Objects are like a dictionary — there are words and their definitions. Similarly, in an object, there is a key and their respective value.

The following is a simple example of how a JavaScript object looks like:

let person = {  
    name: 'John',  
    age: 30,  
    isStudent: false,  
    sayHello: function () {  
        console.log('Hello!');  
    }  
};

In the above object, the name is a key and John is its value. Similarly, age and isStudent are keys, and 30 and false are their respective values.

Finally, sayHello is a key that has a function as its value.

**You can access the properties of an object using dot notation or square bracket notation:**  
  
console.log(person.name); // John  
console.log(person['age']); // 30  
  
**Adding/Modifying Properties:**  
  
person.gender = 'Male'; // Adding a new property  
person.age = 31; // Modifying an existing property

Objects can have functions as properties, which are called methods:  
  
let person = {  
    name: 'John',  
    sayHello: function () {  
        console.log('Hello!');  
    }  
};  
person.sayHello(); // Hello!

**Object Properties and Methods:**

* Properties: Hold data values.
* Methods: Hold functions.
* You can use the for...in loop to iterate over the properties of an object:  
    
  let person = {  
      name: 'John',  
      age: 30,  
      isStudent: false,  
      sayHello: function () {  
          console.log('Hello!');  
      }  
  };
* for (let key in person) {  
      console.log(key + ': ' + person[key]);  
  }  
     
  **Output:**  
  name: John  
  age: 30  
  isStudent: false  
  sayHello: function () {  
             console.log('Hello!');  
          }
* Notice how objects are different from primitive data types, such as number, string, Boolean, null, and undefined.
* Primitive types are immutable, meaning their values cannot be changed after they are created. Any operation that seems to modify a primitive type actually creates a new value.  On the other hand, objects are mutable meaning that any value can be updated.   
    
  let x = 5;  
  x = x + 1; // Creates a new value (6), x is not modified.  
    
  Objects are mutable, and their properties can be changed after they are created.  
    
  let person = { name: 'John', age: 30 };  
  person.age = 31; // Modifies the existing object.  
     
  Primitive values are stored directly in the variable's memory location, whereas object variables store references (memory addresses) to the actual object in memory.
* Copying a primitive type creates an independent copy.  
    
  let a = 10;  
  let b = a; // Creates a copy of the value.  
    
  Copying an object variable only copies the reference, not the entire object.  
    
  let obj1 = { name: 'John' };  
  let obj2 = obj1; // Both point to the same object in memory.  
    
  In the example above, modifying obj2 will also modify obj1.
* [Primitive Comparison](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab1)
* [Object Comparison](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab2)

let a = 5;  
let b = 5;  
console.log(a === b); // true

* [Primitive Comparison](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab1)
* [Object Comparison](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab2)

let obj1 = { name: 'John' };  
let obj2 = { name: 'John' };  
console.log(obj1 === obj2); // false (different references)

let car = {  
    color: 'blue',  
    model: 'sedan',  
    maxSpeed: 120,  
    start: function () {  
        console.log("car started");  
    },  
    stop: function () {  
        console.log("car stopped");  
    },  
    honk: function () {  
        console.log("horn honked");  
    }  
};

let smartphone = {  
    brand: 'iPhone',  
    model: '15',  
    batteryLevel: 80,  
    makeCall: function () {  
        console.log("call made");  
    },  
    sendText: function () {  
        console.log("text sent");  
    },  
    takePhoto: function () {  
        console.log("photo taken");  
    }  
};

Accessing object properties in programming means retrieving the values associated with specific keys within an object. Objects use keys to label pieces of information (properties). Accessing these properties involves fetching the values associated with those keys.

Object properties can be accessed either by using “dot notation” or “square bracket notation.”

*Select each tab to learn more.*

* [Dot Notation](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab1)
* [Square Bracket Notation](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab2)

console.log(person.name); // Dot notation  
  
Dot notation is the simpler and more common way to access object properties.

**Syntax**: objectName.propertyName

Dot notation is easier to read and write, and it is preferred for most scenarios when the property name is a valid identifier (does not contain spaces, starts with a letter, etc.).

Note that both notations can be used interchangeably in many cases, but the choice often depends on the specific use case and the nature of the property names. Dot notation is for simplicity and readability when dealing with static property names. Square bracket notation is used when dealing with dynamic property names or names that are not valid identifiers.

* [Square Bracket Notation](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab2)

console.log(person["age"]); // Bracket notation  
  
Square bracket notation is used when the property name is not a valid identifier or when the property name is dynamic (i.e., determined at runtime).

**Syntax**: objectName["propertyName"] or objectName[expression]

By Using square bracket notation, you can handle dynamic property names or names with special characters. It can be useful when the property name is stored in a variable.  
   
**For example,**  
  
let person = {  
    name: 'John',  
    age: 30,  
    "job title": 'Developer'  
};  
// Dot notation for static properties  
console.log(person.name); // John  
// Bracket notation for dynamic or special properties  
console.log(person["age"]); // 30  
console.log(person["job title"]); // Developer  
// Using variables with bracket notation  
let propName = "name";  
console.log(person[propName]); // John

The for...in loop is used for iterating over the enumerable properties of an object. It should not be used with arrays because it iterates over all enumerable properties, including those added to the prototype chain. It's more suitable for objects.

The following is the basic syntax of how we use for...in loop to loop over an object:  
  
for (const key in object) {  
    // code to be executed for each key  
}  
   
**Example with an object:**  
  
const myObject = { a: 1, b: 2, c: 3 };

for (const key in myObject) {  
    console.log(key, myObject[key]);  
}  
// a 1  
// b 2  
// c 3

An array is a collection of items from the same data type stored at contiguous memory locations. Arrays can be manipulated, which means they can be changed, rearranged, or used to perform various tasks.

You’ll learn to manipulate arrays using methods. You can perform various operations on arrays using predefined array methods, each defined and described below:

*Select each tab to learn more.*

* [Push and Pop](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab1)
* [Concatenation](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab2)
* [Slice](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab3)
* [Map](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab4)
* [Filter](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab5)
* [Reduce](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab6)
* [Sort](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab7)
* [Reverse](https://classroom.emeritus.org/courses/9570/pages/mini-lesson-4-dot-3-objects-and-arrays-30-minutes?module_item_id=2103130#Tab8)

The push method is used to add an element at the end of the array. The pop method requires the element as the parameter that we need to insert at the end of the array. Think of push like adding something to the end of your shopping cart and pop as taking out the last item you added. These operations help you manage your cart effectively.

The pop method is used to remove the element from the end of the array. It expects no parameter.

We can use these methods as follows:

let arr = [1, 2, 3];   
arr.push(4);  // [1, 2, 3, 4] *Adds 4 to the end of the array*  
arr.pop();    // [1, 2, 3]  *Removes 4 from the end of the array*

The concat method is used to concatenate two or more arrays to create a new array that the method returns. It's like taking two separate shopping carts and combining them into one. This is useful when you want to merge two sets of items. They are used to combine two arrays into one, such as merging two lists of names:

const arr1 = [1, 2];  
const arr2 = [3, 4];  
const combinedArr = arr1.concat(arr2);  // [1, 2, 3, 4] *Combines arr1 and arr2 into a new array called combinedArr*

The slice method returns the part of the array between two indices that are specified to the slice method as parameters. Think of a sandwich. Think of slice as slicing the crusts off a peanut butter sandwich to only eat the middle portion.

The syntax to use the slice method is slice(start\_index, end\_index), where start\_index is inclusive, and end\_index is exclusive. Thus, if you want to slice the array from index 1 to 4, you can do it as follows:

const arr = ["James", "Oliver", "Kenneth", "Timothy", "Michael"];  
const slicedArr = arr.slice(1, 4);    
// [ 'Oliver', 'Kenneth', 'Timothy' ] *“Sliced” James and Oliver from the array.*

The map method is used to create and return a new array by applying a function that needs to be provided to it to each element in an existing array. Think of it as a conversion tool. You might use map to convert a list of temperatures from Celsius to Fahrenheit, for example.

Suppose that you want to multiply each element of the array by 2. You can use the map method as follows:

const arr = [1, 2, 3];  
const doubledArr = arr.map((item) => item \* 2);  // [2, 4, 6]*Creates a new array called doubledArr with the multiplied numbers*

The filter method is used to select chosen elements, such as the unread emails from your inbox. Think of filter as when you select the ripe bananas from a bunch of fruit.

The filter method is used to create and return a new array consisting of the elements that pass a Boolean condition specified by a function.

const arr = [1, 2, 3, 4, 5];  
const evenNums = arr.filter((item) => item % 2 === 0);  // [2, 4] *Creates a new array called evenNums that selects only even numbers*

The reduce method uses the elements in an array to return a single value. Think of reduce as when you are calculating the total cost of your grocery cart or the unit cost of a 10-oz watermelon. It is used in conjunction with the accumulator variable, which is like a digital notepad that accumulates the elements as you go.

The reduce method also takes a starting value for the accumulator variable. The following example uses reduce to add all of the numbers in an array. The array is reduced to a single value of 15, which is the sum of the elements of the array. We've specified 0 as the starting value for the accumulator variable.

const arr = [1, 2, 3, 4, 5];  
const sum = arr.reduce((accumulator, item) => accumulator + item, 0);  // 15  
*Creates a new array called sum that adds up all the elements from the arr above.*

The sort method is used to arrange elements in ascending order. You might sort the items in your grocery cart from least expensive to most expensive. The following example takes a set of numbers and sorts them in order:

const arr = [3, 1, 2, 4, 5];  
arr.sort();  // [1, 2, 3, 4, 5]

The reverse method does the opposite of the sort method. It arranges items in descending order (e.g., most expensive to least expensive). In the following example, the set of numbers is sorted in order of largest to smallest.

const arr = [1, 2, 3, 4, 5];  
arr.reverse();  // [5, 4, 3, 2, 1]

Practice:

const student = {

// Your code here

name: "John",

age: 23,

courses: ["Math", "History", "Science"]

};

// Assign a function to the parameter 'greeting'

// It should return a string that introduces the student, mentions their age and the courses the student has taken

// Your code here

student.introduction = function() {

let greeting = "Hi, I am " + this.name + "!. I am " + this.age + " years old, and I have taken the courses: ";

for(let course of this.courses) {

greeting += course + ", ";

}

return greeting

}

// Don't change this line

if(typeof module !== 'undefined') {

module.exports = {

student

};

}

In this task, you’ll work with a simple book object with an isAvailable property. Your goal is to create a function called editBook that toggles the value of the isAvailable property:

* If the book object does not have an isAvailable property, the book object’s isAvailable property should be set to true.  
  Use the “NOT” operator (!).
* if(!variable) {
* // This block gets executed if the variable is false,
* // or even when the variable is undefined
* // Example:
* console.log("Variable is undefined");
* }
* If the book is available (isAvailable is true), set it to false.
* If it’s not available (isAvailable is false), set it to true.

Select the “Try It” button to see the output of your code.

const originalBook = {

title: "Some Title",

author: "Some Author"

}

// Your code here

function editBook(book) {

if(!book.isAvailable) {

book.isAvailable = true;

} else {

book.isAvailable = !book.isAvailable;

}

return book;

}

// Uncomment these lines to test your code:

const newBook = editBook(originalBook);

console.log(newBook);

console.log("Check whether the returned book is the same object as the original book:")

console.log(originalBook === newBook)

// Don't change this line

if(typeof module != 'undefined') {

module.exports = {

editBook

};

}

Implement a function named copyBook that takes a book object as a parameter and returns a new object with the same properties and values. To accomplish this task, do the following:

* Inside the function, create an empty object.
* Use a loop to iterate over the properties of the book object.
* For each property, create a new property in the newly created object with the same key and value.
* Return the new object.

const originalBook = {

  title: "Some Title",

  author: "Some Author",

  isAvailable: true

}

*// Your code here:*

function copyBook(originalBook) {

let book = {};

*// Loop through all properties of the original book*

  for (let key in originalBook) {

    if (originalBook.hasOwnProperty(key)) {

       book[key] = originalBook[key];  *// Copy the property to the new object*

    }

  }

*// Return the newly created object with the same properties*

  return book;

}

*// // Uncomment these lines to test your code:*

const newBook = copyBook(originalBook);

console.log(newBook);

console.log("Check whether the returned book is the same object as the original book:")

console.log(originalBook === newBook)

*// Don't change this line*

if(typeof module != 'undefined') {

  module.exports = {

    copyBook

  };

}

**Task Instructions:**

Write a function describeBook that takes a book object as input and returns a list of descriptions about the book. The book object has the following properties:

* title: the title of the book
* author: the author of the book
* isAvailable: a Boolean indicating whether the book is available

The function should iterate through the properties of the book object and:

* For each property (except isAvailable), generate a description string in the format: "The [property name] of the book is [property value]"
* For the isAvailable property, generate a description string in the format:
  + "This book is available!"if the value is true.
  + "This book is not available" if the value is false.

const originalBook = {

  title: "Some Title",

  author: "Some Author",

  isAvailable: true

}

*// Your code here*

function describeBook(originalBook){

  const descriptions = [];

  for (let key in originalBook){

    if (originalBook.hasOwnProperty(key)) {

      if (key === 'isAvailable') {

*// Special case for the 'isAvailable' property*

        if (originalBook[key]) {

          descriptions.push("This book is available!");

        } else {

          descriptions.push("This book is not available");

        }

      } else {

*// For other properties, create the general description*

        descriptions.push(`The ${key} of the book is ${originalBook[key]}`);

      }

    }

  }

*// Return the array of descriptions*

  return descriptions;

}

*// // Uncomment this line of code to test it:*

const desc = describeBook(originalBook);

console.log(desc);

*// Don't change this line*

if(typeof module != 'undefined') {

  module.exports = {

    describeBook

  };

}

**Task Instructions:**

Your task is to ensure that the printInfo() function returns the right value.

**Here’s what the**infoString**variable should contain:**

1. The book’s title
2. The book’s author
3. Availability information: “Yes” if the book is available. “No” if the book is not available.

const book = {

  title: "Some title",

  author: "Some author",

  isAvailable: true

}

*// Your code here*

book.printInfo = function () {

  let infoString = "";

*// Todo: Modify the `infoString` variable according to instructions.*

*// Your code here*

  infoString += `Title: ${*this*.title}, `;

  infoString += `Author: ${*this*.author}. `;

  infoString += ". Available? "

  infoString += *this*.isAvailable ? "Yes" : "No"

  return infoString;

}

*// Uncomment the next line to see your code in action*

console.log(book.printInfo());

*// Don't change this line*

if(typeof module != 'undefined') {

  module.exports = {

    book

  };

}