1️⃣ JavaScript Objects - Introduction

Problem Without Objects

```
// Storing one person's data
var age2 = 25
var fname = "ali"
var city1 = "alex"
var grade1 = 3.9

// Another person's data
var age1 = 23
var studName = "mohamed"
var city2 = "cairo"
var grade2 = 3.9
```

Problems:

- ❌ Hard to manage multiple variables
- ❌ No relationship between related data
- ❌ Difficult to pass to functions
- ❌ Naming conflicts

Solution: Using Objects

```
var person = {
    fname: "ali",
    age: 23,
    grade: 3.8,
    city: "alex",
    isStudent: true
}

console.log(typeof person)  // "object"
```

Benefits:

- ✅ Groups related data together
- ✅ Easy to manage

- ✅ Easy to pass around
- ✅ Clear structure

Reference: W3Schools - JS Objects

2️⃣ Object Structure & Syntax

Basic Syntax

```
var objectName = {
    property1: value1,
    property2: value2,
    property3: value3
}
```

Key-Value Pairs:

- **Key (Property)**: The name (e.g., `fname`, `age`)
- **Value**: The data (e.g., `"ali"`, `23`)
- Separated by **colon** `:`
- Multiple properties separated by **comma** `,`

Object with Different Data Types

```
var person = {
    fname: "ali",           // String
    age: 23,                // Number
    grade: {                // Nested Object
        science: 35,
        English: 20
    },
    city: "alex",           // String
    isStudent: true,        // Boolean
    colors: ["red", "green", "yellow"],  // Array
    display: function() {   // Function (Method)
        console.log("Display Function")
    }
}
```

Accessing Different Types:

```
console.log(person.fname)              // "ali"
console.log(person.age)                // 23
console.log(person.grade.science)      // 35 (nested object)
console.log(person.colors[1])          // "green" (array element)
person.display()                       // Calls function
```

3️⃣ Accessing Object Properties

Method 1: Dot Notation

```
var person = {
    fname: "ali",
    age: 23,
    city: "alex"
}

console.log(person.fname)   // "ali"
console.log(person.age)     // 23
console.log(person.city)    // "alex"
```

Advantages:

- ✅ Clean and readable
- ✅ Most common method
- ✅ Easy to type

Limitations:

- ❌ Cannot use variables
- ❌ Cannot use property names with spaces or special characters

---

Method 2: Bracket Notation (Subscript)

```
var person = {
    fname: "ali",
    age: 23,
    city: "alex"
```

```
}

console.log(person["fname"])  // "ali"
console.log(person["age"])    // 23

// Using variables
var myKey = "age"
console.log(person[myKey])    // 23 ✅ Works!
console.log(person["myKey"])  // undefined (looks for property named "myKey")
console.log(person.myKey)     // undefined
```

Important Difference:

```
var myKey = "age"

person[myKey]       // ✅ Looks for person["age"] = 23
person["myKey"]     // ❌ Looks for person["myKey"] = undefined
person.myKey        // ❌ Looks for person.myKey = undefined
```

Advantages:

- ✅ Can use variables
- ✅ Can use property names with spaces
- ✅ Can use dynamically generated property names

Reference: W3Schools - Object Properties

Comparison Table

Feature	Dot Notation	Bracket Notation
Syntax	`person.fname`	`person["fname"]`
Use Variables	❌ No	✅ Yes
Spaces in Name	❌ No	✅ Yes
Dynamic Access	❌ No	✅ Yes
Readability	✅ Better	⚠️ Less readable

4️⃣ Reading and Writing Properties

Reading (Get)

```
var person = {
    fname: "ali",
    age: 23,
    city: "alex"
}

console.log(person.age)  // Get: 23
```

Writing (Set)

```
// Modify existing property
person.age = 12
console.log(person.age)  // 12

// Add new property
console.log(person.grade)  // undefined (doesn't exist)
person.grade = 3.5         // Create new property
console.log(person.grade)  // 3.5

// Add with bracket notation
person["address"] = "Menofia"
console.log(person.address)  // "Menofia"
```

Visual Flow:

```
Initial Object:
{
    fname: "ali",
    age: 23,
    city: "alex"
}

After person.age = 12:
{
    fname: "ali",
    age: 12,        ← Modified
    city: "alex"
```

```
}

After person.grade = 3.5:
{
    fname: "ali",
    age: 12,
    city: "alex",
    grade: 3.5      ← Added
}
```

5️⃣ The `this` Keyword

What is `this`?

`this` refers to the **current object** inside a method.

```
var person = {
    fname: "ali",
    age: 23,
    city: "alex",
    display: function() {
        console.log(`${this.fname} - ${this.city} - ${this.age}`)
    }
}

person.display()  // "ali - alex - 23"
```

Why use `this`?

```
// ❌ Without this (won't work)
display: function() {
    console.log(`${fname} - ${city} - ${age}`)  // ReferenceError!
}

// ✅ With this (works)
display: function() {
    console.log(`${this.fname} - ${this.city} - ${this.age}`)
}
```

Reference: W3Schools - this Keyword

Template Literals

```
// Old way (concatenation)
console.log(person.fname + " - " + person.city + " - " + person.age)

// New way (template literals)
console.log(`${person.fname} - ${person.city} - ${person.age}`)
```

Syntax:

- Use **backticks** `` ` `` (not quotes)
- Embed variables with `${variable}`

Advantages:

- ✅ More readable
- ✅ Multiline strings
- ✅ Cleaner syntax

Reference: W3Schools - Template Literals

---

6 Object Methods

Method: Function Inside Object

```
var product = {
    name: "mobile",
    price: 2000,
    discount: function() {
        var newPrice = this.price - (this.price * 0.20)
        console.log(`Price after discount: ${newPrice}`)
    }
}

product.discount()  // "Price after discount: 1600"
```

Breakdown:

```
this.price           // 2000
this.price * 0.20    // 400 (20% discount)
this.price - 400     // 1600 (final price)
```

7️⃣ Object.keys() - Iterating Over Objects

Getting All Keys

```
var product = {
    name: "mobile",
    price: 2000,
    grade: {
        science: 34,
        English: 20
    }
}

var keys = Object.keys(product)
console.log(keys)  // ["name", "price", "grade"]
```

Reference: W3Schools - Object.keys()

Looping Through Object Properties

```
var product = {
    name: "mobile",
    price: 2000
}

var keys = Object.keys(product)

for(var i = 0; i < keys.length; i++) {
    var key = keys[i]           // "name", then "price"
    var value = product[key]    // "mobile", then 2000
    console.log(`${key} = ${value}`)
}

// Output:
// name = mobile
// price = 2000
```

Visual Flow:

```
Iteration 1:
```

```
├─ i = 0
├─ key = keys[0] = "name"
├─ value = product["name"] = "mobile"
└─ Print: "name = mobile"

Iteration 2:
├─ i = 1
├─ key = keys[1] = "price"
├─ value = product["price"] = 2000
└─ Print: "price = 2000"
```

---

8 Math Object

Math Properties (Constants)

```
console.log(Math.PI)       // 3.141592653589793
console.log(Math.E)        // 2.718281828459045
console.log(Math.SQRT2)    // 1.4142135623730951 (√2)
console.log(Math.SQRT1_2)  // 0.7071067811865476 (√0.5)
```

Reference: W3Schools - Math Object

Math Methods

Square Root

```
console.log(Math.sqrt(25))   // 5
console.log(Math.sqrt(16))   // 4
console.log(Math.sqrt(2))    // 1.4142135623730951
```

Absolute Value

```
console.log(Math.abs(-3))    // 3
console.log(Math.abs(0))     // 0
console.log(Math.abs(3))     // 3
```

What it does:

- Returns the **positive** version of a number
- Distance from zero

Sign Function

```
console.log(Math.sign(-3))   // -1 (negative)
console.log(Math.sign(0))    // 0 (zero)
console.log(Math.sign(3))    // 1 (positive)
```

Returns:

- `-1` if number is negative
- `0` if number is zero
- `1` if number is positive

Power

```
console.log(Math.pow(2, 3))  // 8 (2³ = 2×2×2)
console.log(Math.pow(5, 2))  // 25 (5² = 5×5)
console.log(Math.pow(2, 3, 7))  // 8 (extra parameters ignored)
```

Modern Alternative:

```
console.log(2 ** 3)  // 8 (ES6 exponentiation operator)
```

Floor and Ceil

```
console.log(Math.floor(2.9))   // 2 (rounds down)
console.log(Math.floor(2.1))   // 2
console.log(Math.ceil(2.05))   // 3 (rounds up)
console.log(Math.ceil(2.9))    // 3
```

Visualization:

```
Number Line:
    2.0     2.5     3.0
    ↓       ↓       ↓
floor(2.9) = 2  (rounds to left)
ceil(2.05) = 3  (rounds to right)
```

Reference: W3Schools - Math Methods

Random Numbers

```
// Random number between 0 and 1
console.log(Math.random())  // e.g., 0.7234567

// Random number between 0 and 10
console.log(Math.random() * 10)  // e.g., 7.234567

// Random INTEGER between 0 and 10
console.log(Math.floor(Math.random() * 10))  // e.g., 7
```

Random Number in Range Formula

```
// Random integer between min and max (inclusive)
Math.floor(Math.random() * (max - min + 1) + min)

// Example: Random between 5 and 10
console.log(Math.floor(Math.random() * (10 - 5 + 1) + 5))
```

Step-by-step Breakdown:

```
// Goal: Random integer between 5 and 10

Step 1: Math.random()           // 0.0 to 0.999...
Step 2: * (10 - 5 + 1)          // 0.0 to 5.999...
Step 3: + 5                     // 5.0 to 10.999...
Step 4: Math.floor()            // 5, 6, 7, 8, 9, or 10
```

Reference: W3Schools - Math.random()

Trigonometric Functions

```
console.log(Math.sin(90))   // 0.8939966636005579 (expects radians!)
console.log(Math.cos(0))    // 1
console.log(Math.tan(45))   // 1.6197751905438615
```

⚠ **Important:**

- JavaScript expects **radians**, not degrees
- To convert: `radians = degrees × (Math.PI / 180)`

```
var degrees = 90
var radians = degrees * (Math.PI / 180)
console.log(Math.sin(radians))  // 1 (correct!)
```

---

9 DOM - Document Object Model

What is DOM?

The DOM is a **tree structure** representing the HTML document.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document</title>
  </head>
  <body>
    <div>
      <h1>Hello Pd Track</h1>
    </div>
    <div>
      <h2>Welcome Pd Track</h2>
      <p>Lorem ipsum...</p>
    </div>
    <button>Click Me</button>
  </body>
</html>
```

DOM Tree Visualization:

```
document
    │
    └── html
        ├── head
        │   └── title
        │       └── "Document"
        │
        └── body
            ├── div
            │   └── h1
            │       └── "Hello Pd Track"
            │
            ├── div
            │   ├── h2
            │   │   └── "Welcome Pd Track"
            │   └── p
            │       └── "Lorem ipsum..."
            │
            └── button
                └── "Click Me"
```

Reference: W3Schools - HTML DOM

Accessing the Document

```
console.log(document)
```

What you'll see:
```
#document
├── doctype: html
├── html
│   ├── head
│   │   └── title
│   └── body
│       ├── div
│       └── ...
```

The `document` object contains:

- All HTML elements

- Methods to access/modify elements
- Properties about the page

Common DOM Properties

```
console.log(document.title)        // "Document"
console.log(document.URL)          // Current page URL
console.log(document.domain)       // Domain name
console.log(document.body)         // <body> element
console.log(document.head)         // <head> element
```

Reference: W3Schools - DOM Document

🔟 Object Types Summary

1. User-Defined Objects

```
var person = {
    fname: "ali",
    age: 23
}
```

- Created by the programmer
- Custom properties and methods

---

2. Built-in Objects

```
// Math Object
Math.sqrt(25)

// Date Object
var today = new Date()

// Array Object
var arr = [1, 2, 3]
```

```
// String Object
var str = "hello"
```

- Provided by JavaScript
- Ready to use

📊 Summary Tables

Object Access Methods

Method	Syntax	Use Variables?	Dynamic?
Dot	`obj.prop`	❌ No	❌ No
Bracket	`obj["prop"]`	✅ Yes	✅ Yes

Math Methods

Method	Purpose	Example
`Math.sqrt()`	Square root	`Math.sqrt(25)` → `5`
`Math.abs()`	Absolute value	`Math.abs(-3)` → `3`
`Math.floor()`	Round down	`Math.floor(2.9)` → `2`
`Math.ceil()`	Round up	`Math.ceil(2.1)` → `3`
`Math.random()`	Random 0-1	`Math.random()` → `0.7234`

BY. Abdullah Ali

Contact : +201012613453