## Prerequisites

Some familiarity with programming will be helpful, preferably in Python 3
Access to a Windows 7, 8 or 10 computer and permission to install software
A USB flash drive between 8GB and 32GB that can be reformatted

## Safety

Follow the formatting procedures CAREFULLY to avoid doing damage to your computer's hard drive

## Trademark and copyright info

Windows is a trademark of Microsoft, Inc.
Python is  trademark of the Python Software Foundation
The Toy Story transcript is based on the film, copyright 1995, Disney

## Materials

Windows 7, 8 or 10 computer with Python 3 and one free USB port
Python 3 (available at https://www.python.org/)
A USB flash drive between 8 and 32GB in size
Sample Python programs
Sample text files

Images from Stoffregen, Paul, https://www.pjrc.com/tech/8051/ide/fat32.html (used with permission)

## How safe are your files?

Everyone has computer files now. When you wrote that 'A' paper in English class, you used a word processor on a computer. Did you save a copy on a USB flash drive for safety? *If you did not, you ought to get into that habit.* You turned it in and amazed the teacher. Then you deleted the file from the USB drive. Maybe you also had some secrets you erased from that drive? Do you keep an electronic diary? Perhaps you wrote about your secret crush on the new transfer student, copied the file from the USB flash drive to your computer at home and then deleted the file. Is it really safe from prying eyes? Can anyone retrieve it? This project will help you think like a **computer forensic expert** and learn how to get back deleted data.

Computer file systems are a lot like the filing cabinets that business use (or maybe you have one at home) to keep track of a lot of documents. There are several kinds that you usually find on modern computers. Which one your computer has will depend on the age of the computer and the version/manufacturer of the operating system.

| | |
|---|---|
| Apple Macintosh (old) | HFS, HFS+ |
| Apple Macintosh (current) | HPFS |
| Apple iOS | APFS |
| Linux (including Android, Raspberry Pi and other systems) | EXT2, EXT3, EXT4 |
| Microsoft Windows XP | FAT32 *or* NTFS |
| Microsoft Windows Vista, 7, 8, 10 | NTFS |

*Figure 1 - Common file systems by computer type*

Sure, if you put all of Figure 1 in a can you would have alphabet soup but those letters do mean specific things. If you are an technology archeologist you may also have seen computers used by the dinosaurs - old Apple ][s, IBM PCs running MS-DOS, Amigas, Commodores, Coleco ADAMs, CP/M machines and others. As long as we have had computers, we have had a way of keeping track of the data stored in it. Each of these file systems are different from the others but they also have a lot in common. You can find out more about the history of each of these using an internet search. And yes, you will be retrieving data from files stored on file systems at a provider's data center.

So if every computer manufacturer uses a different file system, how can you write a file on a flash drive using your Microsoft Windows laptop and your friend can read it on her Apple? We actually owe that to an ancient file system (in personal computer history), the **File Access Table** or **FAT** file system. This is a simple storage model that first appeared in 1980 in an operating system for computers using the then-popular Intel 8086 microprocessor. It was originally sold as QDOS (Quick and Dirty Operating System), purchased by Microsoft in 1981 and renamed to MS-DOS and licensed to IBM as the system for their first personal computers. Files written in MS-DOS were stored on floppy disks, so the storage structures had to be simple, reliable, and fast.

Over the years we have moved away from floppy disks and onto much larger and more reliable forms of media. So if we have better media and better ways of storing files, why do we care about FAT?

**The skinny on FAT**

FAT file systems offer a huge advantage to drives like USB flash drives. We want to be able to move them from computer to computer without worrying about the brand. Flash storage is often very slow (writing, especially) so we want something simple. We are also careless about ejecting our flash drives before removing them from the computer, so we want some reliable. We already had FAT - so why not keep it around? Variations of FAT have been adapted as the industry standard for small portable drives.

*Figure 2 - Storage in a FAT drive is a lot like storing items in stacked boxes on a shelf*

Just like a smart shopkeeper knows where things are kept in his store, the file system on the flash drive knows where the data is. The directory tells it where to look and the rest of the disk structures describe where all of the data lives on the drive. In wizarding movies the wand may choose the wizard, but a scientist chooses her data thanks to the file system on her computer.

## Retrieving lost or deleted files

If the directory on your flash drive forgets about your file, it is not actually lost. That is an advantage if something happens to your drive - you can get it back. It is a big problem if you deleted a sensitive file and you do not want anyone recovering it. Just how difficult is it to **undelete** a file from a flash drive? How would you go about it? In this project you will research a bit about computer file systems, learn how files are stored on your USB flash drive and see how you might be able to get back a file that you thought was lost and gone forever. You can write a program yourself to recover lost files. This project idea uses code written in Python 3 which is freely available for just about every computer. The sample programs will take you part of the way but you will be able to embellish them and make them your own. If you prefer C or C++, by all means go ahead and use that.

| | |
|---|---|
| Size in bytes | 32 GB = 34,359,738,368 |
| Number of sectors | 512 bytes / sector = 67,108,864 sectors |
| Number of clusters | 16 KB / cluster = 2,097,152 clusters |

*Figure 3 - A common 32 GB flash drive is managed by grouping amounts of storage together into a manageable number of pieces. The FAT file system thinks of the drive as a collection of clusters.*

In order to write a program that undeletes files from your drive we need to understand how the drive stores your files. The drive is a collection of many **sectors**, each typically 512 bytes of data. There are 2 sectors for each kilobyte of data. If you consider that one megabyte (MB) is 1024 kilobytes (KB) - in the computer world of base 2, 1024 is more easily represented than 1000 - and one gigabyte (GB) is 1024 megabyte, then a 32 GB drive has 32 x 2 x 1024 x 1024 or 67,108,864 sectors. That is a lot, so to make it easier to handle, the drive is simplified by taking a number of sectors together and calls that a **cluster**. A cluster is very often 16 KB, or 32 sectors.

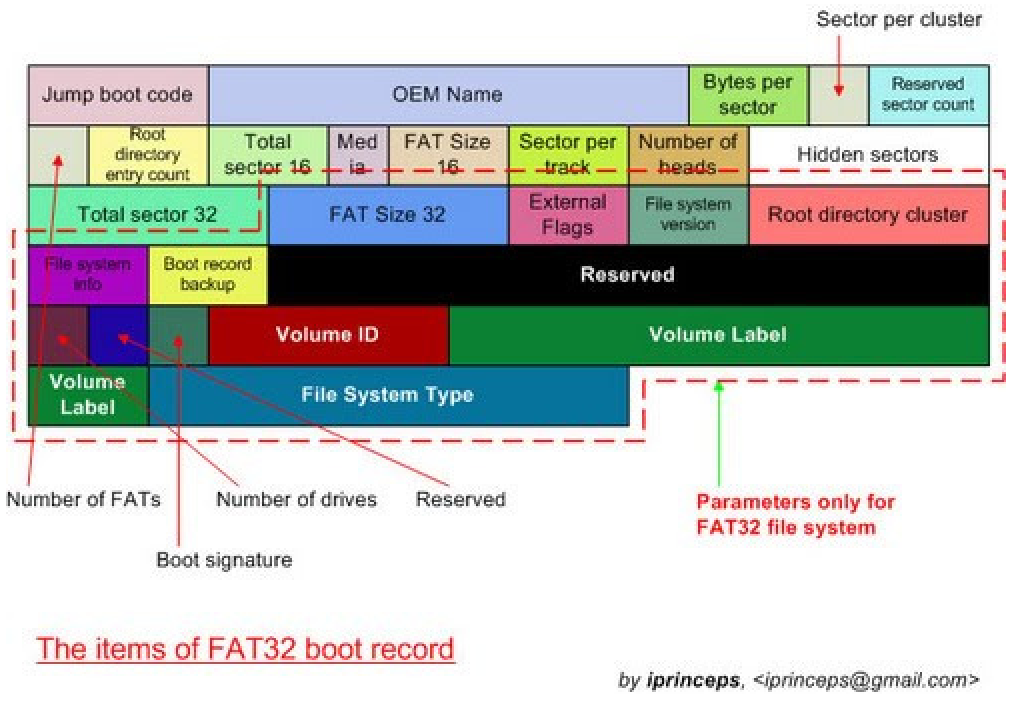


*Figure 4 - The general structure of a FAT drive. A 'boot block' tells the computer that this is a FAT drive, the 'directory' tells where files start, the 'file allocation table' describes how longer files flow from place to place, and the rest of the drive is simply blocks of storage.*

The first sector of the flash drive contains critical information to tell your computer what the drive is and how it is formatted. Data within the sector tells the computer that this is a FAT drive. There is also a piece of system programming called a **boot block** that can tell your computer what to do when it is first powered on. That rarely happens from a flash drive since your computer has a hard drive to boot from. Your hard drive has this same type of information.

There could be some spare sectors allocated right after this boot sector, then one or more **File Allocation Table**s, or FATs are found on the drive. Since the file system is designed to not lose any of your data, it has multiple copies of the FAT. If one copy gets damaged (which was very common in the days of the floppy) the file system can use one of the duplicates. We know how many there are because the boot sector told us. The FAT tells us which clusters are in use and which are available. On ancient MS-DOS systems, each cluster number was stored in 1.5 bytes, or 12 bits to save space. Those standard FAT systems are also referred to as FAT12. As disks became larger, the FAT allocation was expanded to 16 bits and the system was known as FAT16. Modern systems use FAT32 which can handle drives up to 2^32 - 1, or about 4 million clusters. *Since flash drives are nearly always formatted as FAT32, the example code and discussion will be limited to supporting FAT32 drives. We also do not handle drives that are broken into several pieces. Your hard drive in Windows might show up as both drives C: and D:. That is called **partitioning** and flash drives can have that too. It is more complicated to handle a partitioned drive - the example code will not work on one of those. We also do not handle creating directories on the drive. A useful undeleter certainly needs to but the code to do that would be rather hard to understand. After you work with this code for a while, that can be the first improvement you make!*

We said earlier that a cluster is 16K, so drives can be huge in theory. In practice the format is limited to 32GB on many computers. Why? Until recently the concept that anyone would ever have a drive even close to that size was crazy. Now your cellphone probably has more storage than that!

Immediately following the FATs is the root directory of the drive. If your drive loads as “E:\” in Windows or “/Volumes/Science” on your Macintosh, then this is where that directory lives. It will be one or more sectors and each file has a specific data structure that describes it. Using that structure, the file system knows the file’s name and the location of the first cluster of data.

Finally, the rest of the drive is considered to be available space broken up into clusters. Using the directory and the FAT, the file system takes your large spreadsheet full of data you collected in the science lab and stores it in as many clusters as needed. The directory gets the name you called it and associates that with the cluster number of the first 16 KB of data. The FAT is then updated with the location of the second cluster if needed, then a third and so on until the whole file is stored. To read this file, the file system need only to find the file name, locate the first cluster and start following the sequence from cluster to cluster. Sound easy?

**I did not mean to delete that!**

The worst thing possible has happened - you just accidentally deleted a whole semester’s worth of research data. How much mouse food did that mouse consume? Gone. How fast did it run the maze? Gone. Now what?

Fortunately the FAT file system was designed for really slow and somewhat unreliable media. When you delete something it is not really gone. The file system locates the directory entry and marks it as deleted. It does this by replacing the first character of the file name with a character you cannot type from the keyboard. Characters in personal computers are in **ASCII** format, and characters that you can type and read are numbered from 32 to 127. A character is 8 bits in size though, so it can actually range from 0 to 255 (0x00 to 0xFF in hexadecimal or 00000000 to

11111111 in binary). A FAT file system will mark a file as deleted by changing the first character of the file name to 229 (0xE5 in hexadecimal or 11100101 in binary) which is neither recognizable as a character you know or typable on the keyboard. It then removes the entries in the FAT so that the space can be reused, If you realize your mistake and undelete the file immediately, all of your data is still there and undamaged. If, however, you write a new file on the drive first and then realize your mistake, you have to cross your fingers and hope that the data is still there. This attribute of FAT file systems is what we're going to take advantage of to write our undelete program.

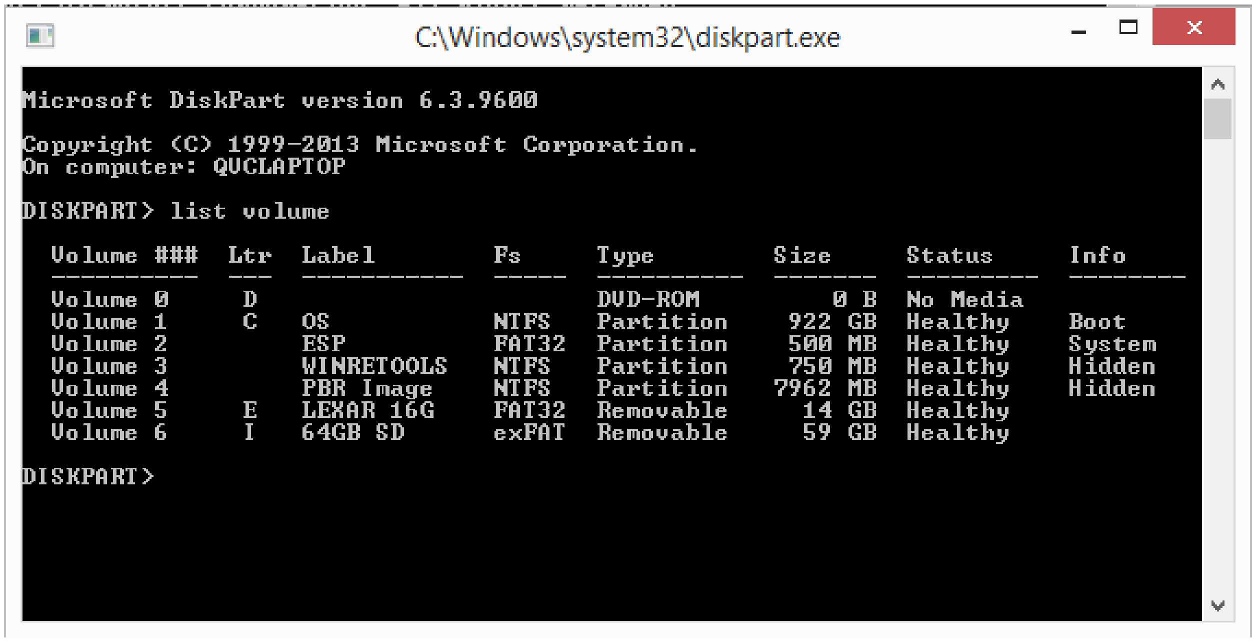
*File systems other than FAT work in conceptually similar but technically very different ways to handle file allocation and deletion. The tricks used to undelete files in a FAT file system will NOT work on a standard Windows NTFS or an Apple HPFS file system. Besides, experimenting with the hard drive in your computer is pretty risky and a great way of losing your data. That is why these experiments are written for FAT, the file system usually used on flash drives.*

(detailed explanations and charts of the data structures follow, then code snippets)

Step 1 - To try this experiment you need Python 3 on a Windows computer and a USB flash drive between 8 and 32 GB in size. Start by installing Python. If you do not have it already, get permission from your parents or the school, and go to https://www.python.org/downloads/ to get the latest version for Windows.

Step 2 - Once you have Python installed you also need PyWin32. That is a module that allows you to access Windows functions from within Python. For this step you need administrator access to your computer. The best way in Windows 8 or Windows 10 is to hold the 'Windows' key down and press 'X'. this will bring up a little menu on the lefthand side. Select "Command Prompt (Admin)" from that menu. If you do not see that or you cannot get access, you may need to ask your school's IT crew for help. Bring up the command prompt window.

Step 3 - On some computers you can simply  use the command "pip3 install pypiwin32" to complete the installations. On others you may need to change into the directory that contains the 'pip3' command. Try "cd /", followed by "dir". If you see a directory called "Python34" or "Python35" you found it. You can find 'pip3' by using the command 'cd \Python35\Scripts' (substituting in whatever directory you actually found). If that doesn't work, Python might also be in your '\Program Files (x86)' or '\Program Files' directory. Installing Python extensions can be confusing so ask for help if you need it.

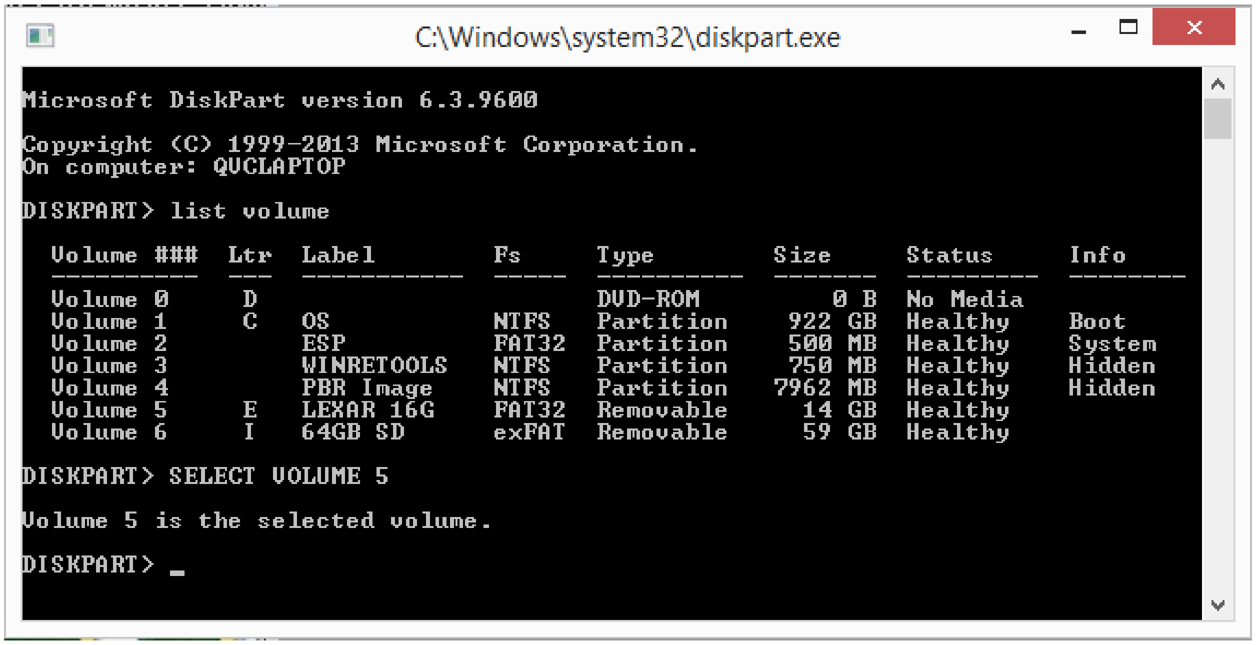


C:\Windows\system32\diskpart.exe

```
Microsoft DiskPart version 6.3.9600

Copyright (C) 1999-2013 Microsoft Corporation.
On computer: QVCLAPTOP

DISKPART> list volume

  Volume ###  Ltr  Label        Fs     Type        Size     Status     Info
  ----------  ---  -----------  -----  ----------  -------  ---------  --------
  Volume 0     D                       DVD-ROM         0 B  No Media
  Volume 1     C   OS           NTFS   Partition    922 GB  Healthy    Boot
  Volume 2         ESP          FAT32  Partition    500 MB  Healthy    System
  Volume 3         WINRETOOLS   NTFS   Partition    750 MB  Healthy    Hidden
  Volume 4         PBR Image    NTFS   Partition   7962 MB  Healthy    Hidden
  Volume 5     E   LEXAR 16G    FAT32  Removable     14 GB  Healthy
  Volume 6     I   64GB SD      exFAT  Removable     59 GB  Healthy

DISKPART>
```

*Figure 4 - Using Windows' DISKPART utility to display the available volumes*

Step 4 - Insert your USB flash drive in an available USB port. In Windows *(this project has been tested on Windows 8 and should work on Windows 7 and Windows 10)* bring up a command prompt. Enter "DISKPART" and press Return. At the "DISKPART>" prompt, enter "LIST VOLUME". This will display all of the drives available to your computer. In that list you should be able to easily identify your USB drive. It will be listed as FAT32 in the "Fs" column, the label should reflect what you think the drive is named. Most importantly, the size will be slightly smaller than what the drive is marked. In figure 5, the 16GB drive is shown as volume 5, or drive E:.

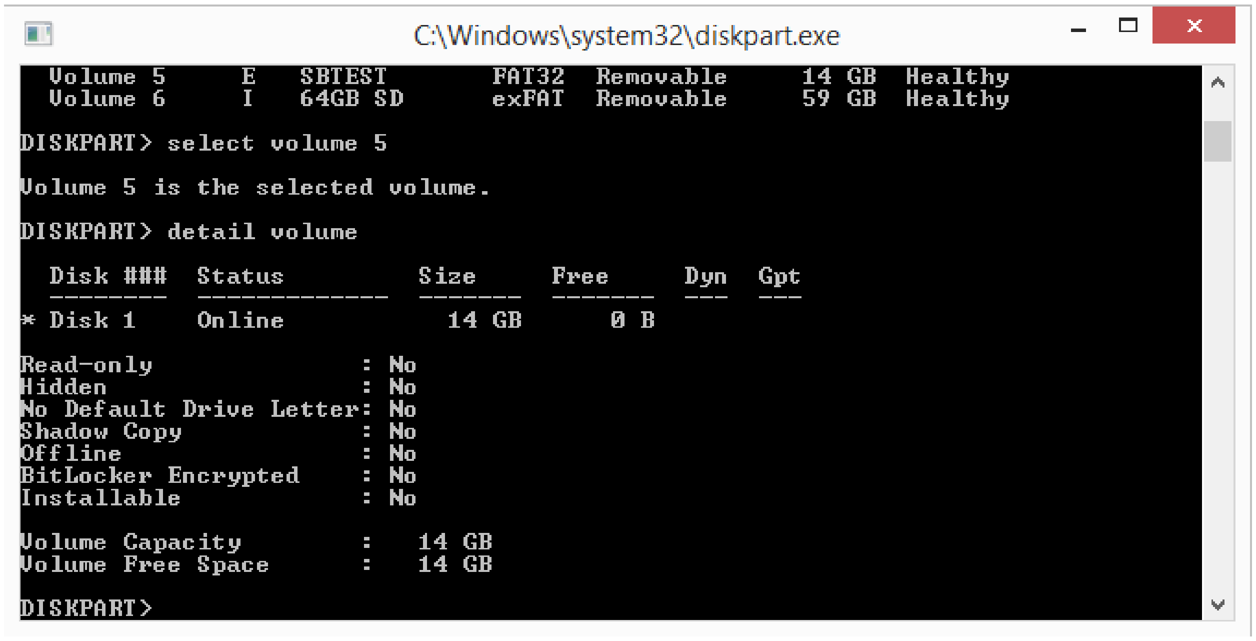


C:\Windows\system32\diskpart.exe

```
Microsoft DiskPart version 6.3.9600

Copyright (C) 1999-2013 Microsoft Corporation.
On computer: QVCLAPTOP

DISKPART> list volume

  Volume ###  Ltr  Label        Fs     Type        Size     Status     Info
  ----------  ---  -----------  -----  ----------  -------  ---------  --------
  Volume 0     D                       DVD-ROM         0 B  No Media
  Volume 1     C   OS           NTFS   Partition    922 GB  Healthy    Boot
  Volume 2         ESP          FAT32  Partition    500 MB  Healthy    System
  Volume 3         WINRETOOLS   NTFS   Partition    750 MB  Healthy    Hidden
  Volume 4         PBR Image    NTFS   Partition   7962 MB  Healthy    Hidden
  Volume 5     E   LEXAR 16G    FAT32  Removable     14 GB  Healthy
  Volume 6     I   64GB SD      exFAT  Removable     59 GB  Healthy

DISKPART> SELECT VOLUME 5

Volume 5 is the selected volume.

DISKPART> _
```

*Figure 5 - Use SELECT VOLUME in DISKPART to select the USB flash drive. Double check that you haven't accidentally selected the wrong drive.*

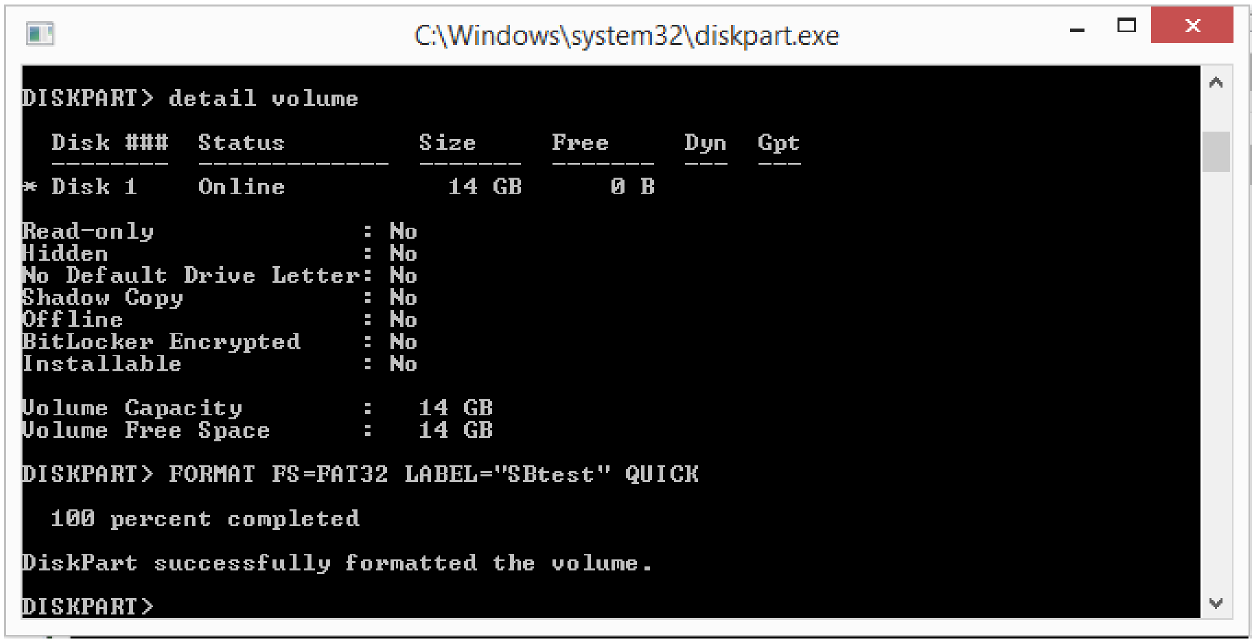
Step 5 - Enter the command "SELECT VOLUME 5", substituting in whatever volume number you just found. **BE VERY CAREFUL** that you have the correct volume number. Double check or have someone look over your shoulder. You do not want to format the wrong drive!


C:\Windows\system32\diskpart.exe

```
  Volume 5     E   SBTEST       FAT32  Removable     14 GB  Healthy
  Volume 6     I   64GB SD      exFAT  Removable     59 GB  Healthy

DISKPART> select volume 5

Volume 5 is the selected volume.

DISKPART> detail volume

  Disk ###  Status         Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
* Disk 1    Online           14 GB      0 B

Read-only              : No
Hidden                 : No
No Default Drive Letter: No
Shadow Copy            : No
Offline                : No
BitLocker Encrypted    : No
Installable            : No

Volume Capacity        :   14 GB
Volume Free Space      :   14 GB

DISKPART>
```

*Figure 6 - Use DETAIL VOLUME in DISKPART to verify you've selected the correct drive. Notice that the size of the drive is the same as your USB drive and much smaller than your hard drive. In this case, the 16GB drive looks to be 14GB which is about right.*

Step 6 - Enter the command "DETAIL VOLUME". You will see some details about the USB drive. It should confirm that you entered the correct volume number. Look at the drive size especially. A drive like the 16GB drive we used will appear to have 14GB, but certainly not the size of the much, much larger hard drive. If you're not sure you have the right drive, you can remove the flash drive and look at the choices available. Then reinsert the USB flash drive and try again. Notice which drive appears when you insert it.



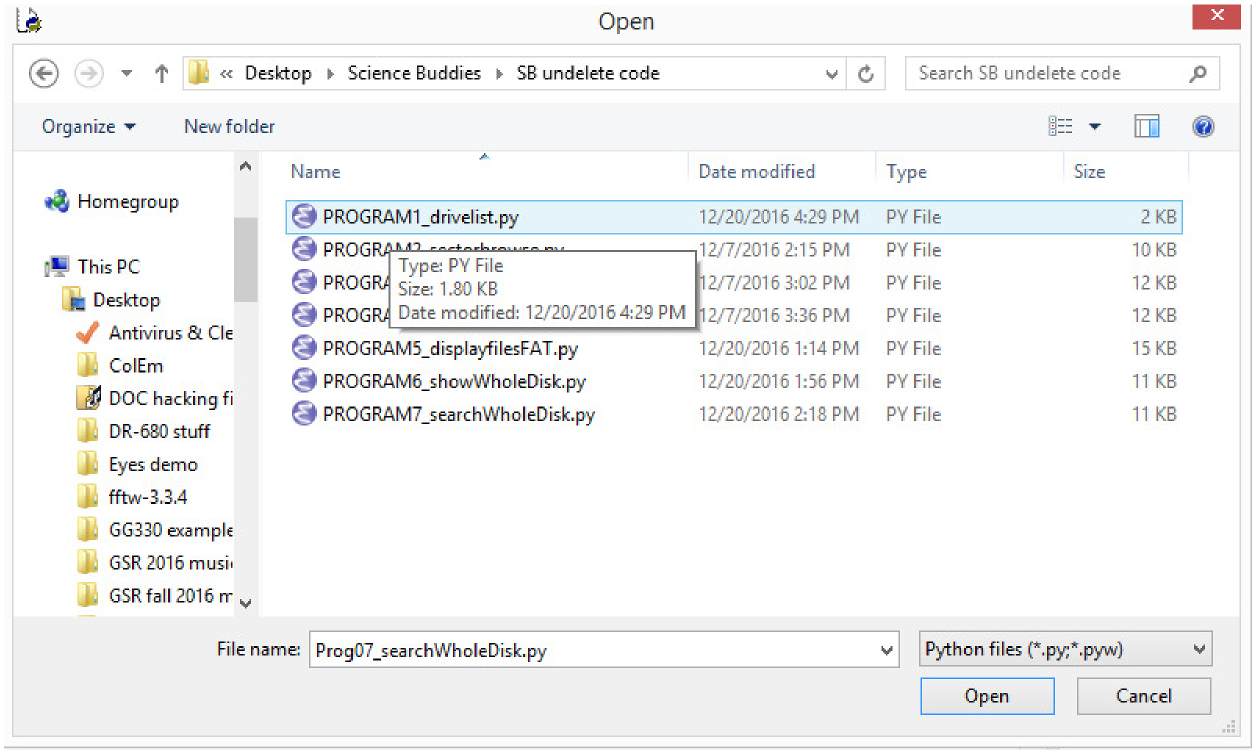
```
DISKPART> detail volume

  Disk ###  Status          Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
* Disk 1    Online           14 GB      0 B

Read-only              : No
Hidden                 : No
No Default Drive Letter: No
Shadow Copy            : No
Offline                : No
BitLocker Encrypted    : No
Installable            : No

Volume Capacity        :   14 GB
Volume Free Space      :   14 GB

DISKPART> FORMAT FS=FAT32 LABEL="SBtest" QUICK

  100 percent completed

DiskPart successfully formatted the volume.

DISKPART>
```

*Figure 6 - Formatting the flash drive. DISKPART uses the current selected drive. The command line tells it to use the FAT32 file system, name it "SBtest" (you'll call yours "Flash530") and simply create the directories without formatting the whole drive. On a 8GB to 32GB flash drive, that takes a really long time.*

Step 7 - Enter the command "FORMAT FS=FAT32 LABEL="Flash530" QUICK" to format the drive. This only takes a few seconds and you will see the USB flash drive LED blink.

Step 8 - Enter the command "EXIT" to leave DISKPART. Close the COMMAND window.

Step 9 - In Windows you can now see your freshly formatted drive. It will have the name Flash530. If you open it, it will appear empty.



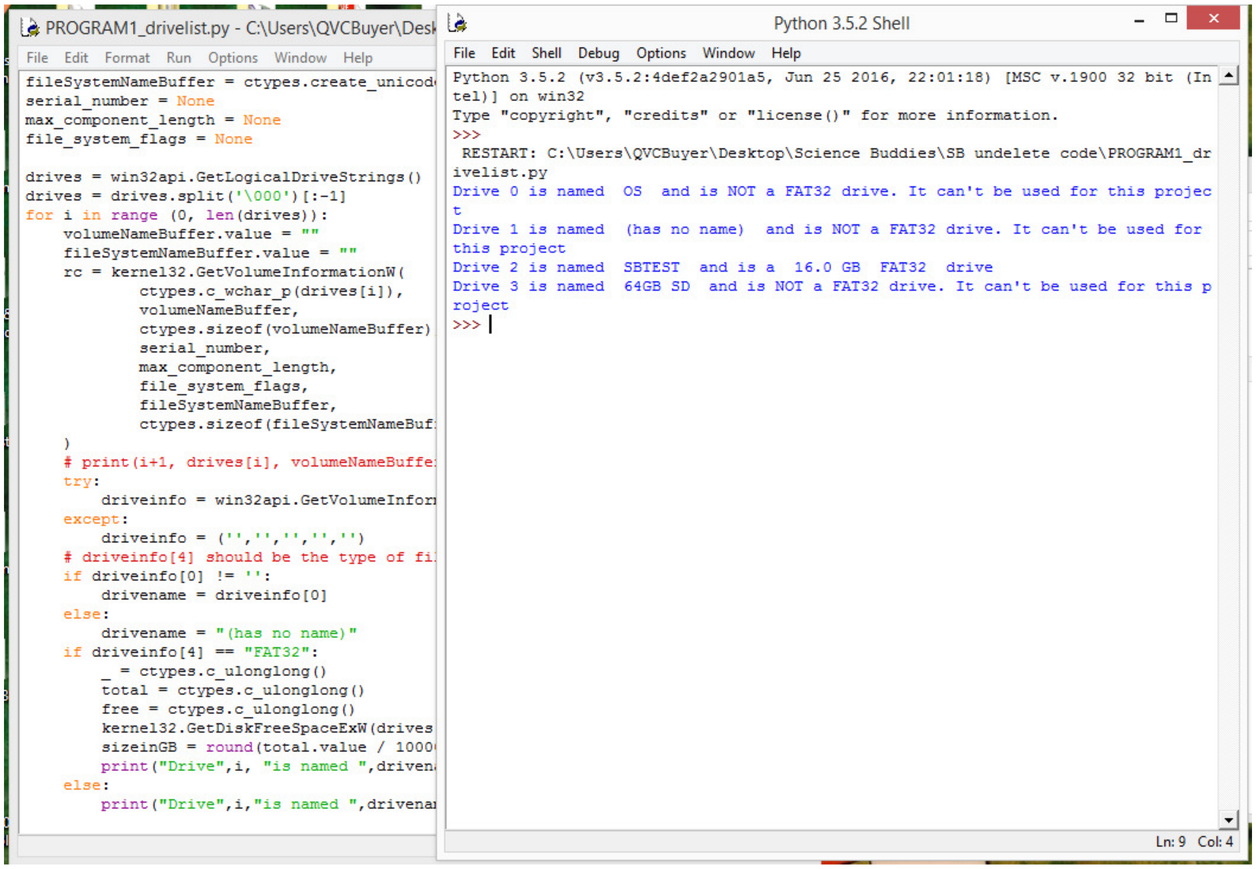
| Name | Date modified | Type | Size |
|---|---|---|---|
| PROGRAM1_drivelist.py | 12/20/2016 4:29 PM | PY File | 2 KB |
| PROGRAM2_sectorbrowse.py | 12/7/2016 2:15 PM | PY File | 10 KB |
| PROGRA | 12/7/2016 3:02 PM | PY File | 12 KB |
| PROGRA | 12/7/2016 3:36 PM | PY File | 12 KB |
| PROGRAM5_displayfilesFAT.py | 12/20/2016 1:14 PM | PY File | 15 KB |
| PROGRAM6_showWholeDisk.py | 12/20/2016 1:56 PM | PY File | 11 KB |
| PROGRAM7_searchWholeDisk.py | 12/20/2016 2:18 PM | PY File | 11 KB |

Type: PY File
Size: 1.80 KB
Date modified: 12/20/2016 4:29 PM

File name: Prog07_searchWholeDisk.py
Python files (*.py;*.pyw)

*Figure 7 - Opening a program in Python's IDLE environment is like any other Windows program. You can navigate to the location of the program and simply select it.*

Step 10 - When you installed Python earlier, Windows will have added a Python icon. Double click on that to launch IDLE, the Python editor and interactive environment. Open up "PROGRAM1_drivelist.py". You can select "Run / Run Module" from the menu, or press F5. But first, let's see what the program does.


PROGRAM1_drivelist.py - C:\Users\QVCBuyer\Desk
File Edit Format Run Options Window Help
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Ln: 9 Col: 4

```
fileSystemNameBuffer = ctypes.create_unicod
serial_number = None
max_component_length = None
file_system_flags = None

drives = win32api.GetLogicalDriveStrings()
drives = drives.split('\000')[:-1]
for i in range (0, len(drives)):
    volumeNameBuffer.value = ""
    fileSystemNameBuffer.value = ""
    rc = kernel32.GetVolumeInformationW(
            ctypes.c_wchar_p(drives[i]),
            volumeNameBuffer,
            ctypes.sizeof(volumeNameBuffer)
            serial_number,
            max_component_length,
            file_system_flags,
            fileSystemNameBuffer,
            ctypes.sizeof(fileSystemNameBuf
    )
    # print(i+1, drives[i], volumeNameBuffe
    try:
        driveinfo = win32api.GetVolumeInfor
    except:
        driveinfo = ('','','','','')
    # driveinfo[4] should be the type of fi
    if driveinfo[0] != '':
        drivename = driveinfo[0]
    else:
        drivename = "(has no name)"
    if driveinfo[4] == "FAT32":
        _ = ctypes.c_ulonglong()
        total = ctypes.c_ulonglong()
        free = ctypes.c_ulonglong()
        kernel32.GetDiskFreeSpaceExW(drives
        sizeinGB = round(total.value / 1000
        print("Drive",i, "is named ",driven
    else:
        print("Drive",i,"is named ",drivena
```

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\QVCBuyer\Desktop\Science Buddies\SB undelete code\PROGRAM1_dr
ivelist.py
Drive 0 is named  OS  and is NOT a FAT32 drive. It can't be used for this projec
t
Drive 1 is named  (has no name)  and is NOT a FAT32 drive. It can't be used for
this project
Drive 2 is named  SBTEST  and is a  16.0 GB  FAT32  drive
Drive 3 is named  64GB SD  and is NOT a FAT32 drive. It can't be used for this p
roject
>>>
```

*Figure 8 - Running PROGRAM1 in Python simply displays the list of available hard drives. It also confirms that you have installed Python correctly.*

Python is a very powerful language but it is not specific to Windows. To make it run on just about any kind of computer, the developers put in the basic functions and let you add in new ones by "importing" new code. This program and all of the ones in this project use some modules that are specific to Windows. The program does that first, then makes some calls to these functions to identify the drives on your computer. You will see your hard drive, the flash drive you just formatted and possibly others. PROGRAM1 simply displays that list. The bottom section of the program uses the data gathered by the top section to display the drives that will work with this project. If it doesn't find any, you will have to find an appropriate USB flash drive.

Step 11 - Close PROGRAM1 and open PROGRAM2. This program uses the code from PROGRAM1 and turns it into a handy function called "findOurFlashDrive()". That function scans the machine and if it finds a drive it can use, it will open up the first sector of the drive and figure out the format. It tells you some of the data - how many bytes there are per sector (always 512), how many sectors there are per disk cluster, then some of the more specific information required by FAT32.

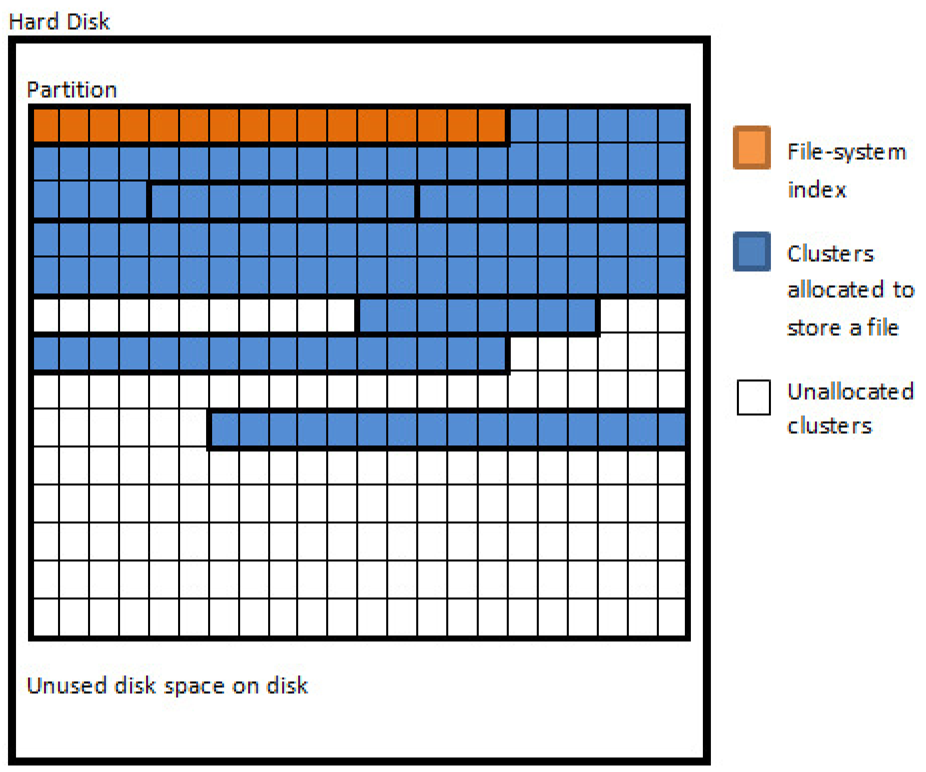


*Figure 9 - Files stored a flash drive are laid out according to the FAT32 format. The directory sits at the beginning of the drive, followed by File Allocation Tables, then finally by the file data itself.*

A drive with multiple GB of storage will have a LOT of sectors. That is hard for a file system to manage, so it breaks them up into contiguous groups called **clusters**. A cluster is the smallest size you can have on the disk - so if you simply write a text file with a friend's phone number, that file will actually be the size of a cluster which might be 8K or 8192 characters. The rest of the cluster is wasted space. You can fit more stuff on a drive if you use fewer files but make the files bigger.

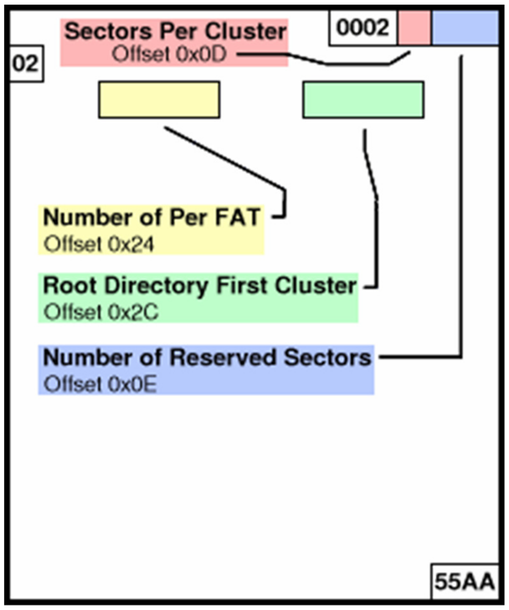


*Figure 10 - the boot sector of the flash drive contains the basic structural information about the drive. Because it is FAT32 we know where things will be stored within that sector. Courtesy of* <a href=”https://www.pjrc.com/tech/8051/ide/fat32.html”>Paul Stoffregen</a>

The directory tells the file system the name of each file you have and which cluster number it starts with on the drive. The first usable cluster is number 2, and the numbers go as high as there is space for data.

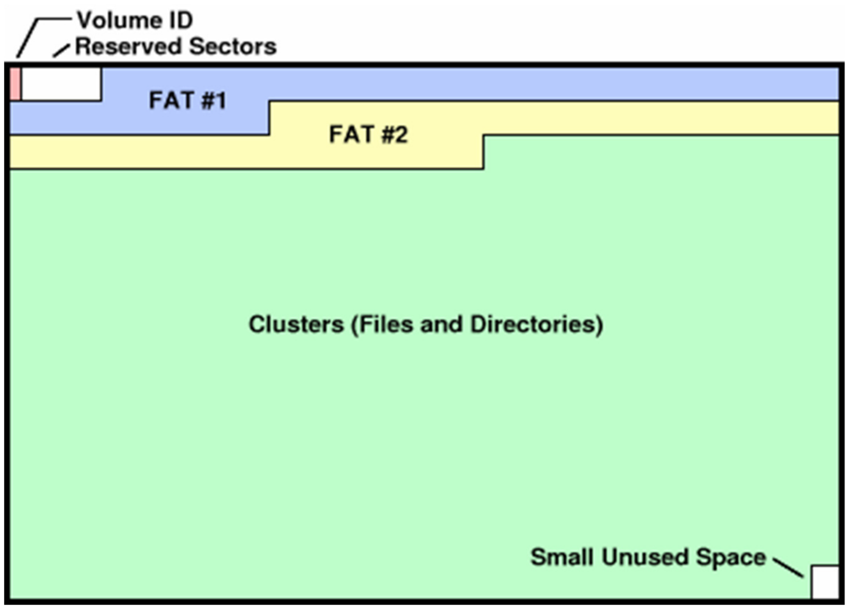


*Figure 11 - A FAT32 drive has two copies of the File Allocation Table (FAT) in case one of them gets scrambled. The directory precedes the two FAT copies and the file data follows it.*

Once you fill up that cluster, the file system looks in the **File Allocation Table** or **FAT** to find another cluster That is not in use. It writes any additional stuff in to that cluster. There is a single entry in the FAT for each cluster on the drive. That entry tells the file system where the next cluster is. Think of a file as links in a chain. Once you know where one link is you can see the next. In a FAT32 file system, the first cluster number is stored in the directory. If it needs a second cluster, the number of that second one will be stored in the FAT entry whose location matches the first cluster. Remember the boxes in our store from the introduction? The file system builds a link from cluster to cluster, storing those links in the FAT. Each entry is 32 bits in length - and That is why this file system is called FAT32.

Step 12 - Use Notepad in Windows and create a little file called "MYTEST.txt" Save it with a single line of text that just says "Hello World!" (programmers like that phrase and you are a programmer now). Close Notepad and open PROGRAM3 in Python. Run it. The drive was empty before you ran Notepad but now it has one file in it. Look at the display from the program. It is looking at the directory structure and identifying files that either have been deleted or are still there. You will certainly see your MYTEST file and perhaps other temporary files used by Notepad. This Python program scans through all of the possible places on the drive that could contain file names but only displays the ones that have files (not directories or other items). Can you follow the program and see how?

Step 13 - Now that you've see how you can display directory entries, let's see if we can read a file. Copy the GETTYSBURG.txt file onto your flash drive. Open PROGRAM4 in Python and run it. You'll see a file list as before but now you can select a file to display. Try selecting your MYTEST file. Run it again and select the GETTYSBURG file. Notice that the name is slightly mangled? That is because FAT32 uses a name system with 8 characters for a name with a 3 character file extension. The longer name Windows uses is stored elsewhere but we do not really care about that. Even though Lincoln's famous speech is a lot longer than your greeting to the world, it is still less than the size of one cluster and simply takes up one. Try a longer text file - copy the MACBETH.txt file onto the flash drive. Display that and you only see the first few pages. What happened? Remember that the directory only knows where the first cluster is. After That is displayed, PROGRAM4 doesn't know what to do next.

Step 14 - Suppose MYTEST.txt contained some juicy gossip about a friend. You do not want them to see it, so you delete it from your flash drive. Do that, then run the program again. Notice that the file name MYTEST.TXT is still there, but the first 'M' is missing? It is been replaced by a character you cannot print, an E5 in hexadecimal (base 16), or 229 in decimal (base 10). Printable characters on computers are always between 32 and 127. FAT32 uses this special unprintable character to indicate a deleted file. See that your program knows it is there anyway but Windows did not? Display it with the Python program - you can still read it!

Step 15 - Open up PROGRAM5. This builds on PROGRAM4 by including use of the FAT. If your drive is formatted to include 8K, or 8192 bytes per cluster, PROGRAM4 would find the first cluster display that many characters and then stop. It simply did not know how to go any farther. PROGRAM5 knows there's  lot more to go. It uses the cluster number of the first cluster and goes to that entry of the FAT which tells it the cluster number of the second. It will display those 8192 bytes, then use the second cluster number to find the third. And so one. Try it. See that it can open your deleted test file, the Gettysburg address and the much longer Macbeth by William Shakespeare.

Step 16 - Experiment a little. Try deleting and adding in longer text files. You will see that adding a new one never overwrites the contents of an old one That is still visible. Once you delete a file though, you might find that Windows will reuse some of those clusters.

What's happening here? To be efficient, WIndows will try and keep chunks of your files as close as possible. It will reuse any clusters it thinks are available because there aren't any directory entries or chains pointing to them. Until it reassigns the clusters, that file is still easily undeleted. It is simply a matter of changing the first character of the name in the directory to one thats readable. These programs do not allow that because Python is blocked by Windows from making changes to the drive like that. Commercial programs and ones you write in other languages will not have that restriction.

Imagine though, that you had a paper for class that you deleted by mistake before copying over new information. The directory may be gone and perhaps some of the clusters have been recycled. Maybe though, some of it is still there? Or looking at this another way - imagine you had a file that you would rather wasn't read by a nosy brother or sister. You deleted it, the directory was written over by another file. But is any of the file still there?

Step 17 - PROGRAM6 explores this by letting you enter a single cluster number or a range of numbers. It will read the cluster and display any characters that can be read by people. Empty space and characters with numbers above 127 will be thrown away. If you look at cluster 2 you will see the directory and recognize names from it. Some of the other programs shows you what clusters a file started on. Enter that number. See the same contents you saw earlier? Try a range by specifying something like 200-250 to display the 50 sectors starting at 200. Depending on what files you copied and deleted, you will find parts of the files interwoven when you read the disk this way. Do not worry - the file system understand what you did but this program is simply dumping the contents to the screen.

So now imagine you had a text file with passwords stored on it. You deleted it and the directory no longer shows it. If the cluster hasn't been used though, you can find it with program 6. Watching as potentially 8 or more GB of data scrolls by isn't fun though. So let's try something else.

Step 18 - Create a text file on your drive called "SECRET.TXT" and write "password = UNH2019" into it. Delete it. If you then copied and deleted enough files, the directory entry would be reused and That is good. Is the password still readable though? Open and run PROGRAM7. This is like program 6, except it will search the whole drive looking for a sequence of characters. In this case, use the word "password" and let it run. It will show you any cluster that contains the word “password”. Oops - does that mean you can write a file, delete it, write new things to the drive and potentially uncover hidden secrets?

Knowing what clusters contain that phrase is one thing. You could display that sector using Program 6, or better yet, write a program yourself that looks through the chains which start in the directory, use the FAT and put the cluster in as part of the file.

**Make it your own**

A program that can walk through a directory and/or a drive and locate lost data isn’t quite the same thing as an undeleter. With a FAT32 drive you can then “unmark” the file as deleted by restoring the first letter of the file name. If the clusters in use haven’t been recycled yet, you’ve successfully undeleted the file. It is usually a better idea to copy the contents of your newly located file to another drive. If you accidentally write over something else on your USB flash drive, you might be unable to undelete another file.

These Python programs are a start. Modify them so that they:

1) Identify a deleted file and not just display the contents, but copy the contents to a file on another drive.
2) When searching for a phrase on the drive, looking sector by sector, go back to the directory and FAT to identify which file might include that cluster.
3) If you cannot find a deleted file that includes that cluster, start with the cluster, use the FAT and see what else was in that file. There may be more treasures!
4) Up to a challenge? If you have administrative permission on the computer you may be able to change the Python code so it writes directory changes on the drive.
5) Reformat your flash drive to have two partitions. Modify your Python program to navigate partitions on a drive.
6) Up to another challenge? Copy your python code over to one of the school’s Debian machines and modify your code so that it works equally well in Linux. In Linux you can query the environment to see what operating system you’re running in. In the code you’re using, you’ve used several Windows-specific functions. You’ll need to locate the Linux equivalents.

```
>>> import os
>>> print os.name
posix
>>> import platform
>>> platform.system()
'Linux'
>>> platform.release()
'2.6.22-15-generic'
```

The output of `platform.system()` is as follows:

- Linux: `Linux`
- Mac: `Darwin`
- Windows: `Windows`