Operations in JS

```
a = 20
b = 10

//addition
sum = a + b

//subtraction
diff = a - b

//multiplication
prod = a * b
```


odd

- Modulo (remainder operator)

12 % 5 = 2

- Exponentiation (power operator)

2**3 = 8

a**b =

NaN in JS

The NaN global property is a value representing Not-A-Number.

```
0 / 0

NaN - 1

NaN * 1

NaN + NaN
```

Operator Precendence

This is the general order of solving an expression.

BODMAS

( )

↑

**

↑

*, /, %

↑

+, -

(5+2) / 7 + 1 * 2

(7) / 7 + 1 * 2

1 + 1

let keyword

Syntax of declaring variables

```
let age = 23 ;

age = age + 1 ;
```

age = 23 + 1 = 24

```
let cgpa ;

cgpa = 8.9
```

```
let num1 = 1 ;

let num2 = 2 ;

let sum = num1 + num2 ;
```

const keyword

values of constants can't be changed with re-assignment & they can't be re-declared

```
const year = 2025;

year = 2026          // Error

year = year + 1      // Error
```

```
const pi = 3.14 ;

const g = 9.8 ;
```

Assignment Operators

```
age = age + 1

age += 1


age = age - 1

age -= 1


age = age * 1

age *= 1
```

Unary Operators

```
age = age + 1
age += 1
age++ // increment
```

```
age = age - 1
age -= 1
age-- // decrement
```

Unary Operators

Pre-increment (Change, then use)

```
let age = 10 ;

let newAge = ++age ;
```

Post-increment (Use, then change)

```
let age = 10 ;

let newAge = age++ ;
```

Identifier Rules

All JavaScript variables must be identified with unique names (identifiers).

- Names can contain letters, digits, underscores, and dollar signs. (no space)
- Names must begin with a letter.
- Names can also begin with $ and _ .
- Names are case sensitive (y and Y are different variables).
- Reserved words (like JavaScript keywords) CANNOT be used as names.

camelCase

Way of writing identifiers

let totalPrice

camelCase (JS naming convention) camelCase let fullName;

snake_case let full_name;

PascalCase let FullName;

Boolean in JS

Boolean represents a truth value -> true or false / yes or no

```
let age = 23 ;
let isAdult = true ;
```

var type change

```
let age = 13 ;
let isAdult = false ;
```

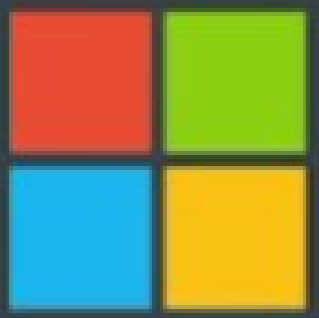
What is TypeScript?

Static Typed, where JS is dynamic typed

fixed

change

Designed by Microsoft



String in JS

Strings are text or sequence of characters

→ word / page

```
let name = "Tony Stark" ;

let role = 'ironman' ;

let char = 'a' ;

let num = '23' ;

let empty = " " ;
```

String Indices

```
let name = "TONY STARK" ;
```

T	O	N	Y		S	T	A	R	K
0	1	2	3	4	5	6	7	8	9

name[0] -> 'T'

name[1] -> 'O' ...

Concatenation

adding strings together

"tony" + " " + "stark" = "tony stark"

"tony" + 1 = "tony1"

null & undefined in JS

undefined

A variable that has not been assigned a value is of type undefined.

```
> let a;
<· undefined
> a
<· undefined
```

null

The null value represents the intentional absence of any object value.

To be explicitly assigned.

```
> let a = null;
<· undefined
> a
<· null
```

console.log( )

To write (log) a message on the console

```
console.log("Apna College");

console.log(1234);

console.log(2+2);

console.log("Apna", "College", 123);
```

Linking JS File

```
<script src="app.js"> </script>
```

Template Literals

They are used to add embedded expressions in a string.

```
let a = 5;

let b = 10;

console.log(`Your pay ${a + b} rupees`);

// console.log("Price is", a+b, "rupees");
```

Template Literals

They are used to add embedded expressions in a string.

```
let a = 5;

let b = 10;

console.log(`Your pay ${a + b} rupees`);

// console.log("Price is", a+b, "rupees");
```

Comparison Operators

Comparison Operators to compare 2 values

`>` → Greater than

`>=` → greater than or equal to

`<` → lesser than

`<=` → lesser than or equal to

`==` → equal to

`!=` → not equal to

Comparison Operators

==

- compares value, not type

```
> "123" == 123
< true
> 1 == '1'
< true
> 0 == ' '
< true
> 0 == false
< true
> null == undefined
< true
```

===

- compares type & value

```
> "123" === 123
< false
> 1 === '1'
< false
> 0 === ' '
< false
> 0 === false
< false
> null === undefined
< false
```

truthy & falsy

Everything in JS is true or false (in boolean context).

This doesn't mean their value itself is false or true, but they are treated as false or true if taken in boolean context.

Falsy values

false, 0, -0, 0n (BigInt value), "" (empty string) , null, undefined, NaN

Truthy values

Everything else

Switch Statement

Used when we have some fixed values that we need to compare to.

```
let color = "red";

switch(color) {
    case "red" :
        console.log("stop");
        break;
    case "yellow" :
        console.log("slow down");
        break;
    case "green" :
        console.log("GO");
        break;
    default :
        console.log("Broken Light");

}
```

Alert & Prompt

Alert displays an alert message on the page.

```
alert("something is wrong!");
```

Prompt displays a dialog box that asks user for some input.

```
prompt("please enter your roll no.");
```

```
console.log("this is a simple log");
console.error("this is an error msg");
console.warn("this is a warning msg");
```

String Methods

let msg = " Hello ";

str.trim()

Trims whitespaces from both ends of string & returns a new one.

```
> let msg = "   Hello   ";
<· undefined
> msg.trim();
<· 'Hello'
> msg
<· '   Hello   '
```

output : "Hello", but value of msg remains same.

String Methods

```
let str = "Random string";
```

```
str.toUpperCase( )        "RANDOM STRING"

str.toLowerCase( )        "random string"
```

indexOf

Returns the first index of occurrence of some value in string. Or gives -1 if not found.

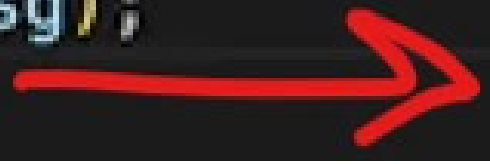
```
let str = "ILoveCoding";
```


```
str.indexOf("love")      // 1

str.indexOf("J")         // -1 (not found)

str.indexOf("o")         // 2 (only 1 index)
```

```
let msg = "    hello    ";
// let newMsg = msg.trim();
// console.log("after trim : ", newMsg);
// newMsg = newMsg.toUpperCase();
// console.log("after uppercase : ", newMsg);
let newMsg = msg.trim().toUpperCase();
console.log(newMsg);
```



slice

Returns a part of the original string as a new string.

```
let str = "IloveCoding";

    str.slice(5)                    // "Coding"

    str.slice(1, 4)                 // "love"

    str.slice(-num)   =   str.slice(length-num)
```

replace

Searches a value in the string & returns a new string with the value replaced.

```
let str = "IloveCoding";

    str.slice("love", "do")    // "IdoCoding"

    str.slice("o", "x")        // "IlxveCoding"
```

repeat

Returns a string with the number of copies of a string

```
let str = "Mango";

str.repeat(3)                    // "MangoMangoMango"
```

```
let students = ["aman", "shradha", "rajat"]
```

Creating Arrays

```
let marks = [99, 85, 93, 76, 62];
let names = ["adam", "bob", "catlyn"];
let info = ["aman", 25, 6.1];    //mixed array


//empty array
let newArr = [];
```

Array Methods

Push : add to end

Unshift : add to start

Pop : delete from end & returns it

Shift : delete from start & returns it

Array Methods

indexOf : returns index of something

```
> primary.indexOf("yellow");
<· 1
> primary.indexOf("green");
<· -1
> primary.indexOf("Yellow");
<· -1
```

includes : search for a value

```
> primary.includes("red");
<· true
> primary.includes("green");
<· false
```

Array Methods

```
> let primary = ["red", "yellow", "blue"];
< undefined
> let secondary = ["orange", "green", "violet"];
< undefined
```

concat : merge 2 arrays ⟶ concatenate

```
> primary.concat(secondary);
< ▶ (6) ['red', 'yellow', 'blue', 'orange', 'green', 'violet']
```

reverse : reverse an array

```
> primary.reverse();
< ▶ (3) ['blue', 'yellow', 'red']
```

Array Methods

slice : copies a portion of an array

```
> let colors = ["red", "yellow", "blue", "orange", "pink", "white"];
```

```
> colors.slice()
<- ▶ (6) ['red', 'yellow', 'blue', 'orange', 'pink', 'white']
> colors.slice(2);
<- ▶ (4) ['blue', 'orange', 'pink', 'white']
> colors.slice(2, 3);
<- ▶ ['blue']
> colors.slice(-2);
<- ▶ (2) ['pink', 'white']
```

Array Methods

```
> let colors = ["red", "yellow", "blue", "orange", "pink", "white"];
```

splice : removes / replaces / add elements in place

splice(start, deleteCount, item0...itemN)

↑ OP

```
> colors.splice(4);
◂ ▸ (2) ['pink', 'white']
> colors
◂ ▸ (4) ['red', 'yellow', 'blue', 'orange']
> colors.splice(0, 1);
◂ ▸ ['red']
> colors
◂ ▸ (3) ['yellow', 'blue', 'orange']
> colors.splice(0, 1, "black", "grey");
◂ ▸ ['yellow']
> colors
◂ ▸ (4) ['black', 'grey', 'blue', 'orange']
```

Array Methods

ascending
descending

sort : sorts an array

```
> let days = ["monday", "sunday", "wednesday", "tuesday"];
<· undefined
> days.sort();
<· ▶ (4) ['monday', 'sunday', 'tuesday', 'wednesday']
```

```
> let squares = [25, 16, 4, 49, 36, 9]
<· undefined
> squares.sort();
<· ▶ (6) [16, 25, 36, 4, 49, 9]
```

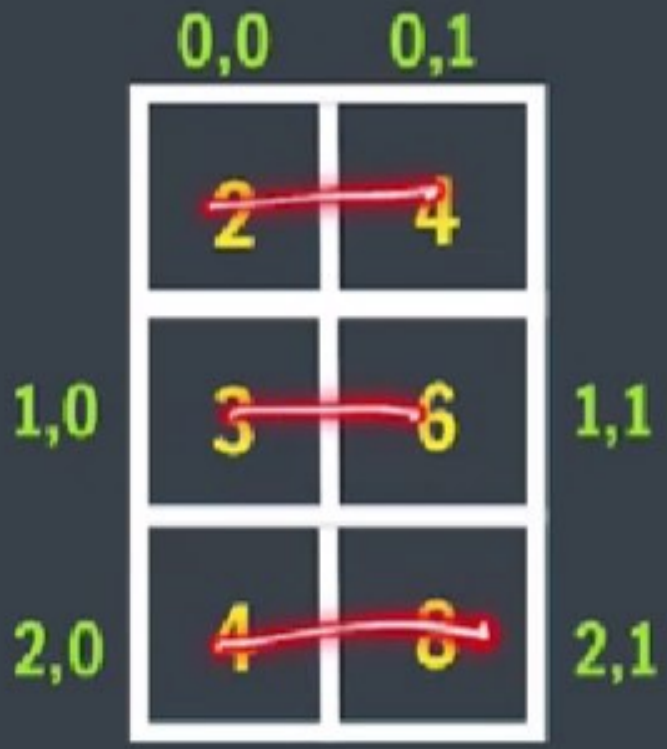
Array References

```
> [1] === [1]
<· false
```

```
> [1] == [1]
<· false
```

Nested Arrays

```
> let nums = [ [2, 4], [3, 6], [4, 8] ];
```



rows = arrays = 3
cols = indx.

nums[0][0] //2

for of loop

```
for (element of collection) {

  //do something

}
```

```
let fruits = ["mango", "apple", "banana", "litchi", "orange"];

for(fruit of fruits) {
    console.log(fruit);
}
```

```
for(char of "apnacollege") {
    console.log(char);
}
```

Nested for of loop

```
let heroes = [ ["ironman", "spiderman", "thor"], ["superman", "wonder woman", "flash"]];

for(list of heroes) {
    for(hero of list) {
        console.log(hero);
    }
}
```

JS Objects Literals

```
let delhi = {
    latitude: "28.7041° N",
    longitude: "77.1025° E"
};
```

```
const student = {
    name: "shradha",
    age: 23,
    marks: 94.4,
    city: "Delhi"
};
```

Get Values

```
let student = {
    name : "shradha",
    marks : 94.4
};

student["name"]

student.name
```

Object of Objects

Storing information of multiple students

```
const classInfo = {
    aman : {
        grade: "A+",
        city: "Delhi"
    },
    shradha : {
        grade: "A",
        city: "Pune"
    },
    karan : {
        grade: "O",
        city: "Mumbai"
    }
};
```

Array of Objects

Storing information of multiple students

```
const classInfo = [
    {
        name: "aman",
        grade: "A+",
        city: "Delhi"
    },
    {
        name: "shradha",
        grade: "A+",
        city: "Pune"
    },
    {
        name: "karan",
        grade: "O",
        city: "Mumbai"
    }
];
```

Math Object

Properties

Math.PI

Math.E

Methods

Math.abs(n)

Math.pow(a, b)

Math.floor(n)

Math.ceil(n)

Math.random( )

Random Integers

From 1 to 10

Step1 :

```
let num = Math.random();
```

```
0.46747741318127045
```

Step2 :

```
num = num * 10;
4.674774131812704
```

Step3 :

```
num = Math.floor(num);
4
```

Step4 :

```
num = num + 1;
5
```

Random Integers

From 1 to 10

```
let random = Math.floor(Math.random() * 10) + 1;
undefined
```

```
random
4
```

Qs

Generate a random number between 1 and 100.

max

```
Math.floor(Math.random() * 100) + 1
```

0 to 99

5

0, 1, 2, 3, 4

+20

20, 21, 22, 23, 24

21, 22, 23, 24, 25

Generate a random number between 1 and 5.

21 to 25

21, 22, 23, 24, 25

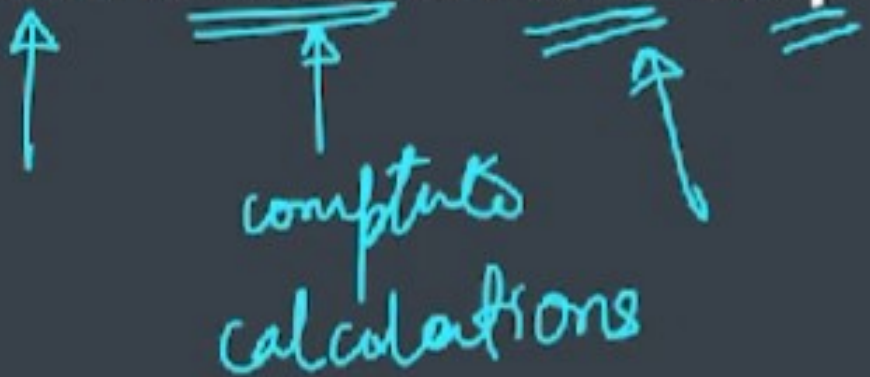
Using the Console

→ vs code

.html
.css
.js

Uses REPL

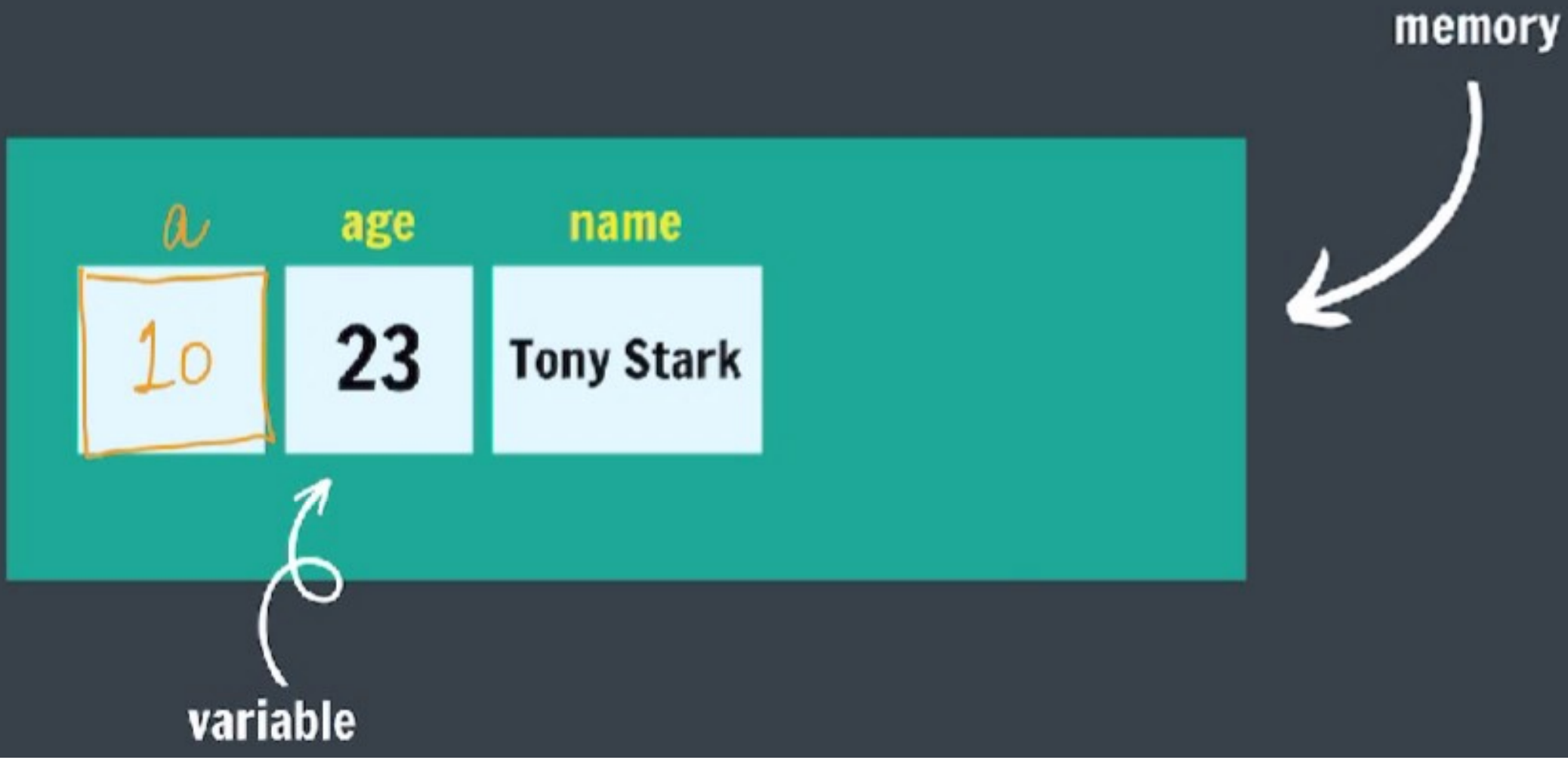
Read-Evaluate-Print-Loop



clear → cmd + K
ctrl + d

What is a Variable?

A variable is simply the name of a storage location.



Data Types in JS

Primitive Types

- Number
- Boolean
- String
- Undefined
- Null
- Bigint
- Symbol

Numbers in JS

- Positive (14) & Negative (-4)
- Integers (45, -50)
- Floating numbers - with decimal (4.6, -8.9)

```
a = 0.9999999999
0.9999999999
a = 0.99999999999999999999
1
```