

JavaScript

Speed up your code

Michael Lappenbusch

IT-SPECIALIST APPLICATION DEVELOPMENT

Table of contents

Introduction to JavaScript	2
Variables and data types	2
Operators and Expressions.....	3
Control structures (if/else, loops)	5
functions.....	7
arrays and objects	7
Dom Manipulation	9
Events and event handling	10
Form validation and submit	11
AJAX and asynchronous programming.....	12
JavaScript libraries and frameworks (e.g. jQuery, AngularJS, React).....	13
Error handling and debugging	14
Best practices and optimization	14
Newer functions of ECMAScript (e.g. arrow functions, promises).....	15
Building JavaScript-based applications (e.g. single page apps).....	16
imprint.....	18

Introduction to JavaScript

JavaScript is a scripting language that is mainly used for developing dynamic and interactive websites. It was originally developed by Brendan Eich in 1995 and has received a variety of new features and enhancements since then.

JavaScript can run on both the client and the server. On the client, it is mainly used to process user interactions on a web page and change the appearance of the page without having to reload the entire page. On the server, it is mainly used with Node.js to develop server applications.

JavaScript is embedded directly in HTML and CSS files and can also be stored in external files, which are then integrated into HTML via a link. It works closely with the Document Object Model (DOM) that allows elements on a web page to be identified and manipulated.

A key part of JavaScript is the use of functions, which can be used as "callbacks" to perform specific actions when specific events or conditions occur. JavaScript also supports the use of objects, arrays, and other data types commonly used in programming.

In recent years, JavaScript has received a multitude of new features and extensions, including ECMAScript 6 and higher, which enable developers to create modern and powerful applications. In addition, there are many popular JavaScript libraries and frameworks such as jQuery, AngularJS, and React that help developers save time and effort in developing applications.

Overall, JavaScript is a versatile and powerful scripting language that is essential for developing dynamic and interactive websites.

Variables and data types

In JavaScript, there are different types of variables and data types that can be used to store and manipulate data.

Variables are placeholders for values used in a program. Variables are declared in JavaScript with the keyword "var", "let" or "const". "var" is the oldest method and is mostly used in older code, while "let" and "const" are used in newer code. The difference between "let" and "const" is that a variable declared with "let" can be subsequently modified, while a variable declared with "const" cannot be modified.

Primitive data types are the basic data types in JavaScript and include:

String(s): e.g. "hello world"

Number: e.g. 42

Boolean (truth values): eg true or false

Undefined: Value assigned to a variable before another value is assigned.

Symbol (New in ECMAScript 6): Unique, immutable values that can be used to denote objects.

Reference data types are more complex data types that can contain multiple values, such as:

Object(s): e.g. {name: "John", age: 30}

Array(s): e.g. [1, 2, 3, 4]

Function (functions): e.g. function add(a, b) {return a + b;}

JavaScript also has a special kind of primitive data type called BigInt that can be used for large integers.

It's important to note that JavaScript has dynamic typing, which means that a variable's type is automatically recognized based on the value it's assigned. However, it's always helpful to explicitly specify the type of a variable when writing code to avoid errors and increase code readability.

Operators and Expressions

In JavaScript, there are different types of operators and expressions that can be used to manipulate data and evaluate conditions.

Arithmetic operators are used to perform arithmetic calculations. Examples of arithmetic operators are:

addition (+)

subtraction (-)

multiplication (*)

Division (/)

Modulo (%)

increment(++)

decrement (--)

Comparison operators are used to compare two values and return a boolean value. Examples of comparison operators are:

equality (==)

Identity (===)

inequality (!=)

non-identity (!==)

Greater than (>)

Less than (<)

Greater than or equal to (>=)

Less than or equal to (<=)

Logical operators are used to evaluate logical expressions. Examples of logical operators are:

Logical AND (&&)

Logical OR (||)

Logical NOT (!)

Assignment operators are used to assign values to variables. Examples of assignment operators are:

equal (=)

Addition and assignment (+=)

Subtraction and Assignment (-=)

Multiplication and assignment (*=)

Division and Assignment (/=)

Modulo and assignment (%=)

Ternary operators (also known as conditional operators) are shorthand for an if-else statement. They are of the form "condition ? expression1 : expression2" and check the condition if true expression1 evaluates and returns the value, otherwise expression2 evaluates and returns the value.

In JavaScript, expressions can also be used to calculate a value, which can then be used by a statement or another function. An expression can consist of a single number or variable, or it can contain multiple operators and variables that combine together to calculate a value.

It's important to note that the order of evaluation of operators, called operator precedence, is important in JavaScript. Some operators have higher precedence than others and are evaluated first. For example, multiplication and division are evaluated before addition and subtraction. If you are unsure of the order of evaluation of operators, you can use parentheses to define the order of evaluation.

Overall, operators and expressions are an important part of programming in JavaScript, allowing developers to manipulate data, check conditions, and calculate values. Understanding and using operators and expressions correctly is an important part of developing effective and error-free code.

Control structures (if/else, loops)

There are various control structures in JavaScript that can be used to control the execution of code and perform specific actions based on specific conditions. The two most important control structures are the if/else statement and loops.

if/else statement allows certain code to be executed only if certain condition is met. The syntax looks like this:

```
if (condition) {  
  // code is executed if condition is true  
} else {  
  // code is executed if condition is false  
}
```

It is also possible to check multiple conditions with else if:

```
if (condition1) {  
  // code is executed if condition1 is true  
} else if (condition2) {  
  // code is executed if condition2 is true  
} else {  
  // code is executed if no condition is true  
}
```

There are various control structures in JavaScript that can be used to control the execution of code and perform specific actions based on specific conditions. The two most important control structures are the if/else statement and loops.

if/else statement allows certain code to be executed only if certain condition is met. The syntax looks like this:

```
if (condition) {  
  // code is executed if condition is true  
} else {
```

```
// code is executed if condition is false  
}
```

It is also possible to check multiple conditions with else if:

Copy code

```
if (condition1) {  
  // code is executed if condition1 is true  
} else if (condition2) {  
  // code is executed if condition2 is true  
} else {  
  // code is executed if no condition is true  
}
```

Loops allow certain code to be executed multiple times in a row. There are several types of loops in JavaScript, including:

for loop: Used to perform a specified number of iterations.

while loop: Used to run code as long as a certain condition is met.

do-while loop: Similar to the while loop, but ensures that the code is executed at least once before the condition is checked.

for-of loop: Used to iterate over an array or other iterable object.

for-in loop: Used to iterate through an object's keys.

In every loop there is a condition that must be met in order to exit the loop and there are also keywords like `break` and `continue` that can be used to prematurely exit or skip the loop.

It's important to ensure that the conditions used in loops and if/else statements are spelled correctly to avoid unexpected behavior. It's also important to ensure that the loops are properly designed to avoid them running endlessly and degrading performance.

Overall, control structures are an important part of programming in JavaScript and allow developers to control the execution of code based on specific conditions and iterations. By properly understanding and applying control structures, developers can write effective and error-free code.

functions

In JavaScript, functions are an important part of programming because they allow code to be broken down into smaller, reusable pieces. A function is a block of code that performs a specific task and optionally returns a value.

Functions are created in JavaScript using the "function" keyword and can optionally have a name. Here's an example of a simple function called "helloWorld" that returns "Hello, world!" outputs to the console:

```
function helloWorld() {  
  console.log("Hello, world!");  
}
```

Functions can also have parameters, which are used as placeholders for values passed to the function when it is invoked. Here's an example of a function called "greet" that takes a name as a parameter and returns "Hello, [name]!" outputs to the console:

```
function greet(name) {  
  console.log("Hello, " + name + "!");  
}
```

Functions can also return a value by using the "return" keyword. Here's an example of a function called "add" that takes two numbers as parameters and returns their sum:

```
function add(a, b) {  
  return a + b;  
}
```

With the ability to create and use functions, developers can break code into smaller, reusable parts, improving code readability and maintainability. Functions can also be used to avoid repeated code and to break the application into several smaller, easy-to-understand parts.

There are also certain types of functions in JavaScript, such as arrow functions and function expressions, that can be useful in certain situations. There are also different ways to invoke functions, such as using parentheses or using methods that refer to an object. There are many ways to use functions in JavaScript, and it's important to know the right techniques to write effective and clean code.

arrays and objects

In JavaScript, arrays and objects are two of the most important data structures used to store and organize data.

Arrays are a type of data structure that allow multiple values to be stored under a single name. An array is created by enclosing the values in square brackets [] and separating them with commas. Here is an example of an array named "colors" containing the values "red", "green" and "blue":

```
let colors = ["red", "green", "blue"];
```

Arrays also have indices, starting at 0, that represent the elements of the array. Thus, a specific element of the array can be accessed by typing the array name followed by the element index in square brackets []. For example, you can reach the first element of the example above with colors[0].

JavaScript also provides a variety of methods that can be applied to arrays, such as push() to add elements to the end of the array, pop() to remove the last element of the array, shift() to remove the first element of the array, unshift() to add elements to the beginning of the array, indexOf() to find the index of an element in the array, and slice() to extract part of the array.

Objects are another important data structure in JavaScript, allowing data to be stored in the form of key-value pairs. An object is created using braces {} and separating key-value pairs with commas. Here is an example of an object named "person" containing the values "name" with the value "John Doe" and "age" with the value 30:

```
let person = {name: "John Doe", age: 30};
```

To access an object's values, use the object's name followed by a period and the name of the key. For example, you can get the value of the name in the example above with person.name.

JavaScript also provides a variety of methods that can be applied to objects, such as Object.keys() to get all keys of an object, Object.values() to get all values of an object, Object.entries() to get all key-value pairs of an object, hasOwnProperty() to check whether a specific key exists in an object, and delete to delete a specific key-value pair from an object.

Another important concept when working with objects in JavaScript is inheritance. JavaScript allows an object to inherit from another object using the "prototype" keyword. You can transfer methods and properties from a "parent object" to a "child object" and thus reuse code.

Overall, arrays and objects in JavaScript are important tools for storing and organizing data. They allow data to be stored and managed in a structured manner and provide a variety of methods to access and manipulate the data. It is important to have a good understanding of the differences between arrays and objects, as well as their available methods and properties, in order to write effective and clean JavaScript code.

Dom Manipulation

The DOM (Document Object Model) is a programming interface that makes it possible to programmatically access and manipulate the structure and content of an HTML or XML page. DOM manipulation allows developers to create dynamic, user-interactive web pages by adding, removing, and modifying elements.

An important aspect of DOM manipulation is selecting elements on a page. This can be done using various methods such as `getElementById()`, `getElementsByTagName()` and `querySelector()`. For example, you can select the first element with ID "header" on a page with `document.getElementById("header")`.

Once selected, one can then use various properties and methods of the selected elements to manipulate them. For example, you can change the content of an element with the `innerHTML` property, change the size and position with the `style` property, or change the CSS class of an element with the `className` property.

Another important aspect of DOM manipulation is adding or removing elements on a page. This can be done using methods like `createElement()`, `createTextNode()` and `appendChild()`. For example, you can create a new element with the tag "p" and the content "Hello, world!" add using the following code:

```
let newParagraph = document.createElement("p");
let newText = document.createTextNode("Hello, world!");
newParagraph.appendChild(newText);
document.body.appendChild(newParagraph);
```

There are also other methods like `removeChild()`, `replaceChild()`, `insertBefore()` that can be used to add or remove elements on a page.

Overall, DOM manipulation enables the creation of dynamic, user-interactive web pages by programmatically selecting, manipulating, and adding or removing elements on a page. It is important to have a good understanding of the different methods and properties of the DOM and the hierarchy of elements on a page in order to write effective and clean code.

Events and event handling

JavaScript allows developers to respond to and take action on events such as mouse clicks, key presses, and page loads. The handling of these events is referred to as event handling.

An event is triggered by a specific behavior of a user or by a change in the DOM (Document Object Model). Examples of events are clicking a button, loading a page, or moving the mouse over an element.

Event handling in JavaScript is primarily done using event listeners. An event listener is a function that is called when a specific event occurs. They are bound to specific elements to respond to the event.

To create an event listener, use the `addEventListener()` or `attachEvent()` methods. Both methods expect two arguments: the type of event and the function to be called. Example:

```
button.addEventListener("click", function() {  
  console.log("Button was clicked");  
});
```

There is also the option of defining event handlers directly as a property of the HTML element. Example:

```
<button onclick="console.log('Button was clicked')">Click me</button>
```

Event handlers can also be removed using the `removeEventListener()` or `detachEvent()` methods. Both methods expect the same arguments as the methods for adding event listeners. Example:

```
button.removeEventListener("click", myFunction);
```

There are many different event types in JavaScript, some examples are "click", "mouseover", "keydown", "load" and "submit". It's important to choose the right event to provide the desired functionality.

Event handling is an important part of JavaScript development and allows developers to react and take action on user interactions and changes in the DOM.

Form validation and submit

Form validation is the process of checking user input for accuracy before the form is submitted. This is important to ensure that the data sent to the server is correct and complete.

Form validation can be performed in a number of ways, including client-side and server-side validation. Client-side validation is performed using JavaScript and allows input to be validated immediately without the need to send a request to the server. Server-side validation is performed using a programming language such as PHP or Ruby and occurs after the form has been submitted.

A simple way to validate form input is to use the "required" attribute. This attribute specifies that a specific field must be filled out before the form can be submitted. Example:

```
<input type="text" required>
```

There are also special input types like "email" and "number" that automatically perform certain validations. Example:

```
<input type="email">
```

JavaScript can be used for more complex validations. For example, the length of a password can be checked or whether certain fields match others. You can use event listeners for this, which react to the submission of the form and check the entries. Example:

```
form.addEventListener("submit", function(event) {  
  if (password.value.length < 8) {  
    event.preventDefault();  
    alert("The password must have at least 8 characters");  
  }  
});
```

Submitting a form can be triggered either by clicking a submit button or by pressing Enter. The default behavior of the form is to send the input to the specified server and reload the current page. However, JavaScript can be used to override the default behavior and send the input to a server via an AJAX request without reloading the page.

In summary, form validation and submit are important aspects of web development that help ensure data integrity and improve user experience. There are different ways to validate forms, from simple

attributes like "required" to advanced JavaScript validations. Also the form submission can be customized to send data to the server using AJAX instead of reloading the page.

AJAX and asynchronous programming

AJAX (Asynchronous JavaScript and XML) is a dynamic web page development technique that retrieves data asynchronously from the server without requiring the entire page to be reloaded. This makes it possible to update content on a page without the user having to leave the page.

Asynchronous programming refers to the way a program runs in which tasks are executed in parallel rather than in a sequential order. In contrast, synchronous tasks are executed in the same order in which they were asked.

In JavaScript, asynchronous programming can be implemented using callbacks, promises, and `async/await`. Callbacks are functions that are passed as arguments to other functions and are called when a specific action is complete. Promises are a more advanced concept that simplifies the management of asynchronous tasks and improves code readability. `Async/Await` is a syntax extension based on promises that allows writing asynchronous code in a synchronous manner.

AJAX requests are typically made using the `XMLHttpRequest` object. This object allows data to be retrieved from the server without requiring a page reload. Example:

```
var xhr = new XMLHttpRequest();  
  
xhr.open("GET", "data.txt", true);  
  
xhr.onreadystatechange = function() {  
  if (xhr.readyState === 4 && xhr.status === 200) {  
    console.log(xhr.responseText);  
  }  
};  
  
xhr.send();
```

In modern JavaScript there is also the `Fetch API` which provides a simpler and modern method for AJAX requests based on promises. Example:

```
fetch("data.txt")  
  .then(response => response.text())  
  .then(data => console.log(data))  
  .catch(error => console.log(error));
```

In both cases, the data is retrieved from the server asynchronously without the need for a page reload. AJAX makes it possible to update dynamic content on a page and improve user interaction without updating the entire page content, thereby improving user experience. It also allows the use of asynchronous programming to run tasks in parallel, thereby improving the application's performance.

An important aspect of using AJAX is error handling. It is important to ensure that the application can properly respond to errors that may occur during a request, such as an invalid URL or an unsuccessful HTTP status code.

Overall, AJAX provides a powerful way to provide dynamic content in web pages and improve user experience. Asynchronous programming allows tasks to run in parallel, thereby optimizing application performance. However, it is important to ensure that the application can properly respond to errors that may occur while using AJAX.

JavaScript libraries and frameworks (e.g. jQuery, AngularJS, React)

JavaScript libraries and frameworks are collections of ready-made code snippets and functions that help developers work faster and more effectively. They often provide advanced functionality that is not part of the core JavaScript, making it easier to develop complex applications.

One of the most popular JavaScript libraries is jQuery. It provides a simple and intuitive API (Application Programming Interface) to access and manipulate the DOM (Document Object Model) and makes it easy to handle events and animations. jQuery also has a large community and many plugins and extensions that allow developers to add complex functionalities quickly and easily.
Example:

```
$("#button").click(function() {  
  
    $("#p").hide();  
  
});
```

Another well-known framework is AngularJS. It is a complete framework for web application development and provides support for building Single Page Applications (SPA). It also offers strong support for data binding, which makes it easier to manage data in the application. AngularJS uses its own syntax called "Directives" which allows to define directives directly in the HTML code. Example:

```
<div ng-app="myApp" ng-controller="myCtrl">  
  
<p>{{ message }}</p>
```

</div>

Another well-known framework is React. It is a JavaScript framework for building user interfaces and is mainly used for developing single-page applications. It provides strong support for managing state and data binding, and allows developers to create custom components that can update themselves as data changes. Example:

```
const element = <h1>Hello, world!</h1>;
```

```
ReactDOM.render(element, document.getElementById("root"));
```

There are many other JavaScript libraries and frameworks to choose from depending on your needs and project.

Error handling and debugging

Error handling and debugging are important aspects of writing JavaScript code.

Error handling is the process of detecting errors in code and preventing them from affecting the program. A common method of error handling is to use exception handlers. These allow the programmer to write code that can react to possible exceptional situations instead of crashing the program. The "try" key value is used to run code that might throw an error, while the "catch" key value is used to catch the error and handle it appropriately.

Debugging is the process of finding and fixing errors in code. There are many tools and techniques that developers can use when debugging, including using breakpoints, printing values to monitor code, and using debugging tools in development environments. A common method is to use console output (`console.log`) in your code to monitor the value of variables and see where errors may be occurring.

It's important to include both error handling and debugging in your JavaScript code to ensure your program is stable and reliable.

Best practices and optimization

Best practices and optimization are important considerations when writing JavaScript code.

Best practices refer to code development recommendations and guidelines that help ensure that the code is readable, maintainable, and scalable. Some examples of best practices in JavaScript are:

Use 4 spaces indentation to improve code readability.

Use camelCase for variable names and function names.

Use consistent spelling for keywords, operators, and other syntax elements.

Use comments to explain and document your code.

Avoid using global variables as this can break code maintainability.

Optimization refers to improving code performance and speed. Some examples of optimizations in JavaScript are:

Use faster alternatives for commonly used functions (e.g. "for" instead of "forEach" for array iterations)

Avoid unnecessary calculations and queries

Avoid wasting memory by quickly removing variables and objects when they are no longer needed.

Use minified versions of libraries and frameworks to reduce loading times.

Use a task runner like Grunt or Gulp to automatically perform tasks like minifying files and monitoring changes.

It's important to incorporate both best practices and optimizations into your JavaScript code to ensure your program is stable, performant, and scalable. There are many resources and tools available to help developers improve and optimize their code.

Newer functions of ECMAScript (e.g. arrow functions, promises)

ECMAScript is the official specification for JavaScript maintained by the European Computer Manufacturers Association (ECMA). There are regularly new versions of ECMAScript that add new features and syntax. Some of the newer features in ECMAScript are Arrow functions and Promises.

Arrow functions are a new type of function syntax introduced in ECMAScript 6 (ES6). They allow functions to be defined in a shorter and more concise way. Compared to traditional functions, Arrow functions have a shorter syntax and automatically preserve the context of the this keyword.

```
// traditional function syntax
```

```
let add = function(a, b) {
```

```
  return a + b;
```



```
}
```

```
// Arrow function syntax
```

```
let add = (a, b) => {  
  return a + b;  
}
```

Promises are another new feature in ECMAScript 6 (ES6) that allow developers to manage asynchronous code execution. A promise represents an asynchronous operation and returns a result once the operation completes. Promises have two main states: "fulfilled" and "not fulfilled" (rejected). A promise can be used to manage asynchronous code execution by processing the results of asynchronous operations as they become available.

```
let promise = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve('Hello World!');  
  }, 1000);  
});  
promise.then((result) => {  
  console.log(result);  
});
```

These examples are just a few of the many new features introduced in recent versions of ECMAScript. There are many more such as template literals, destructuring, default and rest parameters, modules and many more. It is important to become familiar with these new features in order to improve and modernize your JavaScript code.

Building JavaScript-based applications (e.g. single page apps)

JavaScript-based applications, such as Single Page Apps (SPA), are a popular choice for web application development because they provide a smooth user experience by avoiding the need for server-side page reloads.

A SPA is a type of web application that loads the entire page only once, and then dynamically updates the content using JavaScript instead of reloading the entire page. This allows for a smoother user experience as the user does not have to wait for the entire page to reload to access new content.

To create a SPA, there are several steps to consider:

Choose a JavaScript framework or library, such as React, Angular, or Vue.js. These tools offer a variety of features and methods that make it easier to create a SPA.

Design the architecture of the application. This includes identifying the different parts of the application, such as the navigation, the views, and the controls.

Implement navigation. This involves creating routes that link the different views of the application.

Implement the views and controls. This involves using HTML, CSS, and JavaScript to create and style the application's views and controls.

Implement data binding. This involves using technologies such as AJAX to retrieve data from a server and be able to dynamically update the application's views.

Testing and Debugging. It is important to thoroughly test and troubleshoot the application before releasing it.

Publish the application. After the application has been tested and debugged, it can be published to a web server for users to use.

There are many resources and tools available to assist developers in building JavaScript-based applications. It's important to become familiar with the different frameworks and libraries to find the best solution for your needs. It's also important to become familiar with best practices and development standards to ensure code is readable, maintainable, and scalable.

Another important consideration when building JavaScript-based applications is performance. It is important to optimize the application to be fast and responsive to ensure a good user experience. This can be achieved by using technologies such as faster alternatives for commonly used functions, avoiding unnecessary calculations and queries, and avoiding unnecessary use of memory.

Overall, building JavaScript-based applications is a complex process that requires many steps and considerations. It's important to take the time to understand the needs of the application, choose the best technology, and ensure the code is well-structured, optimized, and tested.

imprint

This book was published under the **Creative Commons Attribution-NonCommercial-NoDerivatives (CC BY-NC-ND) license** released.



This license allows others to use and share the book for free as long as they credit the author and source of the book and do not use it for commercial purposes.

Author: Michael Lappenbusch

E-mail: admin@perplex.click

homepage: <https://www.perplex.click>

Release year: 2023