JavaScript

by: Alireza Mohammadi Kordkheili

https://websila.co

Intro & Getting Started

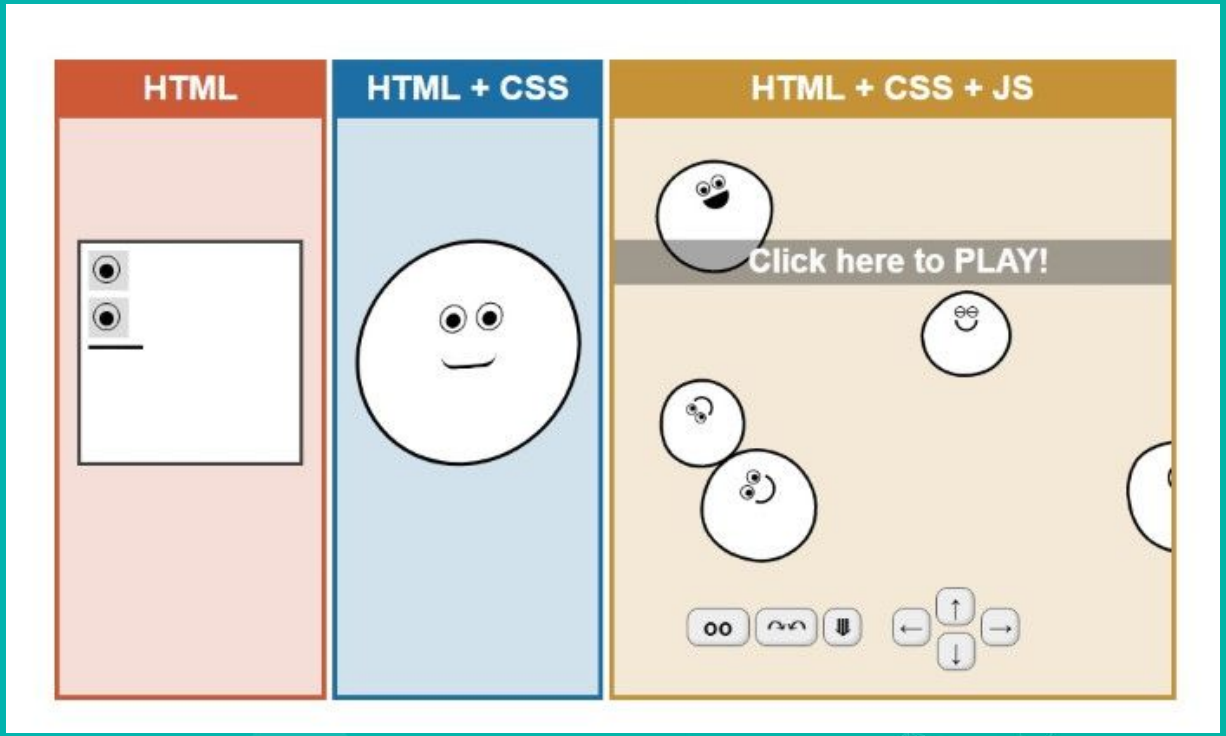
What is JavaScript?

- JavaScript is a **dynamic**, **weakly typed** programming language which is **compiled at runtime.**
- JavaScript is a programming language that allows you to **implement complex features on web pages** — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies.
- JavaScript is **one of the core technologies of the World Wide Web**, alongside HTML and CSS.
- JavaScript was created **to make web pages more dynamic.**
- JavaScript **can update and change both HTML and CSS.**
- JavaScript is totally **independent** from Java.

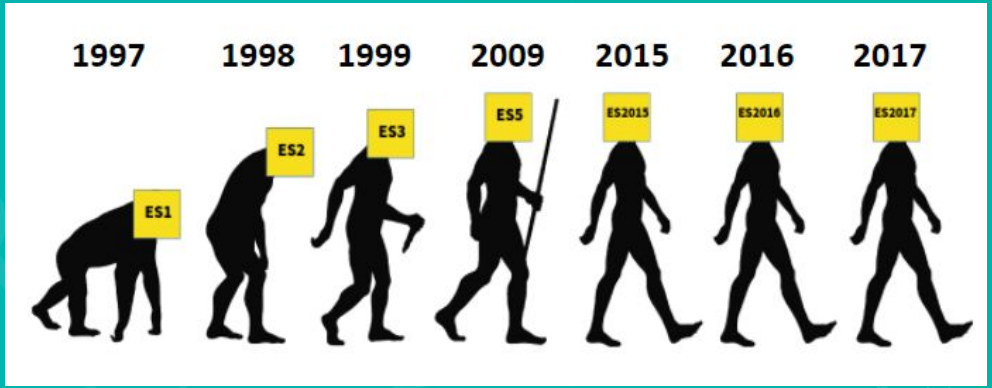
JavaScript is Cross-Platform

- ➔ Front-End Development (Client-Side - Vue.js, React.js, Angular.js)
- ➔ Back-End Development (Server-Side - Node.js)
- ➔ macOS, Windows, and Linux (electron.js)
- ➔ Mobile App Development (React Native)

JavaScript in Action



What is ECMAScript?



Setting Up a Development Environment (Tools & Setup)

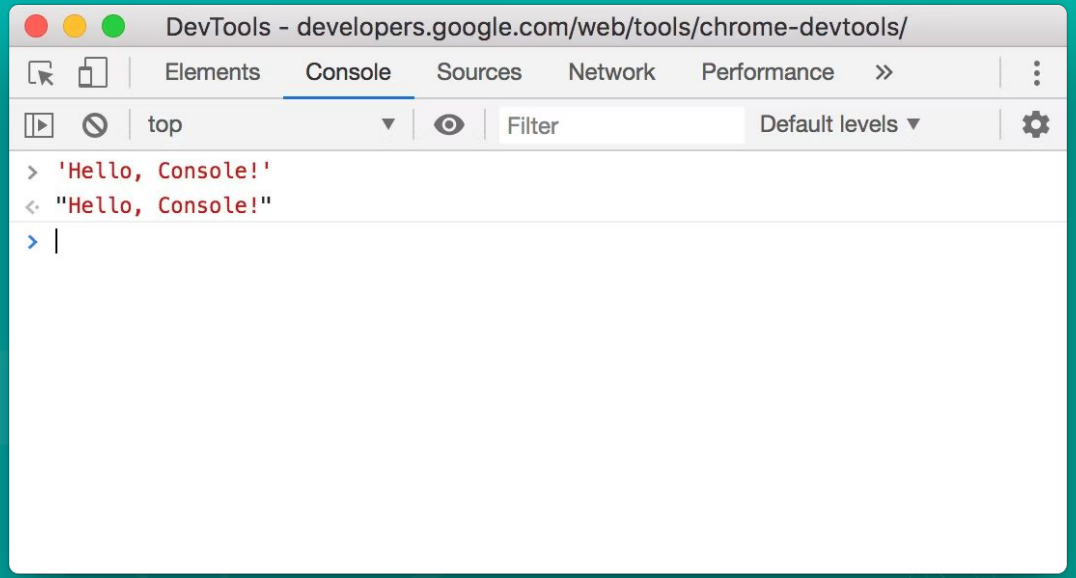
Running JavaScript In The Browser

- ➔ Browser Console
- ➔ Internal
- ➔ External

Github Repository

Variables, Data Types, Operators And More...

Using The Console



Comments & Shortcuts

Ctrl + /
Shift + Alt + A

Values and Variables

- ➔ What are Values?
- ➔ What are Variables?
  - ◆ naming variables rules
    - letters
    - numbers
    - $
    - _
    - should be descriptive
    - about reserved names : new, function, class, name

Case Styles

- ➔ camelCase
- ➔ kebab-case
- ➔ snake_case
- ➔ PascalCase

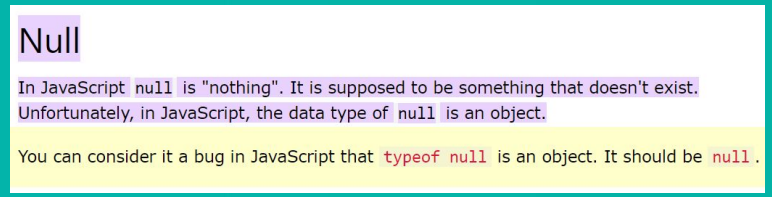
let, const, var

- ➔ which one should i use? by default const
- ➔ declaring variables without let, const, var
- ➔ let is block scope
- ➔ var is function scope

Primitive Data Types

- The "typeof" Operator
- String `let firstName = 'Jonas';`
- Number (decimal or integers) `let age = 23;`
- Boolean `let fullAge = true;`
- Null
- Undefined `let children;`
- Symbol (ES2015) `const sym = Symbol("foo");`
- BigInt (ES2020)

```
const alsoHuge = BigInt(9007199254740991);
const previouslyMaxSafeInteger = 9007199254740991n;
```



Reference Data Type

```
//object
const person1 = {
  firstName: "alireza",
  lastName: "mohammadi",
};

//array
const numbers = [2, 4, 6, 8];

//function
const sayHello = function () {
  console.log("hello from websila academy!");
};
```

Statically vs Dynamically Typed

☝ **JavaScript has dynamic typing:** We do ***not*** have to manually define the data type of the value stored in a variable. Instead, data types are determined **automatically**.

➔ Value has type, NOT Variable!

Basic Operators

- ➔ Arithmetic Operators
  - ◆ +, -, /, *, **, %
- ➔ Assignment Operators
  - ◆ +=
  - ◆ -=
  - ◆ ++
  - ◆ - -
  - ◆ *=
  - ◆ /=
- ➔ Operator Precedence (...)

Alert and Propmt

Strings And Template Literals

- ➔ Using sum operator on strings (“alireza” + “mohammadi”)
- ➔ “hello my name is ” + firstname + “ ” + lastname
- ➔ ` hello my name is ${firstname} ${lastname}`

Type Conversion

- Number()
  - `The Unary Operator (+)`
- String()
- Boolean()
  - just for converting 0 and 1 (explain about truthy and falsy values later on Control Flow Section)

Type Coercion

- 10 + "10"
- 10 + "alireza"
- 10 - "10"
- 10 - "alireza"
- 10 * "alireza"
- 10 / "alireza"
- true + 2
- false + 2

NaN, Infinity

- ➔ NaN : 5 - "alireza"
- ➔ Infinity : 5 / 0
- ➔ typeof NaN is Number
- ➔ typeof Infinity is Number

Working with Control Structures (Control Flow)

Comparison Operators

- ➔ <
- ➔ >
- ➔ <=
- ➔ >=
- ➔ Equality (==)
- ➔ Inequality (!=)
- ➔ Strict equality (===)
- ➔ == vs ===
- ➔ Strict inequality (!==)
- ➔ != vs !==

Operator	Definition	Example	Result
<	Less than	10 < 5	False
>	Greater than	10 > 5	True
<=	Less than or equal to	10 <= 10	True
>=	Greater than or equal to	10 >= 5	True
==	Equal to	10 == '10'	True
!=	Not equal to	10 != 5	True
===	Equal to (including type)	10 === '10'	False
!==	Not equal to (including type)	10 !== '10'	True

Taking Decisions-if,else Statements

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Taking Decisions: if,else Statements

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Taking Decisions: if,else Statements

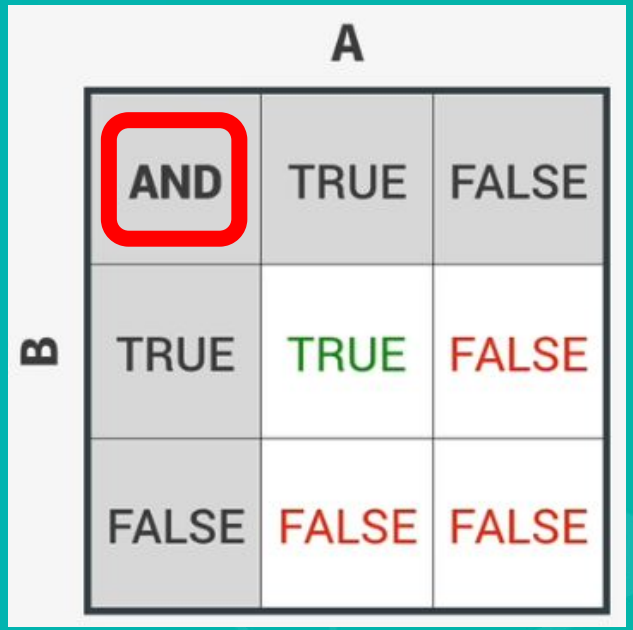
```
1  if (true) {
2    const foo = "foo";
3    console.log(foo); // "foo"
4  }
5
6  console.log(foo); // Uncaught ReferenceError: foo is not defined
```

Truthy and Falsy Values

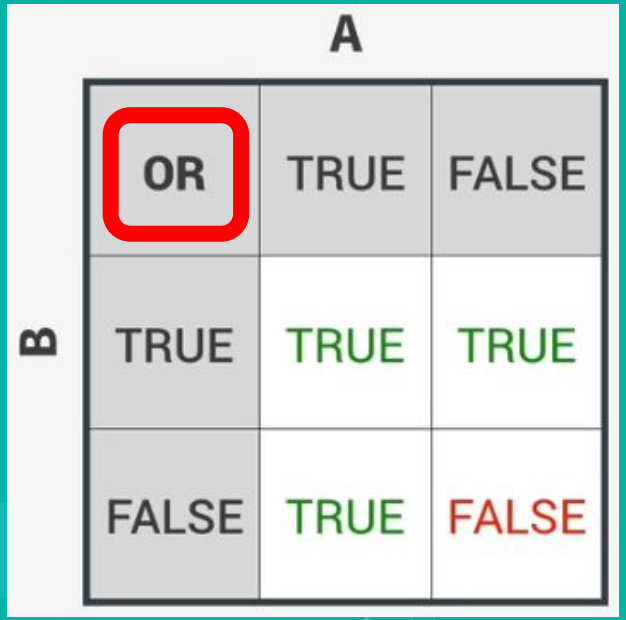
In JavaScript all values are truthy except these 5 :

- ➔ 0
- ➔ ""
- ➔ undefined
- ➔ null
- ➔ NaN

Boolean Logic



B \ A	AND	TRUE	FALSE
TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE



B \ A	OR	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE

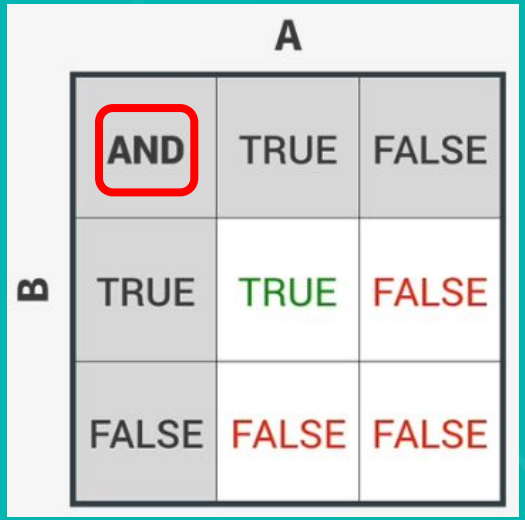
AND

I can leave the house if :

A -> the door is unlocked

B -> my shoes are on

AND



B \ A	AND	TRUE	FALSE
TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE

OR

I will buy the car if :

A -> the car's colour is blue

B -> the car's colour is red

OR

B \ A	OR	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE

Logical Operators

- &&
- ||
- !

&&

```
1  const doorIsUnlocked = true;
2  const shoesOn = true;
3  if (doorIsUnlocked && shoesOn) {
4    console.log("I can leave the house");
5  }
```

||

```
1  const carColor = "blue";
2  if (carColor === "red" || carColor === "blue") {
3    console.log("I will buy the car");
4  }
```

!

```
1  const doorIsLocked = false;
2  const shoesOn = true;
3  if (!doorIslocked && shoesOn) {
4    console.log("I can leave the house");
5  }
6
```

#The Conditional (Ternary) Operator

```
1  const score = 80;
2  let result;
3  if (score > 70) {
4    result = "passed😍";
5  } else {
6    result = "failed😳";
7  }
8
```

```
1  const score = 80;
2  let result = score > 70 ? "passed😍" : "failed😳";
3  console.log(result);
```

#The switch Statement

```
const today = "shanbe";
if (today === "shanbe") {
  console.log("فوتبال");
} else if (today === "2shanbe") {
  console.log("پیانو");
} else if (today === "4shanbe") {
  console.log("برنامه‌نویسی");
} else {
  console.log("بیکار!");
}
```

```
const today = "shanbe";
switch (today) {
  case "shanbe":
    console.log("فوتبال");
    break;
  case "2shanbe":
    console.log("پیانو");
    break;
  case "4shanbe":
    console.log("برنامه‌نویسی");
    break;
  default:
    console.log("بیکار!");
}
```

#The "for" Loop

```
for (let i = 0; i < 9; i++) {
  console.log(i);
}
// Output:
// 0
// 1
// 2
// 3
// 4
// 5
// 6
// 7
// 8
```

"while" Loops

```
let i = 1;
while (i < 9) {
  console.log(i);
  i++;
}
// Output:
// 1
// 2
// 3
// 4
// 5
// 6
// 7
// 8
```

"do-while" Loops

```
let i = 1;
do {
  console.log(i);
  i++;
} while (i < 9);
// Output:
// 1
// 2
// 3
// 4
// 5
// 6
// 7
// 8
```

```
let i = 10;
do {
  console.log(i);
  i++;
} while (i < 9);
// Output:
// 10
```

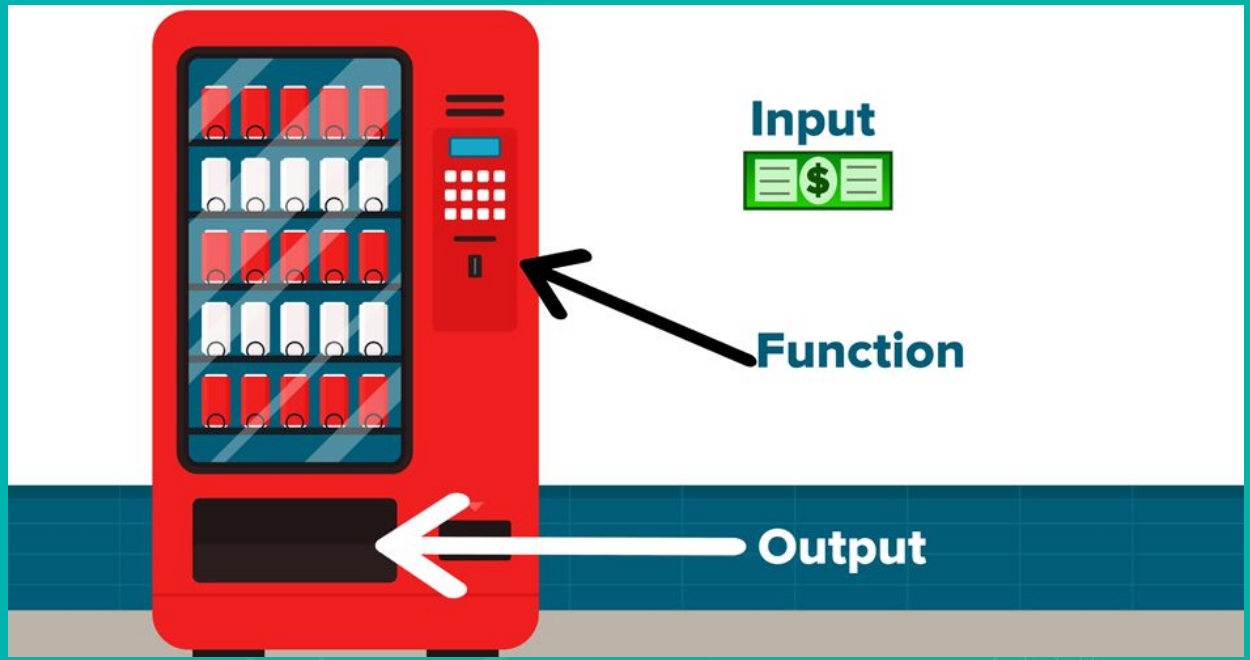
#Breaking and Continuing in Loops

```
for (let i = 0; i < 10; i++) {
  if (i === 3) {
    break;
  }
  console.log(i);
}

// Output:
// 0
// 1
// 2
```

```
for (let i = 0; i < 5; i++) {
  if (i === 3) {
    continue;
  }
  console.log(i);
}

// Output:
// 0
// 1
// 2
// 4
// 5
```

Functions (Basics)

#Introducing Functions



Introducing Functions

#Classic Functions

```
1  function sayHello() {
2    console.log("Hello!");
3  }
4  sayHello(); //execute
```

Function Parameters

```
1  function sayHello(firstName) {
2    console.log(`Hello! My Name is ${firstName}`);
3  }
4  sayHello("Alireza");
5
6  // Output:
7  // Hello! My Name is Alireza
```

Function Parameters

```
function add(a, b) {
  console.log(a + b);
}
add(3, 5); // Output: 8
```

#Returning Values

```
function sayHello() {
  return "Hello!";
}
const result = sayHello();
console.log(result);

// Output:
// Hello!
```

Returning Values

```
function sayHello(firstName) {
  const output = `Hello! My Name is ${firstName}`;
  return output;
}
const result = sayHello("Alireza");
console.log(result);

// Output:
// Hello! My Name is Alireza
```

Returning Values

```
function add(a, b) {
  return a + b;
}
const answer = add(3, 5);
console.log(answer);

// Output: 8
```

#Function Declarations vs Expressions

```
const sayHello = function () {
  return "Hello!";
};
const result = sayHello();
console.log(result);

// Output:
// Hello!
```

```
const add = function (a, b) {
  return a + b;
};
const answer = add(3, 5);
console.log(answer);

// Output: 8
```

Arrays (Basics)

#Introduction to Arrays

```
1  //first method
2  const friends = ["Mostafa", "Hesam", "Arman"];
3
4  //second method
5  const numbers = new Array(2, 4, 6, 8);
```

Get Values from Array

```
1  const friends = ["Mostafa", "Hesam", "Arman"];
2  console.log(friends[0]); //Output: Mostafa
3  console.log(friends[2]); //Output: Arman
```

Mutate Array

```
1  const friends = ["Mostafa", "Hesam", "Arman"];
2  friends[1] = "Ali";
3
4  console.log(friends);
5  //Output:
6  //["Mostafa", "Ali", "Arman"];
```

Some Other Examples

```
const firstName = "Mostafa";
const myArray = [23, "Alireza", firstName, 1402 - 1376];
console.log(myArray[1]); //Output: "Alireza"
console.log(myArray[2]); //Output: "Mostafa"
console.log(myArray[3]); //Output: 26
```

```
const nestedArray = [
  ["salam", "khubi?"],
  [100, 200],
];
console.log(nestedArray[0][0]); //Output: "salam"
console.log(nestedArray[0][1]); //Output: "salam"
console.log(nestedArray[1][0]); //Output: 100
console.log(nestedArray[1][1]); //Output: 200
```

#Add Elements to Array

```
const friends = ["Mostafa", "Hesam"];
console.log(friends);
//Output: ['Mostafa', 'Hesam']
friends.push("Arman");
console.log(friends);
//Output: ['Mostafa', 'Hesam', 'Arman']
```

push

```
const friends = ["Mostafa", "Hesam"];
console.log(friends);
//Output: ['Mostafa', 'Hesam']
friends.unshift("Arman");
console.log(friends);
//Output: ['Mostafa', 'Hesam', 'Arman']
```

unshift

Remove Elements from Array

```
const friends = ["Mostafa", "Hesam"];
console.log(friends);
//Output: ['Mostafa', 'Hesam']
friends.pop();
console.log(friends);
//Output: ['Mostafa']
```

pop

```
const friends = ["Mostafa", "Hesam"];
console.log(friends);
//Output: ['Mostafa', 'Hesam']
friends.shift();
console.log(friends);
//Output: ['Hesam']

```

shift

#Basic Array Methods

```
1  //length
2  const friends = ["Mostafa", "Hesam", "Arman"];
3  console.log(friends.length); //Output: 3
```

```
1  //at
2  const friends = ["Mostafa", "Hesam", "Arman"];
3  console.log(friends.at(1)); // "Hesam"
4  console.log(friends.at(-1)); // "Arman"
```

Basic Array Methods

```
1  //includes
2  const friends = ["Mostafa", "Hesam", "Arman"];
3  console.log(friends.includes("Hesam")); //true
4  console.log(friends.includes("Ahmad")); //false
```

```
1  //indexOf
2  const friends = ["Mostafa", "Hesam", "Arman"];
3  console.log(friends.indexOf("Hesam")); // 1
4  console.log(friends.indexOf("Arman")); // 2
```

Basic Array Methods

```
//concat
const friends = ["Mostafa", "Arman"];
const firends2 = ["Ahmad", "Hamid"];
console.log(friends.concat(firends2));
//Output: ['Mostafa', 'Arman', 'Ahmad', 'Hamid']
```

```
//join
const friends = ["Mostafa", "Arman", "Hesam"];
console.log(friends.join("-"));
//Output: "Mostafa-Arman-Hesam"
```

```
//reverse
const friends = ["Mostafa", "Arman", "Hesam"];
console.log(friends.reverse());
//Output: ['Hesam', 'Arman', 'Mostafa']
```

#Slice and Splice

```
1  // slice(start)
2  // slice(start, end)
3  const friends = ["Mostafa", "Arman", "Hesam", "Ahmad", "Hamid"];
4  console.log(friends.slice(2));
5  //Output: ['Hesam', 'Ahmad', 'Hamid']
6
7  console.log(friends.slice(2, 4));
8  //Output: ['Hesam', 'Ahmad']
```

Slice and Splice

```
// splice(start)
const friends = ["Mostafa", "Arman", "Hesam", "Ahmad", "Hamid"];
friends.splice(3);
console.log(friends);
//Output: ['Mostafa', 'Arman', 'Hesam']
```

```
// splice(start, deleteCount)
const friends = ["Mostafa", "Arman", "Hesam", "Ahmad", "Hamid"];
friends.splice(2, 2);
console.log(friends);
//Output: ['Mostafa', 'Arman', 'Hamid']
```

```
// splice(start, deleteCount, item1, item2, /* …, */ itemN)
const friends = ["Mostafa", "Arman", "Hesam", "Ahmad", "Hamid"];
friends.splice(1, 0, "Jamal", "Kamal");
console.log(friends);
//Output: ['Mostafa', 'Jamal', 'Kamal', 'Arman', 'Hesam', 'Ahmad', 'Hamid']
```

Objects (Basics)

#Introduction to Objects

```
1  //Array
2  const alirezaArray = [
3    "Alireza",
4    "Mohammadi",
5    "Full-Stack Developer",
6    1998,
7    ["Mostafa", "Hesam", "Arman"],
8  ];
9
10 //Object
11 const alirezaObject = {
12   firstName: "Alireza",
13   lastName: "Mohammadi",
14   job: "Full-Stack Developer",
15   birthYear: 1998,
16   Friends: ["Mostafa", "Hesam", "Arman"],
17 };
```

Some Tips :

- Naming rules for properties
- properties order does not matter

Get Value from Object (Dot Notation)

```
const alirezaObject = {
  firstName: "Alireza",
  lastName: "Mohammadi",
  birthYear: 1998,
};

// Dot Notation
console.log(alirezaObject.firstName); // "Alireza"
console.log(alirezaObject.birthYear); // 1998
```

Get Value from Object (Bracket Notation)

```
1  const alirezaObject = {
2    firstName: "Alireza",
3    lastName: "Mohammadi",
4    birthYear: 1998,
5  };
6  const a = "Year";
7
8  //Bracket Notation
9  console.log(alirezaObject["firstName"]); // "Alireza"
10 console.log(alirezaObject["birth" + a]); // 1998
```

Add Elements to Object

```
const alirezaObject = {
  firstName: "Alireza",
  lastName: "Mohammadi",
};
alirezaObject.job = "Full-Stack Developer";
alirezaObject.Friends = ["Mostafa", "Hesam", "Arman"];
alirezaObject["birthYear"] = 1998;
```

Remove Elements from Object

```
1  const alirezaObject = {
2    firstName: "Alireza",
3    lastName: "Mohammadi",
4  };
5  delete alirezaObject.firstName; // or delete alirezaObject["firstName"];
```

Mutate Object

```
1  const alirezaObject = {
2    firstName: "Alireza",
3    lastName: "Mohammadi",
4    job: "Teacher",
5    birthYear: 1994,
6  };
7  alirezaObject.job = "Full-Stack Developer";
8  alirezaObject["birthYear"] = 1994;
```

Challenge

choose custom property with prompt and Get custom data from alireza object by user

#Object Methods

```
const alirezaObject = {
  firstName: "Alireza",
  lastName: "Mohammadi",
  job: "Full-Stack Developer",
  birthYear: 1998,
  Friends: ["Mostafa", "Hesam", "Arman"],
  getAge: function () {
    return 2028 - this.birthYear;
  },
};
console.log(alirezaObject.getAge());
//Output: 30
```

#Combination of Arrays and Objects

```
const users = [
  {
    firstName: "Alireza",
    lastName: "Mohammadi",
    job: "Full-Stack Developer",
  },
  {
    firstName: "Arman",
    lastName: "Shabani",
    job: "Video Editor",
  },
];
```

More on Strings

#Working With Strings - Part 1

```
//! new String()
const wlc = new String("Welcome to Websila!");
console.log(wlc, typeof wlc);
//output: String {'Welcome to Websila!'} 'object'
```

```
//! Strings are also arrays!
const wlc = "Welcome to Websila Academy!";
console.log(wlc[0]); // output: "W"
console.log(wlc[1]); // output: "e"
console.log(wlc.at(-1)); // output: "!"
console.log(wlc.at(-2)); // output: "y"
```

```
//! length
const wlc = "Welcome to Websila Academy!";
console.log(wlc.length); // output: 27
```

```
//! indexOf()
const wlc = "Welcome to Websila Academy!";
console.log(wlc.indexOf("e")); // output: 1
console.log(wlc.indexOf("x")); // output: -1
console.log(wlc.indexOf("m")); // output: 5
```

```
//! lastIndexOf()
const wlc = "Welcome to Websila Academy!";
console.log(wlc.lastIndexOf("!")); // output: 26
console.log(wlc.lastIndexOf("e")); // output: 23
console.log(wlc.lastIndexOf("x")); // output: -1
```

```
//! includes()
const wlc = "Welcome#to#Websila";
console.log(wlc.includes("welcome")); // output: false
console.log(wlc.includes("Websila")); // output: true
console.log(wlc.includes("hello")); // output: false
```

#Working With Strings - Part 2

```
//! toUpperCase()
const wlc = "Welcome to Websila!";
console.log(wlc.toUpperCase());
// output: WELCOME TO WEBSILA!
```

```
//! toLowerCase()
const wlc = "Welcome to Websila!";
console.log(wlc.toLowerCase());
// output: welcome to websila!
```

```
//! trim()
const txt = "    Hello world!  ";
console.log(txt.trim());
// output: "Hello world!"
```

```
//! replace()
const txt = "Hello, My name is Alireza!";
console.log(txt.replace("Hello", "Hi"));
// output: "Hi, My name is Alireza!"
```

```
//! replaceAll()
const wlc = "Welcome#to#Websila";
console.log(wlc.replaceAll("#", "-"));
// output: "Welcome-to-Websila"
```

```
//! slice()
const wlc = "Welcome to Websila Academy!";
console.log(wlc.slice(14)); // output: "sila Academy!"
console.log(wlc.slice(-5)); // output: "demy!"
console.log(wlc.slice(1, 6)); // output: "elcom"
```

#Working With Strings - Part 3

```
//! split()
const wlc = "Welcome#to#Websila";
console.log(wlc.split("#"));
// output: ['Welcome', 'to', 'Websila']
```

```
//! padStart()
const number = "100";
console.log(number.padStart(3, "*")); // output: "100"
console.log(number.padStart(4, "*")); // output: "*100"
console.log(number.padStart(5, "*")); // output: "**100"

const cardNumber = "6037997287544703";
const last4digits = cardNumber.slice(-4);
console.log(last4digits.padStart(16, "*"));
// output: "************4703"
```

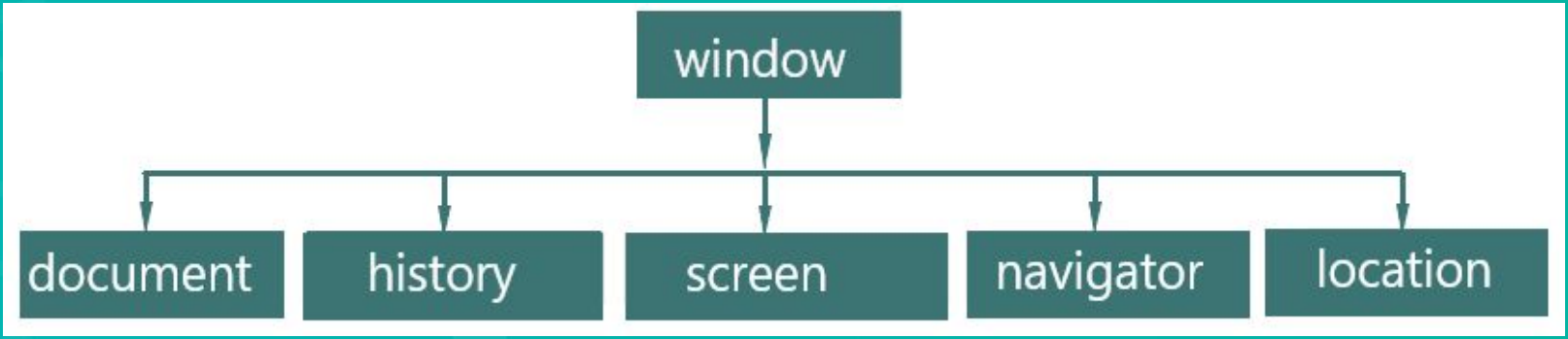
Working With Strings - Part 3

```
//! padEnd()
const number = "100";
console.log(number.padEnd(3, "*")); // output: "100"
console.log(number.padEnd(4, "*")); // output: "100*"
console.log(number.padEnd(5, "*")); // output: "100**"

const mobileNumber = "09121234567";
const first6digits = mobileNumber.slice(0, 6);
console.log(first6digits.padEnd(11, "*"));
// output: "091212*****"
```

```
//! repeat()
const mood = "Happy!";
console.log(mood.repeat(3));
// output: Happy!Happy!Happy!
```

Working with DOM

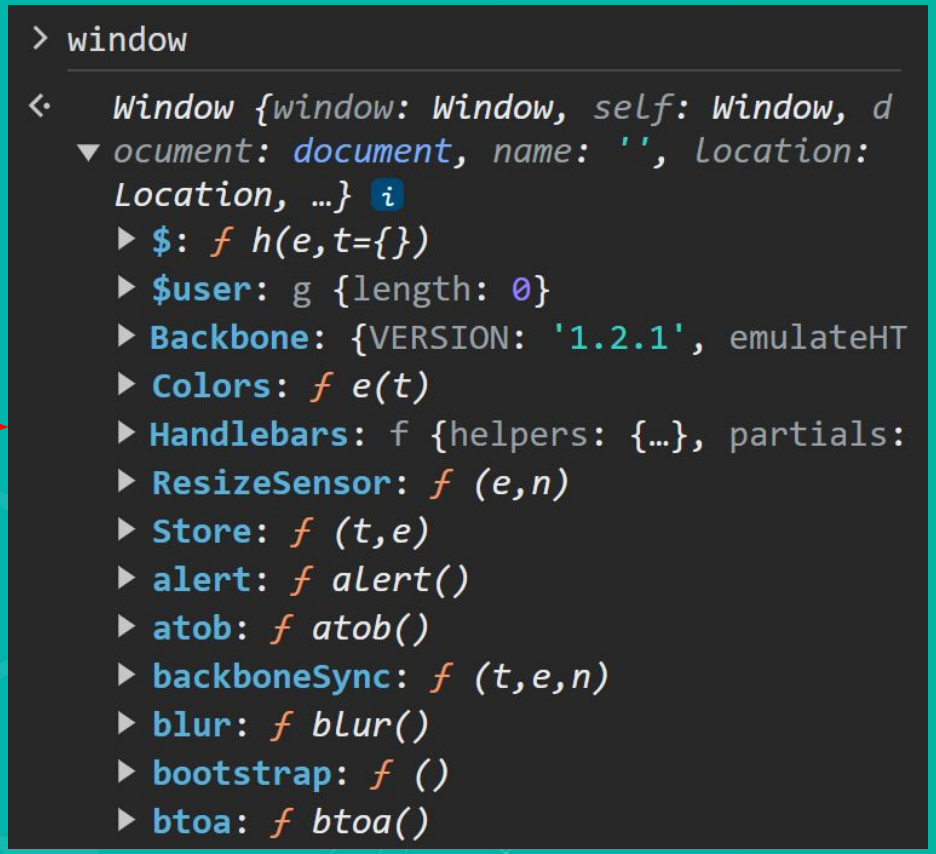
#Window Object in JavaScript



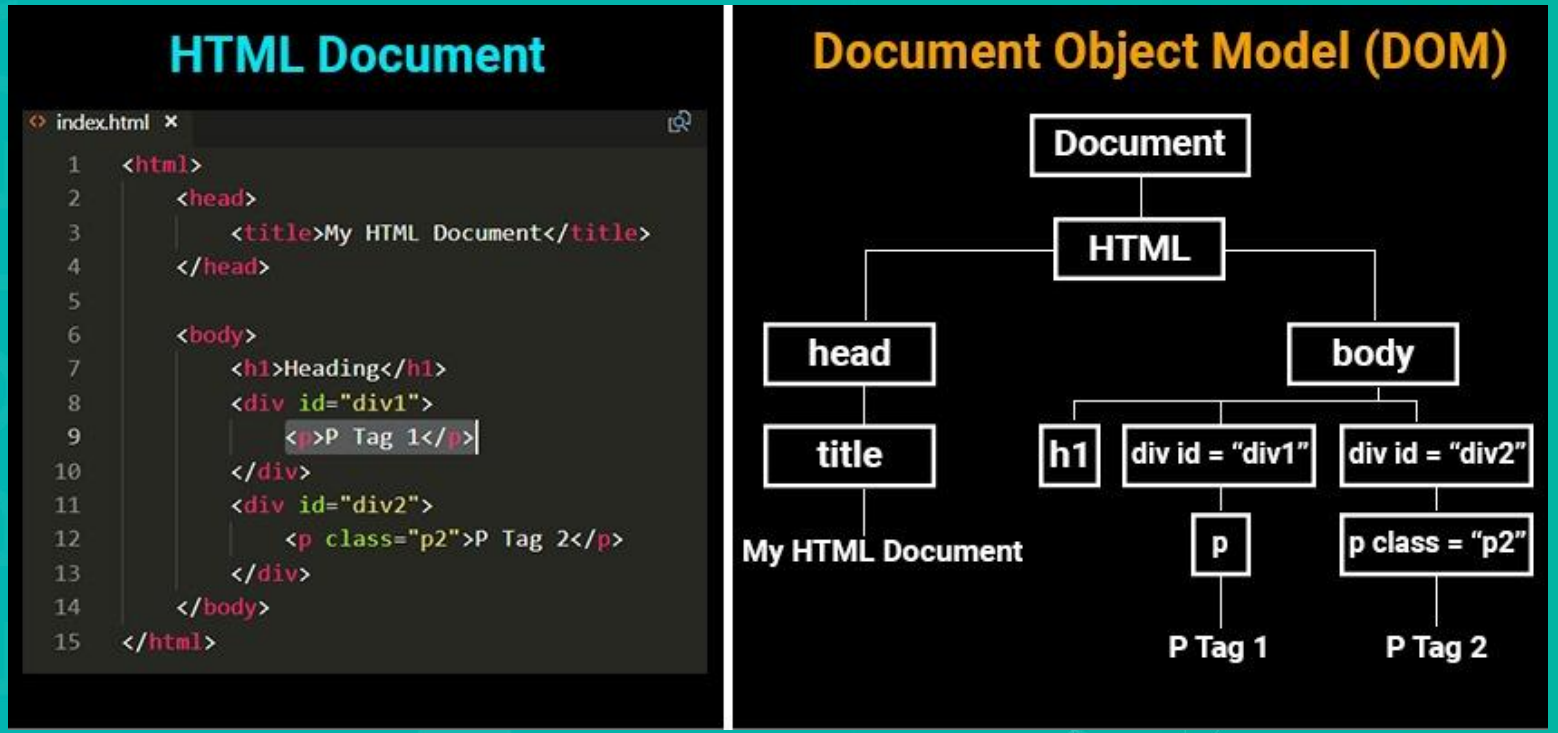

```
1  window.alert("Welcome to Websila JS Course");
2  window.console.log("Welcome to Websila JS Course");
```

Window Object in JavaScript

```
1  console.log(window);
```

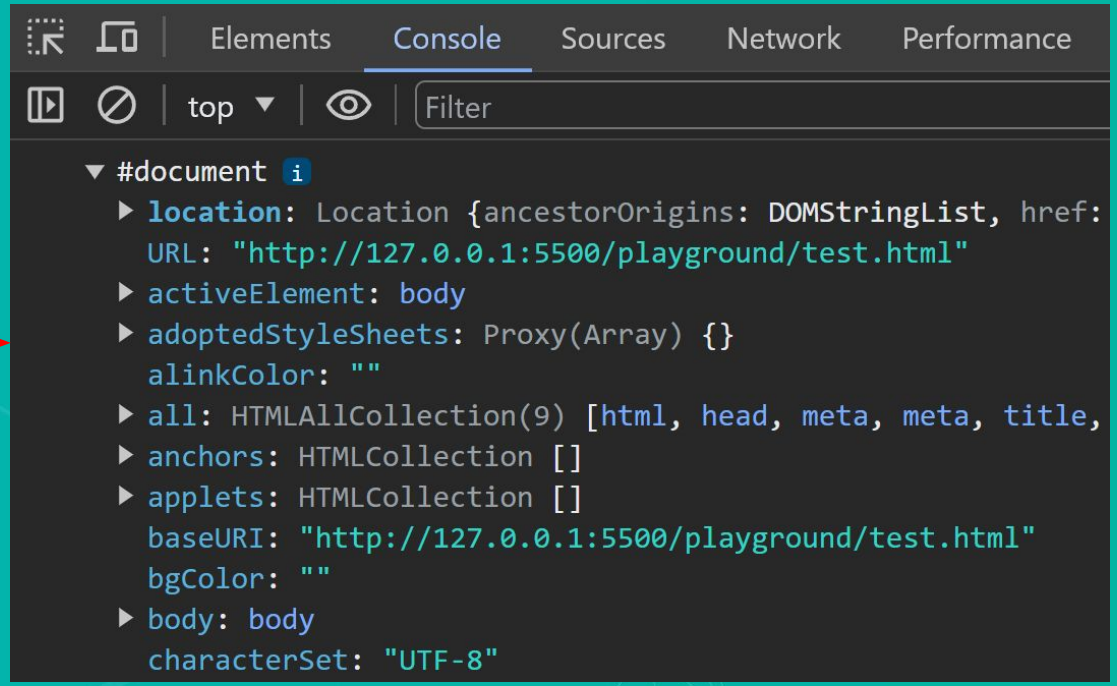


#Document Object Model (DOM)



Document Object Model (DOM)

```
1  console.log(document);
2  console.log(document.head);
3  console.log(document.body);
```



#DOM Selectors - Single Elements

```
1  window.document.getElementById("wrapper");
2  window.document.querySelector(".btn");
```

```
1  document.getElementById("wrapper");
2  document.querySelector(".btn");
```

#DOM Selectors - Multiple Elements

```
1  window.document.querySelectorAll(".btn");
2  window.document.getElementsByClassName("btn");
3  window.document.getElementsByTagName("div");
4  window.document.getElementsByName("email");
```

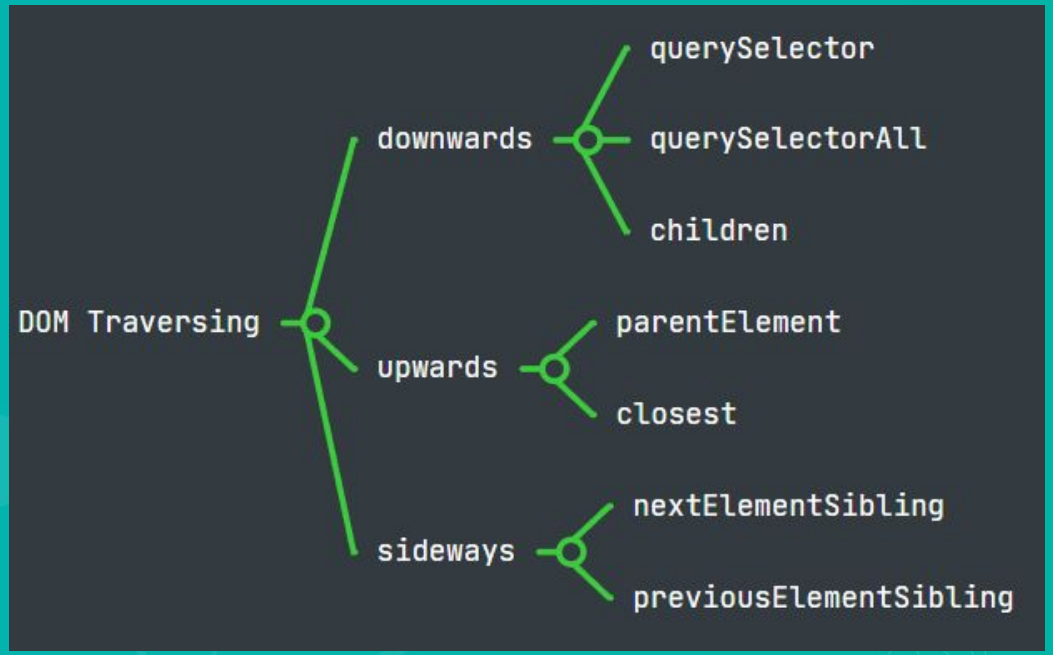
```
1  document.querySelectorAll(".btn");
2  document.getElementsByClassName("btn");
3  document.getElementsByTagName("div");
4  document.getElementsByName("email");
```

#Manipulating Element Text and HTML

```
document.querySelector(".btn").innerText = "ثبت نام";
document.getElementById("wrapper").innerHTML = `<h1>خوش آمدید!</h1>`;
```

```
document.getElementById("wrapper").innerHTML = `
    <div>
        <img class="logo" src="images/logo.png">
    </div>
    <div class="my-4">
        <h1>Welcome to Websila Academy!</h1>
    </div>
    <div style="background-color:#CCC">
        <span>some other stuff here...</span>
    </div>
`;
```

#Traversing the DOM - Overview



#DOM Traversing Downwards

```
1  const tiles_section = document.querySelector(".tiles");
2  const articles = tiles_section.querySelectorAll("article");
3  articles[0].querySelector("h2").innerText = "یک عنوان جدید برای اولین مقاله";
```

```
1  const wrapper = document.getElementById("wrapper");
2  console.log(wrapper.children);
```

```
                                                    script.js:6
HTMLCollection(3) [header#header, div#main,
▼ footer#footer, header: header#header, main: div#main,
  footer: footer#footer] i
  ▶ 0: header#header
  ▶ 1: div#main
  ▶ 2: footer#footer
  ▶ footer: footer#footer
  ▶ header: header#header
  ▶ main: div#main
    length: 3
  ▶ [[Prototype]]: HTMLCollection
```

#DOM Traversing Upwards

```
1  const parentEl = document.querySelector("#wrapper").parentElement;
2  console.log(parentEl);
```

```
▶ <body class="is-preload">…</body>                    script.js:8
```

```
1  const iconEl = document.querySelector(".icon.fa-github");
2  console.log(iconEl.closest("div"));
```

```
▶ div.inner                                            script.js:12
```

#DOM Traversing Sideways

```
1  const mainEl = document.querySelector("#main");
2  console.log(mainEl.nextElementSibling);
3  console.log(mainEl.previousElementSibling);
```

#Adding Elements via HTML in Code

Solution 1:

HTML:

```html
<ul class="list">
  <li class="list-item">Item1</li>
  <li class="list-item">Item2</li>
</ul>
```

JavaScript:

```javascript
const list = document.querySelector(".list");
const newItem = `
  <li class="list-item">
    Item3
  </li>`;
list.innerHTML = list.innerHTML + newItem;
```

Output:

- Item1
- Item2
- Item3

Solution 2:

HTML:

```html
<ul class="list">
  <li class="list-item">Item1</li>
  <li class="list-item">Item2</li>
</ul>
```

JavaScript:

```javascript
const list = document.querySelector(".list");
const newItem = `
  <li class="list-item">
    Item3
  </li>`;
list.insertAdjacentHTML("beforeend", newItem);
```

Output:

- Item1
- Item2
- Item3

insertAdjacentHTML()

```
insertAdjacentHTML(position, text);
```

Positions:

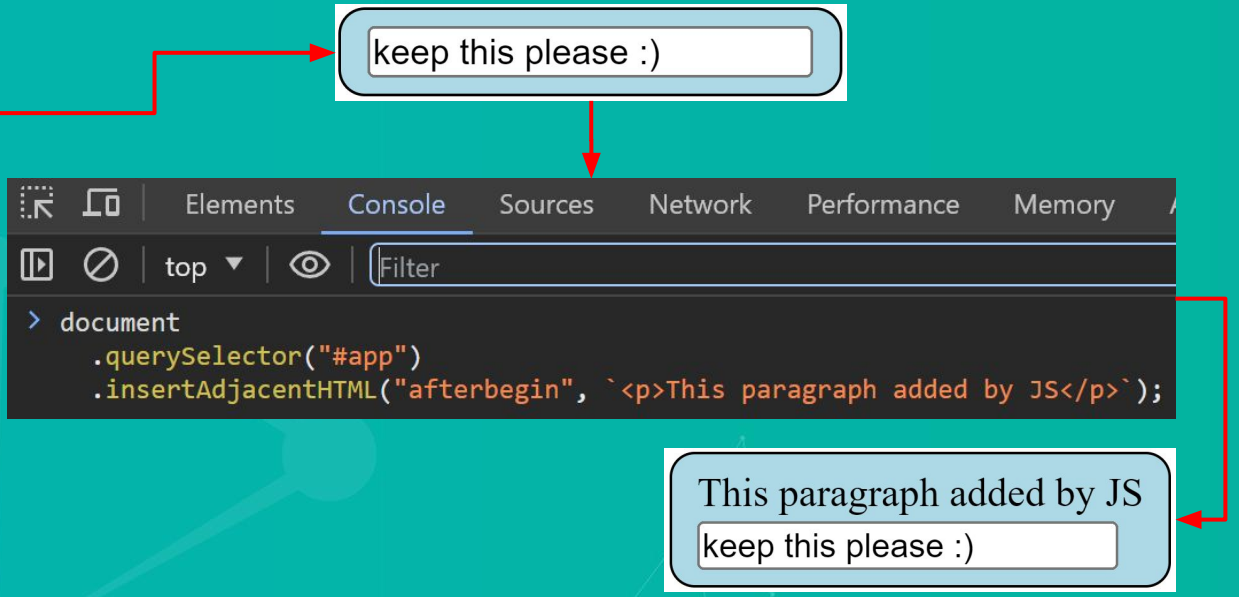
- "beforebegin"
- "afterbegin"
- "beforeend"
- "afterend"

```
<!-- beforebegin -->
<element>
  <!-- afterbegin -->
  text
  <!-- beforeend -->
</element>
<!-- afterend -->
```

Difference between two solutions

With insertAdjacentHTML the entire element will not re-rendered.

```
1  <div id="app">
2    <input type="text" />
3  </div>
```

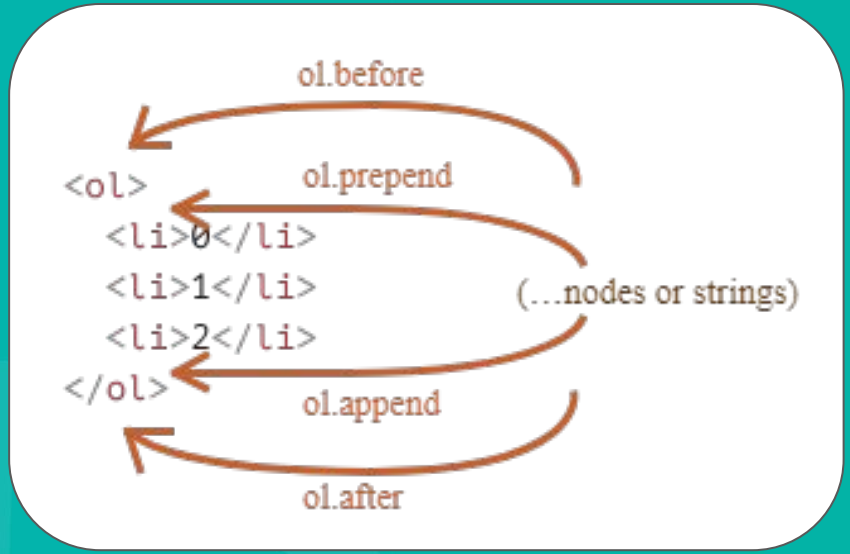


#Create and Remove Elements

```
1  //! create ul
2  const newList = document.createElement("ul");
3
4  //! create li (Item1)
5  const newItem1 = document.createElement("li");
6  newItem1.innerText = "Item1";
7
8  //! create li (Item2)
9  const newItem2 = document.createElement("li");
10 newItem2.innerText = "Item2";
11
12 //! append li to ul
13 newList.append(newItem1, newItem2);
14
15 //! append ul to body
16 document.body.append(newList);
```

Create and Remove Elements

- append()
- prepend()
- before()
- after()



insertAdjacentElement

```
insertAdjacentElement(position, element);
```

Positions:

- "beforebegin"
- "afterbegin"
- "beforeend"
- "afterend"

```
<!-- beforebegin -->
<element>
  <!-- afterbegin -->
  text
  <!-- beforeend -->
</element>
<!-- afterend -->
```

remove()

HTML:

```html
<div id="div-01">Here is div-01</div>
<div id="div-02">Here is div-02</div>
<div id="div-03">Here is div-03</div>
```

JavaScript:

```javascript
const element = document.getElementById("div-02");
element.remove(); // Removes the div with the 'div-02' id
```

Output:

Here is div-01
Here is div-03

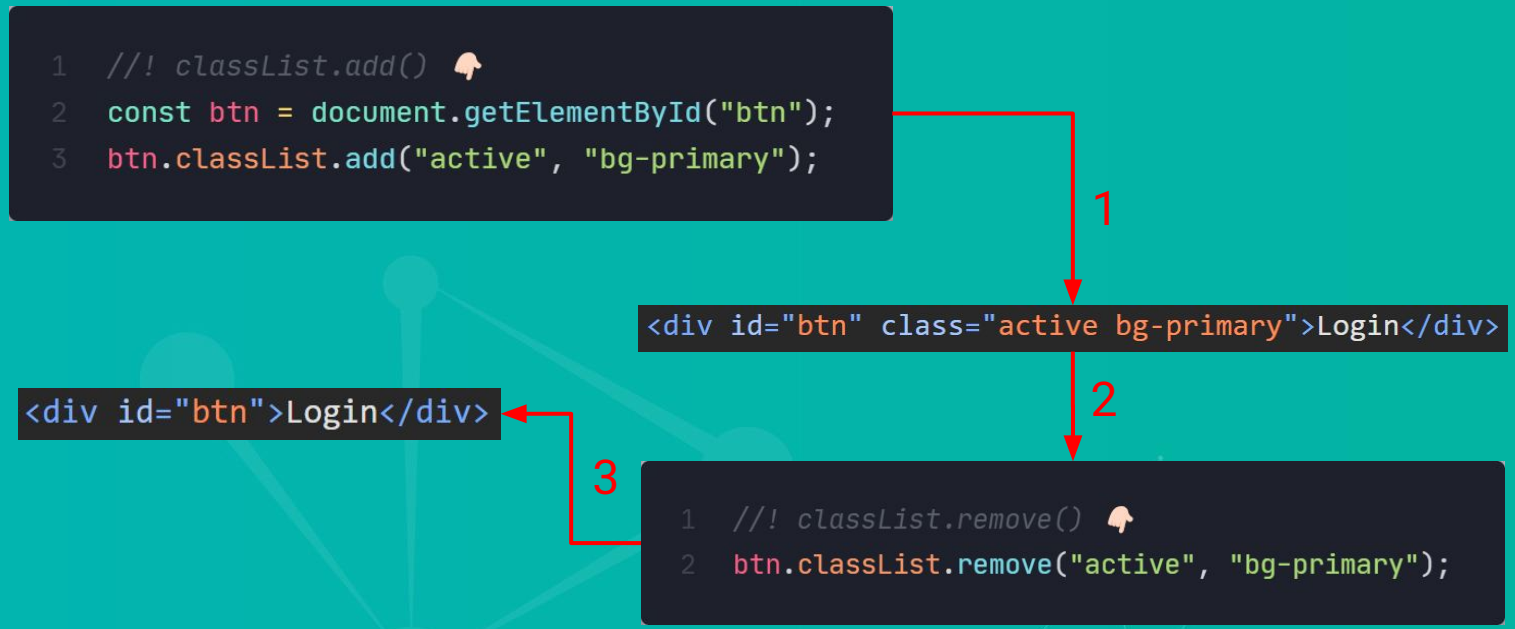
#Manipulating Styles & Classes

style property :

```
1  //! style property 👇
2  const btn = document.getElementById("btn");
3  btn.style.color = "#fff";
4  btn.style.backgroundColor = "#000";
5  btn.style.width = "max-content";
6  btn.style.padding = "5px 15px";
7  btn.style.borderRadius = "10px";
8  btn.style.fontFamily = "Tahoma";
9
```

Manipulating Styles & Classes

classList property (add/remove methods) :



Manipulating Styles & Classes

classList property (contains method) :

```
1  //! classList.contains() 👇
2  const btn = document.getElementById("btn");
3  const result = btn.classList.contains("active");
4  console.log(result); // true or false
```

#Manipulating id, value and other attributes

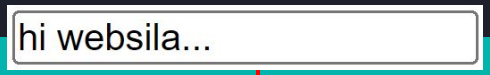
id property :

```
<article id="benz-c200">…</article>
```

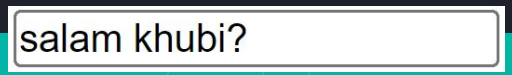
```
const article = document.getElementById("benz-c200");

//! get id
console.log(article.id); //benz-c200

//! set id
article.id = "benz-c300";
console.log(article.id); //benz-c300
```

Manipulating id, value and other attributes ***value property:***

```html
<input id="username" type="text" value="hi websila..." />
```




```js
const input = document.querySelector("#username");

//! get value
console.log(input.value); // "hi websila..."

//! set value
input.value = "salam khubi?";
```



Manipulating id, value and other attributes ***setAttribute method :***

```
1  <button>Hi Websila!</button>
```

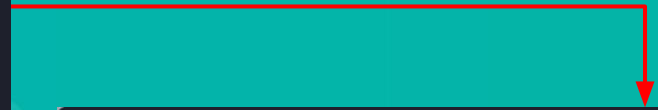

```
1  const button = document.querySelector("button");
2  button.setAttribute("name", "hiButton");
3  button.setAttribute("disabled", "");
```

```
<button name="hiButton" disabled>Hi Websila!</button>
```

Manipulating Other Attributes

getAttribute method :

```html
<div
  id="logout-btn"
  class="btn btn-danger"
  data-btn-id="351"
>
  Logout
</div>
```



```js
//! getAttribute()
const btn = document.querySelector("#logout-btn");
console.log(btn.getAttribute("id")); // "logout-btn"
console.log(btn.getAttribute("class")); // "btn btn-danger"
console.log(btn.getAttribute("data-btn-id")); // "351"
```

Manipulating Other Attributes

removeAttribute method :

```html
<div
  id="logout-btn"
  class="btn btn-danger"
  data-btn-id="351"
>
  Logout
</div>
```

```js
//! removeAttribute()
const btn = document.querySelector("#logout-btn");
btn.removeAttribute("class");
```

```html
<div id="logout-btn" data-btn-id="351">Logout</div>
```

#Using dataset (data-* Attributes)

```html
<article
  id="benz-c200"
  data-columns="2"
  data-index-number="16543"
  data-parent-category="cars"
>
  ...
</article>
```

```js
const article = document.getElementById("benz-c200");

//! get data
console.log(article.dataset.columns); // "2"
console.log(article.dataset.indexNumber); // "16543"
console.log(article.dataset.parentCategory); // "cars"

//! set data
article.dataset.columns = "3";
console.log(article.dataset.columns); // "3"
```

Events (Basics)

#Event Listener

#The Event Object

#Mouse Events

Event	Description
click	Click left mouse button
dblclick	Double click left mouse button
contextmenu	Click right mouse button
mousedown	Click down left mouse button
mouseup	Release left mouse button
wheel	Scroll Wheel button
mouseenter	Hover over element
mousemove	Moving over element
mouseleave	Leave hovering element
dragstart	Click and start to drag element
drag	Drag element around screen
dragend	Stop dragging

#Keyboard Events

- keydown
- keypress
- keyup

#Input Events

- input
- change
- focus
- blur

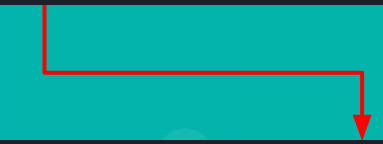
#This Keyword in Events

```
1  document.body.addEventListener("click", function (e) {
2    console.log(e.target);
3    console.log(this);
4  });
```

Data Structures and Modern Operators (ES6+)

#Destructuring Arrays

```
const users = ["Alireza", "Mostafa", "Mohammad"];
const a = users[0];
const b = users[1];
const c = users[2];
console.log(a, b, c); // Output: Alireza Mostafa Mohammad
```



```
const users = ["Alireza", "Mostafa", "Mohammad"];
const [a, b, c] = users; // (Destructuring)
console.log(a, b, c); // Output: Alireza Mostafa Mohammad
```

Destructuring Arrays

Example 1:

```
const users = ["Alireza", "Mostafa", "Mohammad", "Milad"];
const [a, b] = users; // (Destructuring)
console.log(a, b); // Output: Alireza Mostafa
```

Example 2:

```
const users = ["Alireza", "Mostafa", "Mohammad", "Milad"];
const [, , x, y] = users; // (Destructuring)
console.log(x, y); // Output: Mohammad Milad
```

Example 3:

```
const users = ["Alireza", "Mostafa", "Mohammad", "Milad"];
const [, x, , y] = users; // (Destructuring)
console.log(x, y); // Output: Mostafa Milad
```

Destructuring Arrays (Swapping Variables)

```
let [x, y] = ["Alireza", "Behnam"];
const temp = x;
x = y;
y = temp;
console.log(x, y); //output: Behnam Alireza
```

```
let [x, y] = ["Alireza", "Behnam"];
[x, y] = [y, x];
console.log(x, y); //output: Behnam Alireza
```

Destructuring Arrays (Nested)

```
1  //Nested Destructuring
2  const numbers = [10, 20, [1001, 1002]];
3  const [a, b, [c, d]] = numbers;
4  console.log(a, b, c, d); //output: 10 20 1001 1002
```

Destructuring Arrays (Default Values)

```
1  //Default Values
2  const numbers = [1001, 1002];
3  const [num1 = 1, num2 = 1, num3 = 1] = numbers;
4  console.log(num1, num2, num3); //output: 1001 1002 1
```

#Destructuring Objects

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};

const { id, title } = product; // (Destructuring Objects)
console.log(id, title); //output: 10 'HP Pavilion 15-DK1056WM'
```

Destructuring Objects (Rename Props)

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
const { id: code, title: product_name } = product; // (Destructuring Objects)
console.log(code, product_name); //output: 10 'HP Pavilion 15-DK1056WM'
```

Destructuring Objects (Default Values)

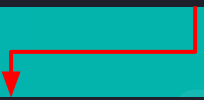
```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
const { id: code = 0, title: product_name, _status = "publish" } = product; // (Destructuring Objects)
console.log(code, product_name, _status); //output: 10 'HP Pavilion 15-DK1056WM' 'publish'
```

Destructuring Objects (Nested)

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [],
};
const {
  rating: { rate, count: rateCount },
} = product;
console.log(rate, rateCount); //output: 4.3 120
```

#The Spread Operator (...)

```
1  //! without using spread operator
2  const arr = [30, 40, 50];
3  const numbers = [10, 20, arr[0], arr[40], arr[50]];
```



```
1  //! using spread operator
2  const arr = [30, 40, 50];
3  const numbers = [10, 20, ...arr];
```

The Spread Operator (...)

```
//! clone array
const arr = [30, 40, 50];
const clone = [...arr];
```

```
//! concat two arrays
const arr1 = [30, 40, 50];
const arr2 = [70, 80, 90];
const joinedArr = [...arr1, ...arr2];
```

The Spread Operator (...)

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
function logger(a, b, c) {
  console.log(a);
  console.log(b);
  console.log(c);
}
logger(product.tags[0], product.tags[1], product.tags[2]);
```

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
function logger(a, b, c) {
  console.log(a);
  console.log(b);
  console.log(c);
}
logger(...product.tags);
```

The Spread Operator (...) - Objects

```
1  //! clone/copy object
2  const product = {
3    id: 10,
4    title: "HP Pavilion 15-DK1056WM",
5    description: "HP Pavilion 15-DK1056WM Gaming...",
6    price: 1099,
7    discountPercentage: 6.18,
8    rating: { rate: 4.3, count: 120 },
9    stock: 89,
10   brand: "HP Pavilion",
11   category: "laptops",
12   tags: ["Tech Product", "Laptop", "Gaming"],
13   thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
14   images: [
15     "https://i.dummyjson.com/data/products/10/1.jpg",
16     "https://i.dummyjson.com/data/products/10/2.jpg",
17     "https://i.dummyjson.com/data/products/10/3.jpg",
18   ],
19 };
20 const productCopy = { ...product };
```

The Spread Operator (...) - Objects

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
const newProduct = {
  ...product,
  id: 11,
  title: "Asus FX507-ZR",
  price: 1500,
  color: "white",
};
```

#Rest Pattern and Parameters

```
1  const [user1, user2, ...others] = ["nima", "amir", "hesam", "ahmad", "ali"];
2  console.log(user1); // nima
3  console.log(user2); // amir
4  console.log(others); // ["hesam", "ahmad", "ali"]
```

Rest Pattern and Parameters

```
function add(...numbers) {
  console.log(numbers);
}
add(5, 6); // [5, 6]
add(1, 4, 9); // [1, 4, 9]
add(7, 8, 99, 33, 5, 10); // [7, 8, 99, 33, 5, 10]
```

```
function add(username, ...numbers) {
  console.log(username, `you have entered ${numbers.length} numbers`);
}
add("hesam", 5, 6); // hesam you have entered 2 numbers
add("behnam", 1, 4, 9); // behnam you have entered 3 numbers
add("milad", 7, 8, 99, 33, 5, 10); //milad you have entered 6 numbers
```

Rest Pattern and Parameters

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 6.18,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  thumbnail: "https://i.dummyjson.com/data/products/10/thumbnail.jpeg",
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
const { id, title, ...others } = product;
```

#Short Circuiting (||)

The OR || operator evaluates expressions from left to right, and returns the right operand if the left evaluates to false. In contrast to AND, when used with many operands, this operator would return the first truthy value if any of the operands are true.

```
1  const expression1 = 7 || "Alireza";
2  console.log(expression1); // 7
3
4  const expression2 = "" || "Alireza";
5  console.log(expression2); // "Alireza"
6
7  const expression3 = 10 * 30 || undefined || "Alireza";
8  console.log(expression3); // 300
9
10 const expression4 = false || null || 0 || "Websila!" || true;
11 console.log(expression4); // Websila!
12
13 const expression5 = null || undefined;
14 console.log(expression5); // undefined
```

Short Circuiting (||)

```
function sum(a, b) {
  console.log(`Summing ${a} and ${b}`);
  return a + b;
}

const exp = 2 > 5 || "" || sum(5, 25) || 5 ** 1000000000;

console.log(exp);

// Summing 5 and 25
// 30
```

Short Circuiting (&&)

The AND && operator evaluates expressions from left to right, and returns the right operand if the left operand evaluates to true. If used with many operands, and any of the operands are false, this operator would return the first falsy value.

```
1  console.log(0 && "Alireza"); // 0
2  console.log(3 && "Alireza"); // "Alireza"
3  console.log(10 < 5 && "Alireza"); // false
4  console.log("Websila" && 10 * 20 && "" && "Alireza"); // ""
```

Short Circuiting (&&)

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 20,
  rating: { rate: 4.3, count: 120 },
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
  //   getTags: function () {
  //     console.log(...this.tags);
  //   },
};

product.getTags && product.getTags();
```

Short Circuiting (&&)

```
function getWeather() {
  console.log("It's really cold today");
  return 0;
}

function sum(a, b) {
  console.log(`Summing ${a} and ${b}`);
  return a + b;
}

const expression = 10 > 5 && "dillion" && getWeather() && sum(2, 5);

console.log(expression);

// It's really cold today
// 0
```

The Nullish Coalescing Operator (??)

The nullish coalescing (??) operator is a logical operator that returns its right-hand side operand when its left-hand side operand is null or undefined, and otherwise returns its left-hand side operand.

Nullish Values in Javascript :

- null
- undefined

```
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 20,
  rating: { rate: 4.3, count: 120 },
  stock: 0,
  brand: "HP Pavilion",
  category: "laptops",
  tags: ["Tech Product", "Laptop", "Gaming"],
  images: [
    "https://i.dummyjson.com/data/products/10/1.jpg",
    "https://i.dummyjson.com/data/products/10/2.jpg",
    "https://i.dummyjson.com/data/products/10/3.jpg",
  ],
};
const stock = product.stock ?? 1000;
console.log(stock); // 0
```

Logical Assignment Operators (||=, &&=, ??=)

```
const course = { title: "", duration: 36 };
course.duration = course.duration || 10;
course.title = course.title || "Title is Empty";
console.log(course.duration); // 36
console.log(course.title); // "Title is Empty"
```

```
const course = { title: "", duration: 36 };
course.duration ||= 10;
course.title ||= "Title is Empty";
console.log(course.duration); // 36
console.log(course.title); // "Title is Empty"
```

```
let person = { firstName: "Jane", lastName: "Doe" };
person.lastName = person.lastName && "Smith";
console.log(person.lastName); // "Smith"
```

```
let person = { firstName: "Jane", lastName: "Doe" };
person.lastName &&= "Smith";
console.log(person.lastName); // "Smith"
```

```
let product = { title: "Galaxy S24 Ultra", stock: 0 };
product.stock = product.stock ?? 100;
console.log(product.stock); // 0
```

```
let product = { title: "Galaxy S24 Ultra", stock: 0 };
product.stock ??= 100;
console.log(product.stock); // 0
```

Enhanced Object Literals

```
const rating = { rate: 4.3, count: 120 };
const images = [
  "https://i.dummyjson.com/data/products/10/1.jpg",
  "https://i.dummyjson.com/data/products/10/2.jpg",
  "https://i.dummyjson.com/data/products/10/3.jpg",
];
const tags = ["Tech Product", "Laptop", "Gaming"];
const attrs = ["color", "battery", "screen", "weight"];
const product = {
  id: 10,
  title: "HP Pavilion 15-DK1056WM",
  description: "HP Pavilion 15-DK1056WM Gaming...",
  price: 1099,
  discountPercentage: 20,
  rating, // Enhanced-Object-Literals
  images,
  tags,
  stock: 89,
  brand: "HP Pavilion",
  category: "laptops",
  [attrs[0]]: "black", // Enhanced-Object-Literals
  [attrs[3]]: "2.1 KG",
  getPrice() {
    // Enhanced-Object-Literals
    console.log(`${this.price}$`);
  },
};
product.getPrice();
```

Optional Chaining

```
if (socialUser.interests && socialUser.interests.addHobby) {
  socialUser.interests.addHobby("coding");
}
```

```
socialUser.interests?.addHobby?.("coding");
```

```
if (socialUser.profile && socialUser.profile.getFullName) {
  socialUser.profile.getFullName();
}
```

```
const fullName = socialUser.profile?.getFullName?.();
```

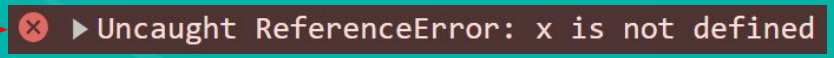
```
const socialUser = {
  // login props
  email: "mkalireza.ir@gmail.com",
  password: "123456",

  // profile props
  profile: {
    firstName: "Alireza",
    lastname: "Mohammadi",
    getFullName() {
      return `${this.firstName} ${this.lastname}`;
    },
  },

  // user interests props
  interests: {
    hobbies: ["hiking", "reading"],
    favoriteMovies: [
      {
        title: "Interstellar",
        rating: 4.8,
      },
      {
        title: "The Shawshank Redemption",
        rating: 4.5,
      },
    ],
    addHobby(interest) {
      this.hobbies.push(interest);
    },
  },
};
```

Behind the Scenes of JavaScript

Strict Mode

```
1  x = 12;
2  console.log(x); // 12
```

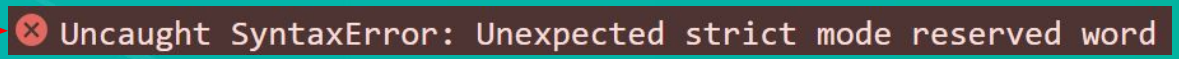
```
1  "use strict";
2
3  x = 12;
4  console.log(x);
```



Strict Mode

```
1  let package = 123;
2  console.log(package); // 123
```

```
1  "use strict";
2
3  let package = 123;
4  console.log(package);
```



#Lexical Scope in JavaScript

Lexical Scope in JavaScript

Type of scopes in JavaScript

1. Global Scope

```
const me = "Kordkheili";
const job = "Software Engineer";
const mood = "Motivated";
```

- outside of any block or function
- variables are accessible **everywhere**

2. Function Scope

```
function logger() {
  const me = "Kordkheili";
  console.log(`Hello ${me}`);
}
console.log(me); // ReferenceError
```

- variables are accessible just inside the function

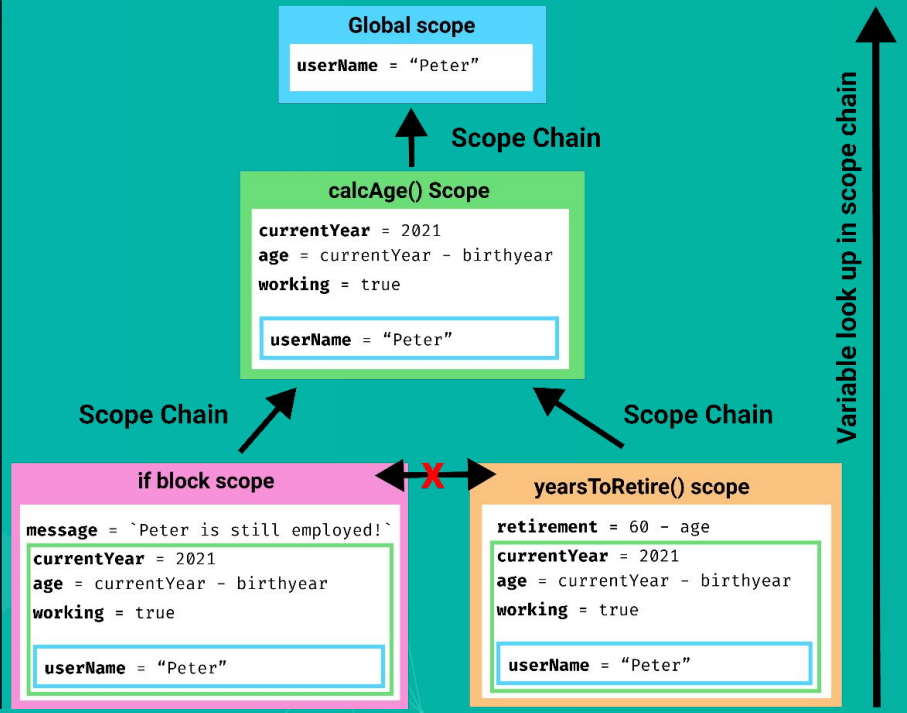
3. Block Scope (ES6)

```
if (20 > 10) {
  const me = "Kordkheili";
}
console.log(me); // ReferenceError
```

- Block is everything that is between **curly braces**.
- variables are accessible just inside block **(not var)**
- **Introduced in ES6**

Lexical Scope in JavaScript : Scope Chain

```
const userName = "Peter";

function calcAge(birthyear) {
  const currentYear = 2021;
  const age = currentYear - birthyear;

  if (age <= 60) {
    var working = true;
    const message = `Peter is still employed!`;
    console.log(message);
  }

  function yearsToRetire() {
    const retirement = 60 - age;
    console.log(`${userName} will be retired in ${retirement} years!`);
  }

  yearsToRetire();
}

calcAge(1975);
```



Lexical Scope in JavaScript : Some Concepts

- **Scoping :** where do variables live?
- **Lexical Scoping :** variables and functions interact based on their **position** within the code.
- Global vs Local Scope
- Variable Lookup
- 'var' is function scoped
- 'let' & 'const' are function & block scoped

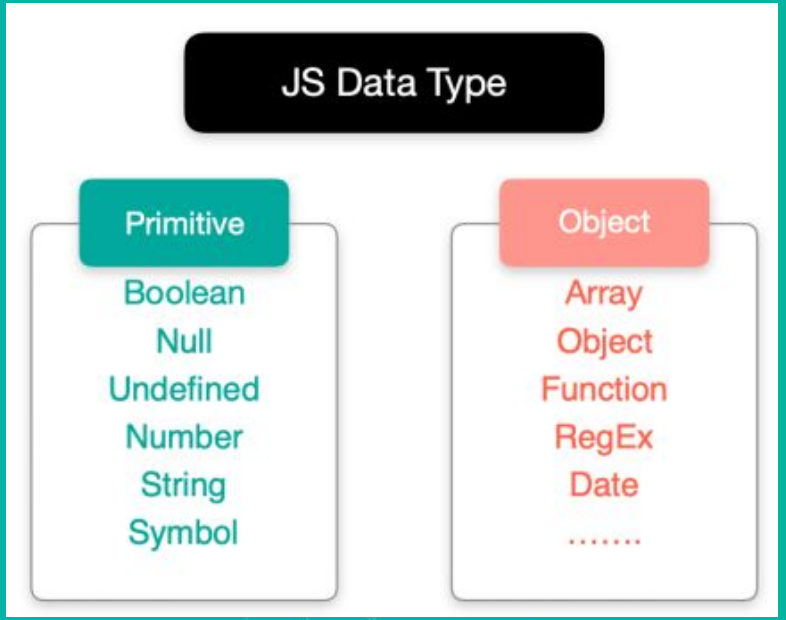
Primitive vs. Reference Types

Primitive (All Data Types except Object & Array)

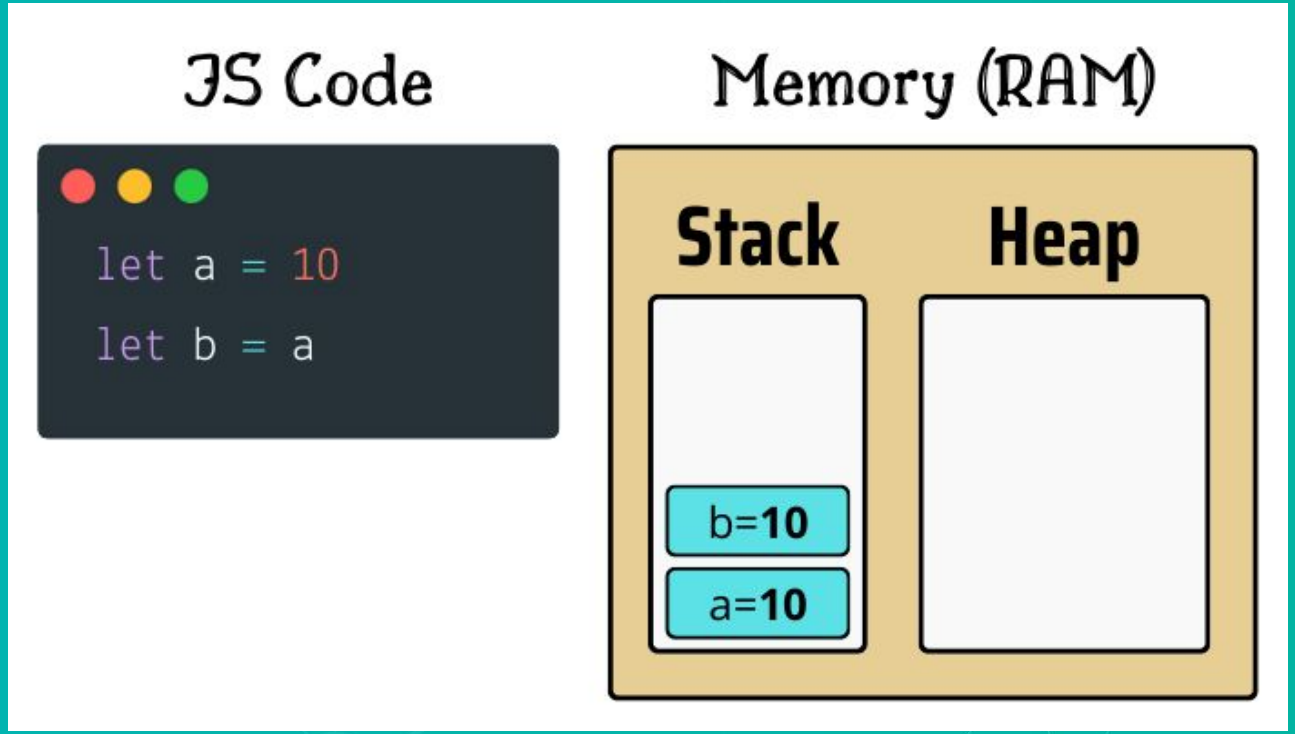
```
let lastname = "Kordkheli";
let lastname_copy = lastname;
lastname = "Mohammadi Kordkheili";
console.log(lastname); // "Mohammadi Kordkheili"
console.log(lastname_copy); // "Kordkheli"
```

Reference (Object & Array)

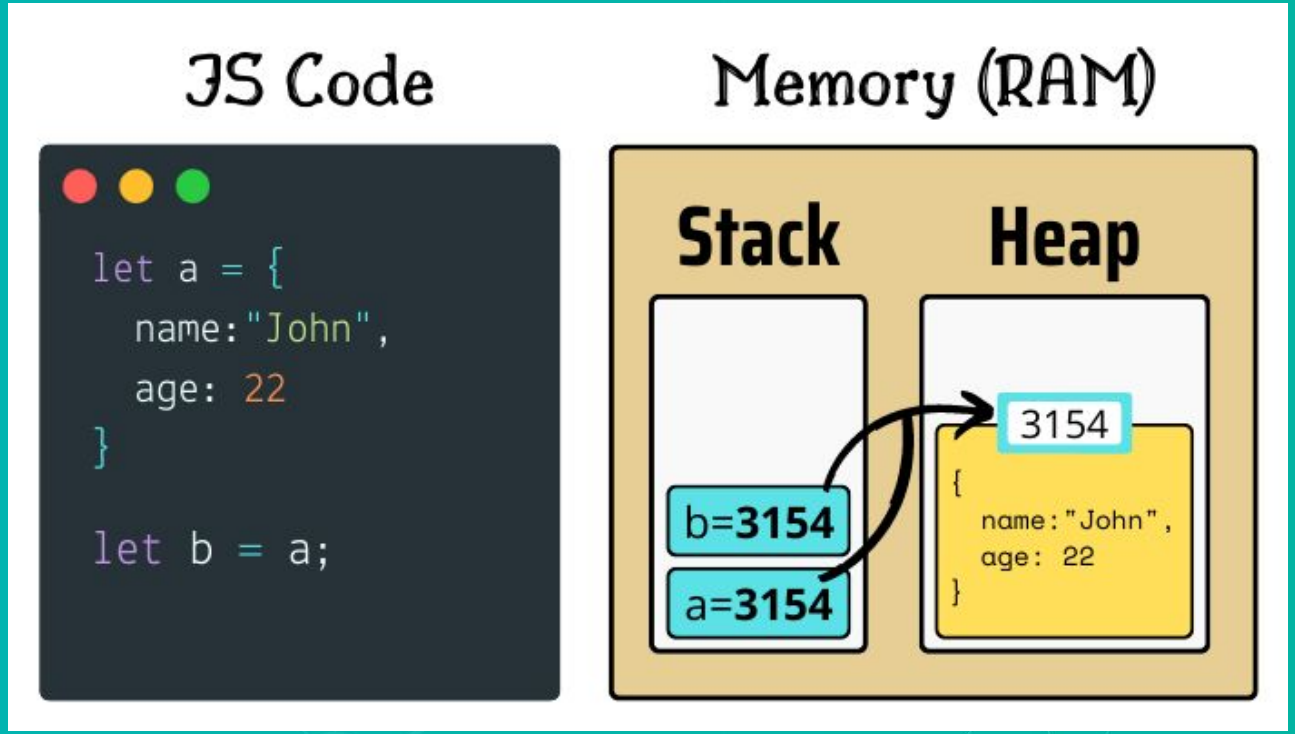
```
const me = { lastname: "Kordkheli" };
const me_copy = me;
me.lastname = "Mohammadi Kordkheili";
console.log(me); // { lastname: "Kordkheli" }
console.log(me_copy); // { lastname: "Kordkheli" }
```



Primitive Types



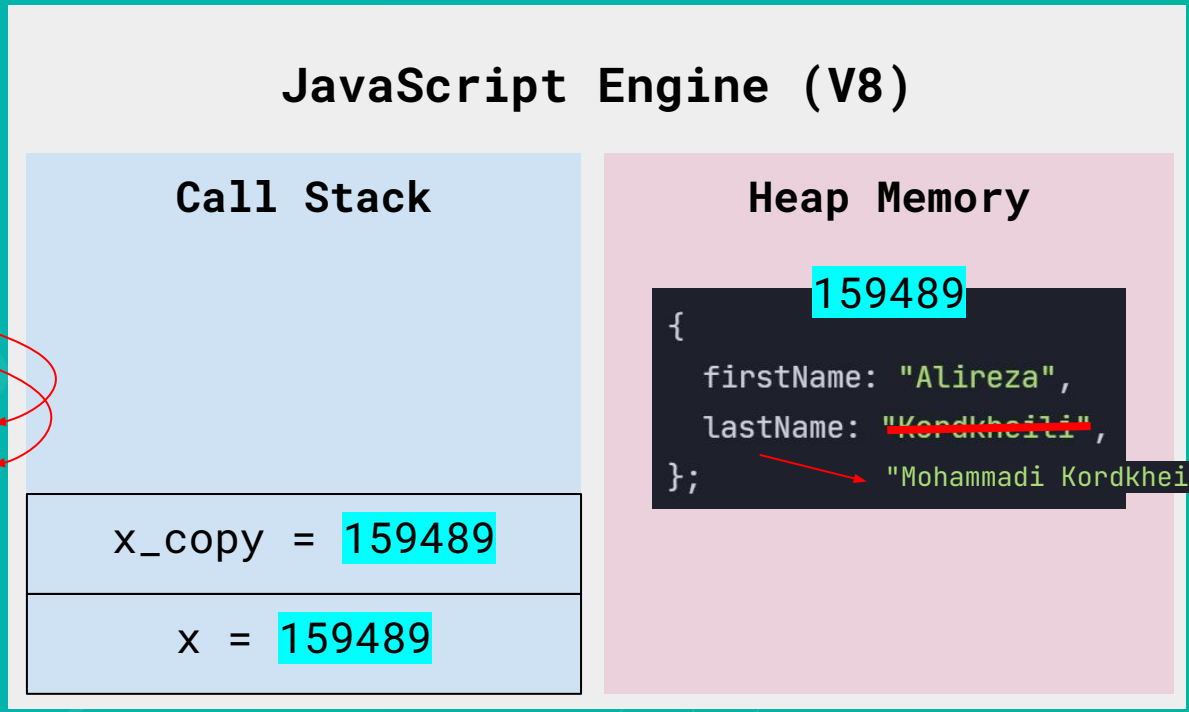
Reference Type (Object)



Reference Type (Object)

```
let x = {
  firstName: "Alireza",
  lastName: "Kordkheili",
};
let x_copy = x;
x.lastName = "Mohammadi Kordkheili";
console.log(x);
console.log(x_copy);
```

```
▶ {firstName: 'Alireza', lastName: 'Mohammadi Kordkheili'}
▶ {firstName: 'Alireza', lastName: 'Mohammadi Kordkheili'}
```



Numbers, Math
Date, Intl, Timers

#Challenges

1. **Lottery function**
2. **Random Color Generator**
3. **Guessing Game :** Write a program that generates a random number between 1 and 100 and allows the user to guess the number. Provide feedback after each guess (higher, lower, or correct) until the user guesses it correctly.
4. **BMI Calculator :** Write a function that takes height and weight as arguments and calculates the Body Mass Index (BMI) using the formula BMI = weight / (height * height). Interpret the result based on WHO's BMI categories (underweight, normal weight, overweight, etc.).
5. Calculate difference between two dates
6. Countdown Timer

#Working with Numbers

- toFixed()
- parseInt()
- parseFloat()
- isNaN()
- isFinite()
- isInteger()
- Numeric Separator (_)

```
const temperature = 25.4321;
const displayedTemperature = temperature.toFixed(1); // "25.4"
```

```
console.log(parseInt("123")); // 123 (default base-10)
console.log(parseInt("123", 10)); // 123 (explicitly specify base-10)
console.log(parseInt("   123 ")); // 123 (whitespace is ignored)
console.log(parseInt("077")); // 77 (leading zeros are ignored)
console.log(parseInt("1.9")); // 1 (decimal part is truncated)
console.log(parseInt("ff", 16)); // 255 (lower-case hexadecimal)
console.log(parseInt("xyz")); // NaN (input can't be converted to an integer)
```

```
const population = 1_234_567_890; // More readable than 1234567890
const distance = 27_381_000_000; // Easier to grasp than 27381000000
```

#Math Object

Constants :

- Math.PI
- Math.E
- Math.LN2

```
1  console.log(Math.PI); // 3.141592653589793
2  console.log(Math.E); // 2.718281828459045
3  console.log(Math.LN2); // 3.141592653589793
```

Math Object

Mathematical Operations :

- Math.abs()
- Math.pow()
- Math.sqrt()

```
1  console.log(2 ** 8); // 256
2  console.log(Math.pow(2, 8)); // 256
3  console.log(Math.sqrt(121)); // 11
4  console.log(121 ** (1 / 2)); // 11
5  console.log(8 ** (1 / 3)); // 2
6  console.log(Math.abs(-10)); // 10
7  console.log(Math.abs(8)); // 8
8  console.log(Math.abs(0)); // 0
```

Math Object

Trigonometric Functions :

- Math.sin()
- Math.cos()
- Math.tan()

```
1  console.log(Math.sin(Math.PI / 2)); // 1
2  console.log(Math.sin(Math.PI / 6)); // 0.49999999999999994
3  console.log(Math.sin(Math.PI / 3)); // 0.8660254037844386
4  console.log(Math.cos(Math.PI)); // -1
5  console.log(Math.tan(Math.PI)); // -1.2246467991473532e-16
```

Math Object

Additional Methods :

- Math.min()
- Math.max()
- Math.random()

```
1  console.log(Math.min(50, 4, 23, 43, 63, 100)); // 4
2  console.log(Math.max(50, 4, 23, 43, 63, 100)); // 100
3  console.log(Math.random()); // 0.41962305794938426
4
5  const numbers = [50, 60, 35, 43, 1, 0, -70];
6  console.log(Math.min(...numbers)); // -70
7  console.log(Math.max(...numbers)); // 60
```

Math Object

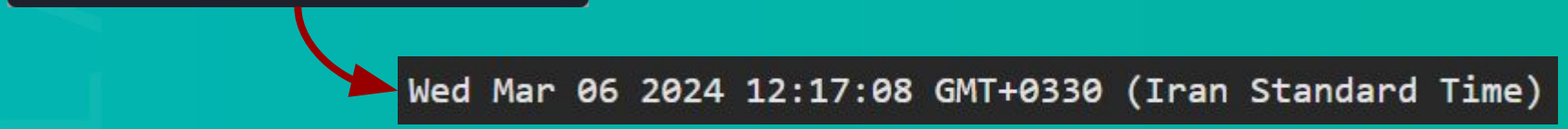
Rounding Methods

- Math.round()
- Math.ceil()
- Math.floor()
- Math.trunc()

```
1  console.log(Math.round(0.9)); // 1
2  console.log(Math.round(1.2)); // 1
3  console.log(Math.round(-5.07)); // -5
4  console.log(Math.ceil(3.9)); // 4
5  console.log(Math.ceil(4.2)); // 5
6  console.log(Math.floor(2.8)); // 2
7  console.log(Math.floor(10.1)); // 10
8  console.log(Math.trunc(15.7)); // 15
9  console.log(Math.trunc(17.3)); // 17
10 console.log(Math.trunc(45.8037)); // 45
```

#Date Object

```
1  console.log(new Date());
```



Different Date constructors :

- new Date(year, month, day, hours, minutes, seconds, milliseconds)
- new Date("2027-03-06")
- new Date(2047408200000)

Date Object

Extracting Date Components

- getFullYear(): Get the year (e.g., 2024)
- getMonth(): Get the month (0-indexed, e.g., 2 for March)
- getDate(): Get the day of the month (1-31)
- getDay(): Get the day of the week (0-6)
- getHours(), getMinutes(), getSeconds(), getMilliseconds(): Get time components
- getTime(): Get the number of milliseconds since epoch (1970-01-01)

#Operations With Dates

```
const date1 = new Date(2034, 10, 18);
console.log(Number(date1)); // 2047408200000
```

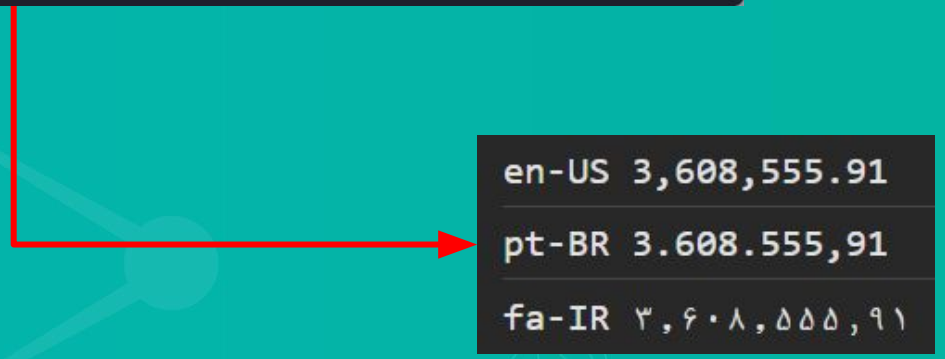
```
const current = new Date();
const date1 = new Date(2034, 10, 18);
const date2 = new Date(2034, 10, 21);

console.log(date2 > date1); // true
console.log(current <= date2); // true
console.log(date1 === date2); // false
console.log(date2 - date1); // 259200000
console.log(Math.max(current, date1, date2)); // 2047667400000
```

#Intl API - Internationalizing Dates and Times

```
1  const today = new Date();
2
3  const options = {
4    weekday: "long",
5    year: "numeric",
6    month: "long",
7    day: "numeric",
8    hour: "2-digit",
9    minute: "2-digit",
10   second: "2-digit",
11 };
12
13 const formatterIR = new Intl.DateTimeFormat("fa-IR", options);
14 const formattedDateIR = formatterIR.format(today);
15 console.log(formattedDateIR);
```

#Intl API - Internationalizing Numbers

```
1  const number = 3608555.91;
2  console.log("en-US", Intl.NumberFormat("en-US").format(number));
3  console.log("pt-BR", Intl.NumberFormat("pt-BR").format(number));
4  console.log("fa-IR", Intl.NumberFormat("fa-IR").format(number));
```



Intl API - Internationalizing Numbers

```
const speed = 100;
const options = {
  style: "unit",
  unit: "kilometer-per-hour",
};
const formattedSpeed = new Intl.NumberFormat("en-US", options).format(speed);
console.log(formattedSpeed); // 100 km/h
```

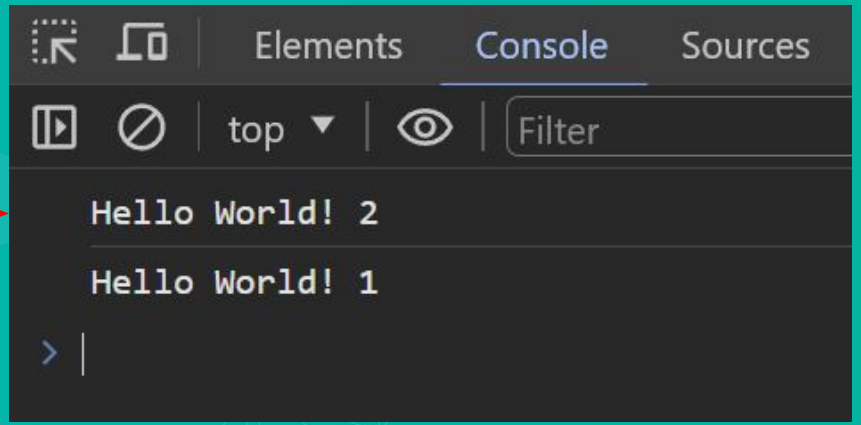
```
const price = 1125.99;
const options = {
  style: "currency",
  currency: "USD",
};
const formattedPrice = new Intl.NumberFormat("en-CA", options).format(price);
console.log(formattedPrice); // "US$1,125.99"
```

```
const percent = 0.25;
const options = {
  style: "percent",
};
const formatted = new Intl.NumberFormat("fa-IR", options).format(percent);
console.log(formatted); // ۲۵٪
```

#Timers - setTimeout and setInterval

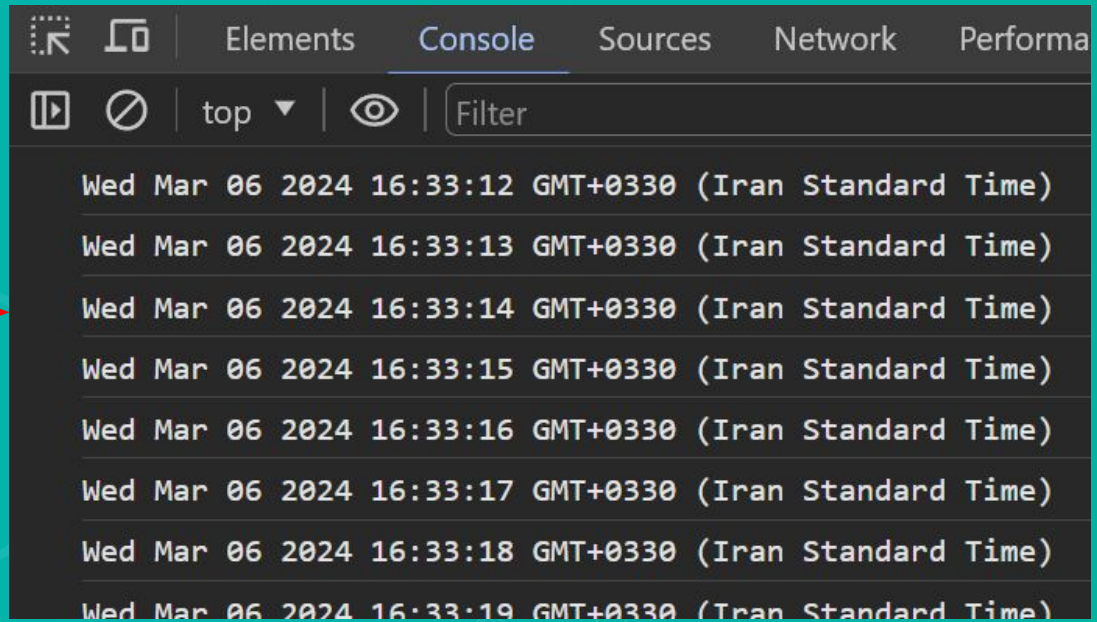
setTimeout

```
1  setTimeout(function () {
2    console.log("Hello World! 1");
3  }, 3000);
4
5  console.log("Hello World! 2");
```



setInterval

```
1  setInterval(function () {
2    const now = new Date();
3    console.log(now);
4  }, 1000);
```



clearTimeout

```
1  const timer = setTimeout(function () {
2    console.log("Hello World! 1");
3  }, 3000);
4
5  if (20 > 10) {
6    clearTimeout(timer);
7  }
```

clearInterval

```
1  const timer = setInterval(function () {
2    const now = new Date();
3    console.log(now);
4  }, 1000);
5
6  if (20 > 10) {
7    clearInterval(timer);
8  }
```

Functions (Advanced)

#Arrow Functions

```
1  // Before Arrow
2  const hello = function () {
3    return "Hello World!";
4  };
```

```
1  // With Arrow Function
2  const hello = () => {
3    return "Hello World!";
4  };
```

Arrow Functions (With Parameters)

```
1  const classicSum = function (a, b) {
2    return a + b;
3  };
4  const result = classicSum(100, 200);
5  console.log(result); // 300
```

```
1  const arrowSum = (a, b) => {
2    return a + b;
3  };
4  const result = arrowSum(100, 200);
5  console.log(result); // 300
```

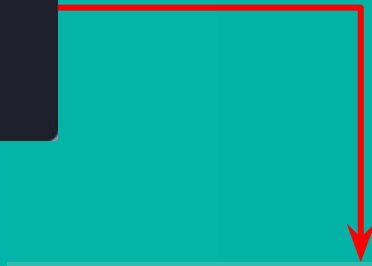
Arrow Function Return Value by Default

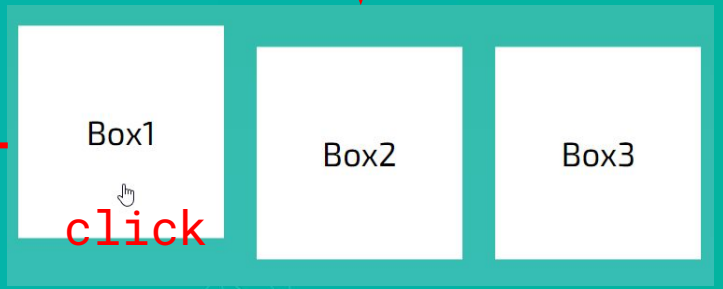
```
1  const hello = () => "Hello World!";
2  const result = hello();
3  console.log(result); // "Hello World!"
```

```
1  const hello = (x) => "Hello " + x;
2  const result = hello("Alireza");
3  console.log(result); // "Hello Alireza"
```

What About `this` ?

```
1  box1.addEventListener("click", function () {
2    console.log(this);
3  });
```

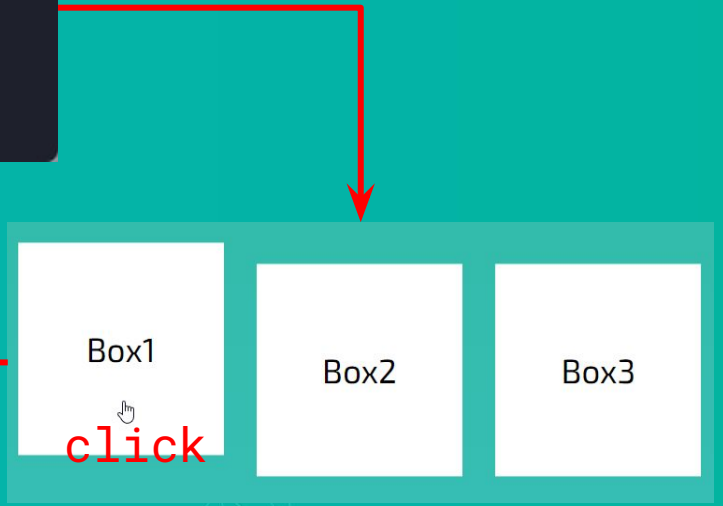





```
                                    script.js:4
<div id="box1" class="box">Box1</div>
```

What About `this` ?

```
1  box1.addEventListener("click", () => {
2    console.log(this);
3  });
```




```
                                        script.js:8
Window {window: Window, self: Window,
▶ document: document, name: '', locatio
n: Location, …}
```

#Default Parameters

example1

```
function logger(x = "Kordkheili") {
  console.log(x);
}
logger("Alireza"); // "Alireza"
logger(); // "Kordkheili"
```

example2

```
const posts = [];
function createPost(
  id = posts.length + 1,
  title = `Post Number ${id}`,
  description = `Description for post ${id}...`,
  publishDate = new Date()
) {
  const newPost = {
    id,
    title,
    description,
    publishDate: Intl.DateTimeFormat("fa-IR").format(publishDate),
  };
  posts.push(newPost);
  console.log(posts);
}
createPost();
createPost(1001, undefined, undefined, new Date("2037-01-01"));
createPost(
  1002,
  "JavaScript Course",
  "Some stuff here...",
  new Date("2037-01-01")
);
```

#Primitive vs Reference Argument

```
1  const user = {
2    firstName: "Alireza",
3    lastName: "Mohammadi",
4  };
5  const job = "Developer 👨🏻‍💻";
6
7  function apply(userObj, JobName) {
8    userObj.lastName = "Kordkheili";
9    JobName = "Software-Engineer";
10   if (userObj.firstName === "Alireza") {
11     console.log("your application request has been approved 😍");
12   } else {
13     console.log("You are not suitable for this position! 😳");
14   }
15 }
16
17 apply(user, job);
18 console.log(user);
19 console.log(job);
```

```
your application request has been approved 😍
▶ {firstName: 'Alireza', lastName: 'Kordkheili'}
developer
```

#First-Class Functions

✅ Functions are simply Values
✅ Functions are another type of object

```
const sum = (x, y) => x + y;
const logger = function () { console.log("Hello!"); };
const counter = {
  value: 1,
  increase: function () { this.value++ },
};
```

```
const clickHandler = () => console.log("click event");
logo.addEventListener("click", clickHandler);
```

#Higher-Order Functions

1. Function **receives** a function

```
const clickHandler = () => console.log("click event");
logo.addEventListener("click", clickHandler);
```

2. Functions **returning** a function

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

This is only possible because of first-class functions.

#Functions Accepting Callback Functions

```
function isOdd(x) {
  return x % 2 === 1;
}
function isEven(x) {
  return x % 2 === 0;
}
function customFilter(arr, fn) {
  const result = [];
  for (let i = 0; i < arr.length; i++) {
    if (fn(arr[i])) result.push(arr[i]);
  }
  return result;
}

const array = [2, 4, 5, 6, 7, 9];
console.log(customFilter(array, isOdd)); // [5, 7, 9]
console.log(customFilter(array, isEven)); // [2, 4, 6]
```

#Functions Returning Functions

```
function greet(prefix) {
  return function (name) {
    console.log(`${prefix} ${name}!`);
  };
}

// Create a greeting function with a specific prefix
const greetWithHello = greet("Hello");
const greetWithHi = greet("Hi");

// Use the returned functions to greet different people
greetWithHello("Alice"); // Output: Hello Alice!
greetWithHi("Bob"); // Output: Hi Bob!
```

#The apply and call Methods => apply()

```
const user1 = { firstName: "Alireza" };
const user2 = { firstName: "Sara" };

function sayHello() {
  console.log(`Hello ${this.firstName}!`);
}

function login(email, password) {
  console.log(`Welcome ${this.firstName}!`);
  console.log(`Login Info : ${email} | ${password}`);
}

sayHello.apply(user1); // Hello Alireza!
sayHello.apply(user2); // Hello Sara!

login.apply(user1, ["alireza@gmail.com", "123456"]);
// Welcome Alireza!
// Login Info : alireza@gmail.com | 123456

login.apply(user2, ["sara@gmail.com", "654321"]);
// Welcome Sara!
// Login Info : sara@gmail.com | 654321
```

The apply and call Methods => call()

```
const user1 = { firstName: "Alireza" };
const user2 = { firstName: "Sara" };

function sayHello() {
  console.log(`Hello ${this.firstName}!`);
}

function login(email, password) {
  console.log(`Welcome ${this.firstName}!`);
  console.log(`Login Info : ${email} | ${password}`);
}

sayHello.call(user1); // Hello Alireza!
sayHello.call(user2); // Hello Sara!

login.call(user1, "alireza@gmail.com", "123456");
// Welcome Alireza!
// Login Info : alireza@gmail.com | 123456

login.call(user2, "sara@gmail.com", "654321");
// Welcome Sara!
// Login Info : sara@gmail.com | 654321
```

The bind Method

```
const user1 = { firstName: "Alireza" };
const user2 = { firstName: "Sara" };

function sayHello() {
  console.log(`Hello ${this.firstName}!`);
}

function login(email, password) {
  console.log(`Welcome ${this.firstName}!`);
  console.log(`Login Info : ${email} | ${password}`);
}

const fn1 = sayHello.bind(user1);
fn1(); // Hello Alireza!

const fn2 = sayHello.bind(user2);
fn2(); // Hello Sara!

const fn3 = login.bind(user1, "alireza@gmail.com", "123456");
fn3();
// Welcome Alireza!
// Login Info : alireza@gmail.com | 123456

const fn4 = login.bind(user2, "sara@gmail.com", "654321");
fn4();
// Welcome Sara!
// Login Info : sara@gmail.com | 654321
```

The bind Method

```
const user1 = { firstName: "Alireza" };
const user2 = { firstName: "Sara" };

function sayHello() {
  console.log(`Hello ${this.firstName}!`);
}

function login(email, password) {
  console.log(`Welcome ${this.firstName}!`);
  console.log(`Login Info : ${email} | ${password}`);
}

const fn1 = sayHello.bind(user1)(); // Hello Alireza!

const fn2 = sayHello.bind(user2)(); // Hello Sara!

const fn3 = login.bind(user1, "alireza@gmail.com", "123456")();
// Welcome Alireza!
// Login Info : alireza@gmail.com | 123456

const fn4 = login.bind(user2, "sara@gmail.com", "654321")();
// Welcome Sara!
// Login Info : sara@gmail.com | 654321
```

apply, call, bind in Practice

```
const user1 = {
  firstName: "Alireza",
  lastName: "Kordkheili",
  introduce: function (job) {
    console.log(`I'm ${this.firstName} ${this.lastName}, ${job}`);
  },
};
const user2 = {
  firstName: "Sara",
  lastName: "Razavi",
};
user1.introduce("Developer"); // I'm Alireza Kordkheili, Developer
user1.introduce.apply(user2, ["Marketer"]); // I'm Sara Razavi, Marketer
```

#Immediately Invoked Function Expressions (IIFE)

```
// Classic IIFE
(function () {
  console.log("I can only run once 😵");
})();

// Arrow IIFE
(() => {
  console.log("me too... 😵‍💫");
})();
```

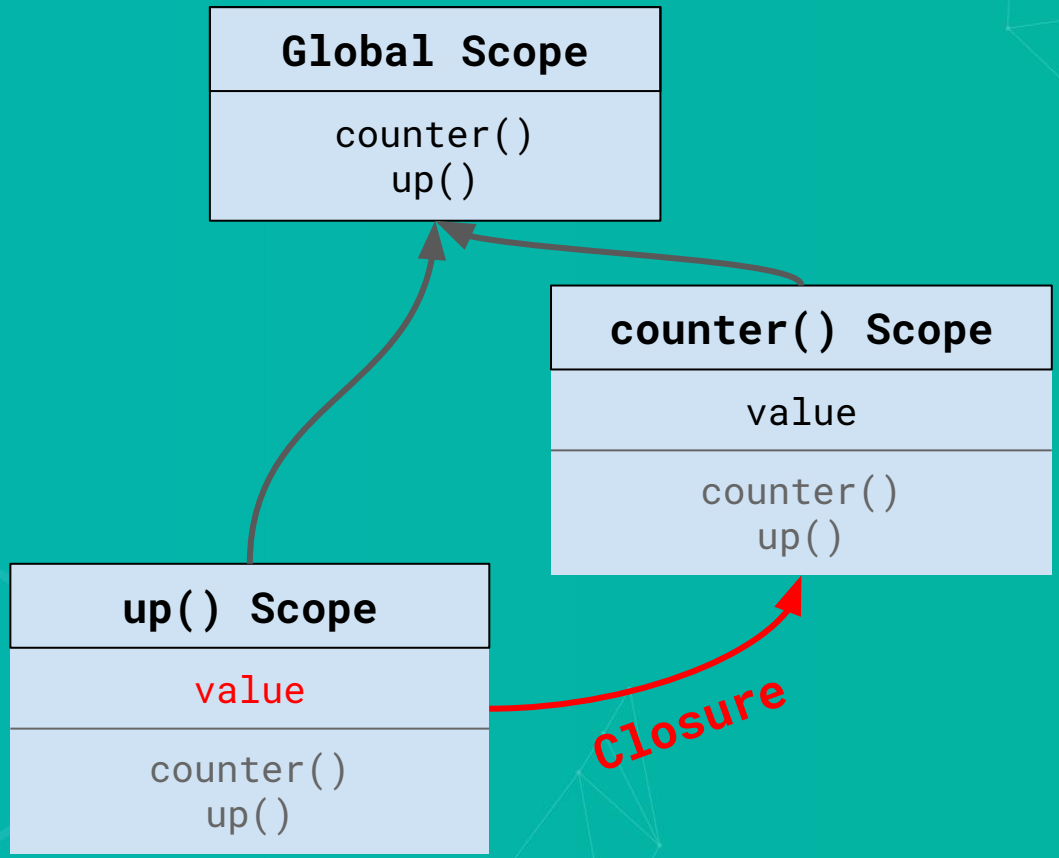
#Closures

a closure makes a function remember all the variables that existed at the functions birth place.

```
function makeFunc() {
  const name = "Kordkheili";
  return function () {
    console.log(name);
  };
}

const myFunc = makeFunc();
myFunc(); // Kordkheili
```

Closures

```
1  function counter() {
2    let value = 0;
3    return function () {
4      value++;
5      console.log(value);
6    };
7  }
8  const up = counter();
9  up(); // 1
10 up(); // 2
11 up(); // 3
```



Arrays (Advanced) & Iterables

#Looping Arrays: forEach

```
// Using a for loop
const numbers = [10, 11, 12, 13, 14, 15];
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}
```

```
// Using forEach method
const numbers = [10, 11, 12, 13, 14, 15];
numbers.forEach(function (number) {
  console.log(number);
});
```

Looping Arrays: forEach

```
// Imagine we have an array of cities
const cities = ["New York", "Los Angeles", "Chicago", "Houston"];

// We want to log each city along with its index and the original array
cities.forEach(function (city, index, array) {
  console.log(`Index: ${index}, City: ${city}, Array: [${array}]`);
});

// Index: 0, City: New York, Array: [New York,Los Angeles,Chicago,Houston]
// Index: 1, City: Los Angeles, Array: [New York,Los Angeles,Chicago,Houston]
// Index: 2, City: Chicago, Array: [New York,Los Angeles,Chicago,Houston]
// Index: 3, City: Houston, Array: [New York,Los Angeles,Chicago,Houston]
```

#Looping Arrays: The for-of Loop

```
// Imagine we have an array of fruits
const fruits = ["apple", "banana", "orange"];

// Using for-of loop to iterate over the array
for (const fruit of fruits) {
  console.log(fruit);
}
// output 👇
// apple
// banana
// orange
```

#Challenge: Price Calculator

Challenge: Price Calculator

You have an array of products, each represented by an object with three properties: name, price, and quantity. Your task is to iterate over this array using a forEach loop, calculate the total cost for each product (price multiplied by quantity), and then display the details of each product along with its total cost. Finally, you need to calculate the total cost of all products combined and display it.

```
const products = [
  { name: "Apple", price: 1.99, quantity: 5 },
  { name: "Banana", price: 0.99, quantity: 10 },
  { name: "Orange", price: 2.49, quantity: 3 },
];

/*
    Expected Output: 👇

    Product: Apple
    Quantity: 5
    Price: $1.99
    Total Cost: $9.95

    Product: Banana
    Quantity: 10
    Price: $0.99
    Total Cost: $9.90

    Product: Orange
    Quantity: 3
    Price: $2.49
    Total Cost: $7.47

    Total Cost of all Products: $27.32
*/
```

#The map Method

```
// Imagine we have an array of numbers
const numbers = [1, 2, 3, 4, 5];

// Using map method to create a new array where each element is doubled
const doubledNumbers = numbers.map(function (number) {
  return number * 2;
});

// Logging the original and new arrays
console.log("Original:", numbers); // Original : [1, 2, 3, 4, 5]
console.log("Doubled:", doubledNumbers); // Doubled : [2, 4, 6, 8, 10]
```

The map Method

```
const names = ["Alice", "Bob", "Charlie"];

// Using map method to add index to each name
const convertedNames = names.map((name, index) => `${index + 1}-${name}`);

// Logging the original and new arrays
console.log("Original:", names);
console.log("Converted:", convertedNames);

// Output👇
// Original: ['Alice', 'Bob', 'Charlie']
// Indexed: ['1-Alice', '2-Bob', '3-Charlie']
```

#Challenge: Celsius to Fahrenheit

Challenge: Celsius to Fahrenheit

You have an array of Celsius temperatures representing the daily temperatures for a week. Your task is to create a function `celsiusToFahrenheit()` that takes an array of Celsius temperatures as input and returns an array of corresponding temperatures in Fahrenheit using the `map()` method.

1. Define a function `celsiusToFahrenheit()` that takes an array of Celsius temperatures as input.
2. Inside the function, use the `map()` method to iterate over each temperature in the array.
3. For each Celsius temperature, convert it to Fahrenheit using the formula `(C × 9 ÷ 5) + 32`
4. Return a new array containing the temperatures converted to Fahrenheit.
5. Test your function with sample input arrays and verify that it produces the correct output.

Example:

```
const temperatureCelsius = [0, 10, 20, 30, 40];
const temperatureFahrenheit = celsiusToFahrenheit(temperatureCelsius);
console.log(temperatureFahrenheit); // Expected Output: [32, 50, 68, 86, 104]
```

#The filter Method

Classic Function

```
const numbers = [1, 2, 3, 4, 5];

// Using filter method to create a new array with even numbers
const evenNumbers = numbers.filter(function (number) {
  return number % 2 === 0;
});
console.log(evenNumbers); // [2,4]
```

Arrow Function

```
const numbers = [1, 2, 3, 4, 5];

// Using filter method to create a new array with even numbers
const evenNumbers = numbers.filter((number) => number % 2 === 0);
console.log(evenNumbers); // [2,4]
```

The filter Method

Classic Function

```
const names = ["Alice", "Bob", "Charlie", "David", "Eve"];

// Using filter method to create a new array with names shorter than 5 characters
const shortNames = names.filter(function (name) {
  return name.length < 5;
});

console.log(shortNames); // ['Bob', 'Eve']
```

Arrow Function

```
const names = ["Alice", "Bob", "Charlie", "David", "Eve"];

// Using filter method to create a new array with names shorter than 5 characters
const shortNames = names.filter((name) => name.length < 5);

console.log(shortNames); // ['Bob', 'Eve']
```

#The reduce Method

Classic Function

```
const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce(function (accumulator, currentValue) {
  return accumulator + currentValue;
}, 0);

console.log(sum); // 15
```

Arrow Function

```
const numbers = [1, 2, 3, 4, 5];

const sum = numbers.reduce(
  (accumulator, currentValue) => accumulator + currentValue,
  0
);

console.log(sum); // 15
```

#The find & findIndex Methods

find()

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.find((element) => element > 10);
console.log(found); // 12
```

findIndex()

```
const array1 = [5, 12, 8, 130, 44];
const found = array1.findIndex((element) => element > 13);
console.log(found); // 3
```

#some and every

some()

```
const numbers = [1, 2, 3, 4, 5];

// Using `some` method to check if any number is greater than 3
const result = numbers.some((number) => number > 3);
console.log(result); // true
```

every()

```
const numbers = [1, 2, 3, 4, 5];

// Using `every` method to check if all numbers are greater than 3
const result = numbers.every((number) => number > 3);
console.log(result); // false
```

#Sorting Arrays
sort()

```
const months = ["March", "Jan", "Feb", "Dec"];
months.sort();
console.log(months); // Array ["Dec", "Feb", "Jan", "March"]
```

```
const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1); // Array [1, 100000, 21, 30, 4]
```

Sorting Arrays
`sort()`

```
let numbers = [1, 100000, 21, 30, 4];

// Using `sort` method to sort the numbers in ascending order
numbers.sort((a, b) => a - b);
console.log(numbers); // Array [1, 4, 21, 30, 100000]
```

```
let numbers = [1, 100000, 21, 30, 4];

// Using `sort` method to sort the numbers in descending order
numbers.sort((a, b) => b - a);
console.log(numbers); // Array [100000, 30, 21, 4, 1]
```

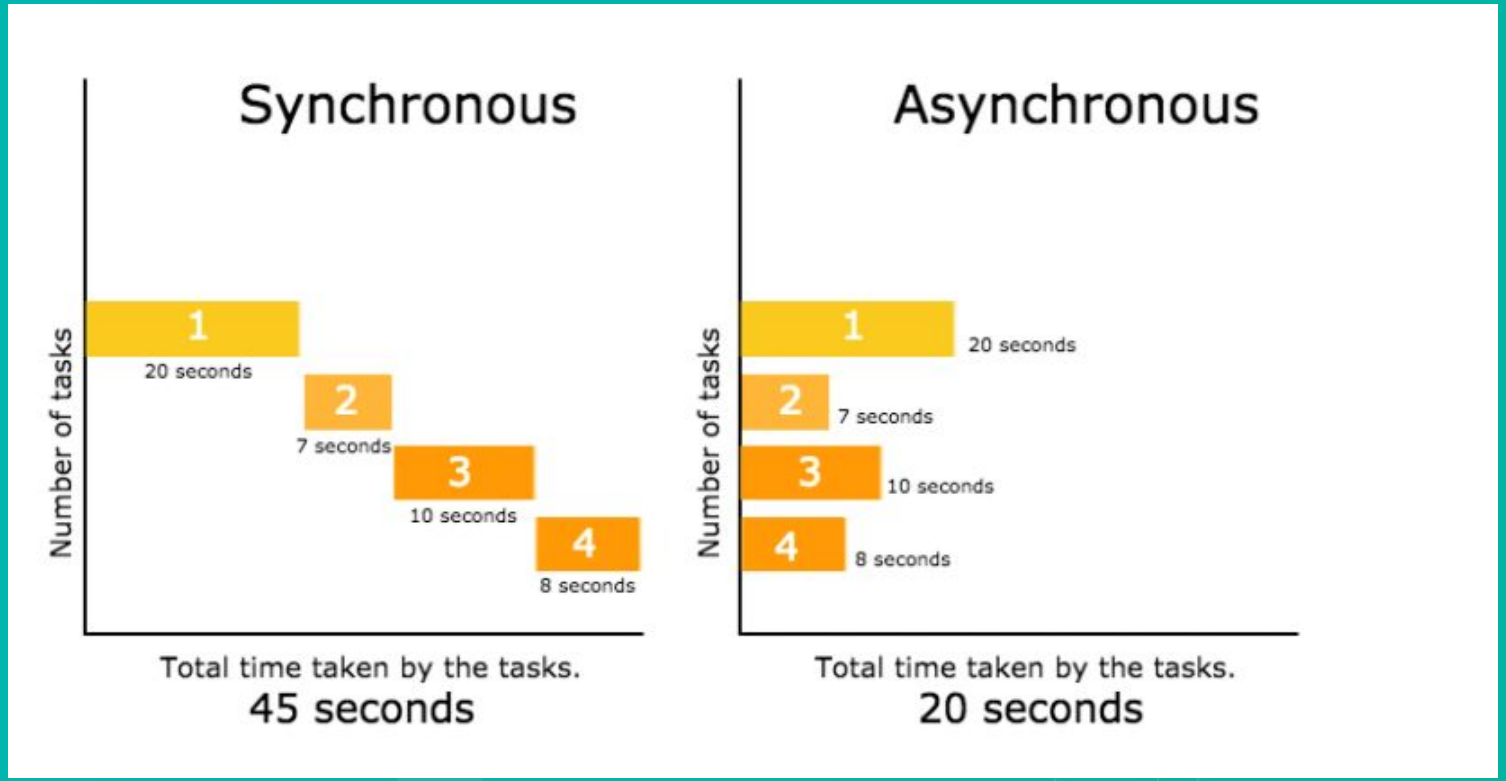
Sorting Arrays
reverse()

```
let array1 = ["Alireza", "Behnam", "Arman", "Sara"];
const reversed = array1.reverse();
console.log(reversed); // Array ['Sara', 'Arman', 'Behnam', 'Alireza']

// Careful: reverse is destructive => it changes the original array.
console.log(array1); // Array ['Sara', 'Arman', 'Behnam', 'Alireza']
```

Project:
Programmers Social Media

Asynchronous JavaScript Promises, Async/Await, and AJAX

Synchronous, Asynchronous



Real-World Examples

Drawing a Picture (Synchronous)

- Get a blank piece of paper.
- Draw a sun in one corner.
- Next, draw clouds next to the sun.
- At the bottom, draw trees.
- Finish by adding grass.

Downloading a File (Asynchronous)

- Start downloading a file.
- Work on other tasks while it downloads.
- Get a notification or find the file in your downloads when it's done.
- Open and use the file.
- Keep working on other tasks.

synchronous = Blocking

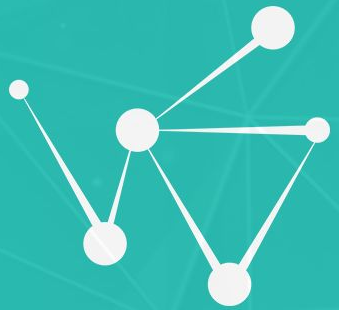
Asynchronous = Non-Blocking

```
1  console.log("Salam");
2
3  setTimeout(function () {
4    console.log("🔥async🔥");
5  }, 2000);
6
7  console.log("Aleyke Salam!");
```

```
1  console.log("Salam");
2  alert("Mozaahem😁");
3  console.log("Aleyke Salam!");
```

The Ultimate JavaScript Course

Lecture:
AJAX, API



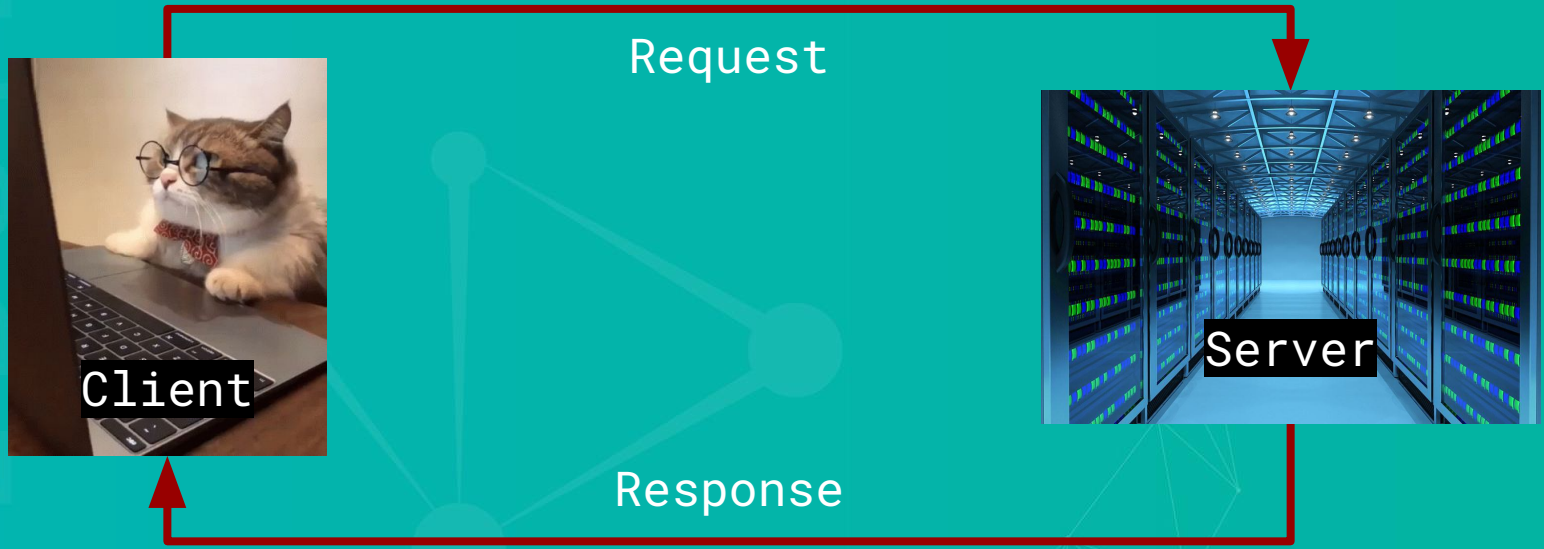
آکادمی وبسیلا

www.WEBSILA.co

مدرس: علیرضا محمدی کردخیلی

AJAX

Asynchronous **J**avaScript **A**nd **X**ML is a set of web development techniques that enables web applications to communicate with a server **asynchronously**. This means that web applications can **send** and **receive** data from a server in the background without interfering with the display and behavior of the existing page.

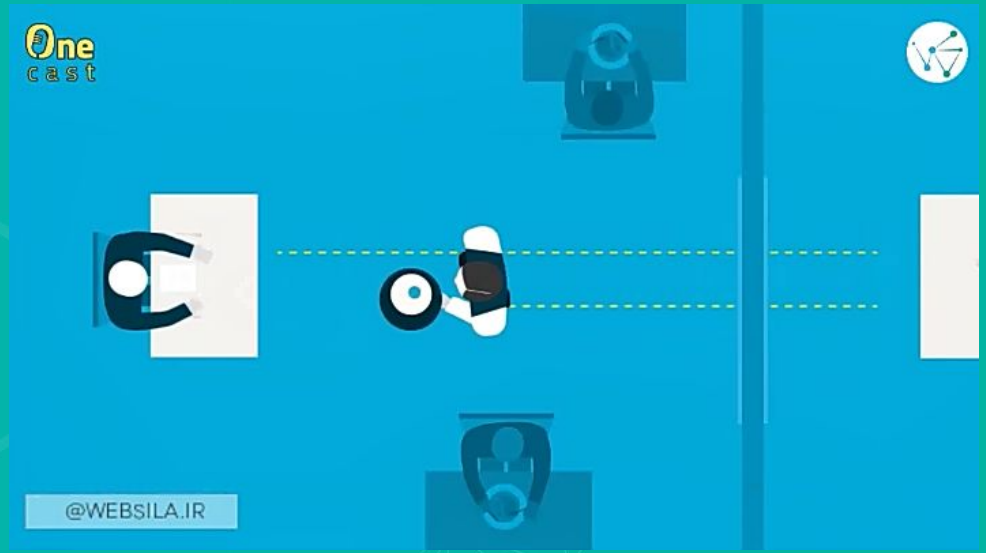


#API

Application **P**rogramming **I**nterface is a set of rules and protocols that allows different software applications to talk to each other. APIs enable developers to access specific features or data from an application, service, or operating system without needing to understand the underlying code or architecture.

⚠️We can 👇

1. build our own web APIs
   (requires back-end development, e.g. with node.js or python fastAPI)
2. or use 3rd-party APIs.



How APIs can help us?

- Weather data☁️
- Get the latest news📰
- Flights data✈️
- Currency exchange💰
- Cryptocurrency market data insights
- APIs for sending SMS📶
- *Endless other possibilities… (visit https://github.com/public-apis/public-apis)*

See a Real API

Some Web API examples: 👇

- openweathermap.org ☁️
- newsapi.org 📰
- api.nasa.gov 🌠
- restcountries.com 🌍
- thecatapi.com 🐈

fakestoreapi.com/products

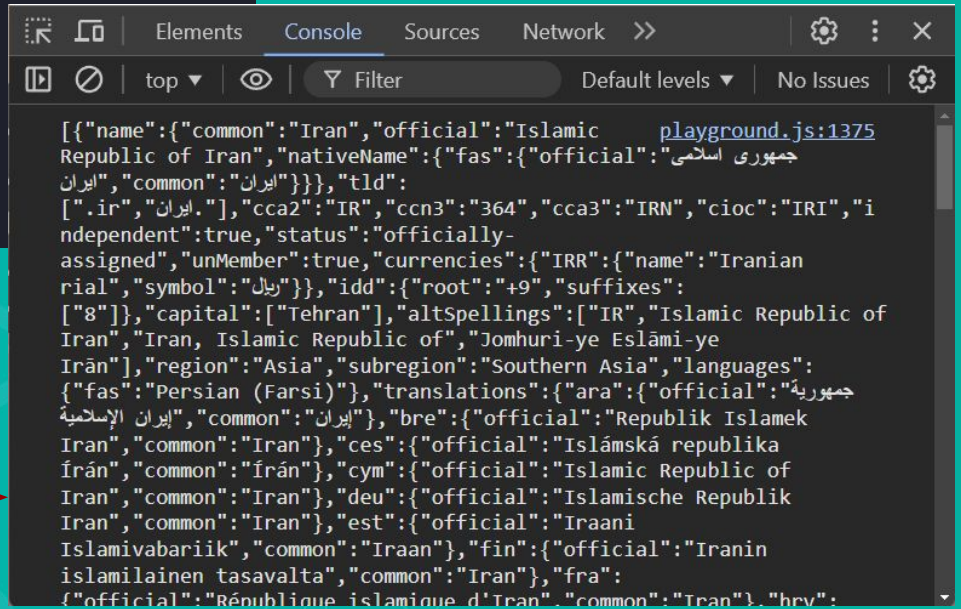
fakestoreapi.com/products

Pretty-print ☑


```
[
  {
    "id": 1,
    "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
    "price": 109.95,
    "category": "men's clothing",
    "image": "https://fakestoreapi.com/img/81fPKd-2AYL._AC_SL1500_.jpg",
    "rating": {
      "rate": 3.9,
      "count": 120
    }
  },
  {
    "id": 2,
    "title": "Mens Casual Premium Slim Fit T-Shirts ",
    "price": 22.3,
    "category": "men's clothing",
    "image": "https://fakestoreapi.com/img/71-3HjGNDUL._AC_SY879._SX._UX._SY._UY_.jpg",
    "rating": {
      "rate": 4.1,
      "count": 259
    }
  },
  {
    "id": 3,
    "title": "Mens Cotton Jacket",
    "price": 55.99,
    "category": "men's clothing",
    "image": "https://fakestoreapi.com/img/71li-ujtlUL._AC_UX679_.jpg",
```

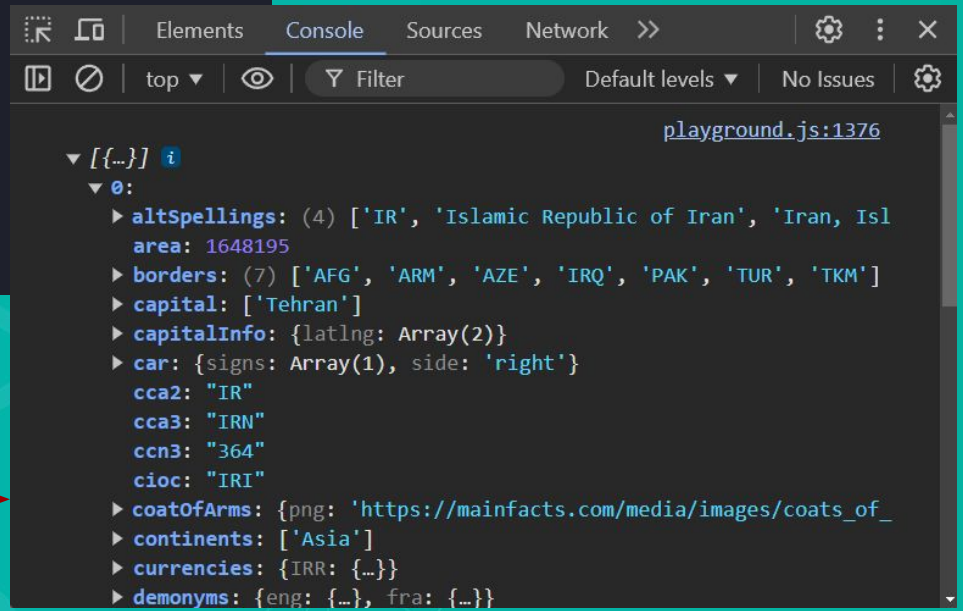
AJAX Request using XMLHttpRequest

```
1  const xhr = new XMLHttpRequest();
2
3  xhr.open("GET", "https://restcountries.com/v3.1/alpha/ir");
4
5  xhr.addEventListener("load", function () {
6    console.log(xhr.responseText);
7  });
8
9  xhr.send();
```



JSON.parse()

```
1  const xhr = new XMLHttpRequest();
2
3  xhr.open("GET", "https://restcountries.com/v3.1/alpha/ir");
4
5  xhr.addEventListener("load", function () {
6    const data = JSON.parse(xhr.responseText);
7    console.log(data);
8  });
9
10 xhr.send();
```



Challenge: Country List

Callback Hell (Example from Country List Challenge)

```
function countryCardGenerator(countryName) {
  //! request(1) -> country data
  const request = new XMLHttpRequest();
  request.open("GET", `https://restcountries.com/v3.1/name/${countryName}`);
  request.send();

  request.addEventListener("load", function () {
    const countryData = JSON.parse(request.responseText)[0];
    if (!countryData.currencies || !countryData.capitalInfo.latlng) return;
    const currencyKey = Object.keys(countryData.currencies)[0];
    const languageKey = Object.keys(countryData.languages)[0];

    //! request(2) -> weather data
    const [lat, lon] = countryData.capitalInfo.latlng;
    const request2 = new XMLHttpRequest();
    request2.open(
      "GET",
      `https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&units=metric&appid=${WEATHER_API_KEY}`
    );
    request2.send();
    request2.addEventListener("load", function () {
      const weatherData = JSON.parse(request2.responseText);
      const countryCard = `...
      `;
      countryListEl.insertAdjacentHTML("afterbegin", countryCard);
    });
  });
}
```

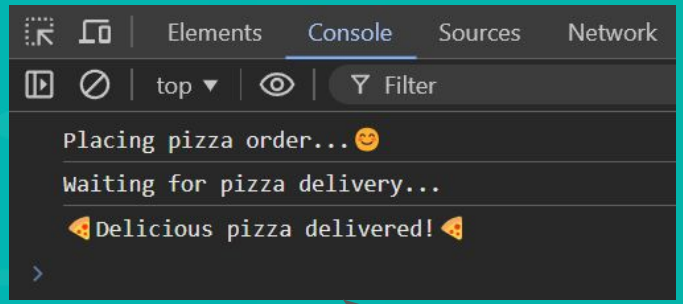
Callback Hell with setTimeout()

```
setTimeout(() => {
  console.log("1st operation completed");
  setTimeout(() => {
    console.log("2nd operation completed");
    setTimeout(() => {
      console.log("3rd operation completed");
      setTimeout(() => {
        console.log("4th operation completed");
        setTimeout(() => {
          console.log("5th operation completed");
        }, 1000);
      }, 1000);
    }, 1000);
  }, 1000);
}, 1000);
```

Promise

```
let myPromise = new Promise((resolve, reject) => {
  // Asynchronous operation
  let success = false; // Simulate success or failure
  if (success) {
    resolve("Operation succeeded!");
  } else {
    reject("Operation failed.");
  }
});

myPromise
  .then((result) => {
    // Operation succeeded!
    console.log(result);
  })
  .catch((error) => {
    // Operation failed.
    console.error(error);
  })
  .finally(() => {
    console.log("Operation completed");
  });
```

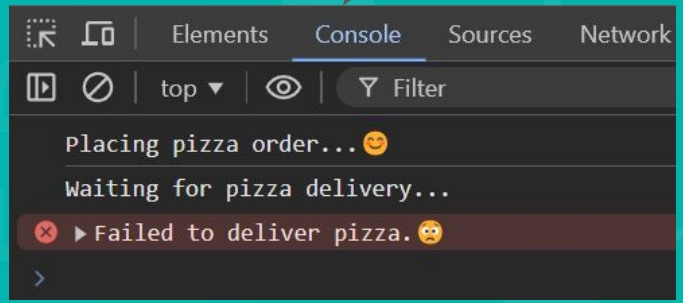
Promise - Pizza example



fulfilled

pending…

rejected




```
let orderPizza = new Promise((resolve, reject) => {
  console.log("Placing pizza order...😊");

  // Simulate pizza preparation
  setTimeout(() => {
    if (Math.random() >= 0.5) {
      resolve("🍕Delicious pizza delivered!🍕");
    } else {
      reject("Failed to deliver pizza.😳");
    }
  }, 3000);
});

orderPizza
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.error(error);
  });

console.log("Waiting for pizza delivery...");
```

Get Rid of Callback Hell Using Promises

```
setTimeout(() => {
  console.log("1st operation completed");
  setTimeout(() => {
    console.log("2nd operation completed");
    setTimeout(() => {
      console.log("3rd operation completed");
      setTimeout(() => {
        console.log("4th operation completed");
        setTimeout(() => {
          console.log("5th operation completed");
        }, 1000);
      }, 1000);
    }, 1000);
  }, 1000);
}, 1000);
```

```
function wait(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}
function start() {
  wait(1000)
    .then(() => {
      console.log("1st operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("2nd operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("3rd operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("4th operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("5th operation completed");
    });
}
start();
```

#async/await

```
function wait(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}
function start() {
  wait(1000)
    .then(() => {
      console.log("1st operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("2nd operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("3rd operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("4th operation completed");
      return wait(1000);
    })
    .then(() => {
      console.log("5th operation completed");
    });
}
start();
```

```
function delay(ms) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

async function start() {
  await delay(1000);
  console.log("1st operation completed");

  await delay(1000);
  console.log("2nd operation completed");

  await delay(1000);
  console.log("3rd operation completed");

  await delay(1000);
  console.log("4th operation completed");

  await delay(1000);
  console.log("5th operation completed");
}

start();
```

try...catch

Fetch API

#An High-Level Overview of JavaScript

- High-Level
- Garbage-Collected
- Interpreted or JIT Compiled
- Multi-Paradigm
- Prototype-Based Object Oriented
- First-Class Functions
- Dynamic
- Single-Threaded
- Non-Blocking Event Loop

#The JavaScript Engine and Runtime