Hema Coding School

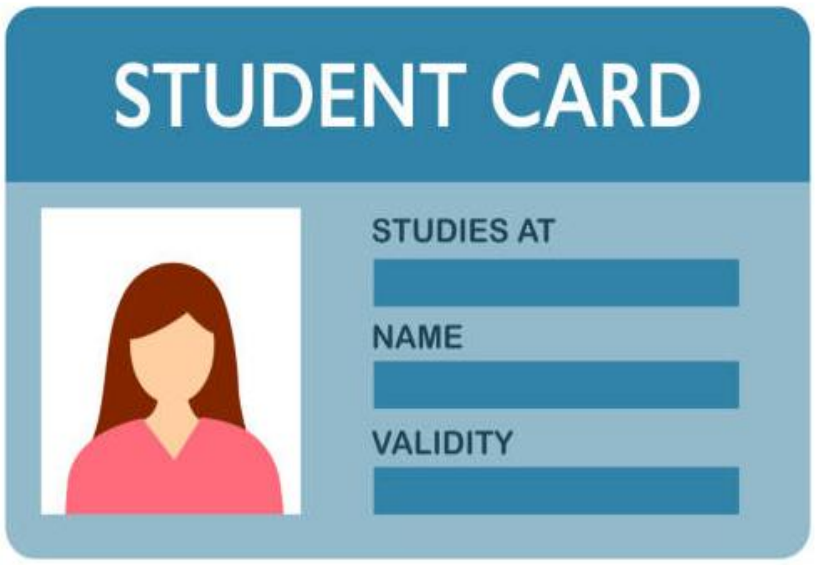
YouTube Link: https://www.youtube.com/@HemaCodingSchool

Object

What is an Object?

Object Literals: An object literal is a way to define an object directly in JavaScript code using curly braces {}.

In JavaScript, an object is a collection of key-value pairs where each key is a string (or symbol) and each value can be any data type, including other objects. Objects are used to store and organize data in a structured way.



While arrays and objects can both store and organize data, they serve different purposes and are suited for different scenarios.

Arrays vs. Objects:

1. **Arrays**:
   - Ideal for storing lists of homogeneous data (e.g., a list of numbers, strings, or objects of the same type).
   - Accessed using numeric indices (0-based).
   - Use when the order of elements matters, such as when dealing with sequential data or collections.
   - Useful for performing operations like iteration, sorting, filtering, and mapping over elements.
2. **Objects**:
   - Ideal for storing collections of heterogeneous data (e.g., key-value pairs representing properties and values).
   - Accessed using keys (property names), which can be strings or symbols.
   - Use when you need to associate unique identifiers (keys) with specific values or when organizing data into structured entities.

- Useful for representing real-world entities (e.g., users, products, contacts) where each entity has distinct properties.

Creation Object:

Understanding the Object Constructor and the new Keyword in JavaScript

In JavaScript, the Object constructor and the new keyword are used to create new object instances. Let's break down what each part does and how they work together.

The Object Constructor

The Object constructor is a built-in JavaScript function that is used to create a new object. When called with the new keyword, it creates an instance of the Object type.

```
var person = new Object();
```

Explanation:

- Object is a constructor function provided by JavaScript to create objects.
- When you use new Object(), it creates an empty object similar to using the object literal {}.

The new Keyword

The new keyword is used to create an instance of an object that has a constructor function. When new is used with a function, it performs the following steps:

1. **Create a New Object**: A new, empty object is created.
2. **Set the Prototype**: The new object's internal [[Prototype]] property is set to the constructor function's prototype property.
3. **Return the New Object**: The new object is returned and assigned to the variable obj.

Creating Objects with Custom Constructor Functions

In addition to using Object as a constructor, you can define your own constructor functions to create objects with specific properties and methods.

```
var student = {
   name:"Hema",
   age:21,
   isStudent:true,
   studentAge:23
}

student.course = "FSWD"
student.age = 22
delete student.isStudent;

student["course"] = "FSWD";
student["age"] = 23;
delete student["isStudent"];
```

- **Dot Notation** is more concise and often used when the property names are known and static.
- **Bracket Notation** is more flexible and versatile, allowing you to access properties dynamically or with non-standard names.

1. **Object.keys(obj)**: Returns an array of a given object's own enumerable property names (keys).

```
// Using Object.keys() to get the keys of the object
var keys = Object.keys(obj);
console.log(keys); // Output: ["name", "age", "email"]
```

2. **Object.values(obj)**: Returns an array of a given object's own enumerable property values.

```
// Using Object.values() to get the values of the object
var values = Object.values(obj);
console.log(values); // Output: ["Mahesh", 30, "Mahesh@example.com"]
```

3. **Object.entries(obj)**: Returns an array of a given object's own enumerable property key-value pairs as arrays.

```
// Using Object.entries() to get key-value pairs of the object
var entries = Object.entries(obj);
console.log(entries); // Output: [["name", "Mahesh"], ["age", 30], ["email",
"Mahesh@example.com"]]
```

4. **Object.assign(target, ...sources)**: Copies enumerable own properties from one or more source objects to a target object.

```
// Using Object.assign() to merge objects
var newObj = { city: 'Hyderabad' };
Object.assign(newObj, obj);
console.log(newObj); // Output: { city: 'Hyderabad', name: 'Mahesh', age: 30, email:
'Mahesh@example.com' }
```

1. **Descriptive Keys**: Objects use descriptive keys (property names) instead of numeric indices, making it easier to understand and access data based on meaningful identifiers. This overcomes the confusion and potential errors that can arise when working with arrays indexed by numbers.
2. **Structured Data**: Objects allow for the storage of structured data with properties and values, mimicking real-world entities more accurately. This structure is essential for organizing complex data and representing entities such as users, products, or contacts with their respective attributes.
3. **Key-Value Flexibility**: Objects offer flexibility in associating unique keys (properties) with specific values, enabling efficient data retrieval and manipulation. This flexibility is especially valuable when dealing with data that requires non-sequential access or varying types of information.
4. **Dynamic Properties**: Objects in JavaScript are dynamic, allowing properties to be added, modified, or removed dynamically. This dynamic nature makes objects more adaptable to changing data requirements compared to the fixed structure of arrays.
5. **Object-Oriented Programming (OOP)**: Objects are fundamental to object-oriented programming principles, facilitating encapsulation, inheritance, and polymorphism. They enable the creation of reusable and modular code structures, enhancing code organization and maintainability.

```
// Using Arrays: Limitations and Problems
var personArray = ['John Doe', 30, 'john.doe@example.com'];
console.log(personArray[0]);  // Accessing data using numeric indices can be less descriptive

// Using Objects: Overcoming Limitations
var personObject = {
   name: 'John Doe',
   age: 30,
   email: 'john.doe@example.com'
};
console.log(personObject.name);  // Accessing data using descriptive keys is more intuitive

// Adding Dynamic Properties to an Object
personObject.address = '123 Main St';  // Dynamically adding a property to an object
console.log(personObject);  // Outputs: { name: 'John Doe', age: 30, email:
'john.doe@example.com', address: '123 Main St' }
```

```
// Object-Oriented Programming (OOP) Principles with Objects
class Person {
    constructor(name, age, email) {
        this.name = name;
        this.age = age;
        this.email = email;
    }

    greet() {
        console.log(`Hello, my name is ${this.name}.`);
    }
}

var newPerson = new Person('Jane Smith', 25, 'jane.smith@example.com');
console.log(newPerson);  // Outputs: Person { name: 'Jane Smith', age: 25, email: 
'jane.smith@example.com' }
newPerson.greet();  // Outputs: Hello, my name is Jane Smith.
```

for...in loop:

- Iterates over the enumerable properties of an object.
- Allows you to access each property name (key) in the object.

```
for (var key in student) {
    console.log( key + ': ' + student[key]);
}
```

Nested Object:

```
// nested object
var person = {
    firstName: 'Mahesh',
    lastName: 'H',
    age: 30,
    address: {
        street: 'Mahima Street',
        city: 'Hyd',
        country: 'India'
    },
    contacts: {
        email: 'Mahesh@example.com',
        phone: '123-456-7890'
    }
};
```

```
console.log(person.address.city); // Output: Anytown
console.log(person['contacts']['email']); // Output: john.doe@example.com
```

Tasks:

Task 1: Object Keys Create an object called **student** with properties for name, age, and grade. Use the **Object.keys()** method to get an array of the keys in the object and log them to the console.

```
// 1
let student = {
   name: 'Alice',
   age: 18,
   grade: '12th'
};

let keys = Object.keys(student);
console.log(keys); // Output: ['name', 'age', 'grade']
```

Task 2: Object Values Using the same **student** object from Task 1, use the **Object.values()** method to get an array of the values in the object and log them to the console.

```
//2
var values = Object.values(student);
console.log(values); // Output: ['Alice', 18, '12th']
```

Task 3: Object Entries Create an object called **car** with properties for make, model, and year. Use the **Object.entries()** method to get an array of arrays, each containing a key-value pair, and log them to the console.

```
//3
var car = {
   make: 'Toyota',
   model: 'Camry',
   year: 2022
};

var entries = Object.entries(car);
console.log(entries); // Output: [['make', 'Toyota'], ['model', 'Camry'], ['year', 2022]]
```

Task 4: Object Assignment Create two objects, **person1** and **person2**, each with properties for name and age. Use the **Object.assign()** method to copy the properties from **person1** to **person2** and log the updated **person2** object.

```
//4
var person1 = { name: 'Alice', age: 25 };
var person2 = { name: 'Bob', age: 30 };

Object.assign(person2, person1);
console.log(person2); // Output: { name: 'Alice', age: 25 }
```

Task 5: Looping through Object Create an object called **fruitBasket** with properties for various fruits and their quantities. Use a **for...in** loop to iterate through the object and log each fruit and its quantity to the console.

```
//5
var fruitBasket = {
   apple: 5,
   banana: 3,
   orange: 7
};

for (var fruit in fruitBasket) {
   console.log(`${fruit}: ${fruitBasket[fruit]}`);
}
// Output:
// apple: 5
// banana: 3
// orange: 7
```

Task 2: Library Catalog Create an object called **libraryCatalog** that represents a library catalog with books as items. Each book should have properties like title, author, genre, and availability. Write a function called **searchBook** that takes the **libraryCatalog** object and a book title as parameters and checks if the book is available in the catalog.

```
//Task 2
var libraryCatalog = {
  books: [
    {
      title: "The Great Gatsby",
      author: "F. Scott Fitzgerald",
      genre: "Classic",
      available: true,
    },
    {
      title: "To Kill a Mockingbird",
      author: "Harper Lee",
      genre: "Fiction",
      available: false,
    },
    {
      title: "1984",
      author: "George Orwell",
      genre: "Dystopian",
      available: true,
    },
  ],
};

function searchBook(catalog, title) {
  for (var book of catalog.books) {
    if (book.title === title) {
      return book.available ? "Book is available" : "Book is not available";
    }
  }
  return "Book not found in catalog";
}

console.log(searchBook(libraryCatalog, "To Kill a Mockingbird")); // Output: Book is not available
```

What is an object in JavaScript?

- **Answer**: An object is a collection of key-value pairs where the keys are strings (or Symbols) and the values can be any type of data, including other objects, arrays, functions, and primitives.

How do you create an object in JavaScript?

- **Answer**: An object can be created using object literal syntax, the Object constructor, or a constructor function.

How do you access properties of an object?

- **Answer**: Properties of an object can be accessed using dot notation or bracket notation.

How do you add or modify properties in an object?

- **Answer**: Properties can be added or modified using dot notation or bracket notation

How do you delete a property from an object?

- **Answer**: Properties can be deleted using the delete operator

What is the difference between dot notation and bracket notation when accessing properties?

- **Answer**: Dot notation is simpler and more readable but can only be used with valid JavaScript identifiers. Bracket notation is more flexible and can be used with property names that contain spaces, special characters, or are dynamically determined.

```
let obj = { 'key-name': 'value', 'key with space': 'value' };
console.log(obj['key-name']);       // Output: value
console.log(obj['key with space']);  // Output: value
```

What is the difference between Object.keys(), Object.values(), and Object.entries()?

- **Answer**: Object.keys() returns an array of the object's own enumerable property names. Object.values() returns an array of the object's own enumerable property values. Object.entries() returns an array of the object's own enumerable property [key, value]

How can you merge two objects in JavaScript?

- **Answer**: Objects can be merged using Object.assign() or the spread operator

```
var obj1 = { a: 1, b: 2 };
var obj2 = { b: 3, c: 4 };

// Using Object.assign()
var mergedObj1 = Object.assign({}, obj1, obj2);
console.log(mergedObj1); // Output: { a: 1, b: 3, c: 4 }

// Using the spread operator
var mergedObj2 = { ...obj1, ...obj2 };
console.log(mergedObj2); // Output: { a: 1, b: 3, c: 4 }
```