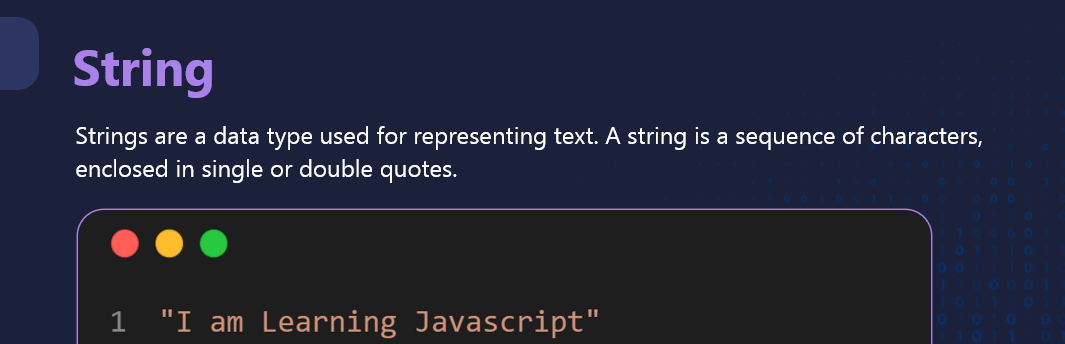
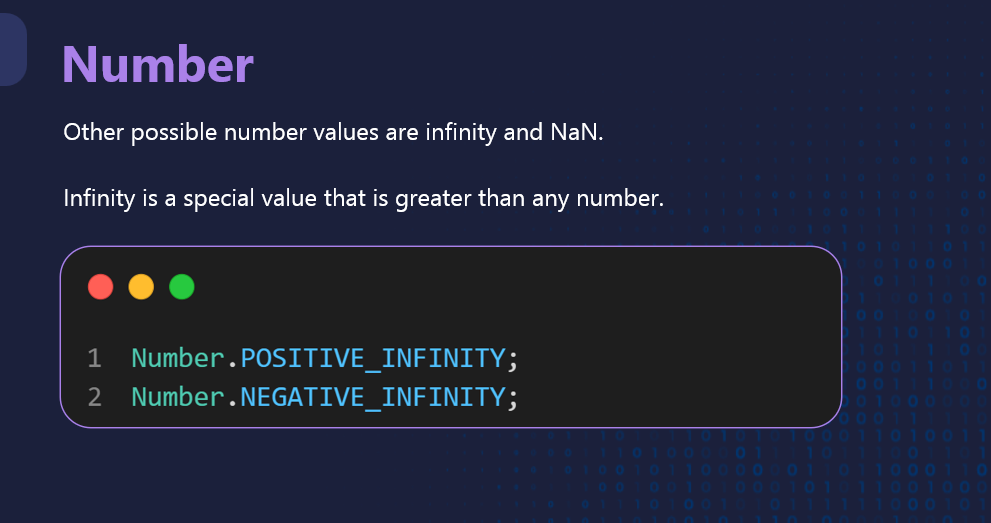
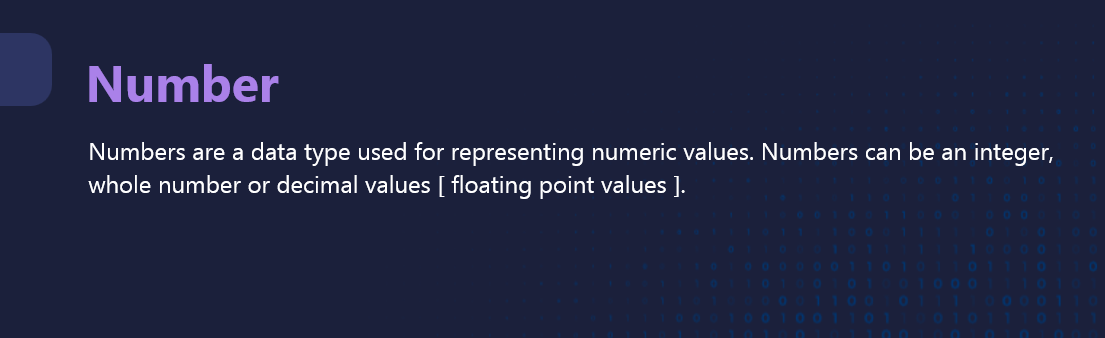


* **JavaScript:**
  + Dynamically typed
  + No need to declare variable types explicitly
  + Type checking happens at runtime
* **Java:**
  + Statically typed
  + Requires explicit declaration of variable types
  + Type checking happens at compile time



**let name = "John";**

**let welcomeMessage = “Hello”+ " Welcome, " + name + "!";**

**Infinity**

Infinity represents a value that is greater than any other number. It can be produced by dividing a number by zero or by performing certain mathematical operations.

javascript

// Positive infinity

let positiveInfinity = 1 / 0; // Infinity

// Negative infinity

let negativeInfinity = -1 / 0; // -Infinity

// Checking for infinity

console.log(positiveInfinity === Infinity); // true

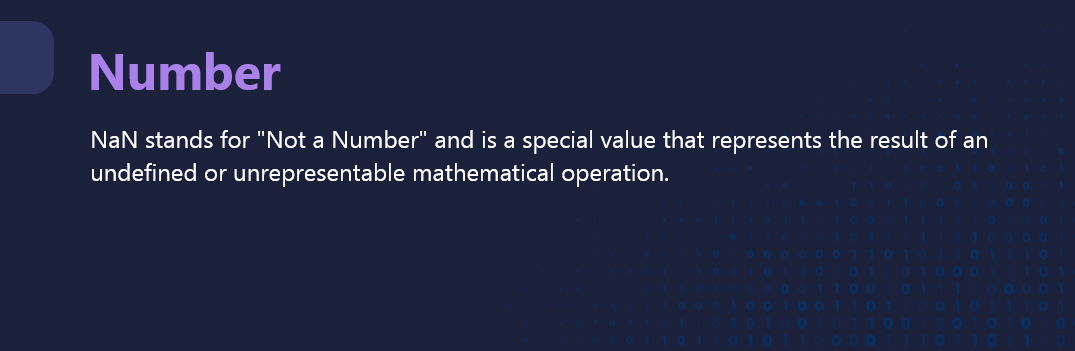
console.log(negativeInfinity === -Infinity); // true

// Operations with infinity

let largeNumber = 1e308;

let infinityResult = largeNumber \* 2; // Infinity

console.log(infinityResult); // Output: Infinity



### **NaN (Not-a-Number)**

NaN is a special value that represents a value that is not a valid number. It can occur when performing arithmetic operations that do not yield a valid number.

javascript

// NaN example

let invalidNumber = parseInt("abc"); // NaN

// Checking for NaN

console.log(isNaN(invalidNumber)); // true

// Operations resulting in NaN

let result = Math.sqrt(-1); // NaN

let invalidOperation = 0 / 0; // NaN

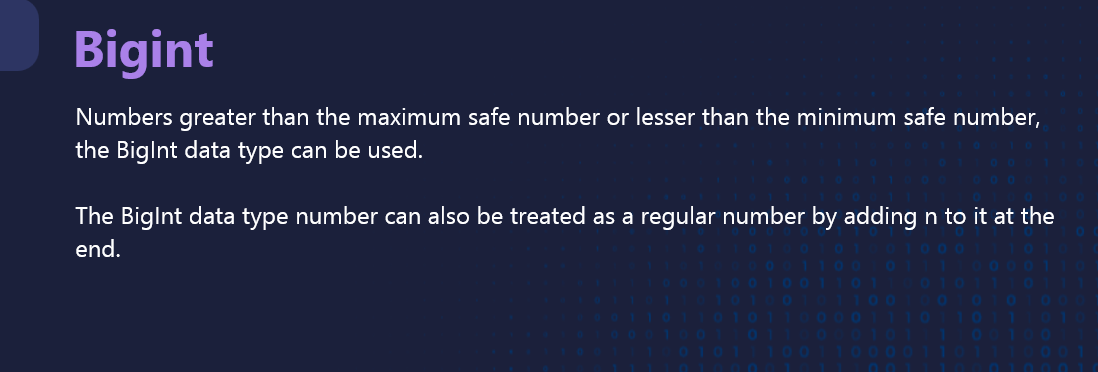
console.log(result); // Output: NaN

console.log(invalidOperation); // Output: NaN

// NaN is not equal to itself

console.log(NaN === NaN); // false

In these examples, Infinity is used to represent values that exceed the largest possible number, while NaN is used to indicate that a value is not a valid number. You can use the isNaN function to check if a value is NaN.



In JavaScript, Number.MAX\_SAFE\_INTEGER is the largest integer that can be safely represented as a Number type, which is 2^53 - 1 (or 9007199254740991). This value is known as the "safe" integer because JavaScript's Number type can accurately represent and compare integers up to this value.

When working with BigInt, you don't need to worry about the safe integer limit, because BigInt can handle integers of arbitrary size without losing precision. Here's an example to illustrate this:

javascript

// Maximum safe integer in JavaScript

let maxSafeInteger = Number.MAX\_SAFE\_INTEGER; // 9007199254740991

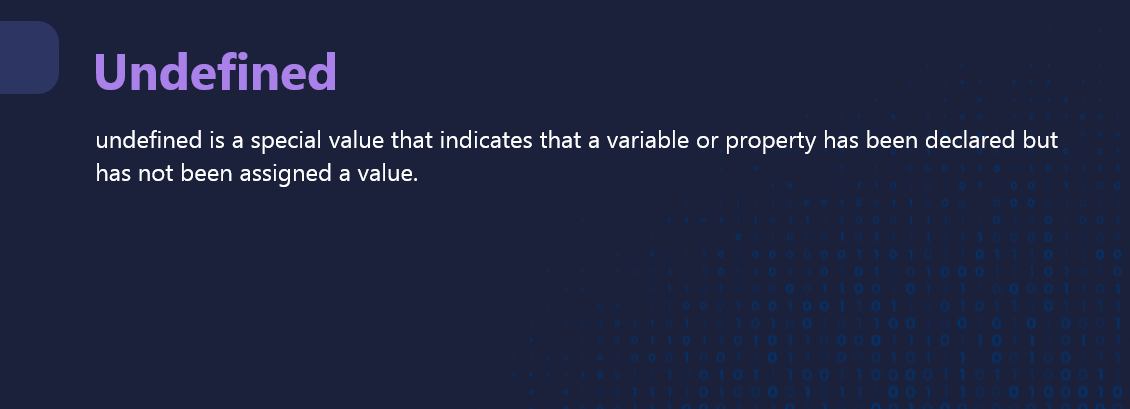
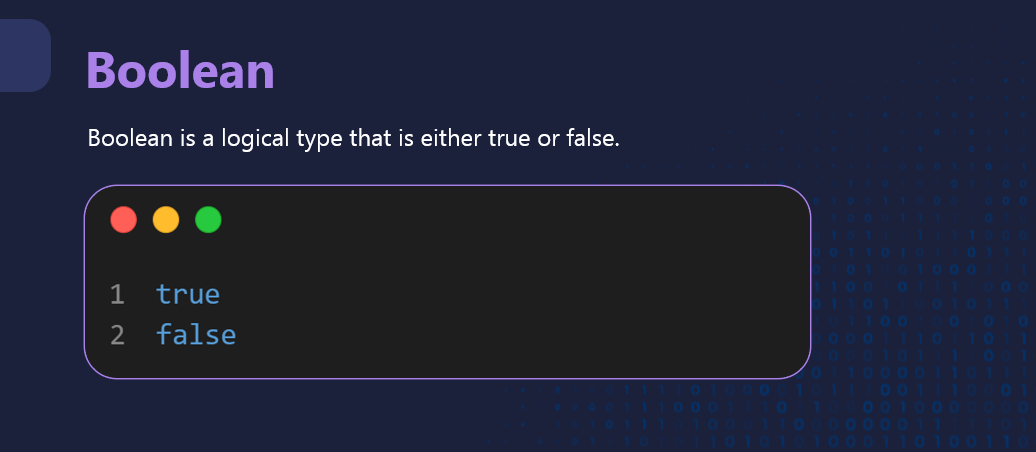
// BigInt beyond the safe integer limit

let bigIntBeyondSafe = BigInt(maxSafeInteger) + 1n; // 9007199254740992n

console.log(maxSafeInteger); // Output: 9007199254740991

console.log(bigIntBeyondSafe); // Output: 9007199254740992n

In this example, maxSafeInteger represents the largest safe integer value, while bigIntBeyondSafe demonstrates a BigInt that can accurately represent values beyond the safe integer limit.



In JavaScript, undefined is a primitive value that represents the absence of a value or the lack of definition. It is commonly encountered in the following situations:

### **1. Variable Declaration without Initialization**

When a variable is declared but not initialized, its value is undefined.

javascript

let myVar;

console.log(myVar); // Output: undefined

### **2. Accessing Non-Existent Object Properties**

When attempting to access an object property that does not exist, the result is undefined.

javascript

let person = { name: "Alice" };

console.log(person.age); // Output: undefined

### **3. Functions without a Return Statement**

If a function does not explicitly return a value, it returns undefined by default.

javascript

function sayHello() {

console.log("Hello!");

}

let result = sayHello(); // Prints "Hello!" to the console

console.log(result); // Output: undefined

### **4. Function Parameters without Arguments**

When a function is called with fewer arguments than it expects, the missing arguments are undefined.

javascript

function greet(name) {

console.log("Hello, " + name);

}

greet(); // Output: "Hello, undefined"

### **5. Array Elements with Empty Slots**

If an array contains empty slots, those slots are undefined.

javascript

let arr = [1, 2, , 4];

console.log(arr[2]); // Output: undefined

### **Checking for undefined**

You can use strict equality (===) to check if a value is undefined.

javascript

let value;

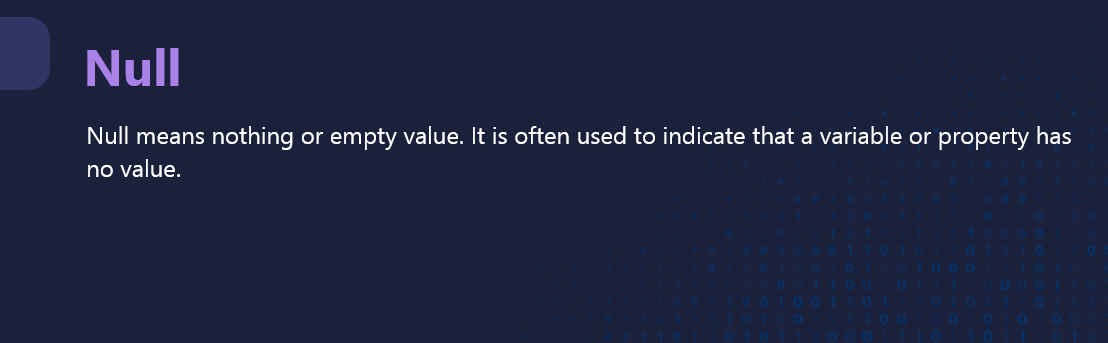
if (value === undefined) {

console.log("The variable is undefined.");

}

### **Important Points**

* undefined is different from null. null represents an intentional absence of value, while undefined indicates that a value has not been assigned or defined.
* Avoid using undefined as a variable name or assigning it directly.



In JavaScript, null is a special value that represents the intentional absence of any object value. It is used to indicate that a variable has been explicitly set to have no value. Here are some common uses and characteristics of null:

### **Assigning null to a Variable**

You can explicitly set a variable to null to indicate that it has no value.

javascript

let myVar = null;

console.log(myVar); // Output: null

### **Checking for null**

You can use strict equality (===) to check if a value is null.

javascript

let value = null;

if (value === null) {

console.log("The variable is null.");

}

### **Differences Between null and undefined**

* null is an assignment value that indicates the intentional absence of any object value.
* undefined indicates that a variable has been declared but not assigned a value.

javascript

let uninitializedVar;

let emptyVar = null;

console.log(uninitializedVar); // Output: undefined

console.log(emptyVar); // Output: null

### **Typical Uses of null**

* Resetting or clearing object references
* Representing a deliberate non-value

javascript

let person = { name: "Alice" };

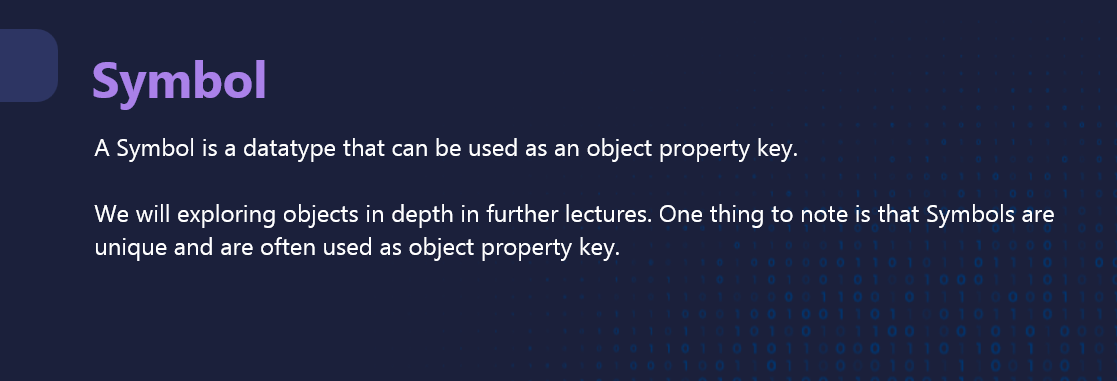
person = null; // Clears the reference to the object

### **Important Points**

* null is considered a falsy value in JavaScript, meaning it evaluates to false in Boolean contexts.
* Using typeof on null will return "object". This is a historical bug in JavaScript.

javascript

console.log(typeof null); // Output: "object"



In JavaScript, a Symbol is a unique and immutable primitive value that can be used as a key for object properties. Symbols are often used to avoid property name collisions in objects, especially when integrating code from different sources. Here's an example to illustrate how Symbol works:

### **Creating Symbols**

You can create a symbol using the Symbol() function. Each time you call Symbol(), you get a unique symbol value.

javascript

// Creating symbols

let sym1 = Symbol();

let sym2 = Symbol("description"); // You can provide an optional description for debugging purposes

// Symbols are unique

console.log(sym1 === sym2); // Output: false

### **Using Symbols as Object Keys**

Symbols can be used as property keys in objects. This allows you to create properties that are not subject to name collisions.

javascript

// Using symbols as object keys

let sym = Symbol("uniqueKey");

let obj = {

[sym]: "Value associated with the symbol",

regularKey: "Regular value"

};

console.log(obj[sym]); // Output: "Value associated with the symbol"

console.log(obj.regularKey); // Output: "Regular value"

### **Symbol.for and Symbol.keyFor**

You can use Symbol.for to create or retrieve a symbol from a global symbol registry. This allows you to share symbols across different parts of your code.

javascript

// Creating or retrieving a symbol from the global registry

let globalSym = Symbol.for("globalKey");

let sameGlobalSym = Symbol.for("globalKey");

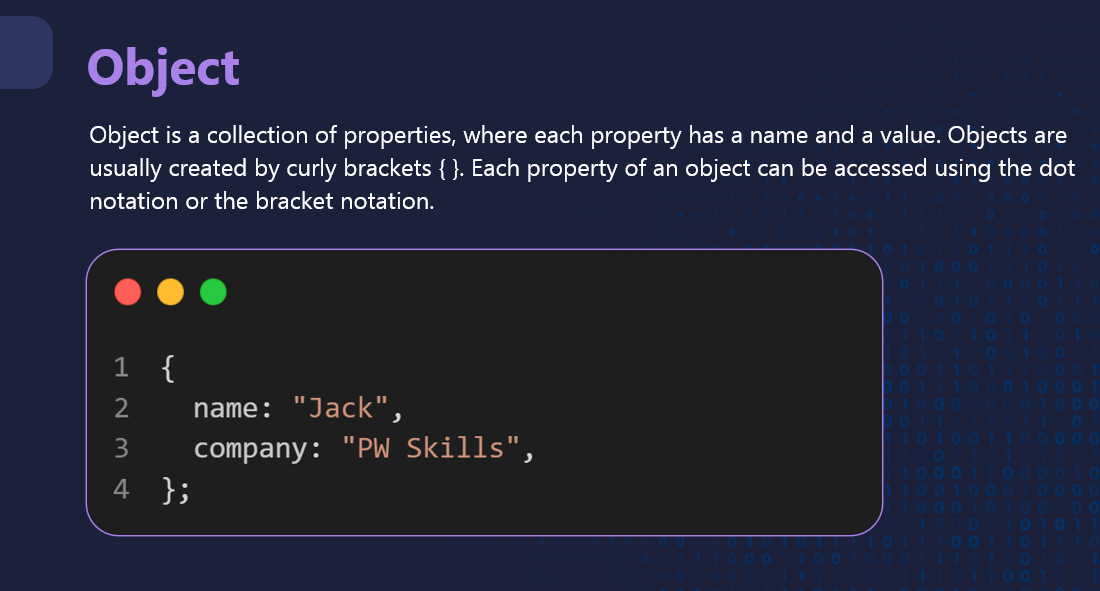
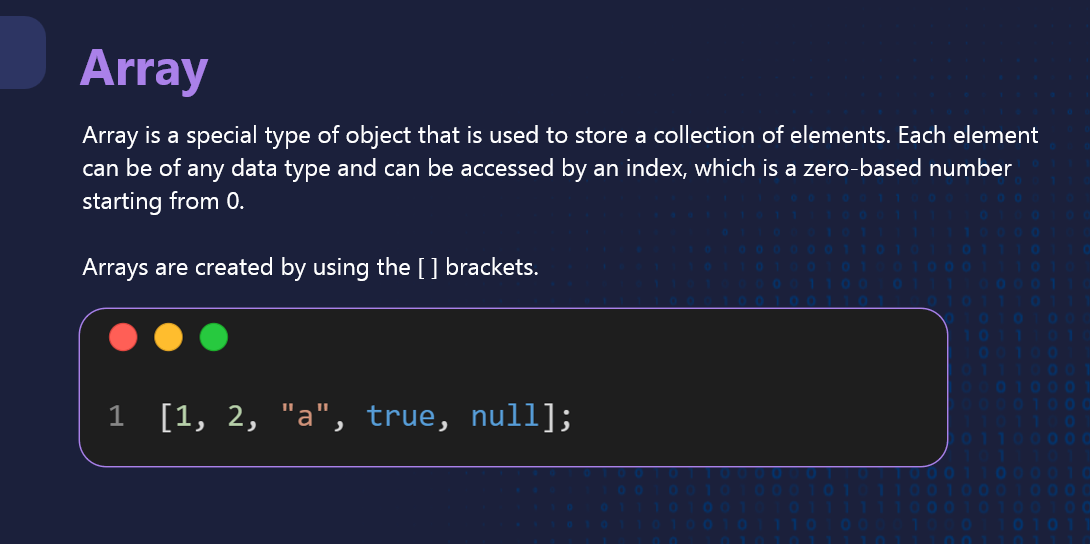
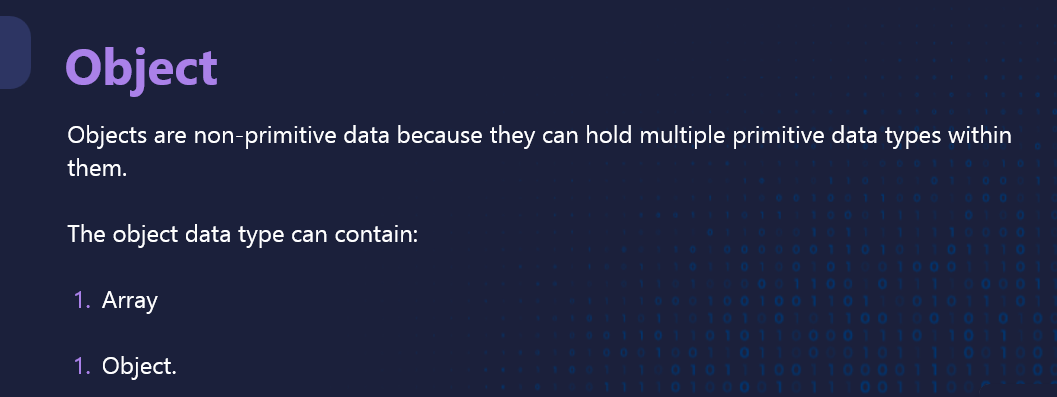
// Symbols retrieved with Symbol.for are the same

console.log(globalSym === sameGlobalSym); // Output: true

// Getting the key for a symbol in the global registry

let key = Symbol.keyFor(globalSym);

console.log(key);



### **Creating an Object**

You can create an object using object literal syntax or by using the Object constructor.

javascript

// Object literal syntax

let person = {

name: "Alice",

age: 30,

job: "Engineer",

greet: function() {

console.log("Hello, my name is " + this.name);

}

};

// Object constructor

let car = new Object();

car.make = "Toyota";

car.model = "Camry";

car.year = 2020;

car.start = function() {

console.log("The car has started.");

};

### **Accessing and Modifying Object Properties**

You can access and modify object properties using dot notation or bracket notation.

javascript

// Accessing properties

console.log(person.name); // Output: Alice

console.log(car["model"]); // Output: Camry

// Modifying properties

person.age = 31;

car.year = 2021;

console.log(person.age); // Output: 31

console.log(car.year); // Output: 2021

### **Adding and Deleting Properties**

You can add new properties to an object or delete existing properties.

javascript

// Adding properties

person.country = "USA";

car.color = "red";

console.log(person.country); // Output: USA

console.log(car.color); // Output: red

// Deleting properties

delete person.job;

delete car.model;

console.log(person.job); // Output: undefined

console.log(car.model); // Output: undefined