**APPLICATION PROGRAM INTERFACE**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**API**: application program interface (API)

Is a set of routines, protocols, and tools for building software applications. Basically, an API specifies how software components should interact. Additionally, APIs are used when programming graphical user interface (GUI) components.

*Intuition*:

See bank analogy

APIs allows you to talk to other software easily.

1. Similarly, if we abstract away the implementation details of operations (such as allocating system memory, activating the garage doors remotely, or launch rockets into space), it will help lower the programmers’ cognitive load (the amount of stuff a person has to remember at one time), therefore increasing productivity.
2. With this abstract of information comes another advantage: Components can be swapped out and replaced as long as they follow the same protocol.
3. Interface is a place where different software components interact. A Protocol is a set of rules defining how they interact, and a Format defines how they talk to each other. Endpoints provide different functionalities within the same interface.
4. **Protocol** is a set of rules defining how components can interact with each other. Both parties must understand and uphold the same protocol for the communication to be successful. In this case, the bank teller understands withdrawing and depositing money, but she wouldn’t know how to go on Wikipedia and print the first result out for Bob.
5. This might not be very obvious, but notice that the language of this exchange was English. We call this a **Format**: it specifies how you can encode the data you want to send over to the other party. In other words, their communication format here was English. As with protocols, both parties must understand and uphold the format. In the real world, common formats for web APIs include XML and JSON, though JSON is winning in popularity because it’s lighter and easier to read than XML.
6. In API Terminology, an API **Endpoint** usually refers to a service provider that provides a specific subset of functions within the same Interface. In this case, Tom (stock broker) and Sally (bank teller) are both endpoints. Different endpoints can have different protocols and different formats.

**Abstract (away**): specifically in computer science, refers to intentionally obscuring the details of how something works in order to simplify things conceptually. Something is abstracted when it acts as a "black box": We put some input into the box, and get some output from the box, but we can't see the inner workings inside the box. Abstraction typically happens in layers, i.e. some system of black boxes become the inner workings of a black box at a higher level of abstraction

**ReST**: Representational State Transfer

It relies on a stateless, client-server, cacheable communications protocol -- and in virtually all cases, the HTTP protocol is used.

**Stateless**: the server only understands current request. It receives and stores data and then forgets that it exists. Servers will need to check the database to confirm existence.

**Servers**: a server is a computer program or a device that provides functionality for other programs or devices, called "clients". This architecture is called the client–server model, and a single overall computation is distributed across multiple processes or devices. Servers can provide various functionalities, often called "services", such as sharing data or resources among multiple clients, or performing computation for a client. A single server can serve multiple clients, and a single client can use multiple servers. A client process may run on the same device or may connect over a network to a server on a different device.[1] *Typical servers are database servers, file servers, mail servers, print servers, web servers, game servers, and application servers.[2]*

**Application server**: Hosts web apps (computer programs that run inside a web browser) allowing users in the network to run and use them, without having to install a copy on their own computers. Unlike what the name might imply, these servers need not be part of the World Wide Web; any local network would do.

**Web Application**: is a client–server computer program which the client (including the user interface and client-side logic) runs in a web browser.

**Rendering in HTML**: Parsing HTML and creating the DOM Tree – HTML is a hierarchal structure that begins with a <html> tag, usually contains a <head> and <body> tag, and elements can be nested within elements. These HTML elements are parsed and turned into a “DOM tree” by the rendering engine.

The Document Object Model (**DOM**) is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.

**JSON** format is often used for serializing and transmitting structured data over a network connection. It is used primarily to transmit data between a server and web application, serving as an alternative to XML. JSON is **JavaScript Object Notation**.

**MACHINE LEARNING**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Epochs vs Steps:** An epoch usually means *one iteration* over all of the training data. For instance, if you have 20,000 images and a batch size of 100 then the epoch should contain 20,000 / 100 = 200 steps. (epoch contains steps) However, I usually just set a fixed number of steps like 1000 per epoch even though I have a much larger data set. At the end of the epoch I check the average cost and if it improved I save a checkpoint. There is no difference between steps from one epoch to another. I just treat them as checkpoints.

People often shuffle around the data set between epochs. I prefer to use the random.sample function to choose the data to process in my epochs. So say I want to do 1000 steps with a batch size of 32. I will just randomly pick 32,000 samples from the pool of training data.

**Lift Chart**: Lift is a measure of the effectiveness of a predictive model calculated as the ratio between the results obtained with and without the predictive model. Cumulative gains and lift charts are visual aids for measuring model performance. Both charts consist of a lift curve and a baseline.

**Corpus:** A collection of digital documents. This collection is used to automatically infer the structure of the documents, their topics, etc. For this reason, the collection is also called a *training corpus*. This inferred latent structure can be later used to assign topics to new documents, which did not appear in the training corpus. No human intervention (such as tagging the documents by hand, or creating other metadata) is required.

**Vector:** In the Vector Space Model (VSM), each document is represented by an array of features. For example, a single feature may be thought of as a question-answer pair:

How many times does the word splonge appear in the document? Zero.

How many paragraphs does the document consist of? Two.

How many fonts does the document use? Five.

The question is usually represented only by its integer id (such as 1, 2 and 3 here), so that the representation of this document becomes a series of pairs like (1, 0.0), (2, 2.0), (3, 5.0). If we know all the questions in advance, we may leave them implicit and simply write (0.0, 2.0, 5.0). This sequence of answers can be thought of as a vector (in this case a 3-dimensional vector). For practical purposes, only questions to which the answer is (or can be converted to) a single real number are allowed.

The questions are the same for each document, so that looking at two vectors (representing two documents), we will hopefully be able to make conclusions such as “The numbers in these two vectors are very similar, and therefore the original documents must be similar, too”. Of course, whether such conclusions correspond to reality depends on how well we picked our questions.

**Sparse Vector:** Typically, the answer to most questions will be 0.0. To save space, we omit them from the document’s representation, and write only (2, 2.0), (3, 5.0) (note the missing (1, 0.0)). Since the set of all questions is known in advance, all the missing features in a sparse representation of a document can be unambiguously resolved to zero, 0.0.

Gensim does not prescribe any specific corpus format; a corpus is anything that, when iterated over, successively yields these sparse vectors. For example, set((((2, 2.0), (3, 5.0)), ((0, 1.0), (3, 1.0)))) is a trivial corpus of two documents, each with two non-zero feature-answer pairs.

**Model:** We use model as an abstract term referring to a transformation from one document representation to another. In genism, documents are represented as vectors so a model can be thought of as a *transformation between two vector spaces*. The details of this transformation are learned from the training corpus.

For example, consider a transformation that takes a raw count of word occurrences and weights them so that common words are discounted and rare words are promoted. The exact amount that any particular word is weighted by is determined by the relative frequency of that word in the training corpus. When we apply this model we transform from one vector space (containing the raw word counts) to another (containing the weighted counts).

**greedy algorithm:** is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage with the intent of finding a global optimum. In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

**PYTHON**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Source Code**: program instructions that must be translated by a compiler, interpreter, or assembler into object code before execution. Since Python is open source you can read the source code.

**Python source file**: Python source files are files that contain Python source code. As Python can be used as a scripting language, Python source files can be considered as scripts.

A "source folder" is a directory that contains source files. Putting .py files into this directory will make them discoverable by PyDev, so that you can for instance import them from other Python files.

**Module**: is a Python object with arbitrarily named attributes that you can bind and reference. Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can *also include runnable code*.

If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a script. As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you’ve written in several programs without copying its definition into each program.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level and in calculator mode).

**A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.** Within a module, the module’s name (as a string) is available as the value of the global variable \_\_name\_\_. For instance, use your favorite text editor to create a file called fibo.py. Once imported, you call the func with the global var fibo.

i.e. fibo.fib(x). Also, fibo\_\_name\_\_ will return the module string name. See notepad for code.

Basically, you can create a given .py file with a few functions. You can import the .py file and then use the functions therein. You do this all the time.

**sys.path**: A list of strings that specifies the search path for modules. Initialized from the environment variable PYTHONPATH, plus an installation-dependent default.

As initialized upon program startup, the first item of this list, path[0], is the directory containing the script that was used to invoke the Python interpreter. If the script directory is not available (e.g. if the interpreter is invoked interactively or if the script is read from standard input), path[0] is the empty string, which directs Python to search modules in the current directory first. Notice that the script directory is inserted before the entries inserted as a result of PYTHONPATH.

A program is free to modify this list for its own purposes. Only strings and bytes should be added to sys.path; all other data types are ignored during import.

**Module Search Path:** When a module named spam is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named spam.py in a list of directories given by the variable ‘sys.path’. ‘sys.path’ is initialized from these locations:

The directory containing the input script (or the current directory when no file is specified).

PYTHONPATH (a list of directory names, with the same syntax as the shell variable PATH).

The installation-dependent default.

After initialization, Python programs can modify sys.path. The *directory* containing the script being run is placed at the *beginning* of the search path, *ahead* of the standard library path. This means that scripts in that directory will be loaded *instead* of modules of the same name in the library directory. This is an error unless the replacement is intended. See section Standard Modules for more information.

Python imports work by searching the directories listed in sys.path. See code notes

**Calculator mode:** Interactive mode and Calculator mode are the same thing

**\_\_init\_\_.py:**  files are required to make Python treat the directories as containing packages; this is done to prevent directories with a common name, such as string , from unintentionally hiding valid modules that occur later (deeper) on the module search path.

**PYW files** are invoked on pythonw.exe instead of python.exe in order to prevent a DOS console from popping up to display the output. **PYW file type:** is primarily associated with 'Python' by Python Software Foundation. PYW files are used in Windows to indicate a script needs to be run using PYTHONW.EXE instead of PYTHON.EXE in order to prevent a DOS console from popping up to display the output.

**Source a File:** When a file is sourced (by typing either source filename or . filename at the command line), the lines of code in the file are executed as if they were printed at the command line. This is particularly useful with complex prompts, to allow them to be stored in files and called up by sourcing the file they are in.

**“Compiled” Python files**: To speed up loading modules, Python caches the compiled version of each module in the \_\_pycache\_\_ directory under the name module.version.pyc, where the version encodes the format of the compiled file; it generally contains the Python version number. For example, in CPython release 3.3 the compiled version of spam.py would be cached as \_\_pycache\_\_/spam.cpython-33.pyc. This naming convention allows compiled modules from different releases and different versions of Python to coexist.

**Packages**: are a way of structuring Python’s module namespace by using “dotted module names”. For example, the module name A.B designates a submodule named B in a package named A. Just like the use of modules saves the authors of different modules from having to worry about each other’s global variable names, the use of dotted module names saves the authors of multi-module packages like NumPy or Pillow from having to worry about each other’s module names.

**hash value:** is a numeric value of a fixed length that uniquely identifies data. Hash values represent large amounts of data as much smaller numeric values, so they are used with digital signatures. You can sign a hash value more efficiently than signing the larger value.

**hash function**: is any function that can be used to map data of arbitrary size to data of a fixed size. The values returned by a hash function are called hash values, hash codes, digests, or simply hashes.

**Absolute vs. Relative Paths:** There are two ways to specify a file path.

An absolute path, which always begins with the root folder

A relative path, which is relative to the program’s current working directory

There are also the dot (.) and dot-dot (..) folders. These are not real folders but special names that can be used in a path. A single period (“dot”) for a folder name is shorthand for “this directory.” Two periods (“dot-dot”) means “the parent folder.”

The .\ at the start of a relative path is *optional*. For example, .\spam.txt and spam.txt refer to the same file.

**The File Reading/Writing Process:**

Plaintext files contain only basic text characters and do not include font, size, or color information. Text files with the .txt extension or Python script files with the .py extension are examples of plaintext files.

Binary files are all other file types, such as word processing documents, PDFs, images, spreadsheets, and executable programs. If you open a binary file in Notepad or TextEdit, it will look like scrambled nonsense

**COMMAND LINE**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Applications**

This folder contains apps, including Mail, Calendar, Safari, and many others. Apps in this folder also appear in Launchpad. If you try to drag an app out of the Applications folder, it isn’t moved; instead, an alias is created. If you want to uninstall an app, see Install, update, and uninstall apps.

**Library**

This folder contains fonts, Internet plug-ins, and other items used by apps that are available to all users of your Mac. Don’t use this folder to store files and folders you create. Instead, use the home folder or the iCloud Drive folder (Desktop or Documents).

**System**

This folder contains the macOS operating system. You can’t change the name or location of the System folder or the items within it.

**Users**

This folder contains the home folders of all the people set up as users on your Mac. When you’re logged in, the icon for your home folder looks like a house, and the other home folders look like regular folders. If you log out and someone else logs in, that person’s home folder then looks like a house. The Users folder also contains the Shared folder. Items in the Shared folder are available to anyone who uses your Mac.

**Home**

Your home folder is named with your user name and has folders for your desktop files, downloads, pictures, documents, movies, music, and any public files. You can create folders in your home folder.

**The Filesystem:** A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem.

"Shareable" files are those that can be stored on one host and used on others. "Unshareable" files are those that are not shareable. For example, the files in user home directories are shareable whereas device lock files are not.

"Static" files include binaries, libraries, documentation files and other files that do not change without system administrator intervention. "Variable" files are files that are not static.

Historical UNIX-like filesystem hierarchies contained both static and variable files under both /usr and /etc. In order to realize the advantages mentioned above, the /var hierarchy was created and all variable files were transferred from /usr to /var. Consequently /usr can now be mounted read-only (if it is a separate filesystem). Variable files have been transferred from /etc to /var over a longer period as technology has permitted.

To boot a system, enough must be present on the root partition to mount other filesystems. This includes utilities, configuration, boot loader information, and other essential start-up data. /usr, /opt, and /var are designed such that they may be located on other partitions or filesystems.

**bin = Essential command binaries**

/bin contains commands that may be used by both the system administrator and by users, but which are required when no other filesystems are mounted (e.g. in single user mode). It may also contain commands which are used indirectly by scripts. See doc filesystem for commands required in /bin.

**boot = Static files of the boot loader**

This directory contains everything required for the boot process except configuration files not needed at boot time and the map installer. Thus /boot stores data that is used before the kernel begins executing user-mode programs. This may include saved master boot sectors and sector map files.

**dev = Device files**

**etc = Host-specific system configuration**

The /etc hierarchy contains configuration files. A "configuration file" is a local file used to control the operation of a program; it must be static and cannot be an executable binary.

**lib = Essential shared libraries and kernel modules**

The /lib directory contains those shared library images needed to boot the system and run the commands in the root filesystem, ie. by binaries in /bin and /sbin.

**media = Mount point for removeable media**

**mnt = Mount point for mounting a filesystem temporarily**

**opt = Add-on application software packages**

**sbin = Essential system binaries**

Utilities used for system administration (and other root-only commands) are stored in /sbin, /usr/sbin, and /usr/local/sbin. /sbin contains binaries essential for booting, restoring, recovering, and/or repairing the system in addition to the binaries in /bin. Programs executed after /usr is known to be mounted (when there are no problems) are generally placed into /usr/sbin. Locally-installed system administration programs should be placed into /usr/local/sbin.

**srv = Data for services provided by this system**

**tmp = Temporary files**

The /tmp directory must be made available for programs that require temporary files.

Programs must not assume that any files or directories in /tmp are preserved between invocations of the program.

**usr = Secondary hierarchy**

/usr is the second major section of the filesystem. /usr is shareable, read-only data. That means that /usr should be shareable between various FHS-compliant hosts and must not be written to. Any information that is host-specific or varies with time is stored elsewhere. Large software packages must not use a direct subdirectory under the /usr hierarchy

**var = Variable data**

/var contains variable data files. This includes spool directories and files, administrative and logging data, and transient and temporary files.

**CLOUD COMPUTING**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Cloud computing** is an information technology (IT) paradigm that enables ubiquitous access to shared pools of configurable system resources and higher-level services that can be rapidly provisioned with minimal management effort, often over the Internet. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a public utility.

Third-party clouds enable organizations to focus on their core businesses instead of expending resources on computer infrastructure and maintenance.[1] Advocates note that cloud computing allows companies to avoid or minimize up-front IT infrastructure costs. Proponents also claim that cloud computing allows enterprises to get their applications up and running faster, with improved manageability and less maintenance, and that it enables IT teams to more rapidly adjust resources to meet fluctuating and unpredictable demand.[1][2][3] Cloud providers typically use a "pay-as-you-go" model, which can lead to unexpected operating expenses if administrators are not familiarized with cloud-pricing models.

FloydHub is a **Platform-as-a-Servic**e for training and deploying your deep learning models in the cloud. FloydHub takes care of the engineering grunt work so you can focus on the core of your problem.

Platform as a Service **(PaaS)** or application platform as a Service (aPaaS) or platform base service is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app

A **container** is a runtime instance of an **image**--what the **image** becomes in memory when executed (that is, an **image** with state, or a user process). You can see a list of your running containers with the command, **docker** ps , just as you would in Linux.

**Docker** is a computer program that performs operating-system-level virtualization, also known as **"containerization".**

Docker is used to run software packages called "containers". In a typical example use case, one container runs a web server and web application, while a second container runs a database server that is used by the web application. Containers are isolated from each other and bundle their own tools, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and are thus more lightweight than virtual machines. Containers are created from "images" that specify their precise contents. Images are often created by combining and modifying standard images downloaded from repositories.

Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. As Bottomley told me, "Containers gives you instant application portability."

**Dockers vs VM:**

Server components: CPU, RAM, Storage, network

VMs share physical components of the servers. Therefore, if a given VM is unused, it renders its allocation of the components unusable – under utilization of resources

Docker: virtualization on top of the OS rather than the physical components.

Docker has its own PID [ (normally referred to as the process ID or PID) is a number used by most operating system kernels—such as those of UNIX, macOS and Microsoft Windows—to uniquely identify an active process], Network stack [software implementation of communication protocols], and root file system. If there are several docker containers using a server, physical components can be dynamically allocated to different dockers.

Docker vs virtual environments:

VEs help with isolation. However, a VE on one computer is not easily replicated on another computer (machine). Issues with environments in mac OS vs Windows – first hand experience.

**Docker reproduces environment, output, and has isolation portability.**

Kaggle uses docker containers

**Floyd** runs standard Docker images for various deep learning frameworks.

In object-oriented programming (OOP), an **instance** is a concrete occurrence of any object, existing usually during the runtime of a computer program. Formally, "instance" is synonymous with "object" as they are each a particular value (realization), and these may be called an instance object; "instance" emphasizes the distinct identity of the object. The creation of an instance is called instantiation.

An object may be varied in a number of ways. Each realized variation of that object is an instance of its class. Each time a process runs, it is an instance of some program. That is, it is a member of a given class that has specified values rather than variables. In a non-programming context, you could think of "dog" as a type and your particular dog as an instance of that class.[1]

An important distinction is between the data type, which is interface, and the class, which is implementation.

The meaning of the term "type" in computer science is rather similar to the meaning of the word "type" in everyday language. For example, a barman can ask a client what type of beverage does he or she want – coffee, tea or beer? A particular cup of coffee that the client receives is in the role of an instance, while two cups of coffee would form a set of two instances of coffee, determining its type at the same time.

**HARDWARE**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

An operating system (**OS**) is system software that manages computer hardware and software resources and provides common services for computer programs.

The central processing unit (**CPU**) in your computer does the computational work—running programs, basically. But modern CPUs offer features like multiple cores and hyper-threading. Some PCs even use multiple CPUs.

A central processing unit (CPU) is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions. The computer industry has used the term "central processing unit" at least since the early 1960s.[1] Traditionally, the term "CPU" refers to a processor, more specifically to its processing unit and control unit (CU), distinguishing these core elements of a computer from external components such as main memory and I/O circuitry.[

**Hyper-threading** was Intel’s first attempt to bring parallel computation to consumer PCs.

A single physical CPU core with hyper-threading appears as two logical CPUs to an operating system. The CPU is still a single CPU, so it’s a little bit of a cheat. While the operating system sees two CPUs for each core, the actual CPU hardware only has a single set of execution resources for each core. The CPU pretends it has more cores than it does, and it uses its own logic to speed up program execution. In other words, the operating system is tricked into seeing two CPUs for each actual CPU core. Hyper-threading allows the two logical CPU cores to share physical execution resources. This can speed things up somewhat—if one virtual CPU is stalled and waiting, the other virtual CPU can borrow its execution resources. Hyper-threading can help speed your system up, but it’s nowhere near as good as having actual additional cores.

**Multiple Cores**

Originally, CPUs had a single core. That meant the physical CPU had a single central processing unit on it. To increase performance, manufacturers add additional “cores,” or central processing units. A dual-core CPU has two central processing units, so it appears to the operating system as two CPUs. A CPU with two cores, for example, could run two different processes at the same time. This speeds up your system, because your computer can do multiple things at once.

Unlike hyper-threading, there are no tricks here — a dual-core CPU literally has two central processing units on the CPU chip. A quad-core CPU has four central processing units, an octa-core CPU has eight central processing units, and so on.

For example, you can see that this system has one actual CPU (socket) and four cores. Hyperthreading makes each core look like two CPUs to the operating system, so it shows 8 logical processors

**Latency** is a time interval between the stimulation and response, or, from a more general point of view, a time delay between the cause and the effect of some physical change in the system being observed.[1]

**INTERNET PROTOCOL**

Secure Shell (**SSH**) is a cryptographic network protocol for operating network services securely over an unsecured network.[1] The standard TCP port for SSH is 22. The best known example application is for remote login to computer systems by users.