless idiot ...
... aren't ya?
Carlson : Look pal, you don't have to be so mean about it ...
Soldier1 : *yells to his buddy who's a fair distance away* Hey dude~!
You're never gonna believe this, but guess who I found~!?
Soldier2 : *voice from a slight distance* Who~!?
Who'd ya find~?!
Soldier1 : Friggen, TUCKER CARLSON, man~!
Soldier2 : *distant* TUCKER CARLSON ...
... isn't that like, the guy from Fox News, TUCKER CARLSON ... ~!?
Soldier1 : Yep~!
It's him alright ... ~!
Soldier2 : *distant* You're right dude~!
I don't believe ya~!
Dudes' a friggen millionaire~!
Soldier1 : Well, come check him out~!
See for your yourself~!
Soldier2 : *distant* Alright, be there in a minute~!
Soldier1 : Heh.
You hear that Tucker ... ?
He's comin'.

He's not gonna believe it either ...
Carlson : Are you guys gonna kill me ... ?
Solider1 : Well, that depends.
Carlson : *gulp* ... on what ... ?
Soldier1 : Buddy, you don't just go runnin around town tryin' to buy supplies in a state of martial law.
Not without some consequences for makin' such a dumb decision, being made readily apparent to you.
Carlson : I don't wanna die ...
Soldier1 : Buddy, you really think I wanna be this guy ... ?
A soldier at war, paranoid as hell ... ?
Carlson : *sniffles* This feels like a really bad dream~!
Soldier1 : Man, shut your mouth.
You're a grown ass man.
Can't be *waves hand around* cryin' like that ...
Carlson : *sniffles*
Soldier2 : *approaches, sees TUCKER CARLSON* Wow!
I thought you were bullshittin' me ...
Soldier1 : See ... ?
Told ya.
Straight-up millionaire LL Cool J-Thompson, Action-Jackson right there.
Soldier2 : ... that's not LL Cool J, bro.
It ain't Thompson or Action Jackson either.
Soldier1 : I didn't literally mean all that, dude.
I was just sayin' it, makin' him sound fancy.
Soldier2 : ... Alright.
Well, that's DEFINITELY TUCKER CARLSON right there.
What do we do with him then ... ?
Soldier1 : I don't know yet, still trying to figure that out ...
Can't just let him GO ...
Soldier2 : Obviously, he's a millionaire.
He's gotta be worth *something* right now other than money ...
Soldier1 : I got a feeling he didn't prepare a god damn thing, though.
Carlson : *sniffles* I've got some property
Soldier1 : Weapons, dipshit.
And supplies.
Carlson : *whispers* This is a nightmare ...
Soldier2 : Heh.
Dude said "this is a nightmare" ...
You're a straight up noob, aren't ya?
Carlson : I guess.
I don't know what that is ...
Soldier1 : Chum, it means you're a straight-up, brand new player in a game you ain't never played.
Carlson : Yeah.
Maybe I am, so what ... ?
Soldier2 : *looks at the soldier* At least he admits it.
Can't be all THAT bad ...
Soldier1 : He's basically helpless.
Doesn't even have a weapon on him, nothin'.
Not sure we can even trust him either ...
... cause of how many times the man has lied on national broadcast television ...
Carlson : I don't lie on Fox News.
Soldier1 : Yeah, you most definitely do, and have, many times dipshit.
Now, shut your mouth.
Soldier2 : What the hell was he doing out there, anyway ... ?
It's dangerous out ...
Soldier1 : Yeh.
This idiot thought he could spend his money whenever.
Didn't realize money's no good now.
Soldier2 : I mean, that's not completely true ...
Soldier1 : Yeah ... ?
Where could he go, right now, and spend that damn money of his ... ?
Soldier2 : The army base sells supplies, still.
Soldier1 : Ah yeh.
They'll still take this dudes' money, won't they ... ?
Soldier2 : That is, if it's not stuck in his bank or something ...
Soldier1 : Is that true Tucker ... ?
You got real fat stacks of cash on hand somewhere ... ?
Hm ... ?
Carlson : *sniffles* Yeh.
I might.
Soldier2 : How much you figure ... ?
Carlson : I've got a few million in cash-money.
Some of the fattest stacks you'll ever lay your eyes on.
Soldier1 : Like, for real ... ?
Not in the bank or whatever ... ?
Carlson : Yeah.
For real.
REAL fat stacks of cash ...
Soldier1 : Buddy ... *turns to Soldier*
Dude might make himself useful after all ...
Get ourselves some weapons from the army supplies store ...
Soldier2 : We could try it.
It'll be dangerous ...
Soldier1 : Extra dangerous.

```
            But, he's talkin' about some sky high stacks or somethin'
    Carlson  : How dangerous ... ?
            And I said some of the fattest stacks of cash you'll ever lay your eyes on ...
            I didn't say sky-high stacks ...
    Soldier1 : Dangerous to the point where you need a kevlar vest to be out the open, at a bare minumum, just to
            stay alive.
            But I mean, if you've got some real fat stacks of cash layin' around,
             ... you could prolly use them, too....
    Carlson  : *sniffling* Yeah ... ?
            Just use some straight-up cash, for body armor ... ?
    Soldier1 : I mean, it's definitely a better idea then gettin' shot dead by somebody's random bullet ...
    Carlson  : *stops sniffling* I'm ready.
            Let's do this ...
            I'm not gonna keep crying like a little girl.
    Soldier1 : Alright.
            You just earned some respect from me by sayin' that.
            You got some kevlar or somethin' ... ?

    \__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/

    Maybe none of these things will happen.
    Maybe everybody that pays all of their bills, and listens to TUCKER CARLSON, they do it because he's just a
    floating voice that echoes and bounces off of the walls. Like a pinball.

    But, like an idiot ...  neither Carlson nor Hannity put much thought to climate change possibly showing up at
    their doorstep someday. Maybe for most people, climate change felt like it COULD be one giant hoax, perpetuated
    by all those PESKY SCIENTISTS that Carlson and Hannity basically undermined for years as they were handed some
    of the fattest stacks of cash that anybody whose ever worked at Fox News, has ever been personally handed.

    They both lived the dream ...  being on modern every day network television.
    Repeating the gospels of fools, while also foreshadowing the certain inevitable arrival ...
     ... of the day where everything began to fall apart.
                                                                                                          _____________/
\_____________________________________________________________________________________________________/ Martial Law
  Judgement Day /-----------------------------------------------------------------------------------------------------\
/---------------

    That's because, the day of reckoning finally arrived, with the skies torched and blackened, and the sun drowned
    out like a faint glimmer in a seemingly endless void, an oasis in the desert ...

    Climate change was in everybody's face that whole entire time ...  it just had to roll up it's sleeves after the
    day of reckoning finally came ...  Judgement Day.

    The question remains, what will that day actually bring forth ... ?
    Will it be hordes of angry mobs, each with a flaming pitchfork and a molotov cocktail ... ?
    Under martial law, seems pretty likely.

    Gotta say, I'd be terrified of even hearing about this story if I were either 1) Carlson, or 2) Hannity.
    Why ... ? Cause I'm certain that some militia has been waiting for the day to visit em'.

    If it gets to that point, I am gonna feel bad for those guys.
    They're gonna have to face the music. It won't be very fun ...
     ... and not a soul will be willing to protect these guys ...
     ... even if they paid a security team a real nice fat stack of cash.

    Because the cash sorta loses it's appeal when the military/government officially declare a state of: MARTIAL LAW.

    Cause that's when they'll scatter like cockroaches and say "Ha. Never liked your ass anyway, bro."
    Then again, maybe not.

    Regardless, for the every day person who isn't a millionaire, even if you happen to be one ... ?
    Once MARTIAL LAW is declared, the state of the world immediately becomes a DREADED NIGHTMARE where MANY PEOPLE DIE.

    At that point, you better hope you have:
    _____________________________________________
    | WEAPONS | AMMUNITION | ARMOR | SUPPLIES | FOOD |
    ---------------------------------------------

    If you don't, you'll already be basically dead. So many people will be caught in this box, and get killed.

    Money will eventually lose all of it's value if these things happen.
    Everything will spiral out of control so fast, people won't realize how grave the situation is ...
     ... until it's too late.

    However, the truth is ...  Guys like Elon Musk exist, and, it stands to reason that he is the only person in the
    world, who is working a lot harder than everybody else is, at preventing all of those nightmares from happening
    in the first place. Though to be clear, he is FAR from the only one. Many people are ALSO helping.

    Still, all of those worst case scenarios could theoretically be completely avoided if people help him, but even
    if people all jump on the bandwagon, start now, and act fast ... ? Time is very slim.

    There will come a day where people won't even care to complain about the car or oil industry anymore.
    It'll just be a lot like the show "The Walking Dead".
```

Everyone's just trying to survive.
Groups of people sticking together, working in colonies, as one.
Some aspects of normal life will try to seep back in ...
... but society as we currently know it ... ?
... it'll be permanently gone.

We could blame the American car and oil industry long before any of that happens ...
There ARE plenty of other industries to call out ... but it all goes right back to GM, being a lousy magician ...

GM used to be the poster child for American car manufacturing, at least it used to be, about 40-50 years ago.
Then, this guy BOB LUTZ came along.

He's probably highly regarded by so many of the people that have come and gone ...
But, even the Germans and Italians over in Europe never really had the problem of being unable to make good cars.
They were very surprised when Bob came along and just started making things a lot easier for the Europeans to dominate the market. They just maintained the idea of "quality" and high standards.

The word "quality" was just never a part of BOB LUTZ vocabulary, not for 50 years.
Not unless it was prefixed by the word "low".

The one word by itself ... ? Oblivious. But together ... ? Heh. They were words that he lived by. Even today.

"Low Quality" ... GM just became a totally different company after that. The Germans were intrigued by how effortlessly this man made their lives, exporting vehicles to the United States.
Even Marc from Marc's VM and Import Service saw how terribly made General Motors vehicles truly were ...

| Marc's VW and Import Service | http://www.marcsvwandimportservice.com |

Since before the age of Hitler, even. Germany has made nothing but good vehicles.
Even their most garbage vehicles last a LOT longer than American cars (unless it's a Tesla).

The reason why, is because BOB LUTZ really did change the entire market, single handedly.
Europe is just a LOT MORE SKILLED at making dependable, durable, higher quality vehicles.

Germany is awesome at it. Same with Italy, with their Ferarri's, Lambo's, Porsches ...

If there's one word that describes European vehicles ... ? They're WELL MADE, or "Good".
"Bad" isn't really a word that ever comes up unless you start talking about American cars (unless it's a Tesla).

European vehicles are durable no matter how hard they try to screw them up.
They're not PERFECT by any means, but ... they're a hell of a lot more well made than American vehicles.

That's saying something. Not only are they durable and last a long time ... ?
But, they're also worth more money, like even if you bought one, and drove it off the lot.

European cars lose a hell of a lot less value.
With an American car (unless it's a Tesla), it's as if 30% of the money spent on them, is immedately thrown into a barrel, soaked with gasoline, and lit on fire. That's because, BOB LUTZ knew how to do something like that, from his very first day at GM.

What I just said, is REALLY BAD. Spend WAY MORE, get a LOT less.
That's the age of internal combustion engine companies, and the WAY they took over a 100 years, to stop destroying the environment and cause climate change.

Lazy/greedy car companies, a handful of morons on Fox News, and oil companies that assisted in the controlled demolition of 3 buildings in Manhattan, New York ... on September 11, 2001

Well, we're getting pretty thrown off course from Programming with PowerShell, aren't we ... ?
I promise, it'll ALL coalesce back into the original lesson plan.

The Central Intelligence Agency, as well as George W. Bush, knew Osama Bin Laden planned the airplane attacks.
But, Big Oil probably planned the controlled demolition, and the collapse of those towers caused people to believe that the AIRPLANES caused the towers to collapse ... not the explosive charges and additional 1000 degrees fahrenheit that would've been required to demolish the builing in the exact fashion that it was destroyed.

But- that's because, many morons exist.
When using a sifter to sift through ash of roasted skeletons, what gets sifted out of the rubble, are details that make the official story that NIST told, ABSOLUTELY IMPOSSIBLE. It's why WTC 7 wasn't hit by a single plane, and yet, anyone who watches videos of these buildings collapsing, they will inevitably realize ...

Robert Korol was correct that whole entire time. Never was incorrect, actually.
Kept being correct from the very first day, and continued to be correct, as well as Leeroy Hulsey, Xili Quan, and Feng Xiao.

Wanna know who was NEVER correct at all ... ? The National Institute of Standards and Technology.

Especially the report that Shayam Sundar signed off on, for former president, George W. Bush.

At which point, the story could go forward in time and talk about violations to the Constitution, committed by President Bush, Michael Hayden, the NSA/CIA/FBI, PRISM, Technology, xKeyScore, Mystic, but ultimately, all of that leads to a thing called: Cyberterrorism, a thing that police don't appear to investigate ... only civilians.

And that's how it ALLLLLLLL ties back into the original lesson plan.

Cyberterrorism.

```
                                                                                                            _______________/
\_________________________________________________________________________________________________________/ Judgement Day
  Cyberterrorism /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

If you contact the local boys, they will laugh at you cause they don't know about stuff like Pegasus.
Some of them are pretty gay about it too.

The most advanced spyware application in the world, incredibly dangerous, allows a remote party the ability to log into a device with no user interaction whatsoever on Apple, and Android.

The State boys do SOME work with actual FBI guys, but the local dispatch office where I live has someone there that seems to have it in for me. I have a list of local sherrifs, and state troopers, who could each stand to be shown how be a lot more attentive to detail and observant.

As it is, some of them really are that stupid. (Though, I can't apply that designation unilaterally)
Same goes for trying to call the FBI, or the NSA, or whatever. (Same as above)

I've come to believe that none of these organizations even do what they tell the general public they do.
They have their tasks and orders, and those orders are to avoid interacting with the general public at all costs. Because if the truth came out, that their main duties are to surveil and control the general population, and lie needlessly about everything that they possibly can, it makes more sense than trying to call them, and ask them why they always seeem to be completely silent recording phone conversations.

Yes. They have made it very obvious to me that once you know exactly what I do...?
They manipulate the general population to think that I must have it all wrong... when I don't have it all wrong at all. If they gave me the INCENTIVE to believe otherwise, they would've done so by now.

There's only one party I know for a fact that I can trust.
Until I see *any* indication otherwise? I remain firm with this assessment.

They are absolutely terrible at it. (Though, I can't apply that designation unilaterally)
Just as terrible at investigating technology based attacks, as BOB LUTZ has always been at making good vehicles at GM for 50 years. A long chapter of time where some blindfolded noob just swung wildly at whatever, and called THAT.... "being a role model citizen".

```
                                                                                                           ________________/
\________________________________________________________________________________________________________/ Cyberterrorism
  Greatest Challenge of our Time /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

But, it could also branch back to global warming and climate change.
Al Gore.
Leonardo DiCaprio.

The two met way back in the year 2000...
Al Gore told young Leonardo...

```
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
```

AG : This global warming thing is gonna get REAL bad, kid.
LD : *nervous gulp* Yeah...?
AG : Oh yeh, it's gonna change everything.
LD : Well, what can we like, DO about it...?
AG : You could vote for me.
     Everyone could.
     I can't explain how bad the problem is in a single minute, but it would be worthy of it's own documentary...
     ...because it's the greatest challenge of our time.
LD : *gulp* The... greatest...
AG : -challenge of our time.
     An existential threat, for certain.

```
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
```

All of these things, when combined into a total narrative that sounds more intelligent than TUCKER CARLSON ever does on any given day, are the reasons why the wildfires are really bad, AND the droughts are getting a lot worse (look at Lake Mead, less than 1/3 of it's normal capacity, and dropping).

But also, the hurricanes are getting worse.
And, the deadly tornado outbreaks.
And, houses and coastal towns slowly receding.
I could probably continue with the story many different ways.

The truth is, America has too many people that blindly believe whatever they're told.
Hence why NEWS versus PROPAGANDA is RATHER IMPORTANT TO UNDERSTAND.

If people blindly believe whatever they're told...?
It immediately causes them to become fools. Effortlessly.

If some important person lies to everyone on a press conference...?
80% of the people will believe the lie that they are told.

Then when you TELL them the actual truth, it is extremely difficult to match the criteria where

they'll consider entertaining a much better, more accurate story, without being offended.

Because if you lose out on getting through to them within this window, they will go right ahead, and prove that they're just too ignorant or careless to dislodge the notion, ... that they've been lied to so many times, the truth is stranger than fiction.

Not just (1) time either. But, repeatedly.
That is why the car industry is still as dominant as it is, Americans don't actually care if they waste their money on a vehicle that was poorly made, or if they waste $1T on a bunch of weapons, equipment, vehicles, helicopters, armaments, and other investments in Afghanistan.

So, many Americans fail to grasp, that every single penny that they gave the government for X amount of time, was thrown into a barrel, soaked with gasoline, and the set on fire. But, they don't get it.

So many people died, and so many of everybody's hard earned tax dollars, they're being pissed away, by people in the government that are CARELESS. When I say careless, I would go so far as to say "brain dead".
Or like, "someone that spent 50 years making terrible vehicles at GM."

When the F-35 Lightning was boasted about being the next heavy contender for the skies ... ?
I knew that Lockheed Martin already made the best fighter jet ever built by a single country, it was the best thing George Bush ever committed to building, as President. Bush has so many stains on his legacy, that it is LITERALLY UNSAFE to write about those stains at all.

Because, someone will censor anybody for discussing those very clear, and real stains.
But- the F-22 Raptor, wasn't a stain at all.
The war in Iraq ... ? Not a BAD idea.
The war in Afghanistan ... ? Literally the worst idea our military has ever committed to, by far.

But, even considering all of those things ... ?
Former President George W. Bush, is a saint, of all sainthood, in comparison to this moron named BOB LUTZ who ran GM for 50 years (though to be clear, GWB is no saint).

_____/ Greatest Challenge of our Time
/ The Adventures of Starman \---/

I realize someone will probably tell BOB LUTZ about this entire document, and if he reads it, he may cry.
And, that's ok. Elon just went right ahead, and upended everything this man BOB LUTZ ever did for 50 years straight ... Elon had the (1) company he owns/runs, one who doesn't make a single bad vehicle ... make a Tesla Roadster.

Then, he had another company he owns/runs, outfit a dummy with a prototype SpaceX suit, and then had them load that vehicle into a payload fairing to send the thing into space. Then ...

The man asked some people who work for him, to spend a lot of time being very creative about Starman, the adventures of Starman, comic books, stories, and making this website that can ...

| The Adventures of Starman | https://spacein3d.com/where-is-starman-live-tracker |

... show BOB LUTZ, or anybody else really, where that car he sent into space, happens to be right *now*.
The whole entire thing just proves to me, that some people don't actually whip out an actual middle finger, nor do they say a single word. Nah. They do something that everyone will remember throughout human history.

Elon Musk, gave a guy BOB LUTZ, the BIGGEST middle finger, in the history of middlefingerhood ...
... and to be fair, giving entire industries full of people that sat around and did nothing ...
... a huge middle finger to them ALL for doubting him immensely, Elon Musk set out to do what no man has ever tried to do before him, like the man was born to do it.

Dude might crack some jokes that piss a lot of people off even more ... ?
But, they never sent a vehicle into space with their own fully paid for rocket, fully paid for car, unique power plant, nah. The COOL thing is that if he had thrown some SOLAR PANELS onto the thing, the car would legitimately be able to TURN THE WHEELS the ENTIRE TIME it was in space. Cause SOLAR ENERGY from the SUN, could've CHARGED THE VEHICLE ... but- Elon didn't wanna take it THAT far ... Nah.

To say that I happen to be highly impressed by the principle ... ?
That's an understatement. Some people are just going to be miserable no matter what, because, they're just as lazy as BOB LUTZ always has been. And that's fine. So it really isn't Elons fault, that the world is full of so many negative people, who couldnt do a fifth of what he's done, if they all tried together ... ?

_____/ The Adventures of Starman
/ Almost \---/

Yeah. George Bush was a perfect president, when compared to this moron, BOB LUTZ.
Bob made lot of really bad vehicles. When I say BAD, I mean ...
... in his entire 50 year tenure, the man never completed (1) single thing to be proud of ...

The truth is- BOB LUTZ just about ALMOST made a LOT of GOOD vehicles, millions of times over. People would say:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

People : Wow, look at you go, Bob~!
Really are phenomenal at ALMOST making good cars~!!"

Lutz : *pause for a moment, snaps back annoyed* Whaddya mean ALMOST ... ?

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

But- those people would say 'A valiant effort.', shrug their shoulders, then leave the room.
All the while, every single time people did this, it eroded his sanity.

Even poor Bob knew that ALMOST only counts in 1) horseshoes and 2) handgrenades.
Otherwise, ALMOST is when you need a minimum of 1.0, but- you can't get past 0.9999999999999
That is *exactly* what the term ALMOST is ALL about.

General Motors has come very close to ALMOST making millions of GOOD vehicles.
The problem is in the way they design the vehicles.
There are only (2) conditions where a General Motors vehicle can be called a GOOD vehicle.

____
| [Condition #1] |

When a mechanic installs parts that GM never 1) made, or 2) touched even once, in that vehicle.
NOW ... the vehicle may theoretically meet GOOD vehicle minimum requirements, like MAGIC.

That's because, technically it's not a GM vehicle if the mechanic uses QUALITY parts.
GM doesn't make QUALITY parts. Used to at some point, but they can't remember how they pulled it off.

So, if it has parts in it that came from the factory? It *automatically* fails the test.

But, if it meets the requirements, and it passes the test ... ?
It can legally be considered GOOD.
This is actually EXTREMELY RARE ...

____
| [Condition #2] |

The vehicle was GOOD for a few seconds at the factory up until it found out who made it.
... which is when it suffered a traumatic event, sustaining permanent engine damage.
This is by design. The vehicle may SOUND fine whenever it is started.

At the moment the vehicle is completed and attributed to BOB LUTZ in any small way whatsoever ... ?
... that's when the car itself says "Wow, BOB LUTZ made my ass ... ?"
... the guys at the factory respond, "Yep. Sure did, pal ... "
... and that's when the car blows a protected head gasket that Bob had the engineers design into the process,
as if it were a person suffering a mini-stroke from the stress involved in being given such bad information.

Every single GM vehicle ever made, under BOB LUTZ leadership ... ?
... they've each endured this traumatic process ...

What makes it all actually work, is the specialized head gasket didn't COMPLETELY blow itself, but it will
remember the location where it ALMOST blew that gasket, where it'll eventually blow itself later on.

Remember how I said ALMOST only counts in horseshoes and handgrenades ... ?
Well, the engine ALMOST suffered permanent damage that THEY would've had to pay to fix.

Since it only ALMOST blew the engine, it'll DEFINITELY happen at some later point at about 100K miles.

While BOB LUTZ himself didn't make that car, the car was NOT impressed when it was told, that BOB LUTZ happened
to be the guy in charge of General Motors when it was marked complete at the factory.

That's ... all it takes.
The vehicle tries to commit suicide, but it's just a car, so it ends there ...

Luckily for BOB LUTZ, people never seem to catch on.
Why ... ? I really have no idea.

To analogize, it's like someone sitting in a room, farting repeatedly, and can't stop ...
And, people being impressed, not disgusted ...

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

People : Wow~!
         points This dude keeps rippin' farts~!
         That's awesome~!

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

These aren't your standard-issue farts either. Nah. These farts ... ?
Saying that they're WAY more impressive ... doesn't do them justice.
Even Barack Obama will chime in ... impressed by such consistently, above-average fartknockery.
Though I'm not fooled at all ... ? Everyone says:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Everyone : That's just your opinion dude ...
           Get over yourself~!

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

As for Elon Musk, this (1), single, non-plural dude, I'd imagine that he didn't think the chain of nasty farts in rapid succession, was anything to be all that impressed with either.

How could he be … ? Dude made the Tesla Model S Plaid, Starship, and just bought Twitter.
So, if he knows something is up, then he and I both know what the hell is going on.

He and I are both EQUALLY perturbed by this disgusting stench.
If anything ? Elon probably could tell that there's a medical emergency goin' on.

He and I, just … (2) like-minded individuals that both feel a threat looming from the other side of the room where BOB LUTZ is endlessly rippin some friggen farts.

When I knew something wasn't right … ? All of these people were saying:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Everyone : You need to grow up bro~!
           You're the only one smelling these 'farts'.

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

But- that's how the "grouped dementia" got em all. They don't know whats happening, but Elon Musk and I sure as hell do. If Elon Musk and I see the same exact problem that nobody else does … ? Well, great minds think alike.

That problem, is BOB LUTZ rippin' an endless array of the nastiest farts any man ever smelled, and this process went on for about 50 years straight over at GM Headquarters. Right under people's noses, too.

I came into the world being very well aware of my surroundings …
Bob thought he was being super slick.
Wasn't that slick at all.
I knew that somebody, somewhere would also be able to detect the problem …

That problem being, BOB LUTZ, showing up on the news, and saying:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Lutz : Hey Elon Musk, how about you show us mere mortals how to run a car company.

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

… after this man shit his pants for over 50 years straight at GM Headquarters.
Maybe it wasn't 50 years straight. Coud've been 30. Or 40.
Maybe he never actually shit his pants at all …
Maybe BOB LUTZ *started* shitting his pants quite *regularly* every day AFTER he went on the news …
Who the hell really knows … ?

Whatever the case may be, I cannot help but envision the fact, that this man has spent more time attempting to build many electric vehicles at GM, than Elon Musk or Tesla has. But, Bob never had the incentive, nor the talent to do it well.

Hence, total failure each time he tried.
Rinse, recycle, repeat … Maybe Bob should take his own advice. Ya know … ?

As for Elon Musk, the man did a lot more than just show BOB LUTZ how a car company could be run, at least, whenever someone with talent decides to take a break from running several other companies, and then show him how to do just that.

But, it's as if this one guy has many talents, and I hate to be the bearer of news …
But, unlike Bob's 50 years of consistently above-average fartknockery … ? This dude:

| CONCEPTUALIZED | FINANCED | TESTED | INVENTED |

… refillable rockets that can land themselves safely without exploding.
That's more difficult than anything BOB LUTZ did for over 50 years at GM Headquarters.

By the way, the newest rocket this dude made, it ALSO happens to be 25ish stories high, and meets the requirements of "first reuseable flying building". NASA never even did that. Did BOB LUTZ do that … ? No.
Not in the first 20 years he ran GM, nor in the entire 50. That right there, says quite a lot.

The fact of the matter is that Bob had a lot more time and experience to set a precedent, but- sometimes people think that doing the bare minimum for 50 years is 'leading by example'. Other people like me might hear or read that, and just start laughing profusely. Not laughing cause the man was trying to be funny, but laughing cause he was trying to be serious.

That'll get ya each and every time.
Dude actually had the nerve to say "Why don't you come down and show us mortals how to run a car company … ?"
And it was at a point in time where BOB LUTZ went from a credible person to so many many people …
… to an absolute laughingstock that shit his pants for 50 years straight. And, that's actually pretty sad.
Bob probably cannot help the fact that he does that … So … what can anybody do … ?

Bob didn't realize that those words would come back to bite him in the ass.
Here's how …

If anyone actually sat down and carefully studied what Elon Musk has accomplished on paper alone, it's easily 20 times the total amount of things BOB LUTZ did in the 50 years he ran a company, one that's ALMOST made a good vehicle millions of times over.

They know how to make good vehicles ... they just prefer not to do that.
They prefer to suck ass at making cars, and nobody knows why.

And, that's actually ok.
Some people like GM and BOB LUTZ, they accept their limitations, and move on with their life.
And, BOB LUTZ isn't the only one out there in many American industries that prefer to rest on their laurels.

BOB LUTZ is one of those people through and through, instead of finally making some decent vehicles that could finally be considered GOOD ? This dude never realized that he ALMOST made a lot of really nice vehicles.

The problem being, that NONE of them ever made it past the 0.99999 part of the 1.0 you'd need, to have made so many decent cars. BOB LUTZ was the director of General Motors for like, 50 years.

GM's made many attempts to build an (electric/gas) vehicle worth a damn.
They can't do it successfully without being FORCED to by TESLA.

Not only have they made so many more EV's than Tesla, but also, all of their gas powered cars too.
A LOT MORE of both. So, it stands to reason that some guy like Bob with a lot of experience wasn't worthy of a response from Elon.

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Musk : Hey, how many electric cars have you tried to build, over the last 50 years ... ?

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Probably at least 40. They didn't have this conversation though. Wanna know why ... ?

Cause Elon would rather one-up BOB LUTZ in the form of using a SpaceX rocket, to launch a Tesla Roadster into space, with StarMan inside ... and that whole process, being the biggest middle finger, to any man throughout history ... ... aimed squarely at BOB LUTZ.

Now, I don't know if BOB LUTZ watches stuff like the news or whatever ... but Elon Musk sent a god damn electric car into space. In his OTHER company's ROCKET, that LANDS ITSELF ...

| The Adventures of Starman | https://spacein3d.com/where-is-starman-live-tracker |

Anybody can go to that link up above and FIND OUT FOR THEMSELVES "Where is Starman ... ?"

I'm gonna repeat what Starman, that website, Tesla, and SpaceX's combined success, means for BOB LUTZ ...
This (1) dude is running like (5)+ companies at the same time, and literally sent an electric vehicle that was actually well built, into space ... using a well built rocket that his OTHER company made ... to give BOB LUTZ the biggest middle finger in human history.

Here's what Bob has to be proud of. Causing guys like me to think that sometimes, old people don't realize how outmatched they are. Even if they were an industry leader for 50 years ... ? It actually just translates to the embarassment factor being even LARGER.

So, the idea of purchasing a vehicle from General Motors, brand new, with a bumper-to-bumper warranty, its a total waste of anyone's money. Why ... ? BOB LUTZ created a lasting company culture that sucks ass at making good cars. Because, even if you were to buy a BRAND NEW car from GM, right now ... ? Chances are, the engine and the transmission might both break down a few thousand miles after that warranty is up.

Even if you bring it to the people at the dealership. Tell em: "Hey. My car is making noises ... "

Well, the ASE certified technicians might be good for (1) potential day out of (5), so, the certified techs might just be there for show, not to do actual work. Because the other (4) days of the week, they may prefer to lie to the customers about any warranty work on these vehicles. That makes things easier and a lot cheaper for them ...

They got paid to provide a warranty, but ... they reserve the right to not service the vehicle if they choose.
Maybe the customer should be told about that and like, get their friggen money back so actual experts can service the vehicle. But, then they start hemming/hawing, not realizing how moronic they sound.

Then, after the engine and the transmission break down, you have to hire an actual mechanic that knows what they're doing, in order to fix it. Which means, spend another $10-15 grand on a car that the dealership "fixed".

But- if they say they "fixed" it, and it still has the same problems from BEFORE you dropped it off ... ?
Maybe you'll start to see the "air quotes" more readily when people talk.

Simply put, I don't believe that GM is capable of making a single vehicle that isn't a total waste of money.
And, I definitely blame BOB LUTZ 100%. Maybe even Bill Cass. From what I've observed, General Motors lacks:

| TALENT | GOOD ENGINEERS | ACTUAL MECHANICS |

They sure as hell APPEAR to have/be PORTIONS of those things, but the CONSISTENCY is SUBSTANDARD.
Because, if they did ... ?
Anybody could buy a brand new car from them and expect that they'll honor the warranty/repair it.

After owning (8) vehicles that General Motors made, the verdict is in.
They are ALL terribly made. Even the CADILLACS BURN OIL.

That is the gods honest truth, the people at GM don't know how to make GOOD vehicles.
It's not the fault of the engineers, they probably know what they're doing ...
It's not the fault of the executives, they might know what they're doing too ...
It's not the fault of the technicians that build the cars, nor the dealers ...

It literally, all boils down to the craftsmanship of the PARTS they use.
That's (Start → Finish), they are all fucking terribly made.

They probably have made quality parts and vehicles at SOME point in time ... ?
But, anyone may as well go ahead and say that General Motors never has, and they never will.
They've had like a 100 year head start over Tesla.

But, they didn't realize the clock was ticking, and it's (too bad/so sad).

They know how to make SOME of them LOOK good, but a car has to do more than just LOOK good ...
... to be considered a good product by actual intelligent people.

These are all of the things I think about when I hear the name "BOB LUTZ".

```
 ________________________________________________________________________________________
| Spend all your money | get a terrible product | they'll get a Bailout check from Obama |
 ----------------------------------------------------------------------------------------
```

No problem.
I wouldn't feel a need to call out a large conglomerate corporation if it had people who knew how to reliably build parts for their car correctly. But, they are terrible at it.

I know I'm coming across very differently than I have throughout the rest of this document, but sometimes people have done more than enough to deserve being openly insulted.

Just like how peanut butter goes hand in hand with jelly ... ?
BOB LUTZ goes hand in hand with "not very innovative or creative", or "sucks at making cars".
The legacy of this man, will be remembered for ETERNITY ... with (1) word.
"Almost."

With CSharp and PowerShell ... ?
You're not ALMOST gettin' the best there is in the industry ... Nah.
You're DEFINITELY gettin' that, and that's all there is to it.

```
                                                                                                          ________/
\________________________________________________________________________________________________________/ Almost
  Back to the Lesson Plan /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

The tangent I just went on, caused me to begin writing Top Deck Awareness - Not News.
But, I'm gonna finish out this lesson plan.

The $Prop variable is an array, so $Prop[0] indicates that I'm selecting the FIRST item in that array.

```
 ____________________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
PS Prompt:\> $Prop[0]

Value           : Microsoft Edge
MemberType      : NoteProperty
IsSettable      : True
IsGettable      : True
TypeNameOfValue : System.String
Name            : DisplayName
IsInstance      : True
____________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

In this instance, we don't care about the Membertype, IsSettable, IsGettable, or IsInstance properties ... but we'll adhere to those values anyway if we use the underlying base type.

So, like I covered before the huge tangent, we can use "[PSNoteProperty]::New($Name, $Value)" to create an object that adheres to PSObject.Properties, and this will ADD a NEW custom property named "EntryUnique" to the custom classes property list, and then it'll cast an empty object array to its value.

This will allow one of the methods we have to write, to return NON-DEFAULT properties to it.
Doing so adheres to the standard class properties, while allowing additional NON-DEFAULT entries to coexist peacefully, and be clean and accessible.

Who doesn't like that idea ... ?

```
 ____________________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
$Prop   += [PSNoteProperty]::New("EntryUnique",@( ))
 _________
| or ... |
 ---------
$Prop   += New-Object PSNoteProperty -ArgumentList EntryUnique, @( )
```

```
_________
| or ... |
---------

$Prop   += New-Object PSNoteProperty EntryUnique, @( )
```

```
We want the class to format itself with the right spacing.
So, to do that, we need to get the maximum TypeNameOfValue string length.
[Again], here's some of my "Voodoo 3 5000" action BELOW, being applied to the $Types variable. (Explained again)

It's a multifaceted ONE-LINER involving:
- ForEach-Object haphazardly piping itself into an array
- $_ token with the property length being greater than 0 in square brackets acts as a switch
- $False selects slot 0 in the array returning the string "String", cause that's binary for ya.
- $True selects slot 1 in the array returning ($_ -Replace "System\.","")

Now, PowerShell does an amazing job of being able to understand when it's dealing with a default system types
like [System.Object], or [System.String]. So, the word System being thrown all over the place is unnecessary.
Removing it makes the code shorter and less complicated/messy looking.
```

```
$Types           = ($Prop.TypeNameOfValue | % { @("String",$_ -Replace "System\.","")[$_.Length -gt 0] })
$TypesMaxLength  = ($Types | Sort-Object Length)[-1].Length
```

```
Now, get maximum Name string length. Also, I'm sorry, but the Voodoo 3 5000 action is all over.
```

```
$Names           = $Prop.Name
$NamesMaxLength  = ($Names | Sort-Object Length)[-1].Length
```

```
Declare a hash table with
1) CLASSNAME,
2) PROPERTY TYPES/NAMES (top portion of class),
3) PARAM TYPE+VALUE (main method),
4) DEFAULT CONSTRUCTOR DEFINITIONS (main portion of the class), and
5) METHODS for self-rereferencing, brevity, processing each individual NON-DEFAULT property,
   as well as writing some output.
```

```
$Def            = @{

    Name        = "Uninstall";
    Type        = @( );
    Param1Type  = "[Object]";
    Param1Value = "`$Registry";
    Const       = @( );
    Method      = @( )
}
```

```
Run through all types, property names, and establish the code to set the class values to the properties of the
(input object/parameter). Add each TYPE+NAME to $Def.Type array, and then each $Name in $Names with spacing for
the $Def.Const array in the same loop.
```

```
ForEach ($X in 0..($Names.Count-1))
{
    $Type        = $Types[$X]
    $Name        = $Names[$X]
    $TypeBuffer  = " " * ($TypesMaxLength - $Type.Length + 1)
    $NameBuffer  = " " * ($NamesMaxLength - $Name.Length + 1)
    $Def.Type   += "    [{0}]{1}{2}`${3}" -f $Type , $TypeBuffer, $NameBuffer, $Name
    $Def.Const  += "        `$This.{0}{1} = {2}.{0}" -f $Name, $NameBuffer, $Def.Param1Value
}
```

```
Get the rank of the line where it matches the PSNoteProperty we added, EntryUnique, then replace.
```

```
$X               = 0..($Def.Const.Count-1) | ? { $Def.Const[$_] -match "EntryUnique" }
$Def.Const[$X]   = $Def.Const[$X] -Replace '= .+', '= $This.GetEntryUniqueProperties($Registry)'
```

```
 ____________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ----------------------------------------------------------------------------------------------------
```

Now we have to write methods (Chunked out for readability). This FIRST method will shorten the process of calling the DEAFULT property names in this class, whereby filtering out "EntryUnique"

```
 ____________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
----------------------------------------------------------------------------------------------------
$Method1         = @(
'    [String[]] Properties()',
'    {',
'        Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name',
'    }'
)
$Def.Method      += $Method1
 ____________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ----------------------------------------------------------------------------------------------------
```

This second method will return the properties of the base class that aren't standard property names that we pulled from the $Edge object template.

```
 ____________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
----------------------------------------------------------------------------------------------------
$Method2          = @(
'    [Object[]] GetEntryUniqueProperties([Object]$Param)',
'    {',
'        Return @(
$Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") | Select Name, Value'
'    }'
)
$Def.Method      += $Method2
 ____________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ----------------------------------------------------------------------------------------------------
```

Now we will create a way for the extended properties to show themselves in a way that is consumable, while still adhering to the default properties.

```
 ____________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
----------------------------------------------------------------------------------------------------
$Method3           = @(
'    [Object[]] Output([UInt32]$Buffer)',
'    {',
'        $Output  = @( )',
'        $Output += ("-" * 120 -join "")',
'        $Output += "[$($This.DisplayName)]"',
'        $Output += ("-" * 120 -join "")',
'        $Output += " "',
'',
'        $This.Properties() | % { ',
'',
'            $Output += "{0}{1} : {2}" -f $_, (" " * ($Buffer - $_.Length + 1) -join ""), $This.$_',
'        }',
'',
'        $Output += (" " * $Buffer -join "")',
'        $This.EntryUnique  | % { ',
'',
'            $Output += "{0}{1} : {2}" -f $_.Name, (" " * ($Buffer - $_.Name.Length + 1 ) -join ""), $_.Value',
'        }',
'        $Output += (" " * $Buffer -join "")',
'',
'        Return $Output',
'    }'
)
$Def.Method      += $Method3
 ____________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ----------------------------------------------------------------------------------------------------
```

Create the Class definition value, this joins together the multiple chunks of the class so that it can be instantiated by the PowerShell (Type/Class) engine.

```
 ____________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
----------------------------------------------------------------------------------------------------
$ClassDefinition = @("Class $($Def.Name)","{",($Def.Type -join "`n"),
                     "    $($Def.Name)($($Def.Param1Type)$   ($Def.Param1Value))",
                     "    {",
                     ($Def.Const -join "`n"),
                     "    }",
                     ( $Def.Method -join "`n"),
                     "}") -join "`n"
 ____________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ----------------------------------------------------------------------------------------------------
```

Instantiate the class definition - Using this command below instantiates the class without having to explicitly

get the variable output.

```
 ____________________________________________________________________________________________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
----------------------------------------------------------------------------------------------------

Invoke-Expression $ClassDefinition
 ____________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
 ----------------------------------------------------------------------------------------------------

Alright, now declare a variable that collects all of the $Apps objects in the registry paths like the pros
suggested up above. Then, look for the longest NON-DEFAULT property name length, and then format ALL of the
classes with that integer to get a steady stream of formatted output.
 ____________________________________________________________________________________________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
----------------------------------------------------------------------------------------------------

$Output  = $Apps | % { [Uninstall]::New($_) }
$Buffer  = ($Output.EntryUnique.Name | Sort-Object Length)[-1].Length
$Output.Output($Buffer)
 ____________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
 ----------------------------------------------------------------------------------------------------

We should probably use an exterior container class to collect all of these items and format them accordingly.
This will include all variables we assigned BEFORE the class definition, and then there will be NO chance, that
an object will be formatted with an inconsistent buffer value.
 ____________________________________________________________________________________________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
----------------------------------------------------------------------------------------------------

Class UninstallStack
{
    [UInt32] $Buffer
    [Object] $Output
    UninstallStack()
    {
        # // We can use some of these variables without assigning them to the class.
        # // This optimizes the System.IO stream since it's not tugging along unnecessary data...
        # // and does automatic garbage cleanup

        # // Apps found in the uninstall registry paths (64-bit/32-bit)
        $Apps        = "\Wow6432Node","" | % {

        GCI "HKLM:\Software$_\Microsoft\Windows\CurrentVersion\Uninstall"
        } | Get-ItemProperty
        # // Pulls MSI object template Edge/Chrome installation
        $Edge        = $Apps | ? DisplayName -match "(^Microsoft Edge$)"
        # // Ignores WMI/PS related properties
        $Prop  = $Edge.PSObject.Properties | ? Name -notmatch "(^_|^PS)"
        # // Add property to include within format
        $Prop += [PSNoteProperty]::New("EntryUnique",@( ))
        # // Allow the Uninstall class to be instantiated via the above work
        # // We will make a class that integrates ALL of these components, soon.
        $This.Output     = $Apps | % { [Uninstall]::New($_) }
        $This.Buffer     = ($This.Output.EntryUnique.Name | Sort-Object Length)[-1].Length
    }
    [Object[]] GetOutput()
    {
        Return @( $This.Output.Output($This.Buffer) )
    }
}
 ____________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
 ----------------------------------------------------------------------------------------------------

Now, cast the variable with UninstallStack, which is the class up above.
This will automatically do the same things that all of those separate variables were able to do.
 ____________________________________________________________________________________________________
/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\
----------------------------------------------------------------------------------------------------

$Output = [UninstallStack]::New()
$Output.GetOutput()
 ____________________________________________________________________________________________________
\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/__\__/
 ----------------------------------------------------------------------------------------------------

You won't want to format the output as a table, but even if you do, everything in the NON-DEFAULT properties will
be caught within an array. There's really no way to get inconsistent properties to work across a bunch of entries
in a table that have varying property types, unless maybe you have a monitor that spans about a football field,
then I guess maybe that might actually work at fitting all of the possible properties that any class might have,
into an incredibly convenient view.

But, I gotta say... I don't think those types of monitors actually exist yet.
Then again, maybe somebody who's been working on just that exact thing I said doesn't exist...?
Well... they just heard me loudly and clearly, and so they felt like letting me know...

FFMD = Football-Field-Monitor-Dude
```

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

FFMD : Look buddy ...
       Those things DO exist.
Me   : Oh yeh ... ?
FFMD : Yeh.
       And you call yourself some type of expert or something ... ?
       scoffs, eyebrows up Unbelievable ...
Me   : I didn't know they did.
FFMD : Yeah pal.
       They most certainly do.
       Everybody knows that ...
Me   : I didn't know that they actually made a monitor that spans a whole entire football field ...
FFMD : Yeah well ...
       ... now you know.
       What, have you been livin' under a rock or somethin ... ?
Me   : Nah.
FFMD : Well buddy ... nice tutorial ...
       But I'm offended about the monitor thing.
Me   : Alright ... ?
       I'm sorry ... ?
FFMD : Wait ...
       ... you're ...
       ... sorry ... ?
Me   : Yeah man, didn't realize I offended you by not knowing a monitor that long actually exists ...
FFMD : Yeah, they do.
       You should totally get one, very convenient, you can see the entire screen no problem ...
Me   : Alright buddy.
       Duly noted.
       Well, I still have one more phase of this tutorial left to go.
FFMD : Oh yeah ... ?
Me   : Yep.
       Gonna throw all of that stuff into a class that generates ... classes.
FFMD : A class that generates classes ... ?
       What are ya, some type of magician or somethin'?
Me   : No ...
FFMD : Buddy, you've got a lot of tricks.
Me   : Well, cool.
       I appreciate that.
FFMD : I guess I'll stick around and keep reading.
Me   : Alright fine.
       Take care, buddy.
FFMD : You too. Get one of those monitors ...

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Those types of monitors don't exist.
I was being facetious about the convenient viewing angle.

Even if they did make them, what practical purpose would anyone use them for ... ?
There's only (1) thing I can think of ... to join a bunch of classes that have no matching properties at all.

It's not unlike buying an Nvidia GTX 3090, so you could play Pogo.com and then brag about it.

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Person : Check it out ...
         points I've got an Nvidia GTX 3090.
         NOW, I am basically unbeatable at Pop-It.
Me   : Wow.
         Impressive.
Person : *chuckles* Heh.
         If you don' have a GTX 3090 to play Pop-It ... ?
         Who the hell even are ya ... ?

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

(^ Pretty sure that's overkill, and that graphics card doesn't do anything for that game/purpose)

Or, taking your paycheck, and then just throwing it in the trash because ... why not ... ?

These things probably sound like they make no sense at all, and that's the point.
That's how much sense it makes to try and use classes that have no matching properties at all.
You won't really have a hard time NOT noticing when they don't match up, either.
You'll see empty cells, and values that extend about 50 miles into the horizon.

Having SOME varying property types from class to class is ok, but try to MINIMIZE that.
Allowing just any old class to settle down in the table ... ?
You're just asking to be confused.

You might find yourself staring at the monitor trying to make sense of what you're seeing ...
But, the truth is, there's no sense or pattern involved after some point.

So, maybe you'll get confused that none of the classes match up in the table anymore ...
Then what ... ?
Start over ... ?

Well, I'll tell ya.
If the properties don't match up, then there's not a whole lot you can do.

It's gonna be a while before anyone is able to readily make use of such an incredibly useful arrangement of class types that USED to have matching properties ... until one day they didn't.

That's just the way the story goes, of every old man that ever lived, who, at the top of their game ... ?
He made certain that HIS classes always had matching properties.
... until he slipped up.

Of course, I'm shooting for dramatic story telling.
But, one way that we could turn the entire script into a useful tool that would help anybody custom build their own classes in a jiffy, is the script. Couldn't be more serious about that actually.

Sometimes I build custom (classes/structs) in either PowerShell or C#, and then I am able to cast some of those classes/structs to PowerShell/.Net type objects, and they work in literally the same way.

The only difference is that the PowerShell code doesn't need to be compiled by MSBuild or anything like that.
That's because PowerShell can compile CSharp code on the fly. It's not all-encompassing like MSBuild is, sometimes CSharp code that works in an MSBuild process doesn't work with the Add-Type cmdlet ...

I think it's just an issue with Roslyn or something. However, writing a bunch of classes/structs in CSharp and then initializing and instantiating them into PowerShell is incredibly useful.
It's the only way to get structs into PowerShell, I think.

What I'm going to do NOW, is, build a class that puts all of this stuff together.
I'll start by stripping away the comments that I made on some of the above content, and then edit it until it's literally picture perfect, and reproduces the same output as above.

| Here are ALL of the above variables thrown into a single section, as well as the page references |

```
# [Page 22]
$Apps            = "\Wow6432Node","" | % {

                     Get-ChildItem "HKLM:\Software$_\Microsoft\Windows\CurrentVersion\Uninstall"

                   } | Get-ItemProperty
# [Page 24]
$Edge            = $Apps | ? DisplayName -match "(^Microsoft Edge$)"

# [Page 25 → TANGENT → Page 51]
$Prop      = $Edge.PSObject.Properties | ? Name -notmatch "(^_|^PS)"
$Prop     += [PSNoteProperty]::New("EntryUnique",@( ))

# [Page 52]
$Types           = ($Properties.TypeNameOfValue | % { @("String",$_ -Replace "System\.","")[$_.Length -gt 0] })
$TypesMaxLength  = ($Types | Sort-Object Length)[-1].Length

# [Page 52]
$Names           = $Properties.Name
$NamesMaxLength  = ($Names | Sort-Object Length)[-1].Length

# [Page 52]
$Def             = @{
    Name         = "Uninstall"
    Type         = @( )
    Param1Type   = "[Object]"
    Param1Value  = "`$Registry"
    Const        = @( )
    Method       = @( )
}

# [Page 52]
ForEach ($X in 0..($Names.Count-1))
{
    $Type        = $Types[$X]
    $Name        = $Names[$X]
    $TypeBuffer  = " " * ($TypesMaxLength - $Type.Length + 1)
    $NameBuffer  = " " * ($NamesMaxLength - $Name.Length + 1)
    $Def.Type   += "    [{0}]{1}{2}`${3}" -f $Type , $TypeBuffer, $NameBuffer, $Name
    $Def.Const  += "        `$This.{0}{1} = {2}.{0}" -f $Name, $NameBuffer, $Def.Param1Value
}

# [Page 53]
$X               = 0..($Def.Const.Count-1) | ? { $Def.Const[$_] -match "EntryUnique" }
$Def.Const[$X]   = $Def.Const[$X] -Replace '= .+', '= $This.GetEntryUniqueProperties($Registry)'
```

```
    # [Page 53]
    $Method1         = @(
    '    [String[]] Properties()',
    '    {',
    '        Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name',
    '    }'
    )
    $Def.Method      += $Method1

    # [Page 53]
    $Method2          = @(
    '    [Object[]] GetEntryUniqueProperties([Object]$Param)',
    '    {',
    '        Return @(
         $Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") | Select Name, Value'
    '    }'
    )
    $Def.Method      += $Method2

    # [Page 53]
    $Method3           = @(
    '    [Object[]] Output([UInt32]$Buffer)',
    '    {',
    '        $Output  = @( )',
    '        $Output += ("-" * 120 -join "")',
    '        $Output += "[$($This.DisplayName)]"',
    '        $Output += ("-" * 120 -join "")',
    '        $Output += " "',
    '',
    '        $This.Properties() | % { ',
    '',
    '            $Output += "{0}{1} : {2}" -f $_, (" " * ($Buffer - $_.Length + 1) -join ""), $This.$_',
    '        }',
    '',
    '        $Output += (" " * $Buffer -join "")',
    '        $This.EntryUnique  | % { ',
    '',
    '            $Output += "{0}{1} : {2}" -f $_.Name, (" " * ($Buffer - $_.Name.Length + 1 ) -join ""), $_.Value',
    '        }',
    '        $Output += (" " * $Buffer -join "")',
    '',
    '        Return $Output',
    '    }'
    )
    $Def.Method      += $Method3

    # [Page 53]
    $ClassDefinition = @("Class $($Def.Name)","{",($Def.Type -join "`n"),
                         "    $($Def.Name)($($Def.Param1Type)$   ($Def.Param1Value))",
                         "    {",
                         ($Def.Const -join "`n"),
                         "    }",
                         ( $Def.Method -join "`n"),
                         "}") -join "`n"
```

_____/ Back to the Lesson Plan

<Class Definition> Template

Now, I'll develop these above variables as (2) separate classes, one for the container, one for the class object, and the class object needs to be written FIRST so that the type can be used in the container class.

Even though the variables are right ABOVE, they will LIKELY have to be rearranged in order to work as a pair of classes, though not necessarily.

And though, reading the material will feel like the above paragraph was SOOOO 30 seconds ago … ?
The truth is that when this was written, I spent about a half hour working on it and tweaking it.

The NAME of this class, is essentially "DefinitionTemplate", because the name of this class doesn't NEED to be COOL, because it's job is to produce a class that includes various elements that would be needed to perform the same exact activity as the variables listed directly above.

However, this class has implemented many changes to the code it's based on up above, while still producing the same output. Some of those changes are details that wouldn't even be seen when using it. But, if you were to debug what it does, you'd see how it molds and shapes the output, and you'd easily see how OTHER techniques were applied, to produce the same result.

Sometimes this process can make the code longer, but other times it can make the code much more responsive or even give it more features so other elements can be added/amended whereby boosting flexibility, capability, or scope.

The fully written class is below, as is.
AFTER this stint, I will break it down and explain the differences between the variables above, and the properties and variables in the class.

```
_____________________________________________________________________________
| Note : This class is just a current derivative, if I were to keep working on it, I would |
| implement various changes to more finely tune the capabilities, properties, values, etc. |
|-------------------------------------------------------------------------------------------|
| Some formatting is being used that I wouldn't normally use due to PUBLISHING CONSTRAINTS |_____________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-------------------------------------------------------------------------------------------------------------------

Class DefinitionTemplate
{
    [String] $Name
    [Object] $Property
    [Object] $Param1Type
    [Object] $Param1Value
    [Object] $Constructor
    [Object] $Method
    DefinitionTemplate([String]$Name,[String]$Param1Type,[String]$Param1Value)
    {
        $This.Name             = $Name
        $This.Property         = @( )
        $This.Param1Type       = $Param1Type
        $This.Param1Value      = $Param1Value
        $This.Constructor      = @( )
        $This.Method           = @( )
    }
    LoadPropertySet([Object[]]$Properties)
    {
        $Types                 = ($Properties.TypeNameOfValue | % {

                                 @("String",$_ -Replace "System\.","")[$_.Length -gt 0]

                                 })

        $TypesMax              = ($Types | Sort-Object Length)[-1].Length
        $Names                 = $Properties.Name
        $NamesMax              = ($Names | Sort-Object Length)[-1].Length
        ForEach ($X in 0..($Names.Count-1))
        {
            $TypeBuff          = " " * ($TypesMax - $Types[$X].Length + 1)
            $NameBuff          = " " * ($NamesMax - $Names[$X].Length + 1)
            $This.AddProperty($Types[$X],$TypeBuff, $NameBuff,$Names[$X])
            $This.SetProperty($Names[$X],$NameBuff)
        }
    }
    AddProperty([String]$Type,[String]$TypeBuff,[String]$NameBuff,[String]$Name)
    {
        $This.Property        += "    [{0}]{1}{2}`${3}" -f $Type, $TypeBuff, $NameBuff, $Name
    }
    SetProperty([String]$Property,[String]$NameBuff)
    {
        $This.Constructor     += "        `$This.{0}{1} = {2}.{0}" -f $Property,
                                                                      $NameBuff,
                                                                      $This.Param1Value
    }
    ChangeProperty([String]$Name,[String]$Value)
    {
        $X                     = 0..($This.Constructor.Count-1) | ? {

                                 $This.Constructor[$_] -match $This.Escape("`$This.$Name")

                                 }
        If (!!$X)
        {
            [Console]::WriteLine("Property [+] Found, altering ... ")
            $This.Constructor[$X]  = $This.Constructor[$X] -Replace "=.+","= $Value"
        }
        Else
        {
            [Console]::WriteLine("Property [!] Not found, skipping ... ")
        }
    }
    [String] AddIndent([UInt32]$Count)
    {
        Return "    " * $Count
    }
    AddMethod([String[]]$Body)
    {
        $Body   = $Body -Replace "^\s*",""
        $I      = 0
        $Return = @( )
        ForEach ($X in 0..($Body.Count-1))
```

```
        {
            $Line = $Body[$X]
            If ($X -eq 0 -and $Line -match "(\s{0})")
            {
                $I ++
                $Line = $This.AddIndent($I) + $Line
            }
            ElseIf ($X -eq 1 -and $Line -match "(\s{0}\{)")
            {
                $Line = $This.AddIndent($I) + $Line
                $I ++
            }
            ElseIf ($X -eq ($Body.Count-1) -and $Line -match "(\s{0}\})")
            {
                $I --
                $Line = $This.AddIndent($I) + $Line
            }
            Else
            {
                $Line = $This.AddIndent($I) + $Line
            }
            $Return += $Line
        }
        $This.Method += $Return -join "`n"
    }
    [String] Escape([String]$Value)
    {
        Return [Regex]::Escape($Value)
    }
    [String] ReturnDefinition()
    {
        $X              = @{
            Name        = "Class {0}" -f $This.Name
            Property    = $This.Property -join "`n"
            Main        = "    {0}({1}{2})" -f $This.Name,
                                               $This.Param1Type,
                                               $this.Param1Value
            Constructor = $This.Constructor -join "`n"
            Method      = $This.Method -join "`n"
        }
        Return @( $X.Name,
                  "{",
                  $X.Property,
                  $X.Main,
                  "    {",
                  $X.Constructor,
                  "    }",
                  $X.Method,
                  "}"
                ) -join "`n"
    }
}
_____________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ---------------------------------------------------------------------------------------------------------

_____________________________________________________________________________________________
| Now that the class is declared, instantiate the class and use its methods to reproduce the result   |
| THEN, load the properties by using the method LoadPropertySet($Properties)                          |_______
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-----------------------------------------------------------------------------------------------------------

$Temp           = [DefinitionTemplate]::New("Uninstall","[Object]",'$Registry')
$Temp.LoadPropertySet($Properties)
_____________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ---------------------------------------------------------------------------------------------------------

_____________________________________________________________________________________________
| Now change property named "EntryUnique", new value "$This.GetEntryUniqueProperties($Registry)" |_____________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-----------------------------------------------------------------------------------------------------------

$Temp.ChangeProperty("EntryUnique","`$This.GetEntryUniqueProperties(`$Registry)")
_____________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ---------------------------------------------------------------------------------------------------------

_____________
| Now add method #1 |_____________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
-----------------------------------------------------------------------------------------------------------

$Temp.AddMethod(@('[String[]] Properties()','{',
'Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name','}'))
_____________________________________________________________________________________________________________
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ---------------------------------------------------------------------------------------------------------

____________________
```

```
| Now add method #2 |___________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
----------------------------------------------------------------------------------------------------------------

$Temp.AddMethod(@('[Object[]] GetEntryUniqueProperties([Object]$Param)',
'{',
('Return @($Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") '+
 '| Select-Object Name, Value'),'}'))
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 --------------------------------------------------------------------------------------------------------------
 __________________________________________________________________________________________________
| Now add method #3                                                                                |
| There is some formatting being used that I wouldn't normally use due to PUBLISHING CONSTRAINTS |_____________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
----------------------------------------------------------------------------------------------------------------

$Temp.AddMethod(@('[Object[]] Output([UInt32]$Buff)','{','$X  = @( )',
'$X += ("-" * 120 -join ""), "[$($This.DisplayName)]", ("-" * 120 -join ""), " "',
'$This.Properties() | % { $X += "{0}{1} : {2}" -f $_, (" " * ($Buff - $_.Length + 1) -join ""), $This.$_ }',
'$X += (" " * $Buff -join "")',
('$This.EntryUnique  | % { $X += "{0}{1} : {2}" -f $_.Name,'+
 ' (" " * ($Buff - $_.Name.Length + 1 ) -join ""), $_.Value }'),
'$X += (" " * $Buff -join "")',
'Return $X','}'))
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 --------------------------------------------------------------------------------------------------------------
 ________________________________________________________________________________________
| Run the method ReturnDefinition(), and cast it's output to variable $ClassDefinition |_______________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
----------------------------------------------------------------------------------------------------------------

$ClassDefinition = $Temp.ReturnDefinition()
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 --------------------------------------------------------------------------------------------------------------
 ____________________________________________________________________________________________
| Copy the variable output to the clipboard, and THEN, let's take a look at the output ...   |__________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
----------------------------------------------------------------------------------------------------------------

$ClassDefinition | Set-Clipboard
                                                                                         _____________________________/
\_______________________________________________________________________________________/ <Class Definition> Template
  Output /¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯\
/¯¯¯¯¯¯¯¯

Class Uninstall
{
    [String]        $DisplayName
    [String]     $DisplayVersion
    [String]            $Version
    [Int32]            $NoRemove
    [String]         $ModifyPath
    [String]    $UninstallString
    [String]    $InstallLocation
    [String]        $DisplayIcon
    [Int32]            $NoRepair
    [String]          $Publisher
    [String]        $InstallDate
    [Int32]        $VersionMajor
    [Int32]        $VersionMinor
    [Object[]]      $EntryUnique
    Uninstall([Object]$Registry)
    {
        $This.DisplayName      = $Registry.DisplayName
        $This.DisplayVersion   = $Registry.DisplayVersion
        $This.Version          = $Registry.Version
        $This.NoRemove         = $Registry.NoRemove
        $This.ModifyPath       = $Registry.ModifyPath
        $This.UninstallString  = $Registry.UninstallString
        $This.InstallLocation  = $Registry.InstallLocation
        $This.DisplayIcon      = $Registry.DisplayIcon
        $This.NoRepair         = $Registry.NoRepair
        $This.Publisher        = $Registry.Publisher
        $This.InstallDate      = $Registry.InstallDate
        $This.VersionMajor     = $Registry.VersionMajor
        $This.VersionMinor     = $Registry.VersionMinor
        $This.EntryUnique      = $This.GetEntryUniqueProperties($Registry)
    }
    [String[]] Properties()
    {
        Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name
    }
    [Object[]] GetEntryUniqueProperties([Object]$Param)
    {
```

```
        Return @(
        $Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") | Select Name, Value
    }
    [Object[]] Output([UInt32]$Buff)
    {
        $X  = @( )
        $X += ("-" * 120 -join ""), "[$($This.DisplayName)]", ("-" * 120 -join ""), " "
        $This.Properties() | % {
        $X += "{0}{1} : {2}" -f $_, (" " * ($Buff - $_.Length + 1) -join ""), $This.$_
        }
        $X += (" " * $Buff -join "")
        $This.EntryUnique  | % {
        $X += "{0}{1} : {2}" -f $_.Name, (" " * ($Buff - $_.Name.Length + 1 ) -join ""), $_.Value
        }
        $X += (" " * $Buff -join "")
        Return $X
    }
}
```

```
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ------------------------------------------------------------------------------------------------------------------
```

So, the output does look slightly different.
At least the content of the method named Output does.

That's because I added a feature that would automatically indent lines within any given method fed to the AddMethod([Object[]]$Object) method. But, then I realized that probably isn't a great use of my time to go the full mile there.

There are many things in the .NET framework that can actually indent some of this stuff FOR you, or anybody else really... but sometimes, learning how to do it without a tool like that can help bolster one's ability to do it themselves. When they do, they'll understand how the "pros" went ahead, and wrote a class that does all of that work, cause they're probably a real nice (guy/girl).

Whether you use one of the StringWriter class types, or I think the Xmlwriter class also does it, can't think off the top of my head what other default classes automatically indent stuff for you, but they're out there.

Things become harder to do when you're writing CODE that does what you as a human would do when editing the content, and that's when design choices might change.

```
                                                                                                                  _______/
\_________________________________________________________________________________________________________________/ Output
  Comparison /-------------------------------------------------------------------------------------------------------------\
/------------
```

```
_____________________________________________________________________________________________________________________
| FIRST CLASS, [UninstallStack] |/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/|
---------------------------------------------------------------------------------------------------------------------

_____________________
| [Before : Part 1] |_____________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
---------------------------------------------------------------------------------------------------------------------

$Apps            = "\Wow6432Node","" | % {

                   Get-ChildItem "HKLM:\Software$_\Microsoft\Windows\CurrentVersion\Uninstall"

                   } | Get-ItemProperty
$Edge            = $Apps | ? DisplayName -match "(^Microsoft Edge$)"
$Properties      = $Edge.PSObject.Properties | ? Name -notmatch "(^_|^PS)"
$Properties     += [PSNoteProperty]::New("EntryUnique",@( ))
```

```
\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
 ------------------------------------------------------------------------------------------------------------------
```

The BEFORE stuff is actually all the variables starting from where the pros suggested that using Get-WMIObject Win32_Product to get installed applications, is not the best way to get that info.

Looking through the registry is actually a WAY snappier way to do the job, and any pro would tell you that.
The thing is, the registry and the WMI objects don't have the same values.

So, since Matthew Caldwell had us imaging THOUSANDS of machines incorrectly ... ?
Whereby forcing is to redo a LOT of work that we had to do OVER again ... ?

WELL ... we had to make some serious changes after writing the WMIClass at the top.
Not to mention, the WMIClass reaches into the HKEY_CLASSES_ROOT for GUID's and stuff that isn't automatically available to PowerShell without opening a new PSDrive to HKCR:*. Pretty sure anyway.

```
_____________________
| [After : Part 1] |_____________________________________________________________________________________________________
/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\
---------------------------------------------------------------------------------------------------------------------

Class UninstallStack
{
    [UInt32] $Buffer
    [Object] $Output
    UninstallStack()
    {
```

```
        $Apps           = "\Wow6432Node","" | % {

                          Get-ChildItem "HKLM:\Software$_\Microsoft\Windows\CurrentVersion\Uninstall"

                          } | Get-ItemProperty
        $Edge           = $Apps | ? DisplayName -match "(^Microsoft Edge$)"
        $Properties     = $Edge.PSObject.Properties | ? Name -notmatch "(^_|^PS)"
        $Properties    += [PSNoteProperty]::New("EntryUnique",@( ))
        $This.Output    = $Apps | % { [Uninstall]::New($_) }
        $This.Buffer    = ($This.Output.EntryUnique.Name | Sort-Object Length)[-1].Length
    }
    [Object[]] GetOutput()
    {
        Return @( $This.Output.Output($This.Buffer) )
    }
}
```

The AFTER stuff is essentially the same code, largely.
Except, there are (2) properties that replace a couple of the variables, like $This.Output and $This.Buffer.
The other variables are still written the same way.

You can get away with this for a large majority of the conversion, UNLESS, having a PROPERTY makes more sense.
Properties of the class will RETAIN the information, whereas variables will NOT, unless they're cast to a property.

SECOND CLASS, [DefinitionTemplate]

[Before : Part 1]

```
$Types          = ($Properties.TypeNameOfValue | % { @("String",$_ -Replace "System\.","")[$_.Length -gt 0] })
$TypesMaxLength = ($Types | Sort-Object Length)[-1].Length
$Names          = $Properties.Name
$NamesMaxLength = ($Names | Sort-Object Length)[-1].Length
$Def            = @{
    Name        = "Uninstall"
    Type        = @( )
    Param1Type  = "[Object]"
    Param1Value = "`$Registry"
    Const       = @( )
    Method      = @( )
}
ForEach ($X in 0..($Names.Count-1))
{
    $Type       = $Types[$X]
    $Name       = $Names[$X]
    $TypeBuffer = " " * ($TypesMaxLength - $Type.Length + 1)
    $NameBuffer = " " * ($NamesMaxLength - $Name.Length + 1)
    $Def.Type  += "    [{0}]{1}{2}`${3}" -f $Type , $TypeBuffer, $NameBuffer, $Name
    $Def.Const += "        `$This.{0}{1} = {2}.{0}" -f $Name, $NameBuffer, $Def.Param1Value
}
```

The Before stuff right here is doing a lot of the work in processing the values in the $Properties variable.
The $Properties variable is an object array of PSNoteProperties based off of the $PSObject.Properties variable.

The hash table is there reserving some collection containers for properties Type, Const, and Method.

Then the loop just goes right ahead and starts processing every item in the $Names array.
It is also calculating the length of the buffer strings so that it formats the code neatly.

[After : Part 2]

```
Class DefinitionTemplate
{
    [String] $Name
    [Object] $Property
    [Object] $Param1Type
    [Object] $Param1Value
    [Object] $Constructor
    [Object] $Method
    DefinitionTemplate([String]$Name,[String]$Param1Type,[String]$Param1Value)
    {
        $This.Name            = $Name
        $This.Property        = @( )
        $This.Param1Type      = $Param1Type
        $This.Param1Value     = $Param1Value
        $This.Constructor     = @( )
```

```
        $This.Method           = @( )
    }
    LoadPropertySet([Object[]]$Properties)
    {
        $Types                 = ($Properties.TypeNameOfValue | % {

                                 @("String",$_ -Replace "System\.","")[$_.Length -gt 0]

                                 })

        $TypesMax              = ($Types | Sort-Object Length)[-1].Length
        $Names                 = $Properties.Name
        $NamesMax              = ($Names | Sort-Object Length)[-1].Length
        ForEach ($X in 0..($Names.Count-1))
        {
            $TypeBuff          = " " * ($TypesMax - $Types[$X].Length + 1)
            $NameBuff          = " " * ($NamesMax - $Names[$X].Length + 1)
            $This.AddProperty($Types[$X],$TypeBuff, $NameBuff,$Names[$X])
            $This.SetProperty($Names[$X],$NameBuff)
        }
    }
    AddProperty([String]$Type,[String]$TypeBuff,[String]$NameBuff,[String]$Name)
    {
        $This.Property        += "    [{0}]{1}{2}`${3}" -f $Type, $TypeBuff, $NameBuff, $Name
    }
    SetProperty([String]$Property,[String]$NameBuff)
    {
        $This.Constructor     += "        `$This.{0}{1} = {2}.{0}" -f $Property,
                                                                     $NameBuff,
                                                                     $This.Param1Value
    }
# // <Continued below>
```

```
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

The After stuff right here looks a lot like the individuals up above.
However, there are many things that have been moved around to make the process more structurally sound.
Mainly because the hash table that was named $Def, was a good idea to turn into the default constructor and class properties.

The intitial constructor requests (3) parameters:
1) name of the class
2) Parameter1Type
3) Parameter1Value
... which can be seen up in the before section, except now the class can accommodate a much more broad range of input without doing a heck of a lot different.

The same can also be done with the hash table. But- if I want to make multiple copies of the class, each with a different set of values, that's a lot easier. With the hashtable, I'd have to copy and paste the hashtable, and then manually enter new values for those new hashtables, otherwise they'll have the same information.

Which is... rather anticlimactic, in all honesty.
Nothing like seeing thousands of the same exact hash table cause a token variable somewhere wasn't being changed.
The hashtables also don't scale well as a table like the classes do.

They CAN, but the class doesn't lose any of it's key arrangements or positioning, so the class is more consistent and reliable. That's not ALWAYS the case, but in this case, it most certainly is.

The method LoadPropertySet($Properties) does the same job as the individual variables in the before stuff above.
However, there are some KEY DIFFERENCES to note here. For starters, the way in which the variables were moved around, allowed the hashtable stuff to be put in it's own constructor.

Then, shifting the remaining variables from that particular block around, allowed moving the type length determination process, the type buffer, type buffer to string, as well as the name length determination process, name buffer, and name buffer to string... not only to the same block, but- there was a perfect opportunity to create a couple of new methods that cleaned up the way that code looked, while also providing some more control.

Maybe it's a matter of preference...? Idk.
But I like the way that portion of the code looks a LOT better than the individual variables.

Suffice to say, adding those methods makes a lot of sense as they're adding properties, and then setting the property values... having a method with a name gives anybody a better sense of what is actually happening behind the code.

```
_____________________
| [Before : Part 3] |___________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
$X               = 0..($Def.Const.Count-1) | ? { $Def.Const[$_] -match "EntryUnique" }
$Def.Const[$X]   = $Def.Const[$X] -Replace '= .+', '= $This.GetEntryUniqueProperties($Registry)'

# [Page 53]
$Method1         = @(
'    [String[]] Properties()',
```

```
'    {',
'        Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name',
'    }'
)
$Def.Method      += $Method1

# [Page 53]
$Method2          = @(
'    [Object[]] GetEntryUniqueProperties([Object]$Param)',
'    {',
'        Return @(
    $Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") | Select Name, Value'
'    }'
)
$Def.Method      += $Method2

# [Page 53]
$Method3           = @(
'    [Object[]] Output([UInt32]$Buffer)',
'    {',
'        $Output  = @( )',
'        $Output += ("-" * 120 -join "")',
'        $Output += "[$($This.DisplayName)]"',
'        $Output += ("-" * 120 -join "")',
'        $Output += " "',
'',
'        $This.Properties() | % { ',
'',
'            $Output += "{0}{1} : {2}" -f $_, (" " * ($Buffer - $_.Length + 1) -join ""), $This.$_',
'        }',
'',
'        $Output += (" " * $Buffer -join "")',
'        $This.EntryUnique  | % { ',
'',
'            $Output += "{0}{1} : {2}" -f $_.Name, (" " * ($Buffer - $_.Name.Length + 1 ) -join ""), $_.Value',
'        }',
'        $Output += (" " * $Buffer -join "")',
'',
'        Return $Output',
'    }'
)
$Def.Method      += $Method3
```

The Before stuff here, generally looks a lot like the after stuff, but, the class has methods where these values are just being changed directly. Like, the first couple of variables are pulling an index number to specifically recall the index of the property that's about to be changed, and then it changes it with some regex.

The after stuff is doing that too … albeit with changes.

The variables named $Method1, $Method2, and $Method3 really aren't all that different from the after stuff either, except here, there's no actual method that's inserting those variable values into the hashtable property named "Method".

Which is fine … ?
But, having a method name that describes the function is really what makes a class even more useful than a slew of variables all taped together. There really is no way to get away from that feeling where using many variables that aren't connected to a larger container object, starts to feel as if they're all operating on their own accord. Obviously, this process is required to build the class types, it's just that if a script writer doesn't make the effort to implement class types and stuff, they may never be able to write code that is able to describe itself a lot more clearly and coherently, and methods and even loop labels go a long way to assist with breaking portions of code off into trunks or branches, rather than one giant soup bowl of variables.

Just, variable soup. I don't see people using many loop labels these days, because it's essentially the same thing as a method, or a switch block.

[After : Part 3]

```
# // <Continued from above>
    ChangeProperty([String]$Name,[String]$Value)
    {
        $X                    = 0..($This.Constructor.Count-1) | ? {

                                $This.Constructor[$_] -match $This.Escape("`$This.$Name")

                                }
        If (!!$X)
        {
            [Console]::WriteLine("Property [+] Found, altering … ")
            $This.Constructor[$X]  = $This.Constructor[$X] -Replace "=.+","= $Value"
        }
```

```
        Else
        {
            [Console]::WriteLine("Property [!] Not found, skipping...")
        }
    }
    [String] AddIndent([UInt32]$Count)
    {
        Return "    " * $Count
    }
    AddMethod([String[]]$Body)
    {
        $Body   = $Body -Replace "^\s*",""
        $I      = 0
        $Return = @( )
        ForEach ($X in 0..($Body.Count-1))
        {
            $Line = $Body[$X]
            If ($X -eq 0 -and $Line -match "(\s{0})")
            {
                $I ++
                $Line = $This.AddIndent($I) + $Line
            }
            ElseIf ($X -eq 1 -and $Line -match "(\s{0}\{)")
            {
                $Line = $This.AddIndent($I) + $Line
                $I ++
            }
            ElseIf ($X -eq ($Body.Count-1) -and $Line -match "(\s{0}\})")
            {
                $I --
                $Line = $This.AddIndent($I) + $Line
            }
            Else
            {
                $Line = $This.AddIndent($I) + $Line
            }
            $Return += $Line
        }
        $This.Method += $Return -join "`n"
    }
    [String] Escape([String]$Value)
    {
        Return [Regex]::Escape($Value)
    }
# // <Continued below>
```

```
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

The after stuff here is a lot longer, no question about it.
Nothing seen in this group of methods is actually reproducing the code above, however-

It's doing additional things that make the code easier to throw OTHER values at, making it even more flexible than it already was. I'm going to paste the portion of code where the class is instantiated and then the methods that produce the same content as the variables up above, will be more readily comparable.

```
 ______________________________________________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

```
$Temp            = [DefinitionTemplate]::New("Uninstall","[Object]",'$Registry')
$Temp.LoadPropertySet($Properties)

$Temp.ChangeProperty("EntryUnique","`$This.GetEntryUniqueProperties(`$Registry)")

$Temp.AddMethod(@('[String[]] Properties()','{',
'Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name','}'))

$Temp.AddMethod(@('[Object[]] GetEntryUniqueProperties([Object]$Param)',
'{',
('Return @($Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") '+
 '| Select-Object Name, Value'),'}'))

$Temp.AddMethod(@('[Object[]] Output([UInt32]$Buff)','{','$X  = @( )',
'$X += ("-" * 120 -join ""), "[$($This.DisplayName)]", ("-" * 120 -join ""), " "',
'$This.Properties() | % { $X += "{0}{1} : {2}" -f $_, (" " * ($Buff - $_.Length + 1) -join ""), $This.$_ }',
'$X += (" " * $Buff -join "")',
('$This.EntryUnique  | % { $X += "{0}{1} : {2}" -f $_.Name,'+
 ' (" " * ($Buff - $_.Name.Length + 1 ) -join ""), $_.Value }'),
'$X += (" " * $Buff -join "")',
'Return $X','}'))
```

```
________________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 ¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯
```

The first two lines here instantiate the class.
Then the method LoadPropertySet injects the PSObject.Properties information.

The third method ChangeProperty is performing the same activity as the first two variables in the before section directly above.

Now I can explain why the code in the class is longer, but- achieves a better end result...
The method ChangeProperty looks for the property in the constructor "EntryUnique", and then sets its value to what is written in the quotes as the second parameter.

It's doing almost the identical thing under the hood as above, however the method ChangeProperty is providing a little bit of error handling, so if the property parameter finds no result, then it doesn't just NOT tell you that nothing was done, it'll say that.

But also, it uses a double not true boolean statement where if the token $X is returned via the double negative check, then that means it found the thing and it made the change.

Otherwise it didn't, and will say "failed..."
As for the method AddMethod(), it is generally identical to the previous variable method, but- I was able to implement spacing changes so that the input strings don't need to include those spaces.

Even if they do, they're gonna get ripped out and then reinserted from the ground up. Why?
Because it made more sense to do it that way.

However, it did mess with the way I like my code to be indented from nested code block to nested code block within a method. But, that's really a matter of preference and it's not a structural defect or anything.

Also, the methods could also be added in the same way as the naked variables, where there was $Method1, $Method2, and $Method3 and then those values were added... It could be done that way here as well.

The main difference is that the method is NAMED, rather than just having some property named "Method" that allows items to be added to it. Yeah, naming a property "method" that isn't actually a method might cause some confusion somewhere...? But then again, maybe it won't...

Either way, it's better to have a legitimate, actual, factual method named "AddMethod" rather than wingin' it with a property named Method then a plus sign and an equals sign, then a variable value.

I know somebody somewhere is going to say "That's not an actual method..."
I'm aware of that... it's a property named method, that just so happens to be inserting a value that represents the string version of a method for a class. A wolf in sheeps clothing, essentially.

It's not unlike showing up to a party in a shirt that says "I'm wearing a black shirt", but...
...your shirt is actually white.

Everyone's that takes their time to read your shirt is going to think...

```
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\

Person 1 : That dude's wearing a white shirt that says 'I'm wearing a black shirt'...
Person 2 : Wow.
           That dude's shirt definitely says that, and it IS white...
Person 3+: Dude's wearin' a white shirt, but then the shirt says "I'm wearing a black shirt"...
(Then somebody will eventually say...)
Person 9 : So, is it like the SHIRT that's wearing a shirt...?
           Or like-
           *shakes head* I don't even know dude...
            ...that shirt is blowing my mind right now...

\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
```

Maybe it won't be that dramatic though.

```
 ____________________
| [Before : Part 4] |______________________________________________________________________________
/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\
---------------------------------------------------------------------------------------------------------------

$ClassDefinition = @("Class $($Def.Name)","{",($Def.Type -join "`n"),
                     "    $($Def.Name)($($Def.Param1Type)$   ($Def.Param1Value))",
                     "    {",
                     ($Def.Const -join "`n"),
                     "    }",
                     ( $Def.Method -join "`n"),
                     "}") -join "`n"
_______________________________________________________________________________________________________________
\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/¯¯\__/
 -------------------------------------------------------------------------------------------------------------
```

There's not much left to go over.
It's literally just the class definition variable, the thing that's supposed to capture the output of the variables up above, so that the class can be loaded into memory, and then it can be instantiated.

There's no question, that having blobs of code written all over the place like what's directly above this paragraph, are a headache to look at. It's long, it's mean looking, doesn't even remotely feel like a cool dude that just so happens to be hanging out at the park during the summer...

The reason for that is because whenyoutrytostuffabunchofthingsintosomethinglikethat-
itkindalookslikeascarymessthatnobodywantstobenearoraroundcauseitslongcomplicatedlookingandnotfun.

I realize, it probably doesn't look THAT BAD ... but that whole "throwing away the spacebar" comment is sorta the vibe that it gives me. All it's doing is generating the single output string that just so happens to be the output of the entire set of variables.

But, there's so many ways it can be optimized:

1) String interpolation
2) directly embedding the values without so many quotes,
3) joining the strings together in separate chunks

It's not BAD for a FIRST ATTEMPT when you're conceptualizing something ...
... but at some point, this will cause anybody that respects well written code, to say (1) word. "Ahhh!"

Do you want somebody who's always been known for respecting well written code, to say "Ahhh!" ...
... when they look at YOURS ... ?
Probably not, right ... ?
Cause "Ahhh!" isn't even a word, really.

At which point, who's the person who feels most insulted ... ? You ... ?
Or some guy that has always been known for respecting well written code ... that said "Ahhh!" ...
... when they looked at yours ... ?

Maybe Linus Torvalds may have to come out from somewhere, and say:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Torvalds : I've seen FAR worse code than that, pal.
           And, I've ALWAYS been known for respecting well-written code ...
(Then I guess I'm gonna have to put my hands up ...)
Me   : Alright ... ~!
           Can't argue with ya."

__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/

Cause who's gonna argue with Linus Torvalds ... the man who wrote Linux? Nobody, that's who.
Everyone reading this will probably agree with me:

/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

Everybody : *shakes head* Yeah.
            That Linus Torvalds guy ... ?
            He literally knows the ins and outs of well written code ...

__/--\

In that case ... ? Then MAYBE it's settled ...
MAYBE the code above isn't THAT bad.

But, PERSONALLY ... ? I feel as if it appears that the author lost track of their space bar, or enter key.
Maybe that's fine sometimes. What do I know?

Whether I AM THE GUY THAT WROTE THAT or not ... ?
It's irrelevant.

Gives me the heebie jeebies.
And, that's why I rewrote it in the class and that whole mess became this thing below ...

____
| [After : Part 4] |____
/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--__/--\

```
# // <Continued from above>
    [String] ReturnDefinition()
    {
        $X              = @{
            Name        = "Class {0}" -f $This.Name
            Property    = $This.Property -join "`n"
            Main        = "    {0}({1}{2})" -f $This.Name,
                                               $This.Param1Type,
                                               $this.Param1Value
            Constructor = $This.Constructor -join "`n"
            Method      = $This.Method -join "`n"
        }
        Return @( $X.Name,
                  "{",
                  $X.Property,
                  $X.Main,
                  "    {",
                  $X.Constructor,
                  "    }",
                  $X.Method,
                  "}"
                ) -join "`n"
    }
```

```
}

$ClassDefinition = $Temp.ReturnDefinition()
```

People may look at this and think...
But- you had all of that in one line... This is like (12+).

Yep.
I know, it APPEARS to be a lot MORE INFORMATION (in this case it is), but sometimes I ask myself how the highway guys can stand working on the highway all day in the heat, those tenth of a mile markers being the thing they gotta drop a bunch of asphalt between, and they'll occasionally look at these markers to gauge their progress.

They probably live for each and every one of those things too.

Highway guy: Ah man.
Just another...
4 miles to go.
Not bad.
It's only Monday though.
Damnit.

You know that they're walking... and they look at these things like a clock or a watch.
Pouring, and laying down some asphalt all day long... living through hell, paycheck to paycheck.

Here's why I prefer the thing above.
If I want to make it shorter, that's easy.
Highway guy can't make his job any shorter if he even wanted to.
With 4 miles of asphalt to lay, there's nothing he can do to make his job less difficult on himself.

But, I definitely CAN make MY job easier on myself.
So, if I want to examine a problem with the output, then I've made it incredibly easy to track down what COULD be causing an issue, adjust it, and then I totally avoid feeling like that dude on the highway in the blistering heat, just pouring asphalt all day long...

Cause, even though those guys typically get paid pretty well...?
I don't think a single one of them dudes really LOVE doing that job...
Maybe some of them do, I don't know.

From what some of my friends tell me they say it's ONLY fun AFTER you get paid.

Conclusion / Comparison

Anyway, that's it for the lesson, hope you enjoyed it.
I've gone ahead and pasted additional copies of the classes below in a comment block.

```
Class Win32_Product
{
    [UInt16]                                $AssignmentType
    [String]                                       $Caption
    [String]                                   $Description
    [String]                                   $ElementName
    [String]                                       $HelpLink
    [String]                                  $HelpTelephone
    [String]                                $IdentifyingNumber
    [String]                                     $InstallDate
    [String]                                    $InstallDate2
    [String]                                $InstallLocation
    [String]                                  $InstallSource
    [Int16]                                    $InstallState
    [String]                                     $InstanceID
    [String]                                       $Language
    [String]                                   $LocalPackage
    [String]                                           $Name
    [String]                                   $PackageCache
    [String]                                    $PackageCode
    [String]                                    $PackageName
    [String]                                      $ProductID
    [String]                                    $RegCompany
    [String]                                      $RegOwner
    [String]                                    $SKUNumber
    [String]                                    $Transforms
    [String]                                  $URLInfoAbout
    [String]                                $URLUpdateInfo
```

```
    [String]                                    $Vendor
    [String]                                   $Version
    [UInt32]                                $WarrantyDuration
    [String]                               $WarrantyStartDate
    [UInt32]                                       $WordCount
    [Management.ManagementScope]                       $Scope
    [Management.ManagementPath]                         $Path
    [Management.ObjectGetOptions]                     $Options
    [Management.ManagementPath]                     $ClassPath
    [Management.PropertyDataCollection]            $Properties
    [Management.PropertyDataCollection]    $SystemProperties
    [Management.QualifierDataCollection]          $Qualifiers
    [ComponentModel.ISite]                              $Site
    [ComponentModel.IContainer]                       $Container
    Win32_Product([Object]$WMIObject)
    {
        $This.AssignmentType     = $WMIObject.AssignmentType
        $This.Caption            = $WMIObject.Caption
        $This.Description        = $WMIObject.Description
        $This.ElementName        = $WMIObject.ElementName
        $This.HelpLink           = $WMIObject.HelpLink
        $This.HelpTelephone      = $WMIObject.HelpTelephone
        $This.IdentifyingNumber  = $WMIObject.IdentifyingNumber
        $This.InstallDate        = $WMIObject.InstallDate
        $This.InstallDate2       = $WMIObject.InstallDate2
        $This.InstallLocation    = $WMIObject.InstallLocation
        $This.InstallSource      = $WMIObject.InstallSource
        $This.InstallState       = $WMIObject.InstallState
        $This.InstanceID         = $WMIObject.InstanceID
        $This.Language           = $WMIObject.Language
        $This.LocalPackage       = $WMIObject.LocalPackage
        $This.Name               = $WMIObject.Name
        $This.PackageCache       = $WMIObject.PackageCache
        $This.PackageCode        = $WMIObject.PackageCode
        $This.PackageName        = $WMIObject.PackageName
        $This.ProductID          = $WMIObject.ProductID
        $This.RegCompany         = $WMIObject.RegCompany
        $This.RegOwner           = $WMIObject.RegOwner
        $This.SKUNumber          = $WMIObject.SKUNumber
        $This.Transforms         = $WMIObject.Transforms
        $This.URLInfoAbout       = $WMIObject.URLInfoAbout
        $This.URLUpdateInfo      = $WMIObject.URLUpdateInfo
        $This.Vendor             = $WMIObject.Vendor
        $This.Version            = $WMIObject.Version
        $This.WarrantyDuration   = $WMIObject.WarrantyDuration
        $This.WarrantyStartDate  = $WMIObject.WarrantyStartDate
        $This.WordCount          = $WMIObject.WordCount
        $This.Scope              = $WMIObject.Scope
        $This.Path               = $WMIObject.Path
        $This.Options            = $WMIObject.Options
        $This.ClassPath          = $WMIObject.ClassPath
        $This.Properties         = $WMIObject.Properties
        $This.SystemProperties   = $WMIObject.SystemProperties
        $This.Qualifiers         = $WMIObject.Qualifiers
        $This.Site               = $WMIObject.Site
        $This.Container          = $WMIObject.Container
    }
}

Class Uninstall
{
    [String]        $DisplayName
    [String]     $DisplayVersion
    [String]            $Version
    [Int32]           $NoRemove
    [String]         $ModifyPath
    [String]    $UninstallString
    [String]    $InstallLocation
    [String]        $DisplayIcon
    [Int32]           $NoRepair
    [String]         $Publisher
    [String]        $InstallDate
    [Int32]        $VersionMajor
    [Int32]        $VersionMinor
    [Object[]]      $EntryUnique
    Uninstall([Object]$Registry)
    {
        $This.DisplayName      = $Registry.DisplayName
        $This.DisplayVersion   = $Registry.DisplayVersion
        $This.Version          = $Registry.Version
        $This.NoRemove         = $Registry.NoRemove
        $This.ModifyPath       = $Registry.ModifyPath
        $This.UninstallString  = $Registry.UninstallString
```

```
            $This.InstallLocation  = $Registry.InstallLocation
            $This.DisplayIcon      = $Registry.DisplayIcon
            $This.NoRepair         = $Registry.NoRepair
            $This.Publisher        = $Registry.Publisher
            $This.InstallDate      = $Registry.InstallDate
            $This.VersionMajor     = $Registry.VersionMajor
            $This.VersionMinor     = $Registry.VersionMinor
            $This.EntryUnique      = $This.GetEntryUniqueProperties($Registry)
        }
        [String[]] Properties()
        {
            Return $This.PSObject.Properties | ? Name -notmatch EntryUnique | % Name
        }
        [Object[]] GetEntryUniqueProperties([Object]$Param)
        {
            Return @(
            $Param.PSObject.Properties | ? Name -notmatch "(^PS|$($This.Properties() -join "|"))") | Select Name, Value
        }
        [Object[]] Output([UInt32]$Buff)
        {
            $X  = @( )
            $X += ("-" * 120 -join ""), "[$($This.DisplayName)]", ("-" * 120 -join ""), " "
            $This.Properties() | % {
            $X += "{0}{1} : {2}" -f $_, (" " * ($Buff - $_.Length + 1) -join ""), $This.$_
            }
            $X += (" " * $Buff -join "")
            $This.EntryUnique  | % {
            $X += "{0}{1} : {2}" -f $_.Name, (" " * ($Buff - $_.Name.Length + 1 ) -join ""), $_.Value
            }
            $X += (" " * $Buff -join "")
            Return $X
        }
    }

    Class UninstallStack
    {
        [UInt32] $Buffer
        [Object] $Output
        UninstallStack()
        {
            $Apps            = "\Wow6432Node","" | % {

                               Get-ChildItem "HKLM:\Software$_\Microsoft\Windows\CurrentVersion\Uninstall"

                               } | Get-ItemProperty
            $Edge            = $Apps | ? DisplayName -match "(^Microsoft Edge$)"
            $Properties      = $Edge.PSObject.Properties | ? Name -notmatch "(^_|^PS)"
            $Properties     += [PSNoteProperty]::New("EntryUnique",@( ))
            $This.Output     = $Apps | % { [Uninstall]::New($_) }
            $This.Buffer     = ($This.Output.EntryUnique.Name | Sort-Object Length)[-1].Length
        }
        [Object[]] GetOutput()
        {
            Return @( $This.Output.Output($This.Buffer) )
        }
    }
    ____________________________________________________________________________________________________________________________
    \__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/--\__/
     ----------------------------------------------------------------------------------------------------------------------------
                                                                                                                  ____________/
\_________________________________________________________________________________________________________________/ Conclusion
```