CS 101: Computer Science

Computer:

* powerful (mass data, speed), stupid (no understanding)
* Code: piece of instructions.
* Humans: Creative, Insight about problem ⟶ algorithm ⟶ code

Pixels:

* RGB: Colour Coding (mixing: form colours): 0 to 255 (r: 10, g: 240, b: 240)

GreyScale:

* when all the R, G, B: have the same value (lower ↓ darker ↑)
* 0 0 0 ⟶ pure black, (255, 255, 255) ⟶ pure white
* (50,50,50) ⟶ dark grey, (120,120,120) ⟶ medium grey, (200,200,200): medium grey.

Regular Color to grey scale:

* Take average of R, G, B : $\left(\frac{R+G+B}{3}\right)$

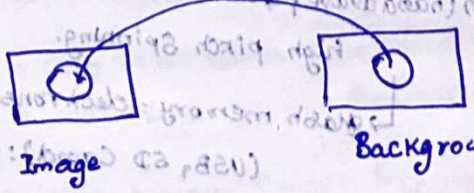
equal Contribution (R, G, B): GreyScale : Same intensity

* Red is more: when actually read, when actually whitish.
* Solution: when red alone is more: will be way greater than average

red > 1.6 * avg (higher: higher restriction)

Bluescreen effect:

* Videos: Series of still Images (20 - 60 per second)



* Bluescreen algorithm: 2 images
  * detect red pixels in image (say);
  * For each (x, y): pixel in image, copy (x, y) of back
  * Red areas: replace by background image area.

'Maintain adjacent pixels'

```
Image image = new SimpleImage("stop.png");
Image back = new SimpleImage("back.png");

for (Pixel pixel : image) {
    avg = (pixel.getRed() + pixel.getBlue() + pixel.getGreen());
    if (avg * 1.5 < pixel.getRed()) {
        x = pixel.getX();
        y = pixel.getY();
        Pixel pixel2 = back.getPixel(x, y);
        pixel.setRed(pixel2.getRed());
        pixel.setGreen(pixel2.getGreen());
    }
}
print(image);
```

Hardware:

* General purpose technology → Transistor (building block)
  → Chip: finger nail sized silicon (Billions of transistors): CPU, memory, flash [No moving parts]

* modern Computers:
  * Chip (transistors) - no moving parts (moore's law)
  * Every 18-24: transistor count doubles in a chip! (cheaper)
  * Computers in cars, thermostats (moore's Law - cheap chips)
  * moore's law: Not Scientific!

* Hardware: CPU → RAM
  → Disk/flash (persistant)

  → CPU: computing part (2 Billion operation/sec)
    → Byte: 8 bits (MB, GB)
  → RAM: Temp storage for computation (volatile), fast!
  → Rom (hard drive, flash drive): Non volatile (magnetic pattern), high pitch spinning.
    → flash memory: electrons in a chip (no moving parts) (USB, SD Card): expensive than hard disk

* Filesystem: organise memory (folders, files)

* Mother board (Complete board: CPU), heat sync, RAM

* micro controller: Small Computer (CPU, RAM, Storage)! (under $1)

Moore's law:

* Adjust brightness (no knob, waterproof)
  ↳ accelerometer (tilt-control)
* Crazy: put a computer in to a torch light!
* Hard disk: Spins, writes patter (north, South: magnetism — circular track)!

Bit & Byte:

* Bit: 0/1 states (group 8 bits: 1 byte)

2^n states (number of bits grouped)

* KB: 1024 bytes, 1 MB: 1024 KB, 1 GB: 1024 MB, 1 TB: 1024 GB.
* Gigahertz: Speed (not bytes): 1 billion cycles per second
  ↳ MHz: million cycles / second
* Rate at which CPU can do its simplest operation / sec.
* High CPU, High power, more heat! (eg: ARM CPUs)

1000 System ⟶ Kilobyte.
1024 System ⟶ Kibi byte, mebi byte.

5% error (but important)

Software:

* Code runs on hardware (Implement machine code)
* Computer: general purpose (seperate memory b/w programs/software)
  ↳ native code (X86: each family ⟶ own idio syncratic machine code)
* Instructions: set of machine code

Program: firefox
Instruction: native code

* CPU runs: fetch/execute cycle
  * Copy instruction into the RAM
  * CPU: Start running instructions (each 4 bytes)
* operating System: (Sanboxing: isolate programe)
  * Starts firefox (Supervisory program, first ran while starting (administration): Start, manage, end: other programs.
  * Keep programs isolated (own RAM, windows)

eg: Laptop

Languages:

* High level (String, loop, DS) ⟶ low level code (Compiler)
* Compiler (source ⟶ native)
  ↳ (.exe ⟶ .c : not possible)
* Dynamic (Java, JS): Implemented by Interpreter (runs code)
  ↳ 1 line at a time

* Compiler: high level ⟶ native (bulk)
* Interpreter: don't produces .exe but intermediate form!

Compiler:

* Faster (lean, less overhead / decisions)

Dynamic):

* Friendly features, more productive, slower code!
* Auto memory management ↓ how many bytes: while running!
  ↳ lighten programmer's workload
  ↳ Trade-off: Programmer (easy), slow running.
  (CPU cheap, evolving)!

JIT (on the fly / just in time compiler):

* Best of both worlds
  ↳ Compile frequent codes (discard when exited)
* Even reducing 10x penalty: still not fast as compiled ones!

Networking:

* Like phone networks (simple outline, complex details)
* Internet: global phone system for computers

LAN:

* Small scale, 1 house / 1 floor (ethernet, wired LAN: wifi)
  ↳ wire: yellow / blue (RJ45 plug)
  ↳ send data packets

Packets:

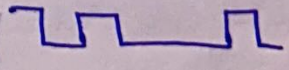
* one hop: divide bytes in to packets (each: 1500 bytes):
* 1500 bytes ⟶ 32 packets. (may be different size)!

Checksum:

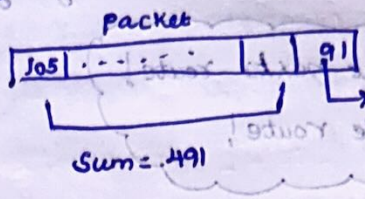
* 1 GB: 1 billion bits per second
* occasional error (add packet bits)
* matches with checksum at end: else: resend the packet! (data corruption)

1 ⟶ 3V
0 ⟶ 0V

Receiver: assemble!

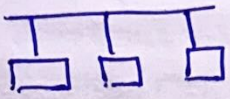


* Not perfect: 2 errors could cancel out! (widely used).



* Bad checksum algorithm: complex, deduck slightest error!
* bad probabilistic: possible for slightest error!

Ethernet design: (Bob Metcalfe – Xerox Parc): minimal hardware, max performance.



→ No Central Controller (distributed, Collaborative)
[Broadband: single medium], simple, cheap

* Send: Each computer has unique address
  packet has addr (sender) [MAC]
  Sender waits for some time (silence): Then send
  packet reaches all computers (broadcast)
* Receive: All computers listen (If not for themselves: ignore)
* **Collision:**
  * Transmit at the same time (garbled)
  * Sender notices the collision & stops sending
  * Both senders wait for a random time then try again
    ↳ wait before transmit protocol (try again when quiet)
    ↳ hard to say how long to take to get a packet!

WiFi:

* Similar to ethernet (every computer has radio, air: medium)
* 1 computer transmits at a time, everyone listens.
* Insecure: hard to listen & pick up intended packets.

TCP/IP: (1974: Govt sponsored research) vendor neutral

* Internet (built on top of TCP/IP: Transmission Control protocol/ Internet protocol): free, open (V4, V6)
* 171.64.64.166 (4 bytes)
  ↓
  Neighbourhood
* Domain name: www.google.com, nick2.stanford.edu
  ↳ easy to remember (domain system look up an IP)

Router:

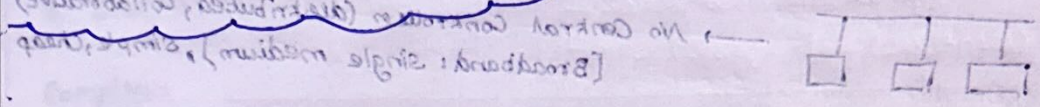
* multiple connections, copies/routes packets b/w them
* Up stream: 171.64.64.166 (router handles connection for few local computers)

* 1 to n hops: Identify shortest path! (From, To) : Delegating to routers

Each router: Figure out next to route!
Not whole route!

→ measure functionality, breakage all the time
→ Destination: put back packets at intended order.

Factors to hop: number of hops
Failures, traffic → distributive, Collaborative!



Special IP:

* 10.x.x.x
* 192.168.x.x

→ translated to a real IP (given out by wifi routers)

(private IP: used within LAN).

↳ By wifi routers!

Being on the Internet:

* Connect to a wifi router (get an IP given by router)
* DHCP: Dynamic host configuration protocol
  - → Automatically configure network settings to work locally.
  - ↳ Feature: get needed config from router.
    (establish temporary connection, get an IP)

Ping (network utility):

* Send ping packet (check for reply): Functioning?
* See hops

Trace route -d 1:

* Identify route in between (Some routers: not visible to traceroute)
* Not entirely reliable.

Table data:

* Rows & Columns (rectangular)
* DB (1:1, 1:n, n:1, n:n) relations
* .csv (Common format)
* get, put (using Python)

```
for (row: table) {
    row.get('field1')
}
```

Some utils:

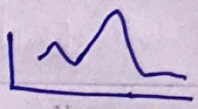
* get (iterate) rows
* get fields, set fields
* Starts with, ends with

Boolean:

* &&
* ||
* !

Analog & digital:

* world: Analog (wave over time)



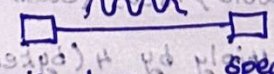
* Sound: vibration (Travel through air, push eardrum, recreate signal)

Oscilloscope:

* Connect to the electrical signal varying over time → show on screen
* 1/400 th of a second: note
* Frequency: How often wave repeats (cycle)
  - → middle A: 440
  - → Higher A: 880
  - → Even higher (octave) up: 1760 cycles
* Amplitude: How high / low

Y → voltage
X → time

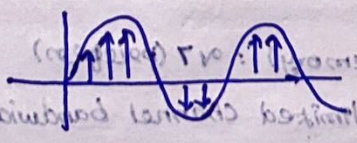
"Signals add up"
- → Chord (doubles): when match up
- → Discord (peaks / valley): don't match



microphone ← speaker ← Noise: Random signal (+, −)

"Analog"

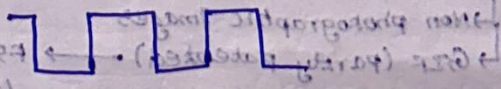
Sampling:



Sample: measure amplitude
44K (CD samples)
↳ −32767 .. 32768 (2 bytes)

Analog → digital.
44K sample: captures: what the human can hear!

Playback: recreate signal from digitalized samples



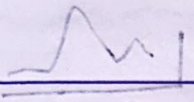
Samples:

* 12000, 12002 : near to each other (distortion)
* Solution: Record first number, then differences alone

12000, +2, +4, +1, +3, -5, +1 (few bytes to store)

↳ 'Compression'

* other scheme:
  * drop every other number (guess (average))
    ↳ saves 2x, but lousy!
    ↳ MP3 (lossy, 10x less space)



Digital media:

* JPEG: free standard (lossy format): widely used
* less detail preserved.

0-10 scale
10 → max detail → v7: Normal!
0 → less detail

JPEG blocks:

* 8×8 blocks: high compression (see boundaries)
* high compression: hard time with crisp edges b/w boundaries (noise, distortion)

v1: 4 times less bytes than v7

* 400x 300 image ⟶ 120K pixels (each 3 bytes): 360KB
* Double the width ⟶ multiply by 4 (bytes)
  ↳ 'Area' of □le
* JPEG saves space (cameras use internally)

Why Compression:

* write into memory (fit in memory): v7 (prefer)
* Agressive compression (when limited channel bandwidth)

web page images:

* Loading issue (slow): But bad: If done v2/v3
* less is more (few images)!
* GIF, PNG: lossless image format (exact recording of pixels)
  ↳ Non photographic images
  ↳ GIF (partly patented) ⟶ Free, open: PNG (Better than PNG)

Audio formats:

* mp3 : lossy like JPEG (10mB/minute) → No Compression
  - → mp3 : 1mB/minute
  - → 2mB/minute / 512 KB/min : space is less less detail.

* mp3 : patented (complex, non-obvious technique)
  - → mp3 software : pay for license

* AAC : used by apple (modern form of mp3) : iPod, iTunes
  - DRm : Digital Right Management feature (control buyer how the content is used)!

* Ogg vorbis : free, open standard, royalty free, better than mp3.

Video formats:

* 20-60 images / sec (lot of bytes)
* DVD Quality (2GB/hour), HD : more space.
* video Compression : Complicated, heavily patented.
* MPEG : Standard in industry (MPEG-LA : handles royalty collection)
* MPEG-2 : Satellite, DVD [1995 : released]
* MPEG-4 (h.264) Compression : good looking, minimum memory
  - → Blue ray disks, video cameras (internal Compression)
  - → patent fee : manufacturers pays for encoders & decoders.

h264 obnoxious royalty:

* Can't distribute unrestricted (own video)
* pay additional royalty for each extra minute!
  (exception : free distribution)

open video format webm:

* mozilla, wiki, google : true, free, open video code (compete h264)
  - → (WEBm project)
* MPEG-LA claims WEBm uses their patents (unable to prove)
* mozila, google, wiki : Supported open source!

Html5 video tag:

* Embed in webpage

Flash video:

* prior flash plugin (Adobe) : supports animated! (h264 Compression)
  - → crashing, security problems (other platforms worse)

Apple Crashes (Due to Adobe) - Apple looked bad.

* Apple, Adobe: Competitors (how motivated: adobe – solve this)
  ↳ Apple writes it!
  (TCP/IP, HTML, HTTP, JPEG, PNG)

Spreadsheet:

* made analysis easy. (2d grids) [VisiCalc, Lotus 123, excel]
* .xlsx (microsoft) : google docs, libreoffice, onedrive
* **Formula:**
  =B15+B16
  =Sum (B3:B16)
  =Average (B3:B16)
* fill right, down : Keep things relative! (C: C3:C16, D: D3:D16)
* Charts,

Computer Security:

* Spear fishing: Intended/Crafted for specific person
* Dictionary attack (Common passwords)
* policy: attempts, 1s for password/login (Slow rate)

* word, misspelling, random, digits, Special Char
* Don't reuse!

phising:

* email, sms (Fraud alert) : Look/Impersonate.
* Real world: Fake ATM (print error, Just record Card no, pin)

HTTPS:

* Secure variant of https (encrypts)
* Eavesdrop:
  * listen to neighbour packets (ethernet broadcast)
  * encrypted (even eavesdropped): useless.
  * https: encryption, URL verification (encrypt before broadcasted).

malware attacks:

* .txt, .jpg, .exe, .doc
  ↳ Can trust passive contents! (don't execute Commands/interact)
  ↳ Active: exe, sh, bat, py, js, vbs, docx, xlsx, pdf (embedded Scripts)

'Doc' → embed macros!

Trojan:

* malware (don't run programs from random Source)

* Bug in adobe: (fed animation, Program breaks, access to the machine)

↳ Keep web facing Software (up to date)

* Firefox (allows machine take over: certain sequence)

CS 105 - Stanford