10 – Arrays and Objects

In JavaScript, Arrays and Objects are two of the most fundamental data types. They both store a collection of data, but differ in how it is stored. Arrays store data sequentially at non-negative indexed positions. Objects store data in the form of key-value pairs.

**Arrays**

In JavaScript, Arrays aren’t primitives but are instead Array objects having the following core characteristics:

* Arrays are resizable and can contain a mix of different data types in JS.
* Arrays are zero-indexed.
* All array-copy operations create shallow copies.

There are three ways to initialize an array in JavaScript:  
  
1) Array Literal (Preferred)

let fruits = [“banana”, “blueberry”, “kiwi”]

2) Array Constructor (Declaration)

Let fruits = new Array(3)

fruits = [“banana”, “blueberry”, “kiwi”]

3) Array Constructor (Declaration and Initialization)

let fruits = new Array(“banana”, “blueberry”, “kiwi”)

When adding an item to a JavaScript array, if the item’s index at which it is added is a valid array index but is out of bounds of the current array length, the array’s length property will update accordingly:

let fruits = [“banana”, “blueberry”, “kiwi”]

fruits[6] = "mango";

console.log(fruits[6]); // 'mango'

console.log(Object.keys(fruits)); // ['0', '1', '2', '6']

console.log(fruits.length); // 7

**Objects**

An object is a collection of data stored in the form of key-value pairs. Each key must be of the type String, and its value can be any data type, such as String, Numbers, Boolean, Arrays, or Objects. Objects are best suited for representing real-world entities as data structures.

let person = {

name: “Shaurya”

age: 21

isStudent: true

skills: ["HTML”, “CSS”, “JS”]

address: {

city: “San Ramon”

state: “CA”

Pin: 93482

}

}

**Accessing and Modifying**

Dot notation 🡪 console.log(person.name)

Bracket notation 🡪 console.log(person["full name”])

**Stored by Reference**

Memory can be divided into two parts: the Stack and the Heap. The stack is where the file's Global Execution Context is created and where the actual program runs. However, a limitation of memory storage in the stack is that it cannot store dynamically changing data types, i.e., Non-Primitive data structures.

Thus, Arrays and Objects are stored in the Heap memory, and have a reference variable pointing towards them from the stack. Such a variable is known as a pointer. Hence, two different pointer variables for the same array/object will be pointing to the same entity in the heap memory. Changes in either will lead to alterations in the original.

**Looping Over Arrays**

1) For Loop

const fruits = ["apple", "banana", "mango"];

for (let i = 0; i < fruits.length; i++) {

console.log(fruits[i]);

}

2) For…Of

for (let fruit of fruits) {

console.log(fruit);

}

3) For-Each

Executes a callback for each element in the array.

fruits.forEach((fruit, index) => {

console.log(`${index}: ${fruit}`);

});

4) while/do…while

let i = 0;

while (i < fruits.length) {

console.log(fruits[i]);

i++;

}

**Looping Over Objects**

1) For…In

for (let key in person) {

console.log(`${key}: ${person[key]}`);

}

Note: Picks up inherited properties too (Defined in the “\_\_proto\_\_” of the object)

2) Object.keys() + forEach()

Iterates only its own property keys (Not inherited ones).

Object.keys(person).forEach(key => {

console.log(`${key}: ${person[key]}`);

});

3) Object.entries + For…Of

Loops over key-value pairs.

for (let [key, value] of Object.entries(person)) {

console.log(`${key}: ${value}`);

}

4) Object.values +forEach()

Loops over just the values.

Object.values(person).forEach(value => {

console.log(value);

});