DSA in JavaScript

JavaScript Objects: -

In real life, objects are things like houses, cars, people, animals, etc...

Object Properties: -

car.name = Fiat, car.model = 500, car.weight = 850kg, car.color = white.

Object Methods: -

car.start(), car.drive(), car.brake(), car.stop().

Defining a JavaScript Object: -

An **object literal** is a list of key:value pairs inside curly braces **{}**.

Const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"}

The below example creates an empty JavaScript object and then adds 4 properties.

```
// Create an Object

const person = {};

// Add Properties

person.firstName = "John";

person.lastName = "Doe";

person.age = 50;

person.eyeColor = "blue";
```

Using the new Keyword: -

```
// Create an Object

const person = new Object();

// Add Properties

person.firstName = "John";

person.lastName = "Doe";

person.age = 50;

person.eyeColor = "blue";
```

Objects written as name value pairs are similar to Associative arrays in PHP, Dictionaries in Python, Hash tables in C, Hash maps in Java.

Accessing Object Properties: -

```
objectName.propertyName

objectName["propertyName"]
```

JavaScript Object Methods: -

Methods are actions that can be performed on objects.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

this refers to the person object.

Note: -

In JavaScript, almost "everything" is an object.

Objects are objects, Maths are objects, Functions are objects, Dates are objects, Arrays are objects, Maps are objects, Sets are objects

All JavaScript values, except primitives, are objects.

Primitive values are immutable. Strings are also immutable. **JavaScript Objects are Mutable**. They are addressed by reference, not by value.

If person is an object, the following statement will not create a copy of person

```
const x = person;
```

The object x is not a copy of person. The object x is person. The object x and the object person share the same memory address. Any changes to x will also change person.

Adding New Properties: -

```
person.nationality = "English";
```

Deleting Properties: -

delete person.age; or delete person["age"];

Nested Objects: -

```
myObj = {
  name:"John",
  age:30,
  myCars: {
    car1:"Ford",
    car2:"BMW",
    car3:"Fiat"
  }
}
```

Access can be done by myObj.myCars.car2 or myObj.myCars["car2"] or myObj["myCars"]["car2"]

```
let p1 = "myCars";
let p2 = "car2";
myObj[p1][p2];
```

If you invoke the fullName property with (), it will execute as a function.

```
name = person.fullName(); //John Doe
```

If you access the fullName property without (), it will return the function definition.

```
name = person.fullName; // function() { return this.firstName + " " + this.lastName; }
```

Adding a Method to an Object: -

```
person.name = function () {
  return this.firstName + " " + this.lastName;
};
```

JavaScript Display Objects: -

```
const person = {
  name: "John",
  age: 30,
  city: "New York"
};
```

The properties of an object can be displayed as a **string**.

```
document.getElementById("demo").innerHTML = person.name + "," + person.age + "," +
person.city;
```

The properties of an object can be collected in a **loop.**

```
let text = "";
for (let x in person) {
  text += person[x] + " ";
};
```

Object.values() creates an array from the property values.

```
const myArray = Object.values(person);
```

Using Object.entries(): -

```
const fruits = {Bananas:300, Oranges:200, Apples:500};
let text = "";
for (let [fruit, value] of Object.entries(fruits)) {
  text += fruit + ": " + value + "<br>";
}
```

JSON.stringify(): -

JavaScript objects can be converted to a string with JSON method JSON.stringify()

```
let myString = JSON.stringify(person);
{"name":"John","age":50,"city":"New York"}
```

JavaScript Object Constructors: -

```
// Constructor Function for Person objects
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

We can use new Person() to create many new Person objects

```
// Create a Person object
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
const mySister = new Person("Anna", "Rally", 18, "green");
```

Also , myFather.nationality = "English";

The new property will be added to myFather. Not to any other Person Objects.

Adding a Property to a Constructor: -

You cannot add a new property to an object constructor

```
Person.nationality = "English"; // wrong
```

To add a new property, you must add it to the constructor function prototype.

```
Person.prototype.nationality = "English";
```

A constructor function can also have methods.

The new method will be added to myMother. Not to any other Person Objects.

```
myMother.changeName = function (name) {
  this.lastName = name;
}
```

Adding a Method to a Constructor: -

```
Person.changeName = function (name) {
  this.lastName = name;
}
myMother.changeName("Doe"); //  TypeError: myMother.changeName is not a function
```

```
Person.prototype.changeName = function (name) {
  this.lastName = name;
}
myMother.changeName("Doe"); // this will work
```

Built-in JavaScript Constructors: -

```
new Object()   // A new Object object
new Array()    // A new Array object
new Map()      // A new Map object
new Set()      // A new Set object
new Date()     // A new Date object
new RegExp()   // A new RegExp object
new Function() // A new Function object
```

JavaScript Object Methods: -

General Methods: -

// Copies properties from a source object to a target object
Object.assign(target, source)

```
// Create Target Object

const person1 = {

  firstName: "John",

  lastName: "Doe",

  age: 50,

  eyeColor: "blue"

};

// Create Source Object
```

const person2 = {firstName: "Anne",lastName: "Smith"};

// Assign Source to Target

Object.assign(person1, person2);

// Creates an object from an existing object
Object.create(object)

// Returns an array of the key/value pairs of an object
Object.entries(object)

// Creates an object from a list of keys/values
Object.fromEntries()

```
const fruits = [
  ["apples", 300],
  ["pears", 900],
  ["bananas", 500]
];
const myObj = Object.fromEntries(fruits);
```

// Returns an array of the keys of an object
Object.keys(object)

// Returns an array of the property values of an object
Object.values(object)

// Groups object elements according to a function
Object.groupBy(object, callback)

```
// Create an Array
const fruits = [
  {name:"apples", quantity:300},
  {name:"bananas", quantity:500},
  {name:"oranges", quantity:200},
  {name:"kiwi", quantity:150}
];
// Callback function to Group Elements
```

```
function myCallback({ quantity }) {
  return quantity > 200 ? "ok" : "low";
}
// Group by Quantity
const result = Object.groupBy(fruits, myCallback);
```