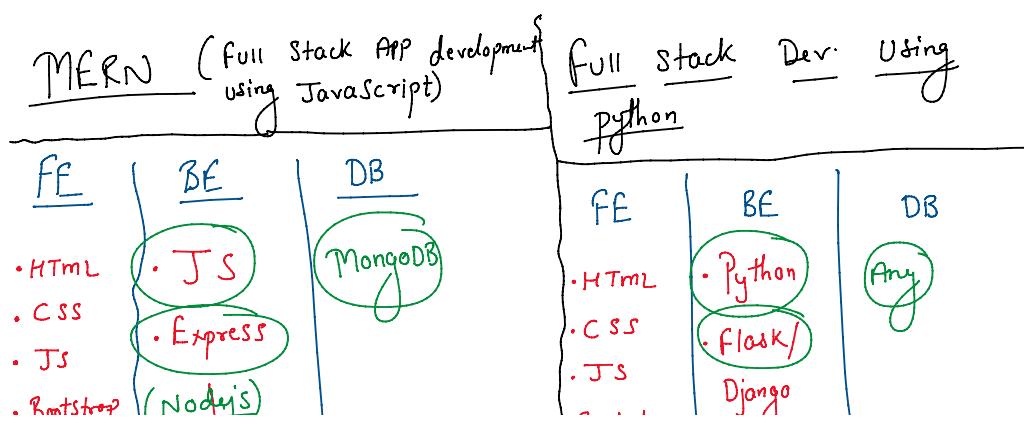
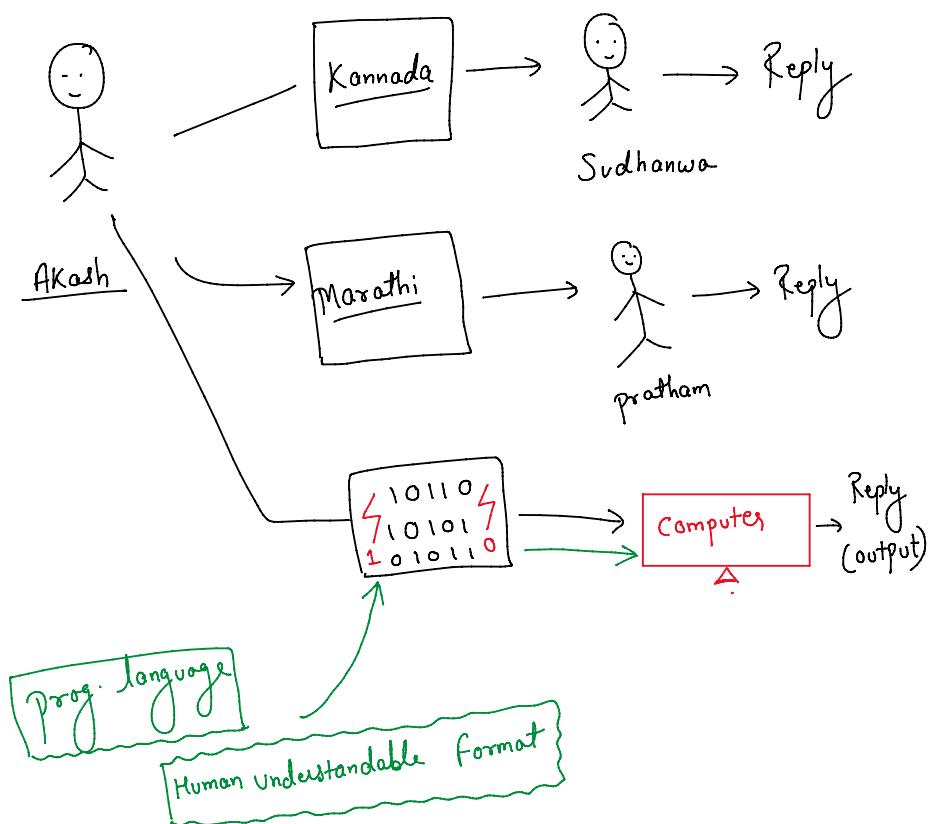
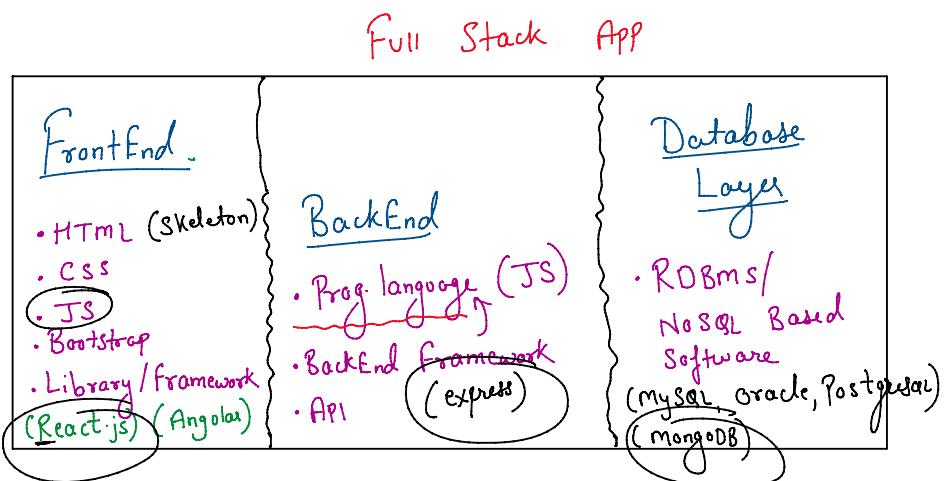
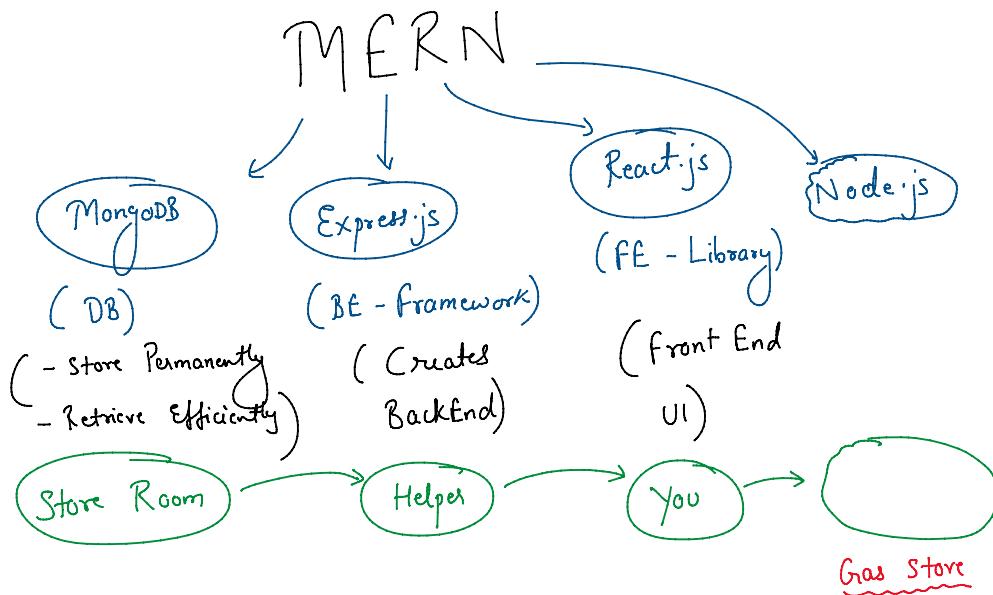


Full Stack

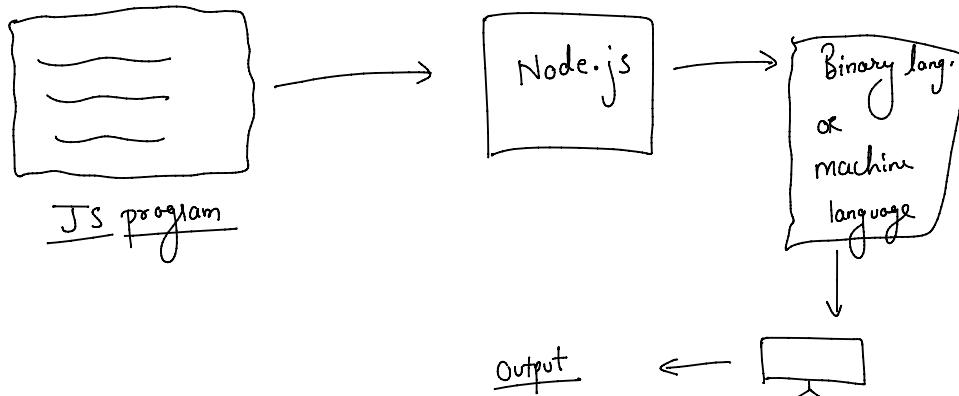




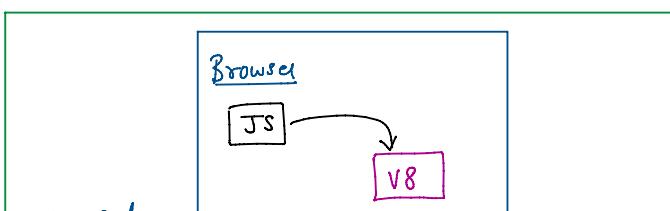
Variables (JS)

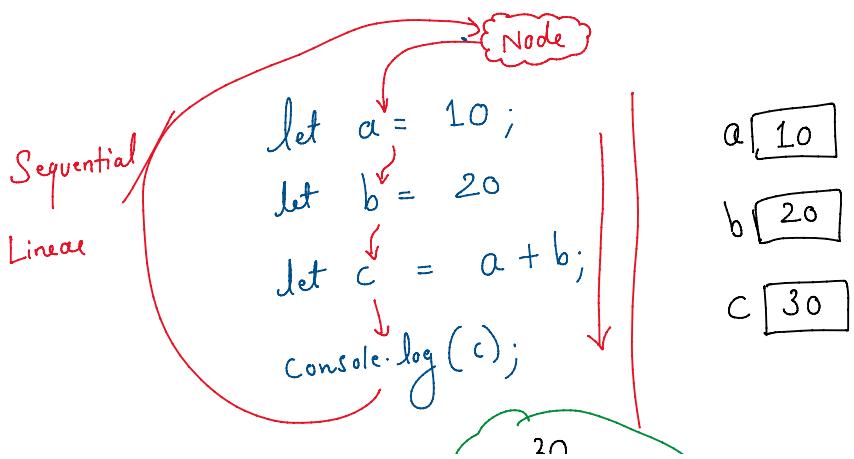
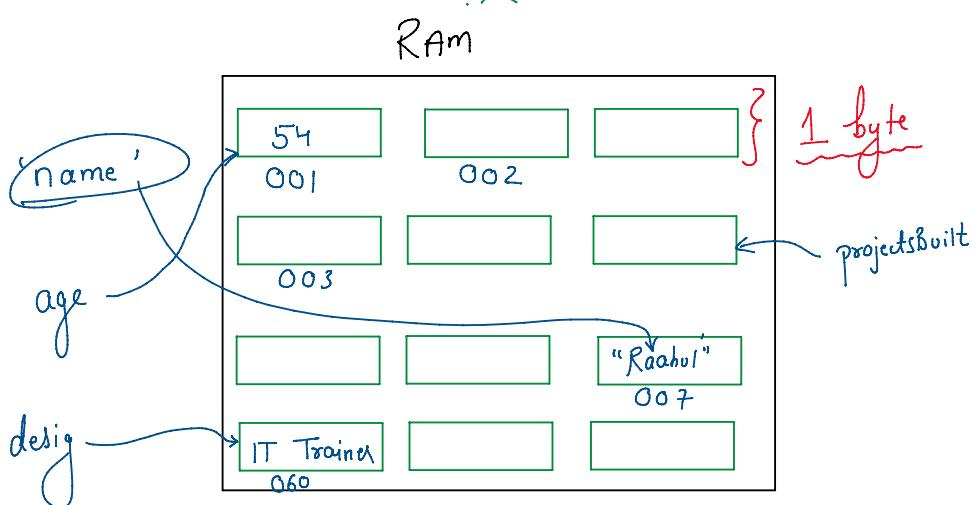
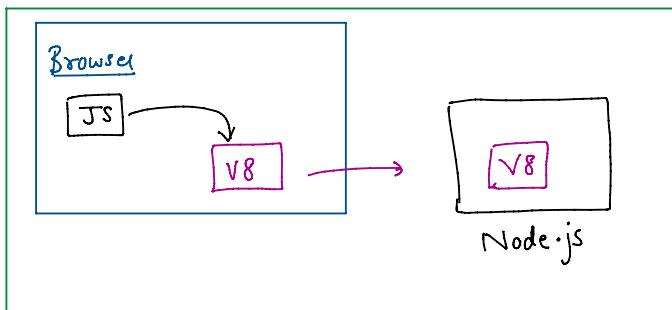
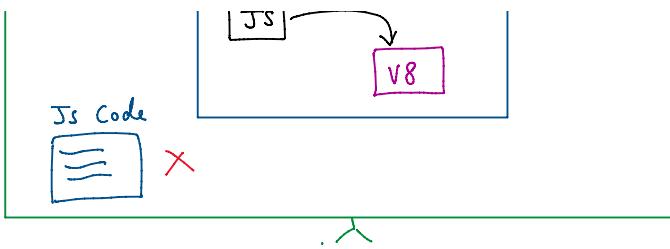
(vessels /
Containers)

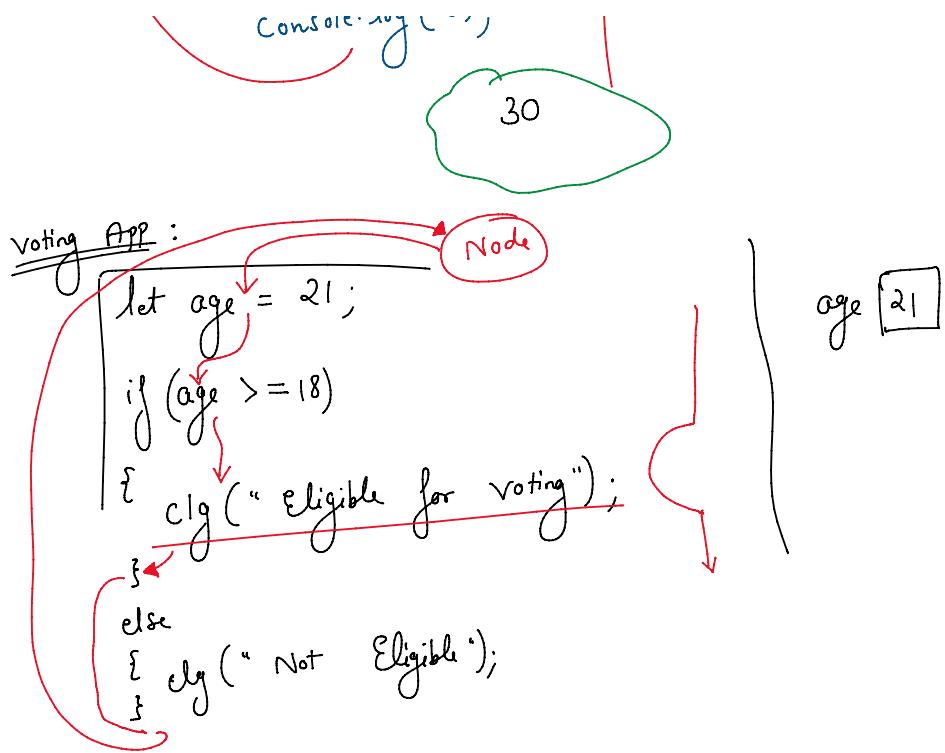
Gas Store (Node.js)



Before 2008:







Functions

makeTea

- . Take a vessel
- . Take water
- . Switch on gas
- . put sugar, tea leaves and milk
- . Strain tea

JS Function

```
function func-name( inputs )
{
```

// Statements

}

function square()
{
 let num = 5;
 let square = num * num;
 clg(square);
}

Node.js

Hello
25

The diagram illustrates the execution flow of a JavaScript function named `Square`. It shows the function definition, its execution, and the resulting output.

function Square(num)

The `num` parameter is highlighted in green. The value `5` is shown in a green box with a red arrow pointing to it from the parameter name.

let squaredNum = num * num;

The expression `num * num` is highlighted in green. The two `5`s are circled in green, with a red arrow pointing to each from the multiplication operator `*`.

return SquaredNum;

The `SquaredNum` return value is highlighted in blue. A red arrow points from the `return` statement to the variable name.

}

let squaredValue = square(5);

The `square(5)` call is highlighted in pink. The `5` argument is circled in green, with a red arrow pointing to it from the parameter name. The result `25` is highlighted in pink, with a red arrow pointing to it from the assignment operator `=`.

clg(squaredValue);

The `squaredValue` variable is highlighted in blue. A red arrow points from the `clg` function to the variable name.

*

* *

* * *

米 米 米 米

* * * *

for(;; i=1; i<=5; i++) {
 clg("VIMEET");
}

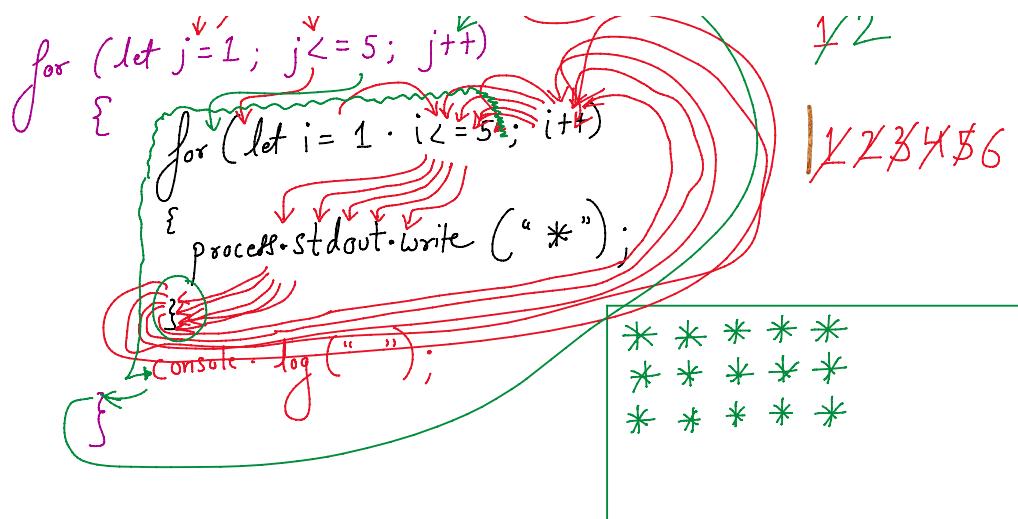
VIMEET
 VIMEET
 VIMEET
 VIMEET
 VIMEET

X284\$6

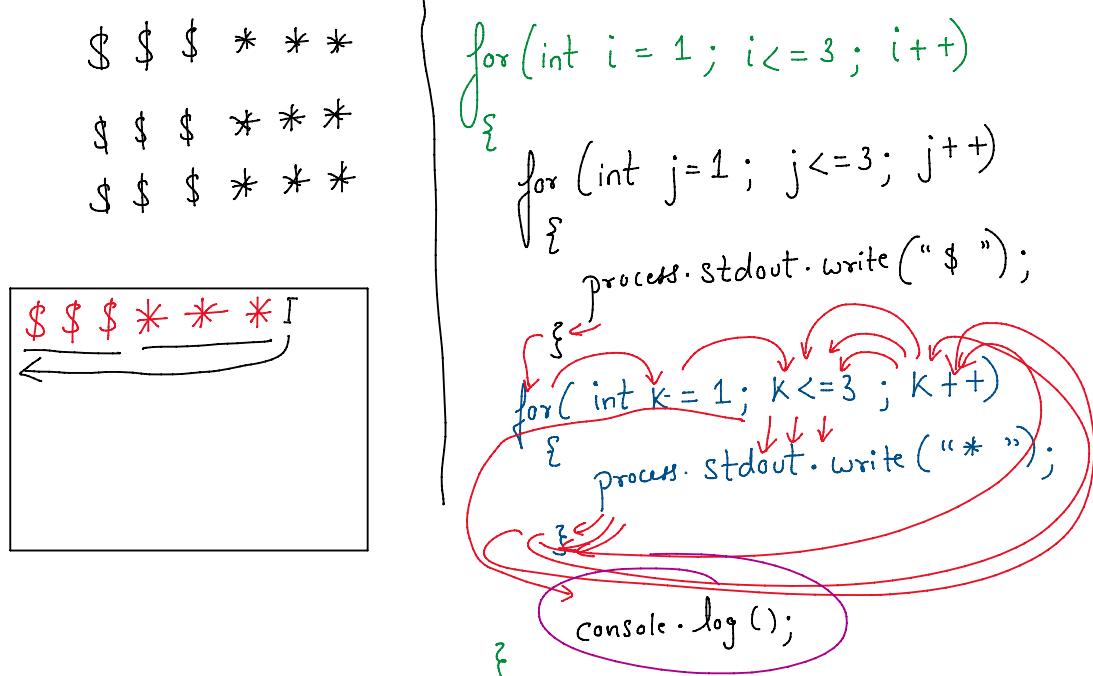
A grid of 15 hand-drawn asterisks arranged in five rows and three columns. The asterisks are drawn with black ink on a white background, showing some variation in size and orientation.

for (*let j=1; j<=5; j++*)
 Node





* → 1
 * * → 2
 * * * → 3
 * * * * → 4
 * * * * * → 5



\$ \$ \$ * → 1
 \$ \$ \$ * * → 2
 \$ \$ \$ * * * → 3

for (let i = 1; i<=3; i++)

{

 for (let j=1; j<=3; j++)

 p.s.w (" \$ ");

 for (let k=1; k<=i; k++)

```

for (let k = 1; k <= i; k++)
{
    p.s.w(" * ");
}
cig();

```

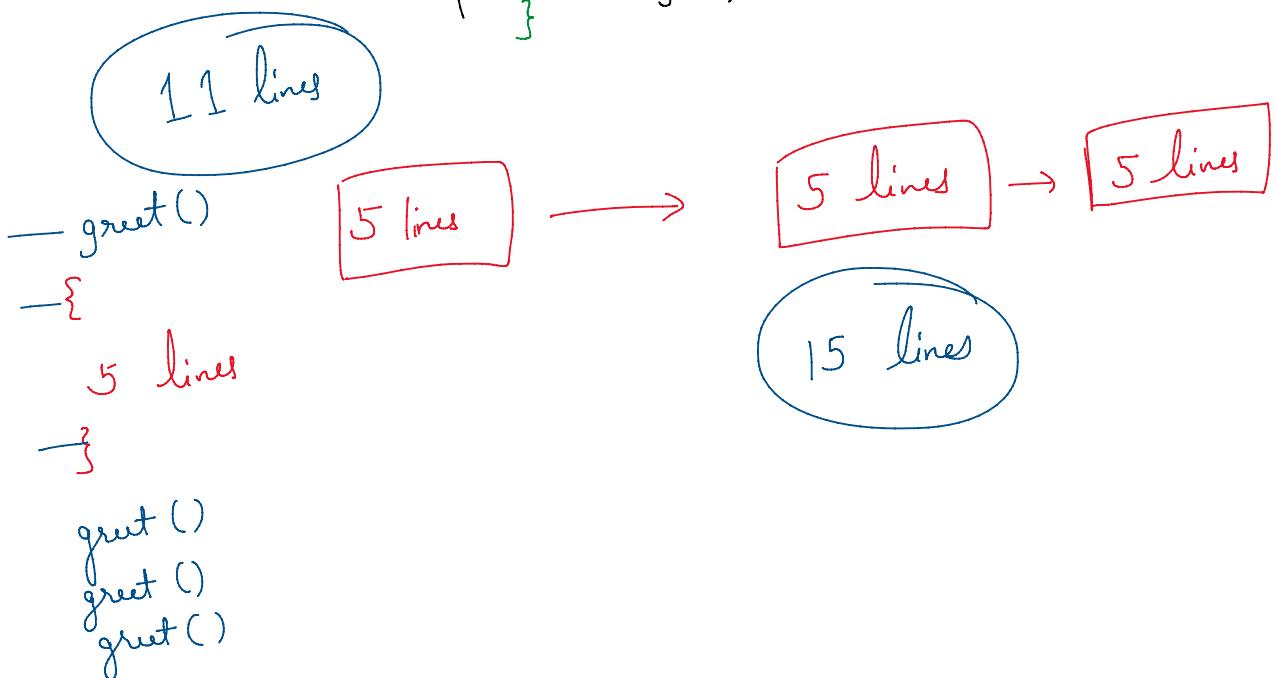
$\begin{array}{l} \text{--- --- * } \rightarrow 1 \\ \text{--- --- * * } \rightarrow 2 \\ \text{--- --- * * * } \rightarrow 3 \\ \text{--- --- * * * * } \rightarrow 4 \\ \text{--- --- * * * * * } \rightarrow 5 \end{array}$

```

for (let i = 1; i <= 5; i++)
{
    for (let j = 5; j >= i; j--)
    {
        p.s.w("   ");
    }
    for (let k = 1; k <= i; k++)
    {
        p.s.w(" * ");
    }
}
cig();

```

$5 - 5$



A screenshot of a code editor window titled "JS ES6". The file "HigherOrderFunctions.js" is open, containing the following code:

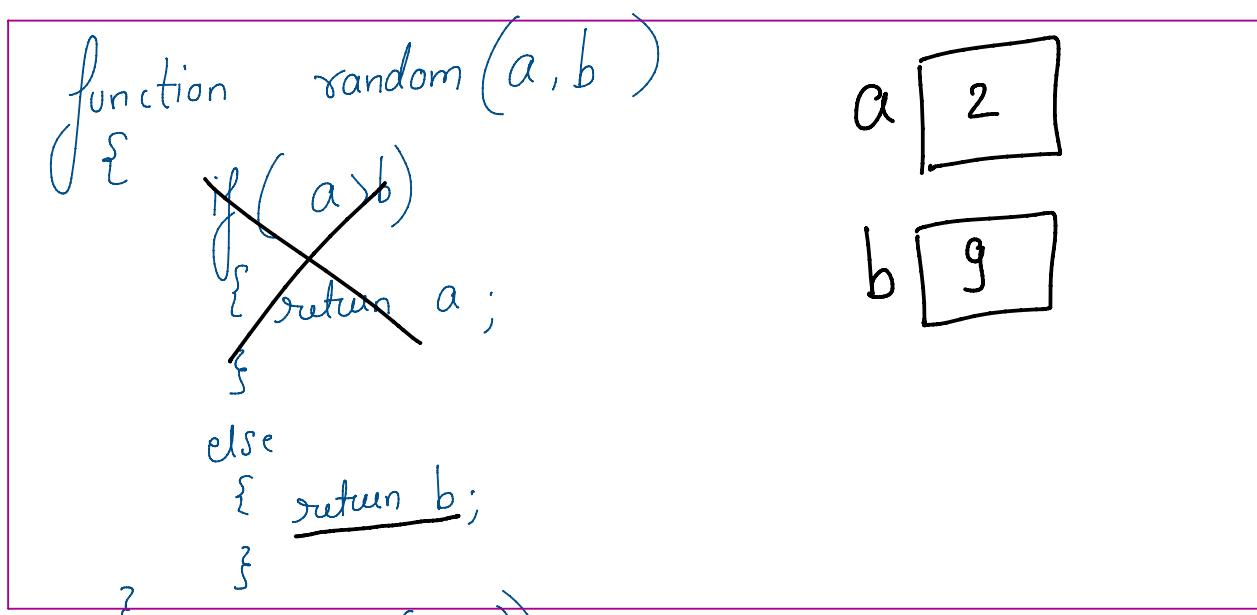
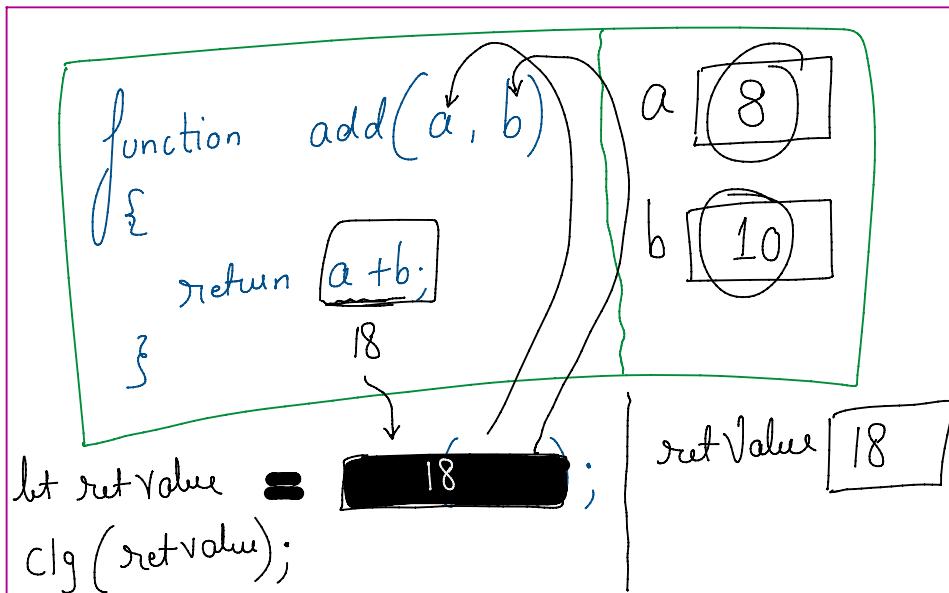
```
function add(a, b) {
    return a + b;
}

let returnedValue = 18;
console.log(returnedValue);
```

Handwritten annotations include:

- A callout bubble points to the line "return a + b;" with the value "18" written next to it.
- Variables "a" and "b" are shown in boxes with values "8" and "10" respectively.
- The variable "returnedValue" is labeled with its value "18" in a box.

Bottom status bar: javascript | ✓ HigherOrderFunctions.js, Babel JavaScript, Go Live, Kite: ready, Prettier



}
 }
 }
 } }
 clg(g);

Higher Order Functions:

The diagram illustrates the execution of a higher-order function. On the left, a screenshot of a code editor shows a file named 'HigherOrderFunctions.js' with the following code:

```
function add(a, b) {
    return a + b;
}

function operate(x, y, operation) {
    return operation(x, y);
}

console.log(operate(5, 3, add));
```

On the right, a hand-drawn diagram shows the state of variables during execution:

- A box labeled "add" contains two boxes: "a [5]" and "b [3]."
- A box labeled "operate" contains three boxes: "x [5]", "y [3]", and "operation [add logic]."
- Arrows point from the "a" and "b" boxes in the "add" box to the "x" and "y" boxes in the "operate" box respectively.
- An arrow points from the "operation" box in the "operate" box to the "add logic" box.
- An arrow points from the "add logic" box to the "add" box.
- An arrow points from the "add" box back to the "operate" box.
- An arrow points from the "operate" box to the final result "8" in the original code editor.

A screenshot of a code editor showing a file named `CallbackFunction.js`. The code defines a function `greet` that logs 'Welcome' followed by a name and then calls a callback function. It also defines a `bye` function that logs a thank you message. A call to `greet` is made with the argument 'Raahul' and a reference to the `bye` function. Handwritten annotations show arrows from the `callback()` call in `greet` to a box containing the string 'Console.log("Thank you, visit again!")', and from the `bye` function definition to the same box.

```
function greet(name, callback) {
  console.log('Welcome ' + name);
  callback();
}

function bye() {
  console.log('Thank you, visit again!');
}

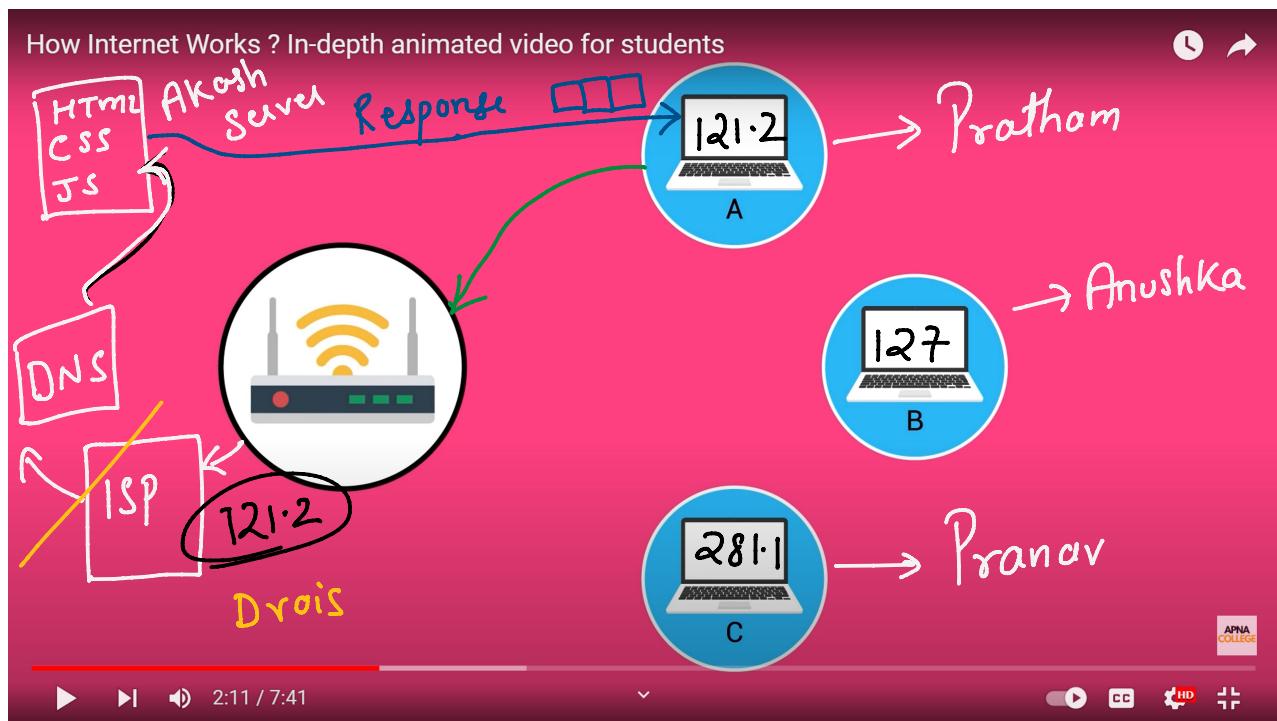
console.log(greet('Raahul', bye));
```

A screenshot of a code editor showing a file named `Arrays.js`. The code declares an array `rollNos` with elements [2, 4, 6, 8, 1, 7, 45, 22, 53]. Handwritten annotations above the array show the numbers 0 through 8 with vertical lines connecting them to the array elements. A large handwritten bracket below the array is labeled 'g Elements'.

```
let rollNos = [2, 4, 6, 8, 1, 7, 45, 22, 53];
```

```
File Edit Selection ... ← → ⚡ JS ES6
ierOrderFunctions.js CallbackFunction.js FunctionExpression.js LetVSConst.js Arrays.js ...
Arrays.js > ...
1 Let rollNos = [2, 4, 6, 8, 1, 7, 45, 22, 53];
2
[4, 8, 12, 16, 2, 14, 96, 44, 106]
```

BackEnd :



DNS

www.akashEngintek.com	<u>172.168.3</u>
-----------------------	------------------

