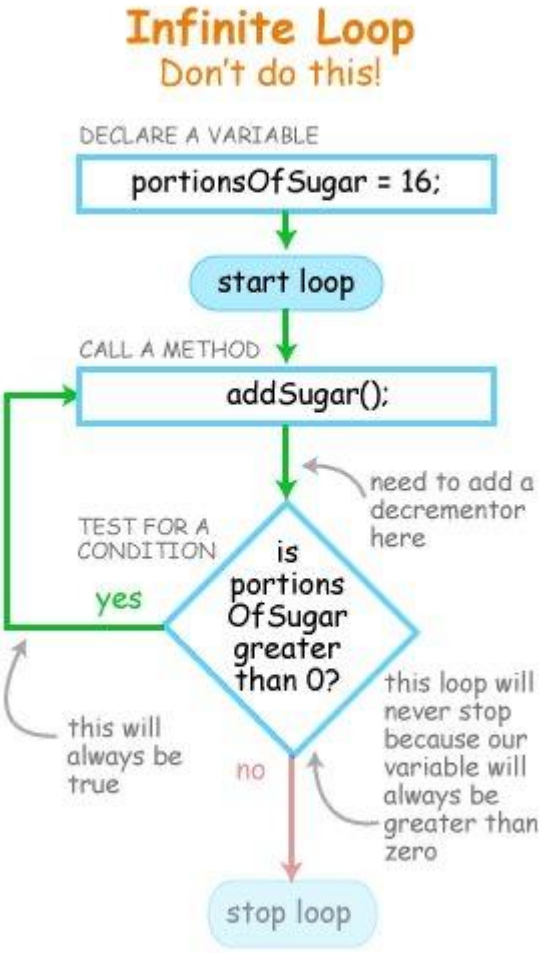
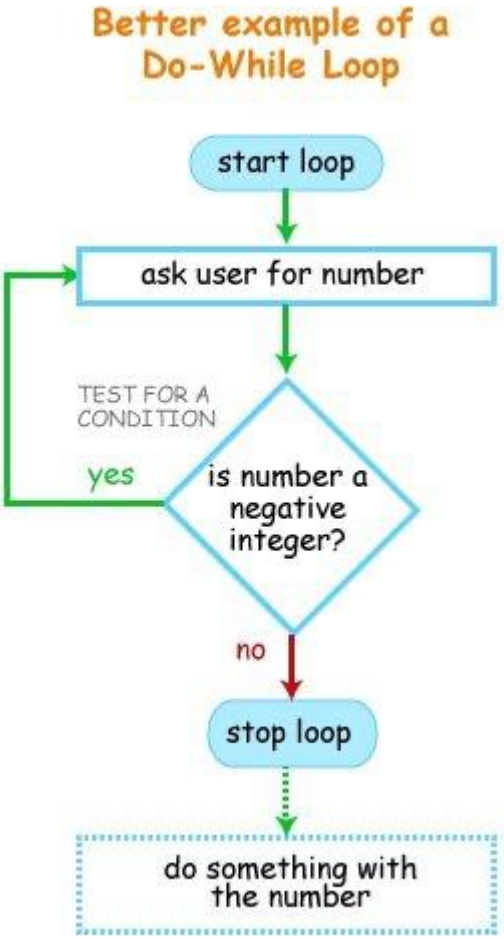
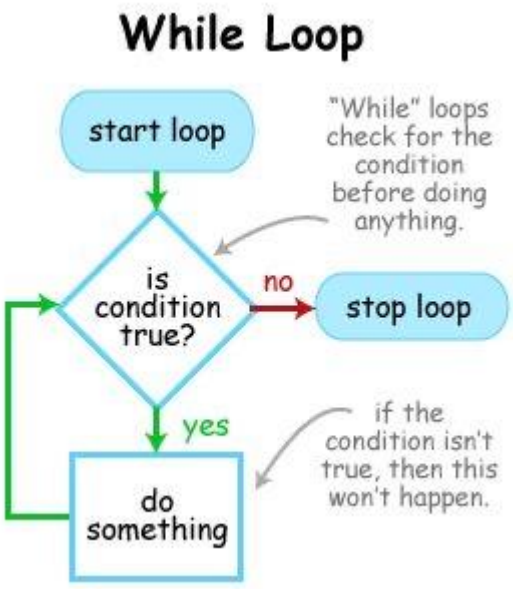
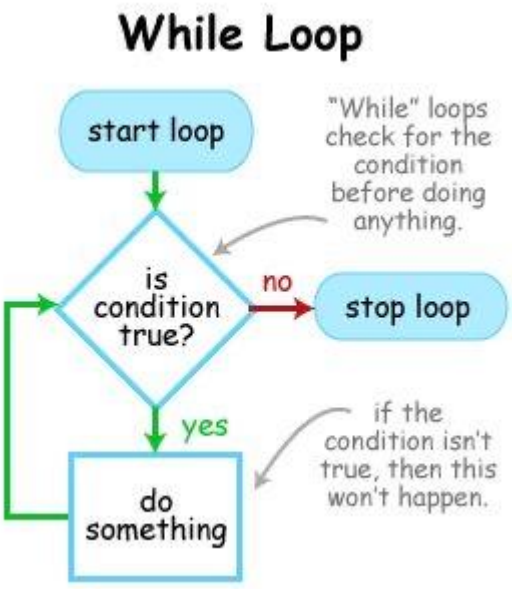
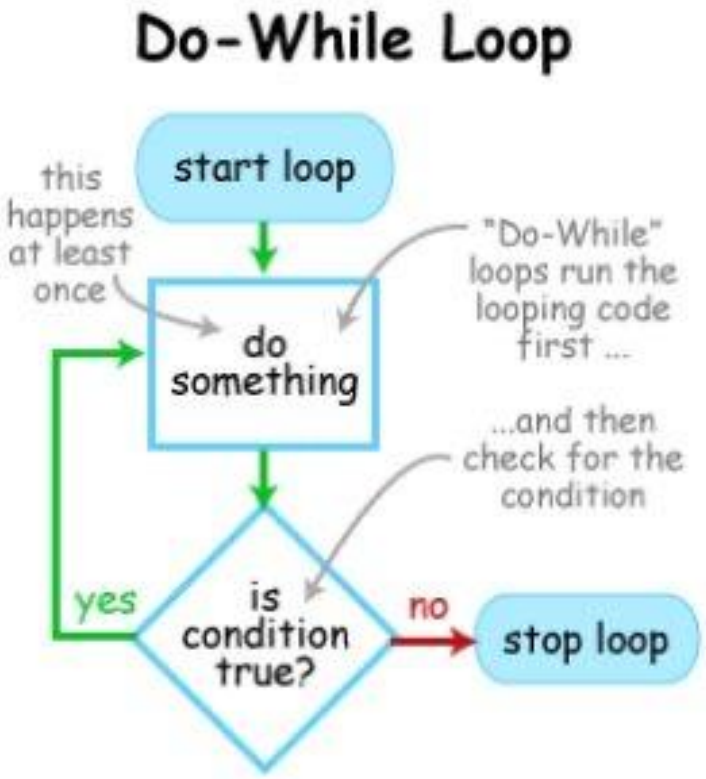

Infinite Loop
Don't do this!
DECLARE A VARIABLE
portionsOfSugar = 16;
start loop
CALL A METHOD
addSugar();
need to add a decrementor here
TEST FOR A CONDITION
yes
is portions Of Sugar greater than 0?
this will always be true
no
this loop will never stop because our variable will always be greater than zero
stop loop

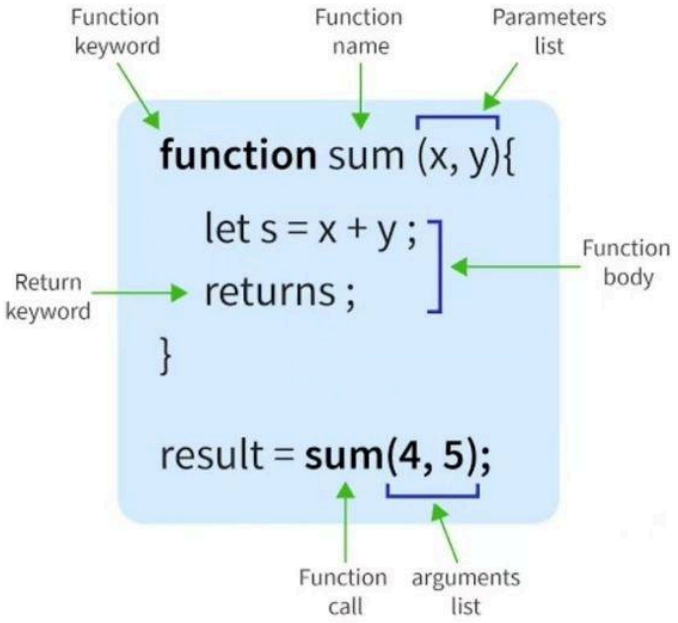

Better example of a Do-While Loop
start loop
ask user for number
TEST FOR A CONDITION
yes
is number a negative integer?
no
stop loop
do something with the number

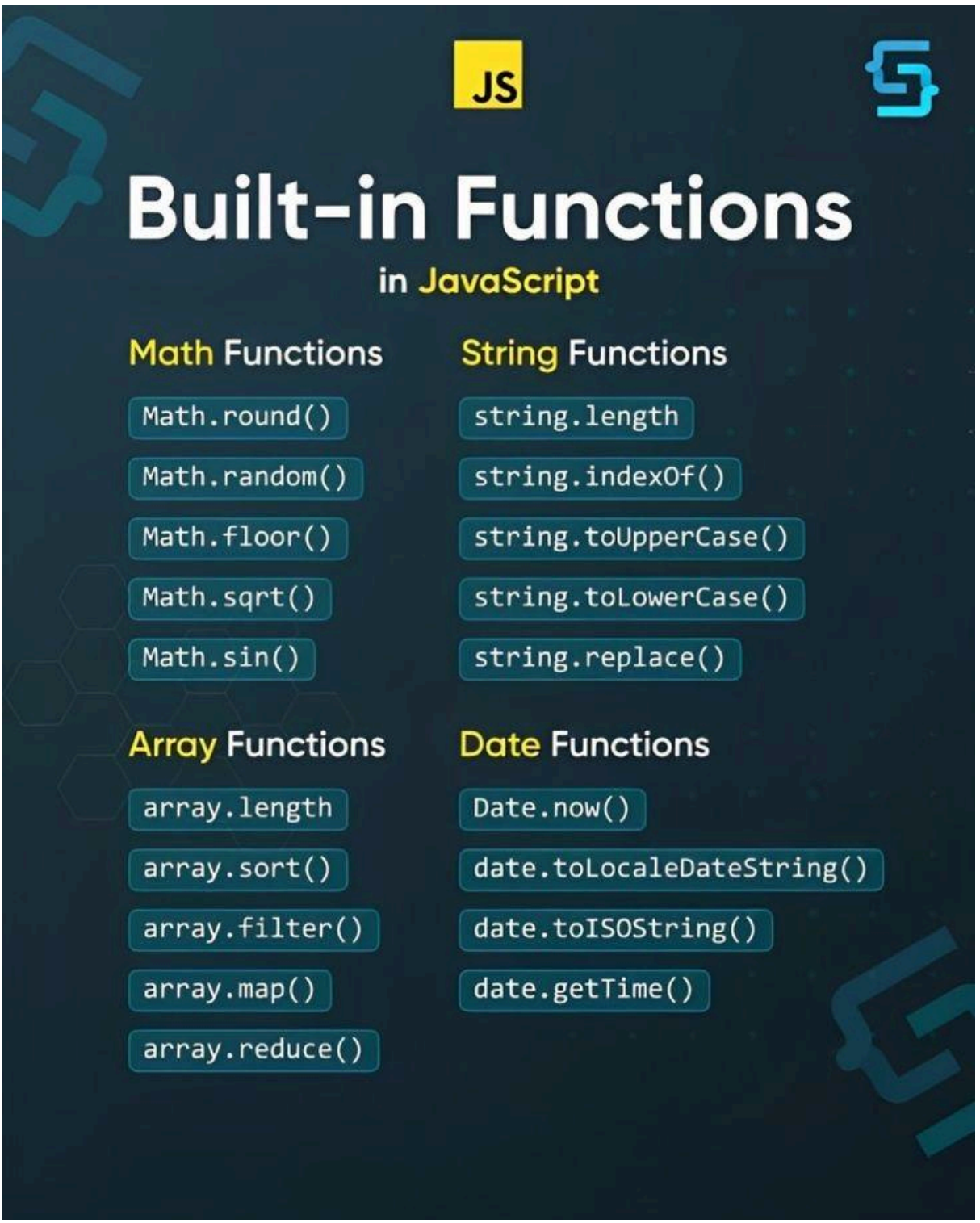

While Loop
start loop
"While" loops check for the condition before doing anything.
is condition true?
no
stop loop
yes
do something
if the condition isn't true, then this won't happen.


While Loop
start loop
"While" loops check for the condition before doing anything.
is condition true?
no
stop loop
yes
do something
if the condition isn't true, then this won't happen.



JavaScript Functions




JS
Built-in Functions
in JavaScript
Math Functions
Math.round()
Math.random()
Math.floor()
Math.sqrt()
Math.sin()
String Functions
string.length
string.indexOf()
string.toUpperCase()
string.toLowerCase()
string.replace()
Array Functions
array.length
array.sort()
array.filter()
array.map()
array.reduce()
Date Functions
Date.now()
date.toLocaleDateString()
date.toISOString()
date.getTime()

Array Methods

mutating original array

Add to original

`.push()`	`[1, 2, 3, 4]`	`.push(8)` →	`[1, 2, 3, 4, 8]`
`.unshift()`	`[1, 2, 3, 4]`	`.unshift(8)` →	`[8, 1, 2, 3, 4]`

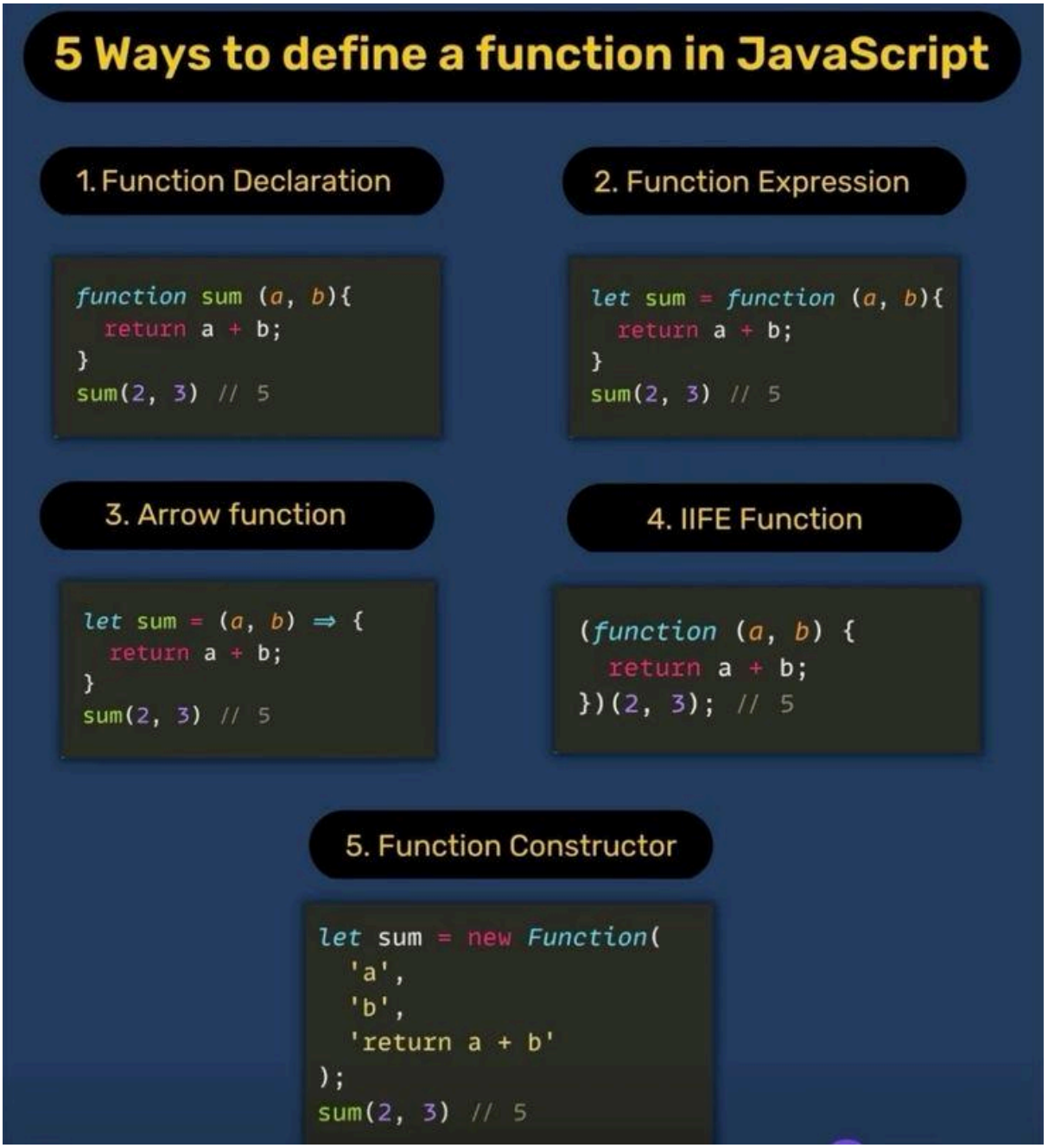
Remove from original

`.pop()`	`[1, 2, 3, 4]`	`.pop()` →	`[1, 2, 3]`
`.shift()`	`[1, 2, 3, 4]`	`.shift()` →	`[2, 3, 4]`
`.splice()`	`[1, 2, 3, 4]`	`.splice(2)` →	`[1, 2]`

Other

`.reverse()`	`[1, 2, 3, 4]`	`.reverse()` →	`[4, 3, 2, 1]`
`.sort()`	`[4, 2, 1, 3]`	`.sort()` →	`[1, 2, 3, 4]`
`.fill()`	`[1, 2, 3, 4]`	`.fill(8)` →	`[8, 8, 8, 8]`

@folizza


5 Ways to define a function in JavaScript
1. Function Declaration
function sum (a, b){
return a + b;
}
sum(2, 3) // 5
2. Function Expression
let sum = function (a, b){
return a + b;
}
sum(2, 3) // 5
3. Arrow function
let sum = (a, b) => {
return a + b;
}
sum(2, 3) // 5
4. IIFE Function
(function (a, b) {
return a + b;
})(2, 3); // 5
5. Function Constructor
let sum = new Function(
'a',
'b',
'return a + b'
);
sum(2, 3) // 5

Operators: Comparison

@codemecca

Operator:	Meaning:
var1 > var2	Greater than
var1 < var2	Less than
var1 != var2	Not equal
var1 == var2	Equal

Loop

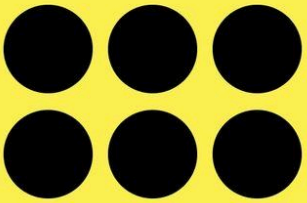
@codemecca

Loop Concept

Draw a circle

Outside a loop:

Within a loop:



Arrays

[ ]

0 1 2 3

fruits = [, , ,]

fruits[0] =

fruits.push

0 1 2 3 4

fruits = [, , , ,]

Objects

{ }

Object:

Properties:

First Name

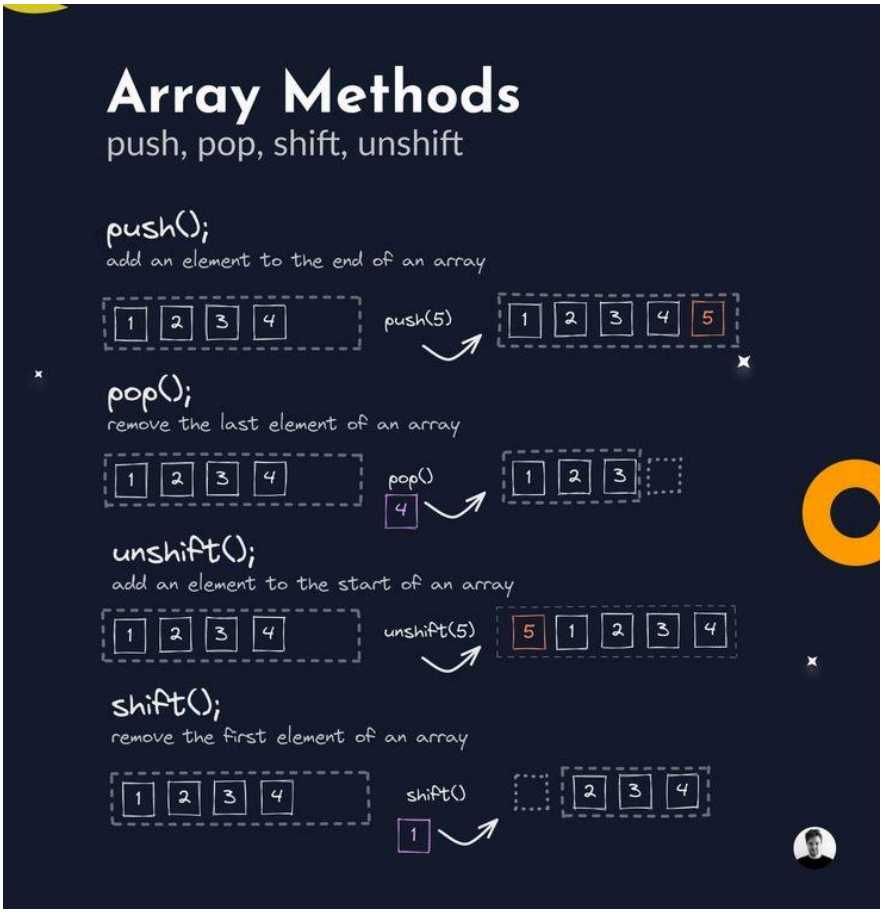
Last Name

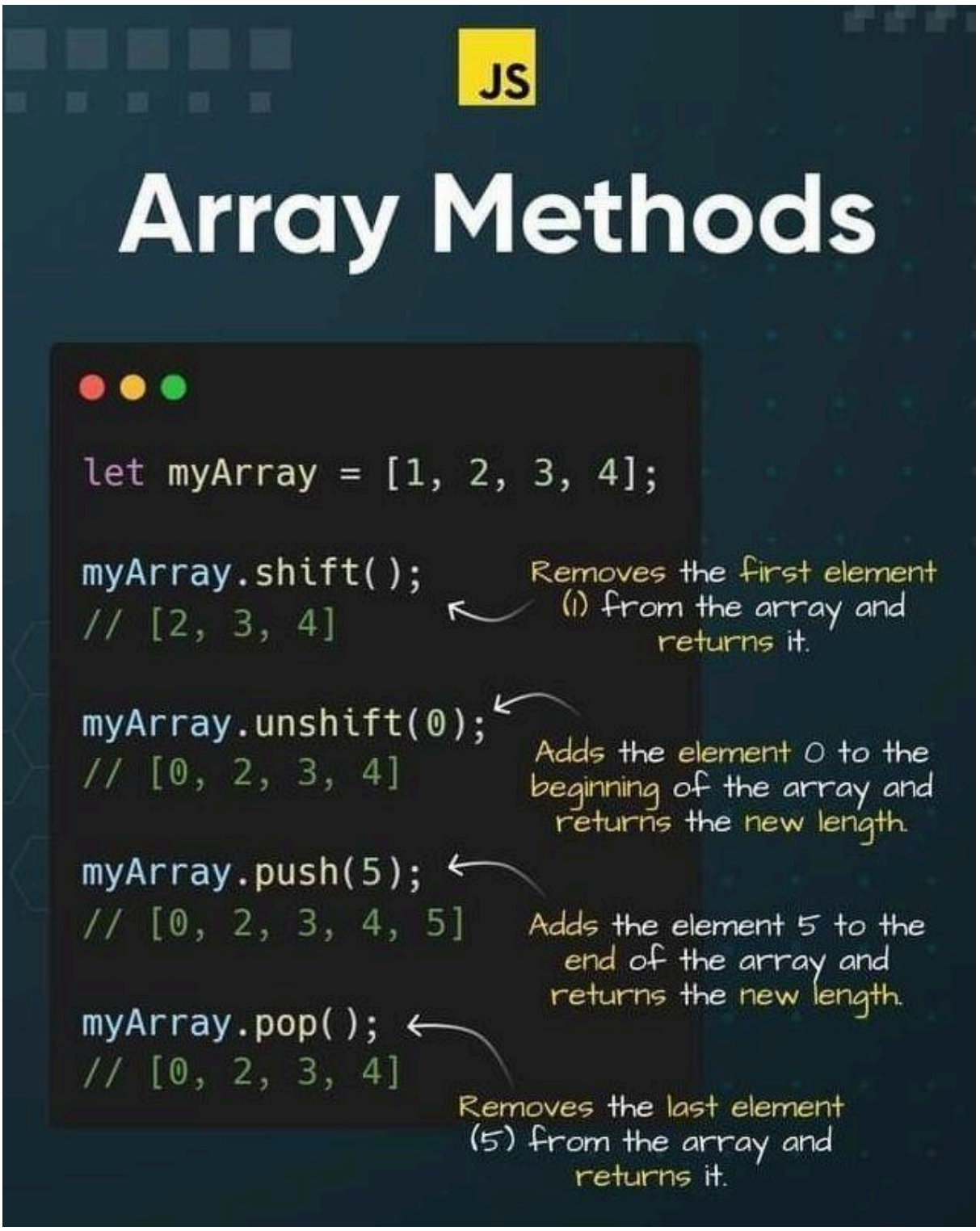
Height

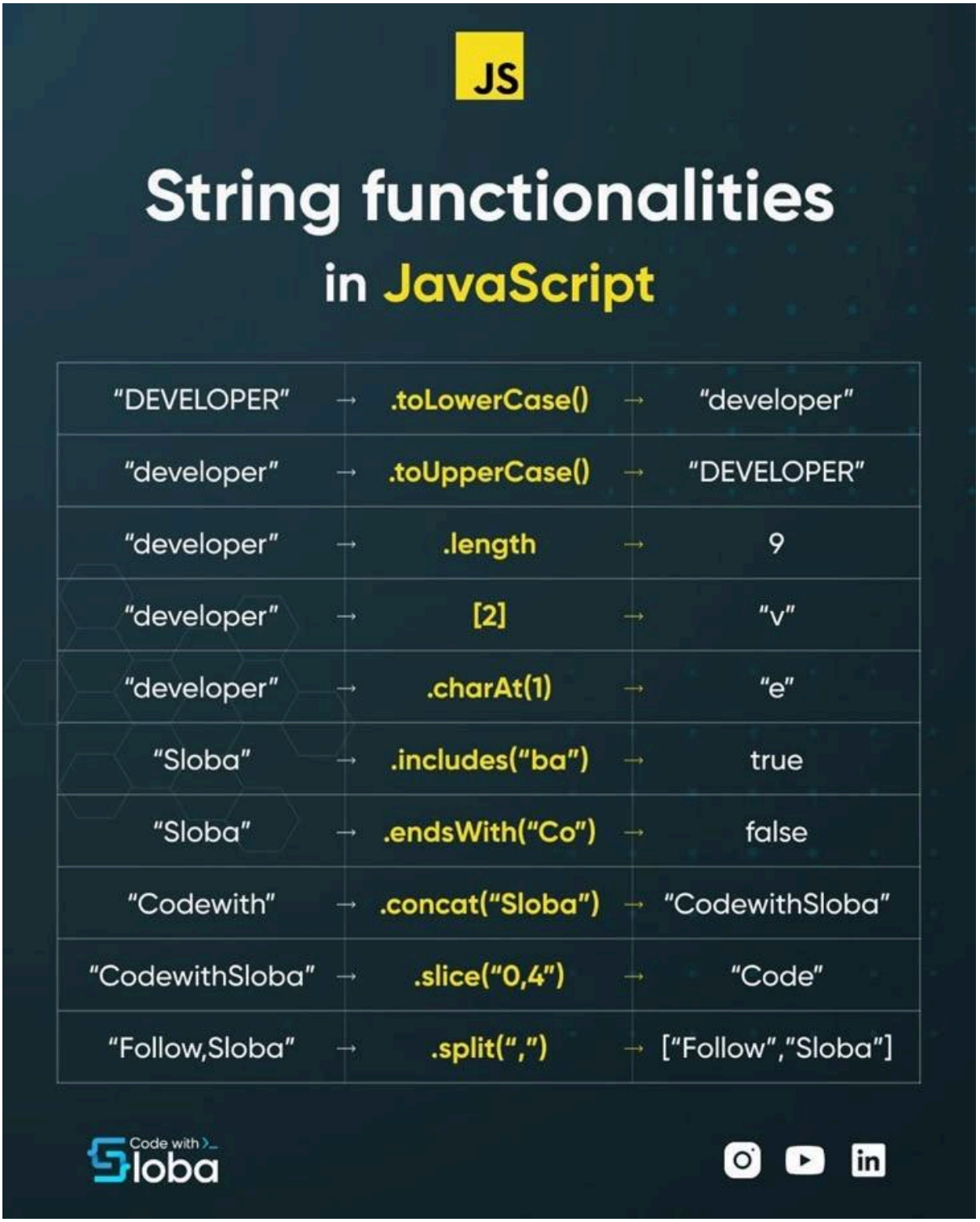
Age

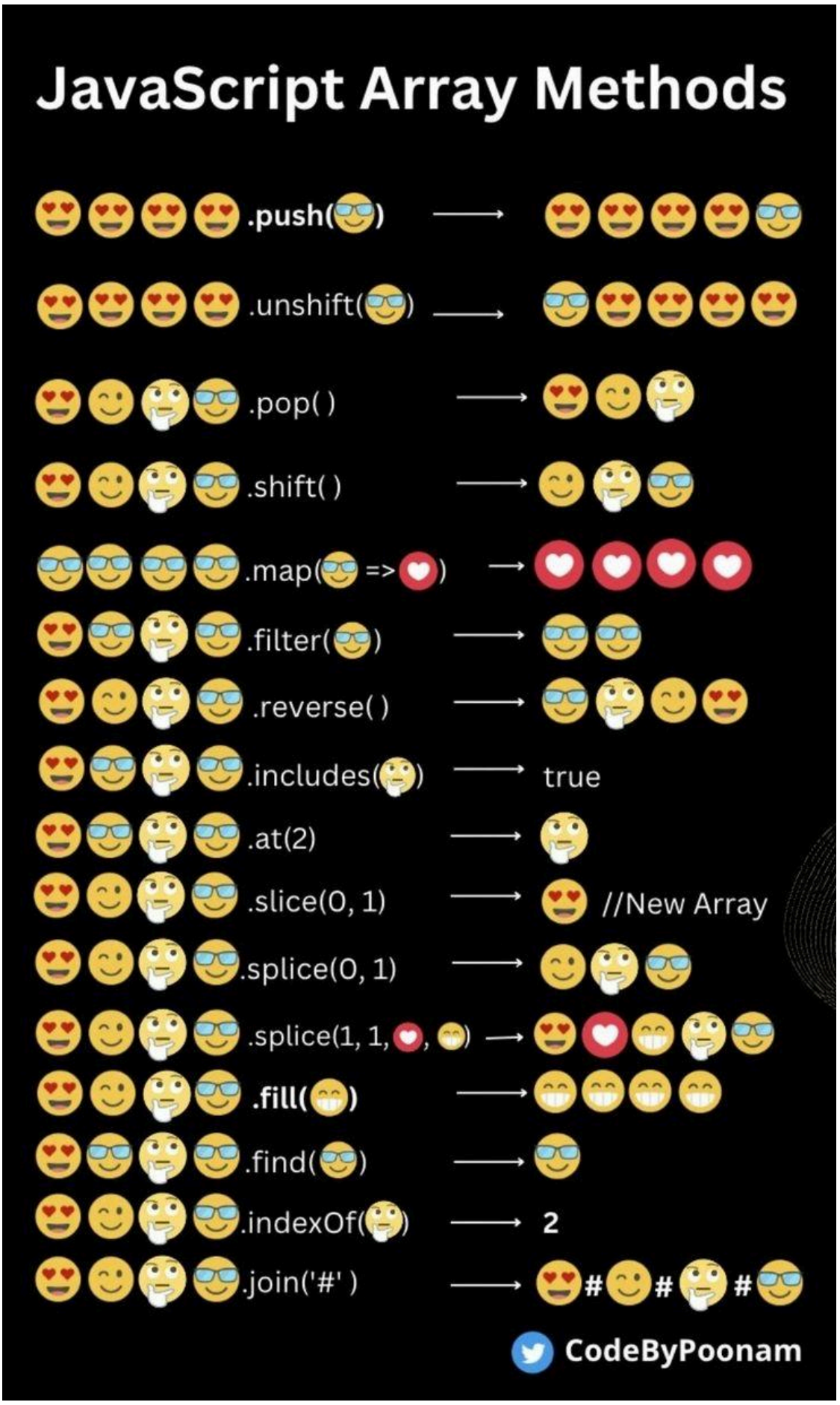
Operators: Logical ✓

Operator:	Meaning:
con1 && con2	Both True
con1 || con2	One True
!con	Not True




JS
Array Methods
let myArray = [1, 2, 3, 4];
myArray.shift();
// [2, 3, 4]
Removes the first element (1) from the array and returns it.
myArray.unshift(0);
// [0, 2, 3, 4]
Adds the element 0 to the beginning of the array and returns the new length.
myArray.push(5);
// [0, 2, 3, 4, 5]
Adds the element 5 to the end of the array and returns the new length.
myArray.pop();
// [0, 2, 3, 4]
Removes the last element (5) from the array and returns it.


JS
String functionalities
in JavaScript
"DEVELOPER" → .toLowerCase() → "developer"
"developer" → .toUpperCase() → "DEVELOPER"
"developer" → .length → 9
"developer" → [2] → "v"
"developer" → .charAt(1) → "e"
"Sloba" → .includes("ba") → true
"Sloba" → .endsWith("Co") → false
"Codewith" → .concat("Sloba") → "CodewithSloba"
"CodewithSloba" → .slice("0,4") → "Code"
"Follow,Sloba" → .split(",") → ["Follow","Sloba"]
Code with
Sloba


JavaScript Array Methods
.push( )
.unshift( )
.pop( )
.shift( )
.map(=>)
.filter( )
.reverse( )
.includes( )
true
.at(2)
.slice(0, 1)
//New Array
.splice(0, 1)
.splice(1, 1, ,)
.fill( )
.find( )
.indexOf( )
2
.join('#')
#
#
#
CodeByPoonam

Comparison Operators

Operators	Operations
==	Equal to
!=	Not Equal to
>	Greater than
>=	Greater than equal to
<	Less than
<=	Less than equal to

Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR (XOR)
<<	Bitwise Left Shift
>>	Bitwise Right Shift
~	One's Complement

Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator	A+B=30
– Subtraction	Subtracts the right-hand operator with left-hand operator	A-B=-10
* Multiplication	Multiplies values on either side of the operator	A*B=200
/ Division	Divides left hand operand with right hand operator	A/B=0
% Modulus	Divides left hand operand by right hand operand and returns remainder	A%B=0

Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator	A+B=30
– Subtraction	Subtracts the right-hand operator with left-hand operator	A-B=-10
* Multiplication	Multiplies values on either side of the operator	A*B=200
/ Division	Divides left hand operand with right hand operator	A/B=0
% Modulus	Divides left hand operand by right hand operand and returns remainder	A%B=0

Assignment Operators

Operator	Operation	Equivalent to
=	num = 5	num = 5
+=	num+=5	num = num+5
-=	num-=5	num = num-5
*=	num*=5	num = num*5
/=	num/=5	num = num/5
%=	num%=5	num = num%5

Logical Operators

Logical Operator	Java Operator
AND	&&
OR	||
NOT	!

```java
public class Test {

    Run | Debug
    public static void main(String[] args) {

        boolean a=true;
        boolean b=false;

        System.out.println("a && b is = " + (a&&b));  // false
        System.out.println("a || b is = " + (a||b));  // true
        System.out.println("!(a||b) is = " + !(a||b)); // false
    }

}
```

TypeScript Cheat Sheet —

Types, Interfaces, Generics, Decorators (ESSENTIAL)

Basic Types

```
let age: number = 21
let username: string = "James"
let isActive: boolean = true
let frults: string[] = ["applece"
let user: null = null
let data: any = "anything
```

Type Aliases

```
type Score = number | string
let testScore: Score = 88
```

Interfaces

```
interface User {
  id: number:
  name: string:
  active?: boolean
}
const ul: User = { id: 1, name "Ali"
```

Extending Interfaces

```
interface Admin extends User{
  role: "super" | "normal"
}
```

Generic Interfaces

```
interface ApiResponse<<T> {
  data: T:
  success: boolean
}
const p = new Persom["lam
```

James code Lab

Union Types

```
let id: number | string
id = 99
id = "A-12"
```

Functions

```
function sum[a: number, b:
number): number {
  return a + b
}
const greet = (name: string) =
string => "Hello
  $(name]
```

Generic Interfaces

```
enum Status {
  PENDING, SUCCESS
  FAILED
}
let current: Status = Satus.SUCC
```

Decorators (Experimental Feature)

```
function Log[target: any,
  key: string) {
  console.log["Property: $key?")
}
```

Narrowing

```
function printId(x number | string)
  if (typeof x === "String")
  console.log(x.toupperCasei())
```

Prototypes

```
function Person(name) {
  this.name = name;
}
Person.prototype.sayHi = function () {
  return `Hi ${this.name}`;
};
```

Classes

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    return `${this.name} makes noise.`;
  }
}
```

1. Basic

Variables

Comment "**JS**" for Full PDF

```
let a = 10;        // block scoped
const b = 20;      // constant
var c = 30;        // function scoped (avoid using)
```

Data Types

```
let str = "Hello";      // string
let num = 123;          // number
let bool = true;        // boolean
let n = null;           // null
let u;                  // undefined
let obj = { key: "val" };
let arr = [1, 2, 3];
```

Type Conversion

```
Number("123");      // 123
String(123);        // "123"
Boolean(0);         // false
```

2. Control Structures

if-else

```
if (a > b) {
  console.log("A");
} else {
  console.log("B");
}
```

switch

```
switch (day) {
  case 1: console.log("Mon"); break;
  default: console.log("Other");
}
```

loops

```
for (let i = 0; i < 5; i++) {}
while (condition) {}
do {} while (condition);
```

3. Operators

```
+ - * / % **        // arithmetic
== === != !==       // equality
< <= > >=           // comparison
&& || !             // logical
= += -= *=          // assignment
```

4. Functions

```
function greet(name) {
  return `Hi, ${name}`;
}

// Arrow Function
const sum = (a, b) => a + b;

// Default Parameters
function greet(name = "Guest") {}

// Rest Parameters
function total(...nums) {
  return nums.reduce((a, b) => a + b);
}
```

5. Arrays

```
let fruits = ["apple", "banana"];

// Methods
fruits.push("mango");
fruits.pop();
fruits.shift();
fruits.unshift("kiwi");
fruits.includes("banana");
fruits.indexOf("apple");

// Iteration
fruits.forEach(f => console.log(f));
let newArr = fruits.map(f => f.toUpperCase());
let filtered = fruits.filter(f => f !== "banana");

// Spread
let all = [...fruits, "grape"];
```

6. Objects

```
let user = {
  name: "John",
  age: 25,
  greet() {
    return `Hi, ${this.name}`;
  }
};

console.log(user.name);
delete user.age;

// Destructuring
const { name } = user;
```

7. ES6+ Features

```
// Template Literals
let msg = `Hello, ${name}`;

// Destructuring
let [a, b] = [1, 2];

// Spread & Rest
let obj2 = { ...user, city: "NY" };

// Optional Chaining
let city = user?.address?.city;

// Nullish Coalescing
let val = input ?? "default";

// Short-circuiting
let isAdmin = user.isAdmin && true;
```

8. DOM Manipulation

```
document.getElementById("id");
document.querySelector(".class");

let el = document.createElement("p");
el.textContent = "Hello";
document.body.appendChild(el);

// Events
el.addEventListener("click", () => alert("Clicked"));
```

9. Async JS

Callback

```
setTimeout(() => console.log("Hi"), 1000);

// Promise
fetch("api/data")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Async/Await

```
async function getData() {
  try {
    let res = await fetch("api/data");
    let data = await res.json();
  } catch (err) {
    console.error(err);
  }
}
```

Comment "**JS**" for Full PDF

10. Advanced Topics

Closure

```
function outer() {
  let count = 0;
  return function inner() {
    count++;
    return count;
  };
}
```

Closure

```
console.log(a); // undefined
var a = 10;
```

this keyword

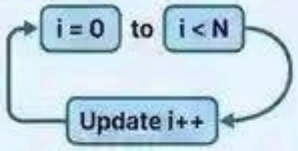
```
const obj = {
  name: "JS",
  print: function () {
    console.log(this.name);
  }
};
```

11. Useful Methods

```
// Array
arr.reduce((a, b) => a + b);
arr.find(e => e === 3);
arr.every(e => e > 0);
arr.some(e => e === 2);

// String
str.includes("H");
str.split(" ");
str.toUpperCase();
str.replace("H", "h");

// Number
parseInt("123");
parseFloat("12.3");

// Object
Object.keys(obj);
Object.values(obj);
Object.entries(obj);
```

TYPES OF LOOPS IN JAVASCRIPT

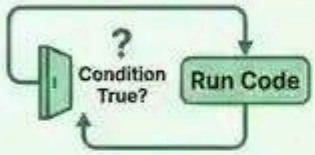
(Simple Explanation)



1. `for` loop

```
for (let i=0; i<5; i++) {
  ...
}
```

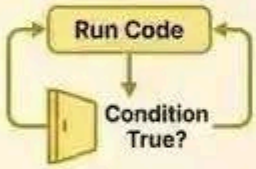
Repeat a specific number of times. Best for **known iterations** (e.g., count from 0 to 9).



2. `while` loop

```
while (condition) {
  ...
}
```

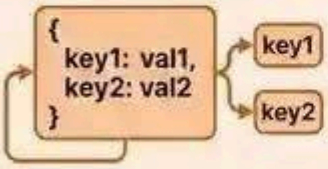
Repeat while a condition is true.
Checks condition *before* each loop.



3. `do...while` loop

```
do {
  ...
} while (condition);
```

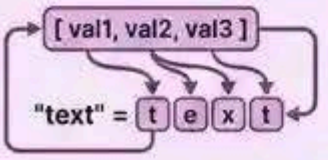
Run once, then repeat while a condition is true.
Guarantees at least one execution.



4. `for...in` loop

```
for (const key in object) {
  ...
}
```

Loop through **object properties** (keys).
Iterates over enumerable properties.



5. `for...of` loop

```
for (const value of array) {
  ...
}
```

Loop through **iterable values** (arrays, strings, etc.). Iterates over the data itself.

Visual Guide to **JavaScript Loop Control Flow.** Note: Modern JS often prefers array methods (map, forEach) for lists.

JavaScript Objects

✪ In JavaScript, almost "everything" is an object.

```
const Person = {
        firstName:"Adarsh",
        secondName:"Gupta",
        age:"19"
}
```

✪ A JavaScript object is a collection of named values

✪ Objects is an unordered collection of key value pair

Object Creation

@adarsh____gupta

You can create objects in multiple ways
1) Using Object Literal
2) Using new Keyword
3) Using Object.create()

```
//object literal
const Person = {
        firstName:"Adarsh",secondName:"Gupta",age:"19"
}
//Using the JavaScript  new Keyword
const Person2=new Object()

Person.firstname="John"
Person.secondname="Cena"

//using object.create()
const person = {
  isHuman: false
  }
const human1= Object.create(person)
human1.isHuman=true
```

Object Properties

✪ Properties are the values associated with a JavaScript object.

✪ Properties are mutable,ie ,it can be changed.

Accesing Properties

```
objectName.property
 // person.age

objectName["property"]
  // person["age"]

objectName[expression]
// x = "age"; person[x]
```

Objects to Array

✪ We can asily convert object to array

```
const person = {
  name: "Adarsh",
  age: 30,
  country: "India"
};

const myArray = Object.values(person);
//myArray =  ["Adarsh",30,"India"]
```

✪In thee same way we can convert objects into Sting using JSON.stringify()

.

this keyword refers to current value inside the object

If you understand objects ,you understand javascript,

Everything is javascript can be a object ,(string,number,array...)

Object Methods

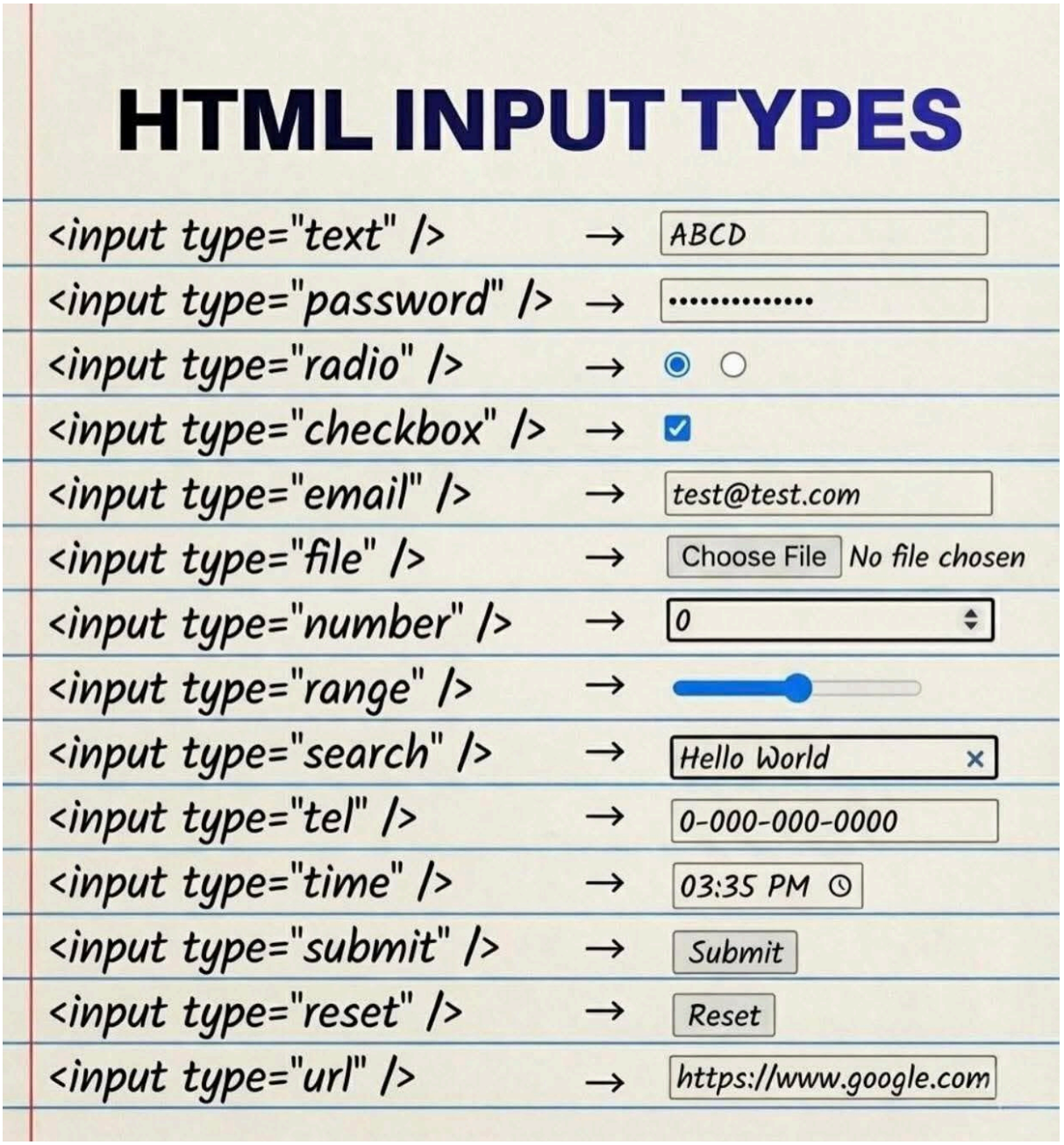
✪ Methods are like function inside objects that will do something

```
const person = {
  firstName: "Adarsh",
  lastName: "Gupta",
  fullName: function() {
    console.log(this.firstName + " " + this.lastName);
  }
};

person.fullName() // Adarsh Gupta
```

Constructor

✪ A blue Print for creating objects

```
function Footballer(Playername, team, Goal) {
  this.Playername = Playername;
  this.team = team;
  this.Goal = Goal;
}

const player1=new Footballer("Messi","PSG",500)
const player2=new Footballer("Ronaldo","M.city",700)
```


HTML INPUT TYPES
<input type="text" /> → ABCD
<input type="password" /> →
<input type="radio" /> →
<input type="checkbox" /> →
<input type="email" /> → test@test.com
<input type="file" /> → Choose File No file chosen
<input type="number" /> → 0
<input type="range" /> →
<input type="search" /> → Hello World
<input type="tel" /> → 0-000-000-0000
<input type="time" /> → 03:35 PM
<input type="submit" /> → Submit
<input type="reset" /> → Reset
<input type="url" /> → https://www.google.com