I-Novotek Academy

Full Stack Web Development Course PDF Guide

website: www.inovotekacademy.com

Youtube: i-novotek academy

Course link: https://www.udemy.com/course/fullstack-web-development-course-projects-base/?referralCode=F8C808368D020D5794BD

The Basics of Javascript: A Beginner's Guide

If you're new to programming, javascript is a great language to start with. It's easy to learn and there are many resources available to help you get started. In section, we'll cover the basics of javascript, including variables, data types, operators, and functions. By the end of this section, you'll have a better understanding of how javascript works and be able to write basic programs.

What is Javascript?

Javascript is a programming language that allows you to add dynamic content to your web pages. That means you can create things like animations, games, and form validation with javascript. It runs on your web browser and doesn't require a separate download or installation like some other languages.

`

Javascript is what's called a "client-side" language. That means the code runs on your computer, not on the server where the website is hosted. This is in contrast to "server-side" languages like PHP or ASP, which run on the server before the page is even sent to your browser.

Client-side languages are convenient because they don't require any special setup on the server. All you need is a text editor and a web browser, and you're ready to start coding!

How Does Javascript Work?

When you view a web page, your browser downloads the HTML code from the server and displays it as a webpage. Javascript code is embedded into the HTML code and runs automatically when the page loads.

The Benefits of Javascript

Javascript is a programming language that is widely used by web developers to create interactive web applications.

Javascript has a number of benefits that make it a popular choice for web development. It is easy to learn, versatile, and can be used to create a wide variety of applications. Additionally, javascript is free to use and is supported by all major web browsers.

If you are considering learning a programming language for web development, javascript is a great choice. It is a versatile language that can be used to create a wide variety of applications.

With Javascript you can become a frontend, backend and fullstack developer

With Javascript you can create web applications, mobile and desktop apps

Javascript Variables: A Comprehensive Guide

What are variables in JavaScript?

JavaScript variables are used to store data values. In JavaScript, data values can be text strings, numbers, arrays, or objects. Variables can be declared using the var, let, or const keywords.

How to create variables in JavaScript

We use the following keywords to create variables

- **var**

- **let**
- **const**

var keyword

the var keyword is used to create a variable that is globally scoped, meaning it can be accessed from anywhere in your code.

let keyword

The let keyword is used to create a variable that is locally scoped, meaning it can only be accessed from within the block of code it was declared in.

const keyword

The const keyword is used to create a variable that cannot be reassigned, making it a constant value.

To declare a variable, you will use the following syntax:

var variableName = value;

Where variableName is the name of your variable and value is the value you are assigning to the variable. You can also declare

multiple variables on the same line using a comma delimiter like this:

```
var name = "John Doe", age = 39, isMarried = true;
```

Once a variable is declared, you can assign it a value using the equal sign (=):

myName = "John Smith"; // Assigns the value "John Smith" to the myName variable.

You can also declare multiple variables without assigning values to them by leaving off the value part of the statement. Variables declared in this way will have a default value of undefined.

var name, age, isMarried; // all variables are assigned the value of undefined by default. /* The following shows data types */

Basic rules of JavaScript syntax

Here are some of the basic rules that govern JavaScript syntax.

1. JavaScript is case-sensitive. This means that language keywords, variables, function names, and any other identifiers must always be

typed with a consistent capitalization of letters. The keyword var is not the same as the keyword var.

2. Statements must end with a semicolon (;). For example, the statement x = 5; is valid, but the statement x = 5 is not.

3. Whitespace (spaces, newlines, and tabs) is generally ignored, except when it is used to separate tokens. For example, the statements x=5 and x = 5 are equivalent.

4. Comments can be added to your code to make it more readable. JavaScript supports two types of comments:

// This is an single-line comment /* This is a multi-line comment */

5. Identifiers can be any combination of letters, digits, underscores (_), and dollar signs ($). However, they cannot start with a digit. In addition, some reserved words cannot be used as identifiers (e.g., class, return, etc.).

6. Variables must be declared before they are used. This can be done using the keyword var. For example:

var x; // declares a variable named x var y = 5; // declares a variable named y and assigns it the value 5

7. Data types in JavaScript include numbers, strings, Booleans (true/false values), and objects. There are also two special data types: undefined and null.

8. Numbers can be written with or without decimals. For example, 3.14, 42, and -99 are all valid numbers.

9. Strings must be enclosed in quotes. Single or double quotes can be used, but they must match (e.g., "hello" and 'goodbye' are both valid, but "hello' is not). Strings can span multiple lines by using the backslash (\ ) as an escape character:

"This is the first line.\nAnd this is the second." // produces "This is the first line.<newline>And this is the second."

10 .Code must be put within <script> tags in order to run

11 . Javascript is a text-based language. This means that it is made up of words, numbers, and punctuation marks that are read by a computer and interpreted into instructions.

Variable naming conventions in JavaScript

- **Variable names can start with letters**, $, or _ .

The following variable names are all valid:

```
$sname, sname, name_sname.
```

- **No spacing**

Variable names should not contain spaces use an underscore instead.

```
var user_name = "John Doe";
```

- **Use descriptive names:**

Using descriptive names for your variables is a good way to make your code more readable and understandable. When choosing a name for your variable, try to think of a word or phrase that accurately describes what the variable is used for.

- **Use camel case:**

Camel case is a naming convention where each word in the name is capitalized, except for the first word. For example, the variable name “myVariable” would be written in camel case.

Do not use reserved words:

When choosing a name for your variable, be sure to avoid using any of the reserved words in JavaScript. Reserved words are words that have special meaning in the language and cannot be used as variable names.

- **Do not use spaces or special characters:**

Spaces and special characters are not allowed in JavaScript variable names. If you need to use multiple words to describe your variable, you can use camel case orunderscores to separate the words (“my_variable”).

- **Keep it short:**

Long variable names can make your code more difficult to read and understand. When possible, try to choose a name that is short but still descriptive.

Javascript data types

The JavaScript language contains two different types of data: primitive values and objects.

A primitive value is a value that has no properties or methods. A primitive value is immutable, which means it cannot be changed. The only way to create a new primitive value is to create a

new variable and assign it a new value.

There are the primitive values in JavaScript: undefined, null, Boolean, Number, String, Symbol, and bigint.

1. undefined

Undefined is a value that represents no value. It is the default value of variables that have not been assigned a value.

2. null

Null is a value that represents no value. It is used to indicate that a variable does not have a value.

3. Boolean

Boolean is a value that represents either true or false.

4. Number

A number is a value that represents a number. All numbers in JavaScript are 64-bit floating-point numbers.

5. String

String is a value that represents a sequence of characters. Strings are immutable, which means they cannot be changed. The only way to create a new string is to create a new variable and assign it a new value.

6. Symbol

Symbol is a unique and immutable value that can be used to identify an object. Symbols are typically used as keys in objects.

7. **Objects (collections of properties)**

Javascript Operators: A Comprehensive Guide

What are operators in JavaScript?

Operators in JavaScript are the symbols that represent certain actions that can be performed on variables. In JavaScript, there are many different kinds of operators, including arithmetic operators, assignment operators, comparison operators, logical operators, and Bitwise operators. Each kind of operator performs a different kind of operation.

Arithmetic operators

Arithmetic operators take two operands (values) and perform an arithmetic operation on them

Arithmetic operators include

- **Addition (+)** to add operands together example let sum = 2+6
- **Subtraction (-)** subtracts operands together let difference = 9-7
- Multiplication (*)
- **Division (/)** let total = 9/2
- **Modulus (%)** division with remainder. Example 12%2 would give you 0 as the remainder while 13%2 would give you 1 as it should be according to division;

Increment (++)

Synthax

- **x++ (** postfix**)** Postfix increment

- **++x (**prefix) Prefix increment

Postfix increment

If used postfix, the increment operator will first increment the value before returning it.

```
let x = 3;
```

y = x++;

```
// y = 3
// x = 4
```

Prefix increment

If used as a prefix operator (for example, ++x), the increment operator will increment the operand and return the new value.

```
let a = 2;
```

b = ++a;

```
// a = 3
// b = 3
```

Assignment operators

The assignment operator in javascript assigns a value to a variable. The most common assignment operator is the = operator, which assigns the value to the left of the = to the variable on the right. For example, if we have a variable called x and we want to give it the value of 5, we would write x = 5.

Other assignment operators include the += operator, which adds the value to the left of the = to the variable on the right, and the -= operator, which subtracts the value to the left of the = from the variable on the right. Example :

int x = 5; // Assigns 5 to x x -= 4; // Subtracts 4 from x, so now x equals 1 int y = 4; y += 2; // Adds 2 to y, so now y equals 6

Combined Assignment Operators

You can combine the assignment operator with basic arithmetic operators using the combined assignment operators (+= , -= , *= , /= , &= , and |=). This lets you write code like this:

let m=0

m = m + 5 is equivalent to m += 5 .

m = m * 3 is equivalent to m *= 3 .

Double equal to vs tripple equal to

Double equal to is == , it compares values.

Tripple euqal to is === , it compares both value and type.

Example of ==

"" == "0" is true then when comparing string , == will compare only values, not type.

```
 0 == false is also true, here as you can see when comparing
value 0 and false then it converts 0 to falsy value.
```

Example of ===

```
= "" === "0" is false, because it compares both values and
their data types(string vs number)
```

Comparison operators

Comparison operators Comparison operators are used to compare two values:

- greater than (>)
- less than (<)
- greater than or equal to (>=)
- and less than or equal to (<=).

NOTE: These operators evaluate expressions such that they return either true or false :

3 < 5 // evaluates to true 3 > 5 // evaluates to false 3 >= 4 // evaluates to true 2 <= 1 // evaluates

Logical operators

Logical operators allow JavaScript to perform boolean logic on values. These operators are:

- **and' ,bb'** : If each of the two operands is true, then the condition becomes true. (a && b) will be true only if both a and b are true.

- **or' ‡ÇÂf** any of the two operands is true, then the condition becomes true. (a || b) will be true if either a or b is true.

- **not' , "** Used with boolean values to reverse the value i.e. from false to true, and from true to false

  var x = 2;

  // x != 7 is TRUE

Conditional Statements in javascript

What is a conditional statement? A conditional statement is a set of commands that will only be executed if a specified condition is met.

```
if (condition) {

  // commands to execute if condition is true

} else {

  // commands to execute if condition is false

}
```

If, else if, and else

are all used to create conditional statements in various programming languages.

Nested conditional statements A nested conditional statement is a conditional statement that contains another conditional statement as one of its components. Code example

Nested conditional statements

Truthy and falsy values in javascript

In JavaScript, a truthy value is a value that is considered true when evaluated in a Boolean context. All values are truthy unless they are defined as falsy. The following values are considered falsy:

- false

- 0

- ""

- null

- undefined

So what does this all mean? Basically, any value that is not one of the falsy values listed above is considered truthy. This includes values like true, 1, and "foo".

JavaScript Loops

JavaScript loops are a powerful programming tool that allows you to execute a block of code multiple times. While there are many different types of loops, they all share a common structure: a condition is checked, and if the condition is true, the code block is executed. This process is then repeated until the condition is no longer true. In this blog post, we'll take a closer look at how JavaScript loops work and how you can use them in your own pr

ograms.

The For Loop

The for loop has the following syntax: > The **for** statement creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement or a block of statements.

```
for ([initialization]; [condition]; [final-expression]){ //
code here... }
```

```
inal-expression ===incrementer/decrementer
```

> The **initialization** expression initializes the loop; it's executed once, as the loop begins.

> When the **condition** returns true, the loop executes the statement.

> The **final-expression** is executed every time the statement (s) runs.

Code example :

```
let i;
for(i=0;i<5; i++){ // code here will run 5 times! }
```

The while loop

The while loop is similar to the for loop, but instead of using an initializer and an incrementer, it only has a condition. The code block is executed repeatedly until the condition evaluates to false.

```
 while (condition) {
  code block to be executed}
  }
```

let i = 0;

```
  while (i < 10) {
    console.log(i);
    i++;
 } // loops ...0, 1, 2,...9
```

The While Loop

The while loop loops through a block of code as long as a specified condition is true.

```
while (condition) {
```

```
  // code  to run
}

let msg = "";
while (x < 10) {
 msg += "The number is " + i;
 i++;
}
```

Javascript Functions

What is a function?

In JavaScript, a function is a piece of code that is written to perform a specific task. Functions are usually self-contained and can be reused across your code. This makes them very important in JavaScript programming.

Why functions

There are many reasons why you should use functions in JavaScript. Here are 10 of the most important reasons:

1. Functions help to make your code more readable.

2. Functions can make your code more reusable.

3. Functions can help to make your code more maintainable.

4. Functions can improve the performance of your code.

5. Functions can help to modularize your code.

6. Functions can make your code more testable.

7. Functions can help to reduce the complexity of your code.

8. Functions can improve the flexibility of your code.

9. Functions can improve the

Defining functions

Functions can be written either as a

- function declaration or a
- function expression.

Function declarations are written as follows:

```
function name() {
  // code to be executed
}
```

Function expressions are written as follows:

```
let name = function() {
  // code to be executed
}
```

Function return keyword

The return keyword is used to exit a function and return a value to the caller.

The return keyword can be used with or without a value. If a value is present, it is returned to the caller. If no value is present, the function simply exits.

The return keyword is essential for creating functions that return values. Without it, functions would only be able to perform actions, not return values to the caller.

This would limit the usefulness of functions and make them much less powerful.

So if you're creating a function that needs to return a value, be sure to use the return keyword. It'll make your function much more useful and powerful.

Differences between function arguement and function parameters

Function arguments and function parameters are often used interchangeably, but there is actually a subtle difference between the two. Function arguments are the values that are passed to a function when it is invoked, while function parameters are the variables that are used to receive those arguments.

Javascript Strings: The Essential Guide

Introduction

Javascript strings are one of the most essential data types in the language. In this guide, we will explore the various properties and methods associated with strings, as well as how to manipulate and format string data.

What are strings?

In Javascript, strings are lines of text that are used to store and represent data. Strings can be anything from a single character to an entire novel—any sequence of characters can be stored as a string. Strings must always be placed within either single or double quotation marks (' ' or " ").

Strings are commonly used for storing user input, such as when a user enters their name into an input field on a website.

How to create strings?

Strings can be created in several different ways. The most common way is simply by assigning a string of characters to a variable:

```
const myName = 'Paige' ;

const greeting = 'Hi there!' ;

 const favoriteNumber = 'My favorite number is 7'

 let str1 = 'Hello' , str2= ', world!' ;
```

toLowerCase()

toUpperCase()

String length()

The length property returns the number of characters in a string.

The length property of an empty string is 0.

```
let str = "Hello World!";

let length = str.length;
```

String trim()

The trim() method is useful for removing whitespace from both sides of a string. This can be helpful for cleaning up user input, or for making sure that two strings are truly equal.

The trim() method does not mutate the original string.

```
let str = "     Coding time       ";
let result = str.trim();
```

String split()

The split() method separates a string into an array of substrings.

The split() method returns a new array.

The split() method does not change the original string.

```
let str3 = "Are you coding today?";
const myArray = str3.split("");
```

string reverse()

The reverse() method changes the order of the elements in an array so that the first element becomes the last, the second element becomes the second to last, and so on.

The reverse() method modifies the original array.

```
const food = ["Pizza", "Congee", "Fufu", "rice"];
const newFood = food.reverse();
```

Array join()

The join() method returns an array converted to a string.

The join() method does not change the original array.

Any separator can be specified. The default is comma (,).

```
const languages = ["English", "Twi", "French", "Fante"];
```

let text = languages.join();

```
console.log(text);
```

String repeat()

The slice() method produces a new string that is a copy of the original string. The original string is not changed. === making copies of strings

```
let str = "Hello world!";

let result = str.repeat(3);
```

String startsWith()

The startsWith() method is useful for determining whether a string begins with a specified string.

This is important because it can help ensure that a string is the correct format.

For example, if you are expecting a string to be in all caps, you can check to see if the string starts with a capital letter.

If the string does not start with a capital letter, you know that it is not in the correct format.

The startsWith() method is case sensitive.

```
let str = "Welcome to i-novotek Academy";
str.startsWith("Welcome");
```

String includes()

- The includes() method determines whether a string contains a specified string.

  If it does, it returns true.Otherwise, it returns false.
- It takes two arguments ('text to search', 'startting point') //default is 0
- The includes() method is case-sensitive.

```
let str = 'Are you a web developer';
let result = str.includes('you');
let result2 = str.includes('you', 4);
```

concat() Method

The concat() method concatenates two or more strings.

The concat() method does not mutate the original strings.

The concat() method returns a new string.

```
let str1 = 'Your';

let str2 = 'order';

let str3 = 'is ready';

let result = str1.concat(' ', str2, ' ', str3);
```

slice() Method

The slice() method extracts a section of a string.

The initial position is 0, the subsequent is 1,

A negative value picks from the end of the string.

The slice() method does not alter the original string.

The slice() method enables you to select from a given start point up to (but not including) a given end point. This method does not change the original data.

```
let str = 'Welcome to javascript methods';

let result = str.slice(0, 5);
```

let ans = str.slice(3);

```
console.log(ans);
```

String comparison

In order to compare whether one string is greater than another, JavaScript employs what is known as "dictionary" or "lexicographical" order.

In other words, strings are compared letter-by-letter.

```
console.log( 'Z' > 'A' ); // true

console.log( 'Glow' > 'Glee' ); // true

console.log( 'Bee' > 'Be' ); // true
```

FACTS ABOUT STRING COMPARISMS

- Compare the first character of both strings

- First it will check the first letters on both sides if one is greater than the other it will return true but if they are equal it will move to the next letter until it returns true /false
- Lowercase letters are greater than uppercase letters of the same type Because the lowercase character has a greater index in the internal encoding table JavaScript uses (Unicode)
- If both strings end at the same length, then they are equal. Otherwise, the longer string is greater

Comparison of different types

- When comparing values of different types, JavaScript converts the values to numbers.

```
console.log( '3' > 2 ); // true, string '3' becomes a number 3

console.log( '05' == 3 ); // true, string '05' becomes a number
5
```

- For boolean values, true becomes 1 and false becomes 0

```
console.log( true == 1 ); // true

console.log( false == 0 ); // true
```

Comparison with null and undefined

For a strict equality check ===
These values are different, because each of them is a different type.

```
console.log( null === undefined ); // false
```

For a non-strict check ==
There's a special rule. These two are a "sweet couple": they equal each other (in the sense of ==), but not any other value.

```
console.log( null == undefined ); // true
```

The complete guide to understanding numbers in Javascript

Number Basics In JavaScript, numbers are a primitive data type. This means that they are not an object and they have no methods.

Numbers can be positive or negative, integers or floats. An integer is a whole number, while a float is a number with a decimal point.

Example fo floating numbers

```
const num = 0.1 + 0.2;

console.log(num);
```

Example fo integer numbers

```
const num = 8;

console.log(num);
```

Example of string concatenation

```
const name = "John";

const surname = "Smith";

const fullName = name + " " + surname;
```

Infinity (or -Infinity)

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

```
const infinity = 1 / 0;

const infinity2 = 1 / Infinity;

const infinity3 = -1 / 0;
```

Numerical operations

```
const number1 = 10;

const number2 = 2;

const number3 = number1 + number2;
```

Numerical string operations

Javascript converts string to number in numerical operations

```
const number4 = "10";

const number5 = 2;

const number6 = number4 + number5;
```

Numerical string operations with strings

This won't work because Javascript +operator to concatenate strings

```
const number7 = "10";
const number8 = "2";
const number9 = number7 + number8;
```

NaN - Not a Number (Not legal number)

Example of NaN

```
const number10 = "10";
const number11 = 2;
const number12 = number10 / number11;
```

Example2

```
const number13 = "10";
const number14 = "foo";
const number15 = number13 / number14;
```

JavaScript Number Methods

Primitive values cannot have properties and methods, but JavaScript treats primitive values as objects when executing methods and properties. This allows for methods and properties to be available to primitive values.

toFixed()

The toFixed() method rounds a number to a certain number of decimals, and returns a string.

```
const num1 = 10.3;
const num2 = 10.6;
const num3 = num1.toFixed(2);
const num4 = num2.toFixed(2);
```

toString()

The toString() method converts a number to a string.

```
const num5 = 10;
const num6 = num5.toString();
```

Converting various types to numbers

parseInt()

- The parseInt() method parses a string and returns an integer.
- Only first number is returned by ignoring the rest of the string

```
const num7 = "10.909";
const num8 = parseInt(num7);
```

parseFloat()

- The parseFloat() method parses a string and returns a floating point number.
- It returns all numbers including the decimal point.

```
const num9 = "10.3";
```

```
const num10 = parseFloat(num9);
```

Number()

The Number() method converts a string to a number.

```
const num11 = "10";
const num12 = Number(num11);
```

JAVSSCRIPT ARRAYS

arrays are a collection of items in a particular order and can be accessed by index number.

create an array

Method 1: using the new keyword
Example:

```
const myArray1 = new Array("css", "html", "javascript");
//or
const myArray2 = new Array();
myArray2[0] = "pizza";
myArray2[1] = "burger";
myArray2[2] = "chicken";
```

Method 3: using the array literal

```
const myArray3 = ["Node", "Express", "MongoDB"];
//or
const myArray4 = [];
```

```
myArray2[0] = "Node";
myArray2[1] = "Express";
myArray2[2] = "MongoDB";
```

Accessing elements in an array

```
const myArray5 = ["Node", "Express", "MongoDB"];
console.log(myArray5[0]);
```

Iterating over an array

method 1: for loop

```
for (let i = 0; i < myArray5.length; i++) {
  console.log(myArray5[i]);
}
```

Advanced: method 2: forEach //We will discus later

```
myArray5.forEach(function (element) {
  console.log(element);
});
```

Array methods

Method 1: push()

adds an element to the end of an array

```
const myArray6 = ["Node", "Express", "MongoDB"];
myArray6.push("React");
console.log(myArray6);
```

Method 2: pop()

removes an element from the end of an array

```
const myArray7 = ["Node", "Express", "MongoDB"];
myArray7.pop();
console.log(myArray7);
```

Method 3: unshift()

adds an element to the beginning of an array

```
const myArray8 = ["Node", "Express", "MongoDB"];
myArray8.unshift("React");
console.log(myArray8);
```

Method 4: shift()

removes an element from the beginning of an array

```
const myArray9 = ["Node", "Express", "MongoDB"];
myArray9.shift();
console.log(myArray9);
```

Method 5: indexOf()

returns the index of the first element in an array that matches the specified value

```
const myArray10 = ["Node", "Express", "MongoDB"];
console.log(myArray10.indexOf("Express"));
```

Method 6: lastIndexOf()

returns the index of the last element in an array that matches the specified value

```
const myArray11 = ["Node", "Express", "MongoDB"];

console.log(myArray11.lastIndexOf("Express"));
```

Method 7: includes()

returns a boolean indicating whether an array includes a certain value

```
const myArray12 = ["Node", "Express", "MongoDB"];

console.log(myArray12.includes("Express"));
```

Javascript objects

What are objects?

Objects are variables that hold multiple pieces of information. In JavaScript, objects are created with curly braces {}.

Creating Objects

method 1: Using the Object() constructor function:

```
const person = new Object();

person.name = "John";

person.age = 30;

person.city = "New York";
```

method 2: Using the object literal syntax:

```
const person2 = {
  name: "John",
  age: 30,
  city: "New York",
};
```

Accessing Object Properties

1. using dot notation:

```
person.name;
person.age;
person.city;
```

2. using bracket notation:

```
person["name"];
person["age"];
person["city"];
```

Updating Object Properties

1. using dot notation:

```
person.name = "Jane";
person.age = 31;
person.city = "Miami";
```

2. using bracket notation:

```
person["name"] = "Jane";
person["age"] = 31;
person["city"] = "Miami";
```

Deleting Object Properties

1. using dot notation:

```
delete person.name;
delete person.age;
delete person.city;
```

2. using bracket notation:

```
delete person["name"];
delete person["age"];
delete person["city"];
```

Adding Methods to Objects

```
const carObj = {
  make: "Ford",
  model: "Mustang",
  year: 1969,
  color: "red",
```

```
  description: function () {
    return `${this.make} ${this.model} (${this.year})`;
  },
};
```

What is this?

- In JavaScript, the this keyword refers to an object.
- The this keyword refers to different objects depending on how it is used:
- In an object method, this refers to the object.
- Alone, this refers to the global object.
- In a function, this refers to the global object.
- In a function, in strict mode, this is undefined.
- In an event, this refers to the element that received the event.

Iterating Over Objects

1. using for...in loop:

2. using Object.keys() method:

3. using Object.values() method:

4. using Object.entries() method:

1. for...in loop:

Syntax:

```
for (let key in obj) {
 //code to run
}
```

.key is the name of the property

.in is the keyword

.obj is the object

```
//Example:
const student = {
  name: "John",
  age: 30,
  city: "New York",
};
//iterating over an object using for...in loop
//-----
for (let key in student) {
  console.log(key);
}
```

2. using Object.keys() method:

Syntax

```
Object.keys(obj)
```

obj is the object

```
//Example:
const student2 = {
```

```
  name: "John",
  age: 30,
  city: "New York",
};
//iterating over an object using Object.keys() method
//-----
//convert the keys of an object into an array
```
const keys = Object.keys(student2);
```
//console.log(keys); //array
```

iterate using forEach loop

Syntax:

```
//obj.forEach(function(value, key) {
//  //code to run
//}
//value is the value of the property
```
//key is the name of the property
```
keys.forEach((key) => {
  console.log(key);
  console.log(`${key}: ${student2[key]}`);
```
});
```
//3. using Object.values() method:
```

Object.values()

/The Object.values() method works opposite to that of Object.key(). It returns the values of all properties in the object as an array. You can then loop through the values array by using any of the array looping methods.

```
//Syntax:
//Object.values(obj)
//obj is the object
//-----
```

Example:

```
const student3 = {
  name: "John",
  age: 30,
  city: "New York",
  height: 1.8,
  weight: 80,
};
```

iterating over an object using Object.values() method

```
//convert the values of an object into an array
const values = Object.values(student3);
console.log(values);
```

looping through an array of objects using forEach loop

```
values.forEach((value) => {
  console.log(value);
```

```
});
```

4. Using object.entries() method:

The object.entries() method returns an array of arrays, where each array contains a key and value pair.

The object.entries() method is useful when you want to iterate over the properties of an object.

Example

```
const student4 = {
  name: "John",
  age: 30,
  city: "New York",
  height: 1.8,
  weight: 80,
};
```

iterating over an object using Object.entries() method

//convert the entries of an object into an array

```
const entries = Object.entries(student4);
console.log(entries);
```

using forEach loop

```
entries.forEach((entry) => {
  console.log(entry);
```

```
});
```

destructuring assignment

```
entries.forEach(([key, value]) => {
  console.log(`${key}: ${value}`);
});
```