1. Why CHIP-8?

a. *So before ppl used simple machine language as ISA (Instruction Set Architecture)*
b. *This Machine Language was used in 8-bit microprocessors.*
c. *To make this communication with the 8-bit microprocessors ez a person named Joe Weisbecker developed chip 8*
d. *And it became popularrrr.*

2. What is CHIP-8?

a. *Memory: CHIP-8 has direct access to up to 4 kilobytes of RAM*
b. *Display: 64 x 32 pixels (or 128 x 64 for SUPER-CHIP) monochrome, ie. black or white*
c. *A program counter, often called just "PC", which points at the current instruction in memory*
d. *One 16-bit index register called "I" which is used to point at locations in memory*
e. *A stack for 16-bit addresses, which is used to call subroutines/functions and return from them*
f. *An 8-bit delay timer which is decremented at a rate of 60 Hz (60 times per second) until it reaches 0*
g. *An 8-bit sound timer which functions like the delay timer, but which also gives off a beeping sound as long as it's not 0*
h. *16 8-bit (one byte) general-purpose variable registers numbered 0 through F hexadecimal, ie. 0 through 15 in decimal, called V0 through VF*
    a. *VF is also used as a flag register; many instructions will set it to either 1 or 0 based on some rule, for example using it as a carry flag.*

3. Damn, now tell me How does it work?

a. *It has smth called programm counter (PC) which used to fetch opcodes (operation codes) from memory.*
b. *Now when it fetches the opcode from the memory it will decode it.*
c. *Now it will process it.*
d. *Now after processing it will increment by 2 bytes fetch a new/next opcode*

4. **Ohhhh damn so can we control this by commands? And do those are like ‘echo /dev/random > big.txt’?**
    a. *Son, you over-thought. But in short the answer is very much yess. In fact you can only control a processor by commands.*
    b. *Well well u have ton of commands let me go through them but first I want to tell you how are those like how to make them*
    c. ***Now pay attention***
        1. ***This command consists of four letters (only numbers and alphabets).***
        2. ***Commands will be like this,***
            a. ***00E0***
            b. ***1NNN***
            c. ***2NNN***
            d. ***3XNN***
            e. ***5XY0***
            f. ***9XY0***

And many more……

3. **Now in this commands the 1st letter is for the number only.**
4. **2nd letter is for X and what is X? So X represents the *VX* register in between V0 to VF .**
5. **well well well 3rd register work is almost the same but it represents *VY* in between V0 to VF .**
6. **And the fourth is a 4 bit number.**
7. **Also this 2nd 3rd and 4th can be in terms of nibble like N or NN or NNN.**
8. **Now what is this Nibble?** So CHIP-8 instructions are divided into broad categories by the first “nibble”, or “half-byte”, which is the first hexadecimal number.

d. Simple Command

1. **00E0 →To clear ur screen**
2. **1NNN → JUMP (increments PC (Program Counter) to NNN and PC jumps directly there (DO NOT INCREMENT PC AFTERWARDS))**
3. **2NNN → just like 1NNN (but a key difference is the current PC will be pushed to the stack and so it can be entered later with using 00EE (stack overflow might happen if stack reaches its limits))**
4. **3XNN → skips one instruction if the value of VX (register I am talking) is NN.**
5. **4XNN → will skip one instruction *if the value of VX is not NN* .**
6. **6XNN → Sets register VX to value NN.**
7. **7XNN → adds NN to register VX.**

e. Logical and Arithmetic instruction

1. **8XY0 → VX is set to the value of VY.**
2. **8XY1 → Binary OR means VX set to VX | VY**
3. **8XY2 → Binary AND means VX set to VX & VY**
4. **8XY3 → Logical XOR means VX set to VX ^ VY**
5. **8XY4 → ADD means VX set to VX + VY (Unlike `7XNN`, this addition *will* affect the carry flag. If the result is larger than 255 (and thus overflows the 8-bit register `VX`), the flag register `VF` is set to 1. If it doesn't overflow, `VF` is set to 0.)**

6. **8XY5 and 8XY7 → For the 8XY5 VX is set to VX - VY. now for the 8XY7 VX is set to VY - VX. (** NOTE : This subtraction will also affect the carry flag, but note that it's opposite from what you might think. If the minuend (the first operand) is larger than the subtrahend (second operand), `VF` will be set to 1. If the subtrahend is larger, and we "underflow" the result, `VF` is set to 0. Another way of thinking of it is that `VF` is set to 1 before the subtraction, and then the subtraction either borrows from `VF` (setting it to 0) or not. **)**
7. **8XY6 and 8XYE → For the 8XY6 it will put the value of VY into VX and then shift the value of VX to 1-bit right. For the 8XYE it is almost the same but the value of VX to 1-bit left.**
8. **ANNN → this sets index register I to the value NNN.**
9. **BXNN → It will jump to address XNN plus the value in register VX (** Like B220 will jump to 220 and then the value of register will be V2 **).**
10. **CXNN → computer will generate binary ANDs with the value NN and put the result in VX.**
11. **DXYN →** A miracle of modern times, not much like a painting but very much like coaxing constellations into place with a whisper of 0's and 1's. Jokes aside, this took me a whileeee to figure out. **hOw It WoRkS? So there is a register called I (literelly I english letter I) which has storage of 16-bit but only 12-bit is used (will tell later on) and stores memory address in the normal Format so it is smth like A230 or anything with no constraints tbh. When DXYN is enters the chat X will represent X coordinate and Y will represent Y coordinate and our dear friend**

N will represent how many lines. Waah, you said with a question “why you explained I register and N and X and Y?” cause the address that I stores address that contains 8 bit binary code where 1 is print and 0 is not print and N is how many addresses to go and ask for theirs data (more like incrementation by 1), let’s say the address stored by I is B500 now pc (programm counter) increments to B501 and read it from there and print on the screen and this incrementation will not affect I’s current state/value and X and Y will tell from where to start, if any problem happens like clipping (going out of screen) VF will be set to 1 from the idle/chill state of 0.

12. EX9E → will skip one instruction if the key corresponding to VX is pressed.
13. EXA1 → will skip one instruction if the key corresponding to VX is not pressed.
14. And this key is determined by the keypad, keys are from 0 to F on keypad.
15. Timers
    a. It is a 8 bit counter (holding values from 0 to 255).
    b. When their value is greater than 0 it will count down at speed of 60 times a second so it is 60Hz.
    c. Let’s say our player has a special shield and it is activated for 2 seconds. In order to achieve 2 seconds of counting time we will set our counter to 120 which will be counted down to 0 and for that counter will take 2 seconds (60 per seconds so 120/60 = 2). This is Delay timer

d. **And for the sound timer it has to make sound until it is 0 in value.**
e. **FX07 → sets VX to the current value of the delay timer.**
f. **FX15 → sets delay timer to the value in VX.**
g. **FX18 → sets the sound timer to the value in VX.**

16. **FX1E → The index register I will get the value in VX added to it.**
17. **FX0A → This instruction blocks the whole flaw and stops executing instructions and awaits for key input (any key) if pressed it will store its value (hex value cause it will be from 0 to F) to the VX register and unpause the output.**
18. **FX29 → register I will look into VX and check what the number is and find its address and store it. (so let's say the number is 8 in VX so I will not store the address of VX but find the address of 8 lets say 0340 and store that one in I)**
19. **FX33 → It takes the number in VX (which is one byte, so it can be any number from 0 to 255) and converts it to three decimal digits, storing these digits in memory at the address in the index register I. For example, if VX contains 156 (or 9C in hexadecimal), it would put the number 1 at the address in I, 5 in address I + 1, and 6 in address I + 2.**
20. **FX55 → will take address values from V0 to VX and store it in register I to register I + X.**
21. **FX65 → does the reverse of what FX55 did, the value of V0 to VX will be changed to whatever I to I + X has. (So V0 → I, V1 → I+1, V2 → I+2, …… ,VX → I +X .**