```powershell
<#
This script is similar to the get-specs script with one difference; it is intended to be run locally.

Save the script to a flashdrive, run it on a machine that you wouldn't be able to contact remotely.

You will have an option to save all of the data to file.
#>

#Modules ======================================================================================================================
if(!(get-installedmodule -name 'PSWRiteColor' -EA ignore)) {
    set-psrepository -name psgallery -InstallationPolicy Trusted
    install-module -name pswritecolor
    import-module -name pswritecolor
}
if(!(get-installedmodule -name 'ImportExcel' -EA ignore)) {
    set-psrepository -name psgallery -InstallationPolicy Trusted
    install-module -name importexcel
    import-module -name importexcel
}

#Hashtables ===================================================================================================================
$script:AcceleratorCapabilities_map = @{
      0 = 'Unknown'
      1 = 'Other'
      2 = 'Graphics Accelerator'
      3 = '3D Accelerator'
}

$script:AdapterTypeID_map = @{
      0 = 'Ethernet 802.3'
      1 = 'Token Ring 802.5'
      2 = 'Fiber Distributed Data Interface (FDDI)'
      3 = 'Wide Area Network (WAN)'
      4 = 'LocalTalk'
      5 = 'Ethernet using DIX header format'
      6 = 'ARCNET'
      7 = 'ARCNET (878.2)'
      8 = 'ATM'
      9 = 'Wireless'
     10 = 'Infrared Wireless'
     11 = 'Bpc'
     12 = 'CoWan'
     13 = '1394'
}

$script:Architecture_map = @{
      0 = 'x86'
      1 = 'MIPS'
      2 = 'Alpha'
      3 = 'PowerPC'
      6 = 'ia64'
      9 = 'x64'
}

$script:Availability_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'Running/Full Power'
      4 = 'Warning'
      5 = 'In Test'
      6 = 'Not Applicable'
      7 = 'Power Off'
      8 = 'Off Line'
      9 = 'Off Duty'
     10 = 'Degraded'
     11 = 'Not Installed'
     12 = 'Install Error'
     13 = 'Power Save - Unknown'
     14 = 'Power Save - Low Power Mode'
     15 = 'Power Save - Standby'
     16 = 'Power Cycle'
     17 = 'Power Save - Warning'
     18 = 'Paused'
     19 = 'Not Ready'
     20 = 'Not Configured'
     21 = 'Quiesced'
}
```

```
$script:BatteryStatus_map = @{

      1 = 'Battery Power'
      2 = 'AC Power'
      3 = 'Fully Charged'
      4 = 'Low'
      5 = 'Critical'
      6 = 'Charging'
      7 = 'Charging and High'
      8 = 'Charging and Low'
      9 = 'Charging and Critical'
      10 = 'Undefined'
      11 = 'Partially Charged'
}

$script:Capabilities_map = @{
      0 = 'Unknown'
      1 = 'Other'
      2 = 'Sequential Access'
      3 = 'Random Access'
      4 = 'Supports Writing'
      5 = 'Encryption'
      6 = 'Compression'
      7 = 'Supports Removeable Media'
      8 = 'Manual Cleaning'
      9 = 'Automatic Cleaning'
     10 = 'SMART Notification'
     11 = 'Supports Dual Sided Media'
     12 = 'Predismount Eject Not Required'
}

$script:Chemistry_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'Lead Acid'
      4 = 'Nickel Cadmium'
      5 = 'Nickel Metal Hydride'
      6 = 'Lithium-ion'
      7 = 'Zinc air'
      8 = 'Lithium Polymer'
}

$script:ConfigManagerErrorCode_map = @{
      0 = 'This device is working properly.'
      1 = 'This device is not configured correctly.'
      2 = 'Windows cannot load the driver for this device.'
      3 = 'The driver for this device might be corrupted, or your system may be running low on memory or other resources.'
      4 = 'This device is not working properly. One of its drivers or your registry might be corrupted.'
      5 = 'The driver for this device needs a resource that Windows cannot manage.'
      6 = 'The boot configuration for this device conflicts with other devices.'
      7 = 'Cannot filter.'
      8 = 'The driver loader for the device is missing.'
      9 = 'This device is not working properly because the controlling firmware is reporting the resources for the device inco
     10 = 'This device cannot start.'
     11 = 'This device failed.'
     12 = 'This device cannot find enough free resources that it can use.'
     13 = 'Windows cannot verify this device''s resources.'
     14 = 'This device cannot work properly until you restart your computer.'
     15 = 'This device is not working properly because there is probably a re-enumeration problem.'
     16 = 'Windows cannot identify all the resources this device uses.'
     17 = 'This device is asking for an unknown resource type.'
     18 = 'Reinstall the drivers for this device.'
     19 = 'Failure using the VxD loader.'
     20 = 'Your registry might be corrupted.'
     21 = 'System failure: Try changing the driver for this device. If that does not work, see your hardware documentation. W
     22 = 'This device is disabled.'
     23 = 'System failure: Try changing the driver for this device. If that doesn''t work, see your hardware documentation.'
     24 = 'This device is not present, is not working properly, or does not have all its drivers installed.'
     25 = 'Windows is still setting up this device.'
     26 = 'Windows is still setting up this device.'
     27 = 'This device does not have valid log configuration.'
     28 = 'The drivers for this device are not installed.'
     29 = 'This device is disabled because the firmware of the device did not give it the required resources.'
     30 = 'This device is using an Interrupt Request (IRQ) resource that another device is using.'
     31 = 'This device is not working properly because Windows cannot load the drivers required for this device.'
}
```

```
$script:CpuStatus_map = @{
    0 = 'Unknown'
    1 = 'CPU Enabled'
    2 = 'CPU Disabled by User via BIOS Setup'
    3 = 'CPU Disabled By BIOS (POST Error)'
    4 = 'CPU is Idle'
    5 = 'Reserved'
    6 = 'Reserved'
    7 = 'Other'
}

$script:CPUStatusInfo_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'Enabled'
      4 = 'Disabled'
      5 = 'Not Applicable'
}

$script:DriveType_map = @{
    0 = 'Unknown'
    1 = 'No_Root_Directory'
    2 = 'Removable Disk'
    3 = 'Local Disk'
    4 = 'Network Drive'
    5 = 'Compact Disk'
    6 = 'RAM Disk'
}

$script:DitherType_map = @{
      1 = 'No dithering'
      2 = 'Dithering with a coarse brush'
      3 = 'Dithering with a fine brush'
      4 = 'Line art dithering'
      5 = 'Device does gray scaling'
}

$script:Family_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = '8086'
      4 = '80286'
      5 = '80386'
      6 = '80486'
      7 = '8087'
      8 = '80287'
      9 = '80387'
     10 = '80487'
     11 = 'Pentium(R) brand'
     12 = 'Pentium(R) Pro'
     13 = 'Pentium(R) II'
     14 = 'Pentium(R) processor with MMX(TM) technology'
     15 = 'Celeron(TM)'
     16 = 'Pentium(R) II Xeon(TM)'
     17 = 'Pentium(R) III'
     18 = 'M1 Family'
     19 = 'M2 Family'
     24 = 'K5 Family'
     25 = 'K6 Family'
     26 = 'K6-2'
     27 = 'K6-3'
     28 = 'AMD Athlon(TM) Processor Family'
     29 = 'AMD(R) Duron(TM) Processor'
     30 = 'AMD29000 Family'
     31 = 'K6-2+'
     32 = 'Power PC Family'
     33 = 'Power PC 601'
     34 = 'Power PC 603'
     35 = 'Power PC 603+'
     36 = 'Power PC 604'
     37 = 'Power PC 620'
     38 = 'Power PC X704'
     39 = 'Power PC 750'
     48 = 'Alpha Family'
     49 = 'Alpha 21064'
     50 = 'Alpha 21066'
```

```
      51 = 'Alpha 21164'
      52 = 'Alpha 21164PC'
      53 = 'Alpha 21164a'
      54 = 'Alpha 21264'
      55 = 'Alpha 21364'
      64 = 'MIPS Family'
      65 = 'MIPS R4000'
      66 = 'MIPS R4200'
      67 = 'MIPS R4400'
      68 = 'MIPS R4600'
      69 = 'MIPS R10000'
      80 = 'SPARC Family'
      81 = 'SuperSPARC'
      82 = 'microSPARC II'
      83 = 'microSPARC IIep'
      84 = 'UltraSPARC'
      85 = 'UltraSPARC II'
      86 = 'UltraSPARC IIi'
      87 = 'UltraSPARC III'
      88 = 'UltraSPARC IIIi'
      96 = '68040'
      97 = '68xxx Family'
      98 = '68000'
      99 = '68010'
     100 = '68020'
     101 = '68030'
     112 = 'Hobbit Family'
     120 = 'Crusoe(TM) TM5000 Family'
     121 = 'Crusoe(TM) TM3000 Family'
     122 = 'Efficeon(TM) TM8000 Family'
     128 = 'Weitek'
     130 = 'Itanium(TM) Processor'
     131 = 'AMD Athlon(TM) 64 Processor Family'
     132 = 'AMD Opteron(TM) Family'
     144 = 'PA-RISC Family'
     145 = 'PA-RISC 8500'
     146 = 'PA-RISC 8000'
     147 = 'PA-RISC 7300LC'
     148 = 'PA-RISC 7200'
     149 = 'PA-RISC 7100LC'
     150 = 'PA-RISC 7100'
     160 = 'V30 Family'
     176 = 'Pentium(R) III Xeon(TM)'
     177 = 'Pentium(R) III Processor with Intel(R) SpeedStep(TM) Technology'
     178 = 'Pentium(R) 4'
     179 = 'Intel(R) Xeon(TM)'
     180 = 'AS400 Family'
     181 = 'Intel(R) Xeon(TM) processor MP'
     182 = 'AMD AthlonXP(TM) Family'
     183 = 'AMD AthlonMP(TM) Family'
     184 = 'Intel(R) Itanium(R) 2'
     185 = 'Intel Pentium M Processor'
     190 = 'K7'
     200 = 'IBM390 Family'
     201 = 'G4'
     202 = 'G5'
     203 = 'G6'
     204 = 'z/Architecture base'
     250 = 'i860'
     251 = 'i960'
     260 = 'SH-3'
     261 = 'SH-4'
     280 = 'ARM'
     281 = 'StrongARM'
     300 = '6x86'
     301 = 'MediaGX'
     302 = 'MII'
     320 = 'WinChip'
     350 = 'DSP'
     500 = 'Video Processor'
 }

 $script:FormFactor_Map = @{
     0 = 'Unknown'
     1 = 'Other'
     2 = 'SIP'
     3 = 'DIP'
```

```
    4 = 'ZIP'
    5 = 'SOP'
    6 = 'Proprietary'
    7 = 'SIMM'
    8 = 'DIMM'
    9 = 'TSOP'
    10 = 'PGA'
    11 = 'RIMM'
    12 = 'SODIMM'
    13 = 'SRIMM'
    14 = 'SMD'
    15 = 'SSMP'
    16 = 'QFP'
    17 = 'TQFP'
    18 = 'SOIC'
    19 = 'LCC'
    20 = 'PLCC'
    21 = 'BGA'
    22 = 'FPBGA'
    23 = 'LGA'
}

$script:ManufacturerRAM_Map = @{
      '01980000802C' = 'Kingston'
      '80AD00000000' = 'SK Hynix'
}

$script:MemoryErrorCorrection_map = @{
      0 = 'Reserved'
      1 = 'Other'
      2 = 'Unknown'
      3 = 'None'
      4 = 'Parity'
      5 = 'Single-bit ECC'
      6 = 'Multi-bit ECC'
      7 = 'CRC'
}

$script:MemoryType_map = @{
      0 = 'Unknown'
      1 = 'Other'
      2 = 'DRAM'
      3 = 'Synchronous DRAM'
      4 = 'Cache DRAM'
      5 = 'EDO'
      6 = 'EDRAM'
      7 = 'VRAM'
      8 = 'SRAM'
      9 = 'RAM'
     10 = 'ROM'
     11 = 'Flash'
     12 = 'EEPROM'
     13 = 'FEPROM'
     14 = 'EPROM'
     15 = 'CDRAM'
     16 = '3DRAM'
     17 = 'SDRAM'
     18 = 'SGRAM'
     19 = 'RDRAM'
     20 = 'DDR'
     21 = 'DDR2'
     22 = 'DDR2 FB-DIMM'
}

$script:NetConnectionStatus_map = @{
      0 = 'Disconnected'
      1 = 'Connecting'
      2 = 'Connected'
      3 = 'Disconnecting'
      4 = 'Hardware Not Present'
      5 = 'Hardware Disabled'
      6 = 'Hardware Malfunction'
      7 = 'Media Disconnected'
      8 = 'Authenticating'
      9 = 'Authentication Succeeded'
     10 = 'Authentication Failed'
     11 = 'Invalid Address'
```

```
      12 = 'Credentials Required'
}

$script:PowerManagementCapabilities_map = @{
      0 = 'Unknown'
      1 = 'Not Supported'
      2 = 'Disabled'
      3 = 'Enabled'
      4 = 'Power Saving Modes Entered Automatically'
      5 = 'Power State Settable'
      6 = 'Power Cycling Supported'
      7 = 'Timed Power On Supported'
}

$script:ProcessorType_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'Central Processor'
      4 = 'Math Processor'
      5 = 'DSP Processor'
      6 = 'Video Processor'
}

$script:RAMpartnumber_map = @{
}

$script:StatusInfo_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'Enabled'
      4 = 'Disabled'
      5 = 'Not Applicable'
}

$script:TypeDetail_map = @{
      1 = 'Reserved'
      2 = 'Other'
      4 = 'Unknown'
      8 = 'Fast-paged'
     16 = 'Static column'
     32 = 'Pseudo-static'
     64 = 'RAMBUS'
    128 = 'Synchronous'
    256 = 'CMOS'
    512 = 'EDO'
   1024 = 'Window DRAM'
  16512 = 'Synchronous and Unbuffered (unregistered)'
   2048 = 'Cache DRAM'
   4096 = 'Non-volatile'
}

$script:UpgradeMethod_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'Daughter Board'
      4 = 'ZIF Socket'
      5 = 'Replacement/Piggy Back'
      6 = 'None'
      7 = 'LIF Socket'
      8 = 'Slot 1'
      9 = 'Slot 2'
     10 = '370 Pin Socket'
     11 = 'Slot A'
     12 = 'Slot M'
     13 = 'Socket 423'
     14 = 'Socket A (Socket 462)'
     15 = 'Socket 478'
     16 = 'Socket 754'
     17 = 'Socket 940'
     18 = 'Socket 939'
}

$script:Use_map = @{
      0 = 'Reserved'
      1 = 'Other'
      2 = 'Unknown'
      3 = 'System memory'
```

```
      4 = 'Video memory'
      5 = 'Flash memory'
      6 = 'Non-volatile RAM'
      7 = 'Cache memory'
}

$script:VideoArchitecture_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'CGA'
      4 = 'EGA'
      5 = 'VGA'
      6 = 'SVGA'
      7 = 'MDA'
      8 = 'HGC'
      9 = 'MCGA'
     10 = '8514A'
     11 = 'XGA'
     12 = 'Linear Frame Buffer'
    160 = 'PC-98'
}

$script:VideoMemoryType_map = @{
      1 = 'Other'
      2 = 'Unknown'
      3 = 'VRAM'
      4 = 'DRAM'
      5 = 'SRAM'
      6 = 'WRAM'
      7 = 'EDO RAM'
      8 = 'Burst Synchronous DRAM'
      9 = 'Pipelined Burst SRAM'
     10 = 'CDRAM'
     11 = '3DRAM'
     12 = 'SDRAM'
     13 = 'SGRAM'
}


$script:space = @() # stores results from get-space
$script:OS = @() # stores results from get-OS
$script:BIOS = @() # stores results from Get-BIOS
$script:CPU = @() # stores results from get-CPU
$script:RAM = @() # stores results from get-RAM
$script:GPU = @() # stores results from get-GPU
$script:Network = @() # stores results from get-network
$script:Health = @() # wip - storing health status for checked components
$script:healthRes = @() # name for customobject that makes up health
$script:Devices = @() # stores results from get-devices
$script:Basics = @() # stores basic/general stats about enclosure
$script:Battery = @() # stores information about installed batteries (if present)
$script:AllStats = @() # array of arrays

function get-filename {
      $script:Directory = "C:\Output\"
  Write-Color "Would you like to save your output to ","${script:Directory}" -Color Yellow,Green
  $Confirm = Read-Host -Prompt '[Y]es or any key for No'
  Write-Color "${Confirm}" -Color Cyan
  if(!($Confirm -eq 'Y')) {
    $script:Directory = Read-Host -Prompt "Where would you like to save your output?"
    Write-Color "${Directory}" -Color Cyan
  }
  $repChar = '_'
  $tmpName = Read-Host -Prompt 'Please enter report title; note that spaces will be replaced by undescores'
  $script:Report = $tmpName -replace ' ', $repChar
  $script:Directory = $script:Directory.trimend("\")


  $script:CheckPath = "${script:Directory}\${script:Report}.xlsx"

  write-host "$script:checkpath"


  if(test-path $script:Checkpath) {

    Write-Color "A file already exists by this name!","`r`tWould you like to ","[O]","verwrite ","or have the script ","[R]",'
    $script:ChkChoice = Read-Host -prompt "[O]verwrite, [R]ename, or any other key to cancel and end the script"
```

```
    switch($script:chkchoice) {
      'O' {
            remove-item -path $script:checkpath -force
            Write-Color "Overwriting ","${script:checkpath}" -color red,yellow
      }
      'R' {
            $script:updateTime = get-date -format "MM-dd-yy"
            $script:Report = -join($script:Report,"_")
            $script:Report = -join($script:Report,$script:updateTime)
            $script:CheckPath = "${script:Directory}\${script:Report}.xlsx"
            write-color "New Filename: ","${script:checkpath}" -color cyan,green
      }
    }
  }

  Write-Color -foregroundcolor Green "Report will be saved as ${script:Report}.xlsx in directory ${script:Directory}"
  return $script:checkpath
}

function write-excel {

      $script:xlpkg = $script:trash       | export-excel -path $script:checkpath    -worksheetname 'TRASH'        -tablename
      $script:xlpkg = $script:Basics      | Export-excel -excelpackage $xlpkg       -worksheetname 'Basics'       -tablename
      $script:xlpkg = $script:Battery     | Export-excel -excelpackage $xlpkg       -worksheetname 'Battery'      -tablename
      $script:xlpkg = $script:OS          | Export-excel -excelpackage $xlpkg       -worksheetname 'OS'           -tablename
      $script:xlpkg = $script:BIOS        | Export-excel -excelpackage $xlpkg       -worksheetname 'BIOS'         -tablename
      $script:xlpkg = $script:CPU         | Export-excel -excelpackage $xlpkg       -worksheetname 'CPU'          -tablename
      $script:xlpkg = $script:GPU         | Export-excel -excelpackage $xlpkg       -worksheetname 'GPU'          -tablename
      $script:xlpkg = $script:RAM         | Export-excel -excelpackage $xlpkg       -worksheetname 'RAM'          -tablename
      $script:xlpkg = $script:Network     | Export-excel -excelpackage $xlpkg       -worksheetname 'Network'      -tablename
      $script:xlpkg = $script:Space       | Export-excel -excelpackage $xlpkg       -worksheetname 'Space'        -tablename
      $script:xlpkg = $script:Health      | Export-excel -excelpackage $xlpkg       -worksheetname 'Health'       -tablename


      $script:xlpkg.workbook.worksheets.delete('TRASH')
      set-excelrange    -worksheet $script:xlpkg.Health -Range "E:E" -wraptext
      set-excelrange    -worksheet $script:xlpkg.Health -Range "G:G" -wraptext
      set-excelrange    -worksheet $script:xlpkg.Health -Range "H:H" -wraptext
      set-excelrange    -worksheet $script:xlpkg.OS -Range "H:H" -wraptext
      set-excelrange    -worksheet $script:xlpkg.OS -Range "Q:Q" -wraptext
      close-excelpackage $script:xlpkg
}

Function Get-FriendlySize {
    Param([bigint]$bytes)
    switch($bytes){
        {$_ -gt 1PB}{"{0:N2} PB" -f ($_ / 1PB);break}
        {$_ -gt 1TB}{"{0:N2} TB" -f ($_ / 1TB);break}
        {$_ -gt 1GB}{"{0:N2} GB" -f ($_ / 1GB);break}
        {$_ -gt 1MB}{"{0:N2} MB" -f ($_ / 1MB);break}
        {$_ -gt 1KB}{"{0:N2} KB" -f ($_ / 1KB);break}
        default {"{0:N2} Bytes" -f $_}
    }
}

function Check-AD {
    param ([string]$comp)

    $script:ADFixFlag = $false
    $script:inADFlag = $null
    $script:LastLogon = $null
    $script:tmpName = $null
    $script:OldName = $null
    $numeric = ($comp -match "^[\d\.]+$")


    #if $comp is numeric (OT) add wildcards to searchstr
    if($numeric -eq $true) {
        $searchstr = -join("*",$comp,"*")
    }
    else {
        $searchstr = $comp
    }

    #filter search AD for names that contain $comp
    $script:inAD = get-adcomputer -filter {name -like $searchstr} -properties *
```

```
    if($script:inAD -ne $null) {
        $script:tmpName = $script:inAD.Name
        $script:LastLogon = $script:inAD.lastlogondate
        $script:inADFlag = $true
        $script:ados = $script:inAD.OperatingSystem
        $script:ou   = ($script:inAD.distinguishedName).split(',')[-6].split('=')[1]
        $script:adDesc = $script:inAD.description

        if($script:ou -eq $comp) {

                $script:ou = ($script:inAD.distinguishedName).split(',')[-5].split('=')[1]
            }

            if(!($script:ados -like '*Windows*')) {
                $script:ados = "Non-PC"
                $script:Err += "Non-PC"
            }

            #if found in AD, check if $comp and AD name are the same
            if($script:tmpName -ne $comp) {
                #only set fix flag to true if searchstr was for OT
                if ($numeric -eq $true) {
                    $script:OldName = $script:Node
                    $script:Node = $script:tmpName
                    $script:ADFixFlag = $true
                }
            }
        }

    else {
        $script:inADFlag = $false
    }

    return $script:inADFlag
}

function Get-ADinfo {

    param ( [string]$comp)

    if($script:Err -ne $null) {

        $status = $script:Err
    }
    else {
        $status = "Successful"
    }

    check-ad $comp

    $script:Obj = (Get-ADComputer -Identity $comp -properties canonicalname, MemberOf)
    $cname   = $script:Obj.canonicalName
    $hname  = $script:Obj.Name
    $groups = $script:Obj.MemberOf -replace '^CN=([^,]+).+$','$1'

    if($ou -eq $hname) {

            $ou = ($script:Obj.distinguishedName).split(',')[-5].split('=')[1]
        }
        $script:all_groups = "Groups:"

    foreach($g in $groups) {
      $script:all_groups = -join($script:all_groups,"`r`t${g}; ")

    }
    $script:all_groups = $script:all_groups.trimend("; ")

    $nsforward = resolve-dnsname -name $comp -erroraction silentlycontinue
        if($nsforward -eq $null) {
            $reverse = $null
        }
        else {
            $reverse = resolve-dnsname -name $nsforward.ipaddress.tostring() -erroraction silentlycontinue
        }

     $script:Res = [PSCustomObject]@{
```

```
        Host = $comp
        OT = $OT
        Query = 'Get-ADinfo'
        Desc = $script:adDesc
        OS = $script:ados
        OU = $script:ou
        Groups = $all_groups
        NSlookup = $nsforward.IPAddress
        ReverseNS = $reverse.Namehost
        Created = $script:inad.created
        Last_Logon = $script:inad.lastlogondate
        Status = $status
      }

       $desc = $script:addesc

       $enabled = $script:inad.enabled
       $locked = $script:inad.lockedout
       $logcount = $script:inad.logoncount

       $status = -join("Enabled: ",$enabled)
       $status = -join($status,"`rLocked_Out: ")
       $status = -join($status,$locked)
       $status = -join($status,"`rLogon_Count: ")
       $status = -join($status,$logcount)

       $name = -join("Name: ${comp}","`rAD_Desc: `r`t")
       $name = -join($name,$desc)

       $serialmac = -join("SID: ",$script:inad.objectsid)

         $script:healthres = [pscustomobject]@{
             Host = $script:res.host
             OT = $script:res.ot
             Query = 'Get-Health'
             Component = 'AD_Info'
             Health = $status
             Name = $name
             Detail = $script:all_groups
             Serial_Mac = $serialmac

         }
         $script:health += $script:healthres
     $script:AD_Info += $script:res
}

function Get-Space {
      param([string]$comp)

     $script:volumes = get-volume | where-object {$_.filesystemtype -ne 'Unknown' -and $_.driveletter -ne $null}

     foreach($vol in $volumes) {

         $drive = $vol.driveletter

         $infov = get-volume | where driveletter -like $drive | get-partition | select *
         $infops = get-psdrive $drive
         $infodrive = get-disk | where disknumber -like $infov.disknumber | select *
         $infohealth = get-physicaldisk | where {$_.driveletter -like $infov.driveletter} | select *
         $free = (($infops.free / $infov.size)).tostring("P")
         $health = $infodrive.healthstatus
             $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


         $script:res = [pscustomobject]@{

           Host = $comp
           OT = $OT
           Query = 'Get-Space'
           Drive = $drive
           DriveModel = $infodrive.friendlyname
           DriveType = $vol.drivetype
           Health = $health
           BusType = $infodrive.bustype
           AdapterSerial = $vol.adapterserialnumber
           SerialNumber = $infodrive.serialnumber
           Signature = $infodrive.signature
```

```
            FileSystem = $vol.filesystem
            FileSystemLabel = $vol.filesystemlabel
            DiskNumber = $infodrive.disknumber
            NumberPartitions = $infodrive.numberofpartitions
            PartitionStyle = $infodrive.partitionstyle
            OperationalStatus = $infodrive.operationalstatus
            isBoot = $infodrive.isboot
            FirmwareVersion = $infodrive.firmwareversion
            Location = $infodrive.location
            Size = $infov.size
            Used = $infops.used
            Free = $infops.free
            Size_Friendly = get-friendlysize $infov.size
            Used_Friendly = get-friendlysize $infops.used
            Free_Friendly = get-friendlysize $infops.free
            '%_Free' = $free

        }
        $fsl = $vol.filesystemlabel
        $btype = $infodrive.bustype
        $friendly = $infodrive.friendlyname
        $name = "Label: "
        $name = -join($name,$fsl)
        $name = -join($name," `r${btype} ")
        $name = -join($name,$friendly)

        $tmp = get-friendlysize $infops.free
        $detail = $tmp
        $detail = -join($detail,' free of ')
        $tmp = get-friendlysize $infov.size
        $detail = -join($detail,$tmp)
        $detail = -join($detail,' total.')
        $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


        $script:healthres = [pscustomobject]@{

            Host = $comp
            OT = $OT
            Query = 'Get-Health'
            Component = 'Space'
            Health = $health
            Name = $name
            Detail = $detail
            Serial_Mac = $infodrive.serialnumber

        }

        $script:health += $script:healthres
        $script:space += $res

    }
    return $script:space | out-null
}

function Get-Basic {
      param([string]$comp)
    $Servicetag = (get-ciminstance -computername $comp win32_bios).serialnumber
        $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


    $script:obj = get-computerinfo | select *

      $script:res = [pscustomobject]@{

            Host = $comp
            OT = $OT
            Query = 'Get-Basic'
            Chassis_Type = $script:obj.cschassisskunumber
            ChassisBootstate = $script:obj.cschassisbootupstate
            PowerSupplyState = $script:obj.cspowersupplystate
            Manufacturer = $script:obj.csmanufacturer
            ServiceTag = $servicetag
            Family = $script:obj.cssystemfamily
            SystemSKU = $script:obj.cssystemskunumber
            Model = $script:obj.csmodel
            PC_Type = $script:obj.cspcsystemtype
```

```
            Basic_Status = $script:obj.csstatus
            Thermal_Status = $script:obj.csthermalstate
      }

      $cshealth = "PSU: "
      $cshealth = -join($cshealth,$script:res.PowerSupplyState)
      $cshealth = -join($cshealth," `rThermalState: ")
      $cshealth = -join($cshealth,$script:res.thermal_status)

      $details = $script:res.family
      $details = -join($details," - ")
      $details = -join($details,$script:res.PC_Type)
      $chassis = $script:res.chassis_type
      $details = -join($details," (")
      $details = -join($details,$chassis)
      $details = -join($details,")")

      $script:healthres = [pscustomobject]@{

            Host = $script:res.host
            OT = $OT
            Query = 'Get-Basic'
            Component = 'Basic'
            Health = $cshealth
            Name = $script:res.model
            Detail = $details
            Serial_Mac = $servicetag
      }

      $script:Basics += $script:res
      $script:health += $script:healthres
}

function Get-Battery {
      param([string]$comp)
      $script:obj = get-ciminstance -computername $comp win32_battery | select *
          $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


      if($script:obj -ne $null) {

            $runtime = $script:obj.Estimatedruntime
            if($runtime -eq 71582788) {
                  $runtime = 'AC Power'
            }
            else {
                  $runtime = (new-timespan -minutes $runtime).tostring()
            }

            $bstatus = $script:BatteryStatus_map[[int]$script:obj.BatteryStatus]
            $chem = $script:Chemistry_map[[int]$script:obj.chemistry]

            $bathealth = "Battery Status: "
            $bathealth = -join($bathealth,"`r`t ${bstatus}")
            $bathealth = -join($bathealth,",`rEst. Charge: ")
            $charge = $script:obj.estimatedchargeremaining
            $bathealth = -join($bathealth,$charge)
            $bathealth = -join($bathealth,",`rEst. Runtime: ")
            $bathealth = -join($bathealth,$runtime)

            $batdetail = "Chemistry: "
            $batdetail = -join($batdetail,$chem)
            $batdetail = -join ($batdetail," `rVoltage: ")
            $voltage = $script:obj.designvoltage
            $batdetail = -join($batdetail,$voltage)


            $script:res = [pscustomobject]@{

                  Host = $comp
                  OT = $OT
                  Query = "Get-Battery"
                  Name = $script:obj.caption
                  Model = $script:obj.name
                  DeviceID = $script:obj.deviceid
                  Battery_Status = $bstatus
                  Availability = $script:availability_map[[int]$script:obj.availability]
```

```
                  Chemistry = $chem
                  Design_Voltage = $script:obj.designvoltage
                  Estimated_Charge = $script:obj.estimatedchargeremaining
                  Estimated_Runtime = $runtime

            }

            $script:healthres = [pscustomobject]@{

                  Host = $comp
                  OT = $OT
                  Query = 'Get-Health'
                  Component = 'Battery'
                  Health = $bathealth
                  Name = $script:res.name
                  Detail = $batdetail
                  Serial_Mac = $script:res.deviceid
            }
      }
      else {
            $script:res = [pscustomobject]@{

                  Host = $comp
                  OT = $OT
                  Query = 'Get-Battery'
                  Name = 'NA'
                  Model = 'NA'
                  DeviceID = 'NA'
                  Battery_Status = 'No Batteries Detected'
                  Availability = 'NA'
                  Chemistry = 'NA'
                  Design_Voltage = 'NA'
                  Estimated_Charge = 'NA'
                  Estimated_Runtime = 'NA'

            }

            $script:healthres = [pscustomobject]@{

                  Host = $comp
                  OT = $OT
                  Query = 'Get-Health'
                  Component = 'Battery'
                  Health = 'No Batteries Detected'
                  Name = 'NA'
                  Detail = 'NA'
                  Serial_Mac = 'NA'

            }

      }

      $script:Battery += $script:res
      $script:Health += $script:healthres
}

function Get-OS {
      param(
  [string]$comp)

    $script:obj = get-computerinfo | select-object '*os*'
    $windows = get-computerinfo | select-object '*Windows*'
    $hotfixes = $script:obj.oshotfixes | sort-object -property hotfixid -descending | select-object -first 5
    $hotfixes = $hotfixes | select-object -expandproperty hotfixid
    $admins = get-localgroupmember -group "Administrators" | select-object Name
    $admin = "Admins - "
        $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


    foreach($a in $admins) {
      $admin = -join($admin," `r`n")
      $admin = -join($admin,$a.name)
    }

    $string = "Hotfixes - "
    foreach($h in $hotfixes){
        $string = -join($string,";`r`n ")
```

```
        $string = -join($string,$h)
    }

      $script:res = [pscustomobject]@{

        Host = $comp
        OT = $OT
        Query = 'Get-OS'
        OS_Name = $script:obj.osName
        OS_Version = $script:obj.osversion
        OS_Build = $script:obj.osbuildnumber
        OS_Serial = $script:obj.osserialnumber
        Last_5_Hotfixes = $string
        OS_LocalTime = $script:obj.oslocaldatetime
        OS_BootTime = $script:obj.oslastbootuptime
        OS_InstallDate = $script:obj.osinstalldate
        OS_Org = $script:obj.osorganization
        OS_Arch = $script:obj.osarchitecture
        OS_SP_Major_Version = $script:obj.osservicepackmajorversion
        OS_SP_Minor_Version = $script:obj.osservicepackminorversion
        OS_Status = $script:obj.osstatus
        Admin_Group = $admin

      }
      $over = $script:res.OS_Version
      $obuild = $script:res.OS_Build

      $name = -join("OS Name: ",$script:res.os_name)
      $name = -join($name,"`rOS_Version: ")
      $name = -join($name,$over)
      $name = -join($name,"`rOS_Build: ")
      $name = -join($name,$obuild)

      $script:healthres = [pscustomobject]@{
            Host = $script:res.host
            OT = $OT
            Query = 'Get-Health'
            Component = 'OS'
            Health = $script:res.os_status
            Name = $name
            Detail = $admin
            Serial_Mac = $script:res.os_serial

      }


      $script:health += $script:healthres
      $script:OS = $script:res
}

function Get-BIOS {

    $script:obj = get-computerinfo | select-object '*BIOS*'
    $mobo = get-ciminstance -computername $comp win32_baseboard
    $version = ($script:obj.biosversion)
        $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


    $script:res = [pscustomobject]@{

        Host = $comp
        OT = $OT
        Query = 'Get-BIOS'
        BIOS_Version = $version
        BIOS_Release = $script:obj.biosreleasedate
        BIOS_Primary = $script:obj.biosprimarybios
        BIOS_Serial = $script:obj.biosseralnumber
        BIOS_Firmware_Type = $script:obj.biosfirmwaretype
        SMBIOS_Present = $script:obj.biossmbiospresent
        SMBIOS_Version = $script:obj.biossmbiosbiosversion
        BIOS_Status = $script:obj.biosstatus
        Motherboard_Status = $mobo.status
        MB_Serial = $mobo.serialnumber
        MB_Version = $mobo.version
        MB_Product = $mobo.product

    }
```

```
      $detail = "BIOS FirmwareType: "
      $firm = $script:res.BIOS_Firmware_Type
      $release = $script:res.BIOS_Release
      $detail = -join($detail,$firm)
      $detail = -join($detail,"`rBIOS Release: ")
      $detail = -join($detail,$release)

      $tmp = $script:res.bios_version

      $hname = -join("Bios Name: ",$tmp)
      $tmp = $mobo.product
      $hname = -join($hname,"`rMotherboard Name: ")
      $hname = -join($hname, $tmp)
      $tmp = $mobo.version
      $hname = -join($hname, ", Version ${tmp}")

      $bhealth = "BIOS Status: "
      $bstatus = $script:res.bios_status
      $mstatus = $mobo.status
      $bhealth = -join($bhealth,$bstatus)
      $bhealth = -join($bhealth,"`rMotherboard Status: ")
      $bhealth = -join($bhealth,$mstatus)

      $script:healthres = [pscustomobject]@{
              Host = $script:res.host
              OT = $OT
              Query = 'Get-Health'
              Component = 'BIOS'
              Name = $hname
              Detail = $detail
              Serial_Mac = $script:res.bios_serial
              Health = $bhealth
          }


          $script:health += $script:healthres

      $script:BIOS += $script:res
  }

  Function Get-CPU {
        param(
    [string]$comp)

      $script:obj = get-ciminstance -classname win32_processor | select-object *
      $fan = get-ciminstance -classname win32_fan | select-object *
      $fandetail = ""
      $numfan = $fan.count
          $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag

      foreach($f in $fan) {
              $mapped = $script:availability_map[[int]$f.availability]
            $fandetail = -join($fandetail,"`r`t${mapped}")
            $fandetail = -join($fandetail,", ")
            $fandetail = -join($fandetail,$f.status)
        }
        $mapped = $script:cpustatus_map[[int]$script:obj.cpustatus]
      $health = -join("CPU Status: `r`t",$mapped)
      $cstatus = $script:obj.status
      $health = -join($health,"; ${cstatus}")
      if($numfan -gt 1) {
            $health = -join($health,"`rFan_Status: ${numfan} Fans")
      }
      else {
        $health = -join($health,"`rFan_Status: ${numfan} Fan")
        }
      $health = -join($health,$fandetail)

      $serial = $script:obj.SerialNumber.value
      if($serial -eq $null) {
        $serial = "Unavailable"
      }

      $script:res = [pscustomobject]@{

          Host = $comp
          OT = $OT
```

```
        Query = 'Get-CPU'
        Model = $script:obj.name
        Caption = $script:obj.caption
        Device_ID = $script:obj.deviceid
        PartNumber = $script:obj.partnumber
        ProcessorID = $script:obj.processorid
        SerialNumber = $script:obj.serialnumber
        Type = $script:ProcessorType_map[[int]$script:obj.processortype]
        Architecture = $script:architecture_map[[int]$script:obj.architecture]
        Family = $script:family_map[[int]$script:obj.family]
        CPU_Status = $script:cpustatus_map[[int]$script:obj.cpustatus]
        CPU_Fan = $fandetail
        Availability = $script:availability_map[[int]$script:obj.availability]
        Socket = $script:obj.socketdesignation
        Upgrade_Method = $script:UpgradeMethod_map[[int]$script:obj.upgrademethod]
        '#_Cores' = $script:obj.numberofcores
        Current_Voltage = $script:obj.currentvoltage
        Load_Percent = ($script:obj.loadpercentage)
        Max_ClockSpeed = $script:obj.Maxclockspeed
        Current_ClockSpeed = $script:obj.currentclockspeed
        L2_CacheSize = $script:obj.l2cachesize
        L3_CacheSize = $script:obj.l3cachesize
        ThreadCount = $script:obj.threadcount


    }
    $detail = "Device_ID - "
    $detail = -join($detail, $script:res.device_id)
    $detail = -join($detail," `r`n")
    $detail = -join($detail,"Caption - ")
    $detail = -join($detail,$script:res.caption)
    $serial = "ProcessorID - "
    $serial = -join($serial,$script:res.ProcessorID)
    $serial = -join($serial," `r`n")
    $serial = -join($serial,"Serial - ")
    $serial = -join($serial,$script:res.serialnumber)

    $script:healthres = [pscustomobject]@{

            Host = $script:res.host
            OT = $OT
            Query = 'Get-Health'
            Component = 'CPU'
            Health = $health
            Name = $script:res.Model
            Detail = $detail
            Serial_Mac = $serial

        }

       $script:health += $script:healthres
    $script:CPU += $script:res
}

Function Get-RAM {
      param([string]$comp)

      $script:mem = get-ciminstance win32_physicalmemory
      $script:memarray = get-ciminstance win32_physicalmemoryarray | select *
          $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


      foreach($m in $script:mem) {

            $manu = $script:ManufacturerRAM_Map[[string]$m.manufacturer]
            if($manu -eq $null) {
                  $manuMap = $m.manufacturer
            }
            else {
                  $manuMap = $manu
            }

            $script:res = [pscustomobject]@{

                  Host = $comp
                  OT = $OT
                  Query = 'Get-RAM'
```

```
                Name = $m.tag
                Part_Number = $m.partnumber
                Serial_Number = $m.serialnumber
                Manufacturer = $manumap
                Form_Factor = $script:FormFactor_Map[[int]$m.formfactor]
                Capacity = $m.capacity
                Capacity_friendly = get-friendlysize $m.capacity
                Data_Width = $m.datawidth
                Memory_Type = $script:MemoryType_map[[int]$m.memorytype]
                Type_Detail = $m.typedetail
                Memory_ErrorCorrection = $script:MemoryErrorCorrection_map[[int]$memarray.memoryerrorcorrection]
                Memory_Use = $script:use_map[[int]$memarray.use]
                Speed = $m.speed
                Config_Clockspeed = $m.configuredclockspeed
                Config_Voltage = $m.configuredvoltage
                Location = $m.devicelocator
            }

            $name = $script:res.name
            $name = -join($name,"; ")
            $name = -join($name,$script:res.location)
            $pnumber = $script:res.Part_Number
            $detail = "Manufacturer: "
            $detail = -join ($detail,$manumap)
            $detail = -join($detail,"`rPart_Number: ${pnumber}")

            $script:healthres = [pscustomobject]@{

                Host = $script:res.host
                OT = $OT
                Query = 'Get-Health'
                Component = 'RAM'
                Health = 'Unreported; memtest for memory problems'
                Name = $name
                Detail = $detail
                Serial_Mac = $script:res.Serial_number
            }


            $script:health += $script:healthres
            $script:ram += $script:res
        }
}

Function Get-GPU {
      param(
  [string]$comp)

    $script:vid = get-ciminstance win32_videocontroller
        $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag


    foreach($v in $script:vid) {

        $script:res = [pscustomobject]@{

            Host = $comp
            OT = $Ot
            Query = 'Get-GPU'
            GPU = $v.name
            Driver = $v.driverversion
            DriverDate = $v.driverdate
            Adapter_RAM_GB = $v.adapterram/1GB
            RAM_Type = $script:VideoMemoryType_map[[int]$v.videomemorytype]
            Adapter_DAC_Type = $v.adapterdacetype
            Current_VideoMode = $v.videomodedescription
            Video_Processor = $v.videoprocessor
            Availability = $script:availability_map[[int]$v.availability]
            GPU_Status = $v.status
            Dither_Type = $script:DitherType_map[[int]$v.dithertype]
            Video_Architecture = $script:videoarchitecture_map[[int]$v.videoarchitecture]

        }
        $driver = $script:res.driver
        $ddate = $script:res.driverdate
        $gstatus = $script:res.GPU_Status
        $gavail = $script:res.availability
```

```
        $detail = "Driver: "
        $detail = -join($detail,$driver)
        $detail = -join($detail,"`rDriverDate: ")
        $detail = -join($detail,$ddate)

        $hdetail = "Status: "
        $hdetail = -join($hdetail,$gstatus)
        $hdetail = -join($hdetail,"`rAvailability: ")
        $hdetail = -join($hdetail,$gavail)

        $script:healthres = [pscustomobject]@{

            Host = $script:res.host
            OT = $OT
            Query = 'Get-Health'
            Component = 'GPU'
            Health = $hdetail
            Name = $script:res.gpu
            Detail = $detail
            Serial_Mac = 'Unavailable for GPUs'

        }

        $script:health += $script:healthres

        $script:GPU += $script:res
    }
}

function Get-Network {
      param(
  [string]$comp)

         $OT = (get-ciminstance -computername $comp win32_systemenclosure).smbiosassettag



    $script:net = get-ciminstance win32_networkadapter | where {$_.physicaladapter -like 'True'} | select *
    foreach($script:n in $script:net){

        $script:a = get-netadapter | where InterfaceIndex -eq $script:n.interfaceindex | select *
        $script:c = get-ciminstance win32_networkadapterconfiguration | where interfaceindex -eq $script:n.interfaceindex | se

        $script:res = [pscustomobject]@{

            Host = $comp
            OT = $OT
            Query = 'Get-Network'
            Name = $script:n.productname
            NetConnectionID = $script:n.netconnectionid
            NetEnabled = $script:n.netenabled
            Health = $script:NetConnectionStatus_map[[int]$script:n.netconnectionstatus]
            Device_ID = $script:n.deviceid
            Availability = $script:availability_map[[int]$script:n.availability]
            LinkSpeed = $script:a.linkspeed
            Speed = $script:n.speed
            Speed_Friendly = get-friendlysize $script:n.speed
            Adapter_TypeID = $script:AdapterTypeID_map[[int]$script:n.AdapterTypeID]
            Installed = $script:n.installed
            InterfaceIndex = $script:n.interfaceindex
            MAC = $script:n.macaddress
            Manufacturer = $script:n.manufacturer
            PhysicalAdapter = $script:n.physicaladapter
            ServiceName = $script:n.servicename
            Driver_Version = $script:a.driverversion
            Driver_Date = $script:a.driverdate
            SCSI_Interface = $script:a.iscsiinterface
            DHCP_Enabled = $script:c.DHCPEnabled
            DHCP_LeaseObtained = $script:c.dhcpleaseobtained
            DHCP_LeaseExpires = $script:c.DHCPLeaseExpires

        }
        $cid = $script:res.netconnectionid
        $atype = $script:res.adapter_typeid

        $detail = "Connection_ID: "
        $detail = -join($detail,$cid)
```

```
        $detail = -join($detail,"`rAdapter_Type:")
        $detail = -join($detail,$atype)

        $script:healthres = [pscustomobject]@{

            Host = $script:res.host
            OT = $OT
            Query = 'Get-Health'
            Component = 'Network'
            Health = $script:res.health
            Name = $script:res.name
            Detail = $detail
            Serial_Mac = $script:n.macaddress

        }
        $script:health += $script:healthres
        $script:network += $script:res
    }

    return $script:network | out-null
}

function run-stats {
      param(
  [string]$comp)

  $script:Task = 'Get-Battery'
  get-battery -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-Basic'
  Get-Basic -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-BIOS'
  get-BIOS -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-CPU'
  get-CPU -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-GPU'
  get-GPU -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-OS'
  get-OS -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-Network'
  get-Network -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-RAM'
  get-RAM -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  $script:Task = 'Get-Space'
  get-space -comp $comp
  Write-Color "`t${script:Task}" -color darkgray

  Write-Color "`tCreating Health Report" -color darkgray
  write-host ""

  $script:Task = 'Run-All'

  $trash = [pscustomobject]@{
      This = 'is just'
      The = 'hacky'
      Garbage = 'to delete'
      Query = 'TRASH'
  }

  $all = @(
    $trash
    $script:Health
```

```
    $script:Basics
    $script:Battery
    $script:OS
    $script:BIOS
    $script:CPU
    $script:RAM
    $script:GPU
    $script:Network
    $script:space

    )
  $script:allstats += $all

  return $script:allstats | out-null
}

#Main program ================================================================================================================
cls

$script:psver = $psversiontable.PSVersion
if (!($script:psver.Major -gt 6)) {

    Write-color "Warning! ","This script was written for PowerShell version 7 or greater." -color red,yellow
    Write-Color "You have version ","${script:psver}" -color yellow,red
    Write-Color "Some formulas and commands may not behave as expected. " -color yellow
    Write-Color "For the best experience, please make sure you run this script in Powershell version 7.0 or greater. " -color
}

$comp = hostname

Write-Color "Gathering data on ","$comp . . ." -color blue,green
Write-Host ""
run-stats -comp $comp

Write-Color "Please select which ","results"," to display: " -color blue,green,blue
Write-Host ""
do {
      Write-Color "`r`n`t[1]"," - Battery" -color green,blue
      Write-Color "`t[2]"," - Basic" -color green,blue
      Write-Color "`t[3]"," - BIOS" -color green,blue
      Write-Color "`t[4]"," - CPU" -color green,blue
      Write-Color "`t[5]"," - GPU" -color green,blue
      Write-Color "`t[6]"," - OS" -color green,blue
      Write-Color "`t[7]"," - Network" -color green,blue
      Write-Color "`t[8]"," - RAM" -color green,blue
      Write-Color "`t[9]"," - Space" -color green,blue
      Write-Color "`t[10]"," - Overall Health" -color green,blue
      Write-Color "`t[W]"," - Write to file" -color cyan,green
      Write-Color "`t[Q]"," - Quit" -color yellow, red
      Write-Color "`Results will appear in another window; closing the window will not affect this session." -color cyan
      Write-Host ""
      $option = read-host -prompt "Please pick your results"

      switch ($option) {

            '1'{
                  $choice = $script:Battery
                  $title = "Battery Stats'"
            }
            '2'{
                  $choice = $script:basics
                  $title = "Basic Stats'"
            }
            '3'{
                  $choice = $script:BIOS
                  $title = "BIOS Stats"
            }
            '4'{
                  $choice = $script:cpu
                  $title = "CPU Stats"
            }
            '5'{
                  $choice = $script:gpu
                  $title = "GPU Stats"
            }
            '6'{
                  $choice = $script:os
```

```
                $title = "OS Info"
            }
            '7'{
                $choice = $script:network
                $title = "Network Adapters"
            }
            '8'{
                $choice = $script:RAM
                $title = "RAM Stats"
            }
            '9'{
                $choice = $script:space
                $title =  "Storage Stats"
            }
            '10'{
                $choice = $script:health
                $title =  "Overall Health"
            }
            'W'{

                Write-Color "You have chosen to write to file." -Color cyan
                $rep_char = '_'

                do {
                    $confirm = ""

                    $directory = read-host -prompt "Please enter your desired directory/filepath"
                    Write-color "You have entered ","${directory}" -color blue,cyan
                    Write-Color "Please confirm by hitting ","[Y]","es or any key for ","No." -color blue,green,blue,red
                    $confirm = read-host
                    } while($confirm -ne 'Y')

                do {

                    $confirm = ""

                    $filename = read-host -prompt "Please enter your desired filename. (Please note that spaces will be re
                    $filename = $filename -replace ' ',$repchar

                    Write-color "You have entered ","${filename}" -color blue,cyan
                    Write-Color "Please confirm by hitting ","[Y]","es or any key for ","No." -color blue,green,blue,red
                    $confirm = read-host
                    } while($confirm -ne 'Y')

                    $filename = $filename.trimend(".xlsx")

                write-color "Filename ","${filename}"," will be saved at ","${directory}" -color blue,green,blue,green
                $directory = $directory.trimend("\")

                $script:checkpath = "${directory}\${Filename}.xlsx"

                write-excel

                Write-Color "Report written to ","${script:CheckPath}" -color blue,cyan
                Write-Color "Goodbye!"
                exit
            }
            'Q'{
                Write-Color "Goodbye!"
                exit
            }
            default{
                Write-Color "${script:continue}"," is an invalid selection."," `rPlease try again." -color red,yellow,blue
            }
        }

        $choice | out-gridview -title $title
        Write-Host ""

        Write-Color "Displaying current ${title}" -Color blue
        Write-Host ""
        $script:continue = $null
        write-color "Would you like to continue? ","[Y]","es or ","any key"," for no: " -color blue,green,blue,red,yellow
        $script:continue = read-host
        Write-Host ""

    } while($script:continue -eq 'Y')
```