Javascript document

Certainly! ES6 (ECMAScript 2015) introduced several new features and enhancements to JavaScript. Here's a list of some key topics in ES6:

- **let and const:** Block-scoped variable declarations.
- **Arrow Functions:** Shorter syntax for writing function expressions.
- **Template Literals:** String literals with embedded expressions.
- **Default Parameters:** Default values for function parameters.
- **Rest Parameters:** Gather remaining function arguments into an array.
- **Spread Syntax:** Spread elements of an iterable (like arrays) into places like function arguments or array literals.
- **Destructuring Assignment:** Extracting values from arrays or objects and assigning them to variables in a concise syntax.
- **Object Literal Enhancements:** Shorthand for defining object properties and methods.
- **Classes:** Syntactical sugar over JavaScript's existing prototype-based inheritance.
- **Modules:** Built-in module system for organizing and reusing JavaScript code.
- **Promises:** Objects representing the eventual completion or failure of an asynchronous operation.
- **Symbol:** A new primitive data type for creating unique identifiers.
- **Iterators and Generators:** Protocol and function for iterating over data collections.
- **Map and Set:** New built-in data structures for storing key-value pairs and unique values, respectively.

- **Array and Object Methods:** New methods added to arrays and objects for common operations.
- **Async/Await:** Syntactic sugar for writing asynchronous code using promises.
- **Proxy and Reflect:** New objects providing low-level hooks for intercepting operations on objects.
- **Typed Arrays:** Arrays with a specified element type, useful for working with binary data.
- **BigInt:** A new primitive type for representing arbitrarily large integers.
- **String Methods:** New methods added to strings for common operations like searching and manipulation.
- **Object.setPrototypeOf and Object.getPrototypeOf:** Methods for setting and getting the prototype of an object.
- **String Interpolation:** Simplified syntax for concatenating strings and variables.
- **Array.find and Array.findIndex:** Methods for finding elements in arrays based on a condition.
- **Array.includes:** Method for checking if an array includes a certain value.
- **Array.from:** Method for creating a new array from an iterable object.
- **Array.of:** Method for creating a new array with a variable number of arguments.
- **Math enhancements:** New methods added to the Math object, such as Math.trunc, Math.sign, Math.log10, etc.
- **Number enhancements:** New static properties and methods added to the Number object, such as Number.isInteger, Number.isNaN, Number.parseFloat, etc.
- **Object.entries and Object.values:** Methods for extracting key-value pairs and values from objects.
- **Reflect API:** A set of methods for performing meta-programming operations on objects.
- **String padding:** Methods like padStart and padEnd for padding strings with spaces or other characters.
- **Unicode Support:** Enhanced support for Unicode characters and strings, including Unicode code point escapes.

- **Tail Call Optimization:** Support for optimizing tail-recursive function calls in certain environments.
- **RegExp Enhancements:** New RegExp features such as named capture groups, lookbehind assertions, etc.
- **Deprecated Features:** Some features from earlier versions of ECMAScript, like arguments.caller and arguments.callee, have been deprecated.
- **Proxying built-in objects:** Proxy objects that allow you to intercept and define custom behavior for fundamental operations like property lookup, assignment, and enumeration on objects.
- **Reflect API:** A collection of reflective methods on the Reflect object that provide the same functionality as some Object static methods and operators, but in a functional programming style.
- **Global object enhancements:** New methods and properties added to the global object, such as globalThis, globalThis.setTimeout, globalThis.parseInt, etc.
- **RegExp improvements:** Additional RegExp features and flags like the sticky (y) flag for sticky matching and the Unicode (u) flag for handling Unicode characters properly.
- **Function.prototype.toString():** Now returns the exact source code of the function, making it more useful for debugging and introspection.
- **ArrayBuffer and SharedArrayBuffer:** New types for working with raw binary data, allowing efficient manipulation and storage of binary data directly in memory.
- **Atomics:** A set of static methods in the Atomics object for performing atomic operations on shared memory locations, useful for concurrent programming.
- **Promise.prototype.finally():** A method added to the Promise prototype for specifying a callback to be executed when the Promise is settled, regardless of its outcome.
- **BigInt support in Number methods:** Additional methods added to the Number object that support BigInt values, such as Number.parseInt and Number.parseFloat.
- **Intl API enhancements:** Additional functionality and options added to the Intl API for internationalization and localization, including support for relative time formatting, calendar formatting, and more.

- **Optional chaining (?.):** An operator that allows you to safely access nested properties of an object without worrying about null or undefined values causing errors.
- **Nullish coalescing operator (??):** An operator that provides a default value when encountering null or undefined, unlike the logical OR operator which also considers falsy values like 0 or an empty string.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the given promises have either resolved or rejected, allowing you to handle a batch of promises and their outcomes collectively.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the given promises resolves, or rejects if all of the promises are rejected.
- **Logical Assignment Operators:** Compound assignment operators combined with logical operators like &&= (logical AND assignment) and ||= (logical OR assignment), providing a concise way to update variables based on conditions.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a substring within a string with another substring, similar to String.prototype.replace() but replacing all occurrences by default.
- **BigInt and Exponentiation Operator:** Support for BigInt literals and BigInt arithmetic operations, as well as the exponentiation operator (**) for calculating exponentiation in a concise manner.
- **GlobalThis:** A new global variable globalThis that provides a standardized way to access the global object (window in browsers, global in Node.js, etc.) across different JavaScript environments.
- **Private Class Fields and Methods:** Syntax for declaring private instance variables and methods within class definitions using the # prefix, ensuring encapsulation and data privacy.
- **Top-level await:** Support for using the await keyword at the top level of modules, allowing asynchronous module initialization and simplifying module code that depends on asynchronous operations.
- **String.prototype.trimStart() and String.prototype.trimEnd():** Methods for trimming whitespace from the beginning (trimStart()) or end (trimEnd()) of a string.
- **Array.prototype.flat() and Array.prototype.flatMap():** Methods for flattening nested arrays (flat()) and mapping and flattening arrays in one step (flatMap()).

- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Optional catch binding:** Syntax for omitting the catch parameter when it's not needed, allowing you to catch errors without binding them to a variable.
- **ArrayBuffer.transfer():** A method for transferring ownership of an ArrayBuffer from one object to another without copying its contents, improving performance when working with large binary data.
- **Intl.DateTimeFormat.formatRange():** A method for formatting date and time ranges in a locale-specific manner, providing a standardized way to format intervals between two dates or times.
- **Logical Assignment Operators:** Compound assignment operators combined with logical operators, such as &&= and ||=, providing a concise way to update variables based on conditions.
- **Numeric Separators:** Syntax for adding underscores (_) as separators within numeric literals to improve readability, especially for large numbers.
- **Error Stacks:** Error objects now include a stack property that contains information about the call stack at the time the error was thrown, aiding in debugging.
- **String.prototype.matchAll():** A method for obtaining an iterator of all matches for a regular expression within a string, providing detailed information about each match.
- **Promise.prototype.allSettled():** A method similar to Promise.all(), but it waits for all promises to settle (either resolve or reject) and returns an array of objects representing the outcome of each promise.
- **Promise.prototype.any():** A method that takes an array of promises and returns a promise that resolves as soon as one of the promises in the iterable resolves, or rejects if all of the promises are rejected.
- **BigInt:** A new primitive type in JavaScript for representing arbitrarily large integers, providing a way to accurately represent and perform arithmetic operations on large numbers.
- **Optional chaining (?.) and nullish coalescing (??) operators**:** These operators provide concise and safe ways to access nested properties and handle null or undefined values.

- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).
- **Named capture groups in regular expressions:** Syntax for naming capturing groups in regular expressions, making it easier to extract specific pieces of matched text.
- **Intl.Locale:** A constructor for creating locale objects that represent a specific language, region, and other locale-related information, providing a standardized way to work with locales and perform locale-sensitive operations.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Logical assignment operators:** Compound assignment operators combined with logical operators, such as &&= and ||=, providing a concise way to update variables based on conditions.
- **Top-level await:** Support for using the await keyword at the top level of module code, allowing modules to asynchronously initialize their dependencies.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **String.prototype.matchAll():** A method that returns an iterator of all matches for a regular expression within a string, providing detailed information about each match.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified substring within a string with another substring.
- **Array.prototype.at():** A method for accessing an element at a specified index in an array, providing a more concise and readable alternative to array indexing.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Optional catch binding:** Syntax for omitting the catch parameter in a try...catch statement when the caught error is not needed.
- **Array.prototype.sortStable():** A method for performing a stable sort (i.e., maintaining the order of equal elements) on an array, ensuring predictable sorting behavior.

- **Temporal:** A new global object for working with dates, times, and time zones in a modern and standardized way.
- **Private methods and accessors in classes:** Syntax for declaring private methods and accessors in class definitions, providing encapsulation and data privacy.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **Array.prototype.groupBy():** A method for grouping elements of an array based on a specified criterion, returning an object where keys are the result of applying the criterion and values are arrays of corresponding elements.
- **WeakRefs and FinalizationRegistry:** New features for managing weak references to objects and registering finalization callbacks for objects that are garbage-collected.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **String.prototype.matchAll():** A method that returns an iterator of all matches for a regular expression within a string, providing detailed information about each match.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified substring within a string with another substring.
- **Array.prototype.at():** A method for accessing an element at a specified index in an array, providing a more concise and readable alternative to array indexing.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Optional catch binding:** Syntax for omitting the catch parameter in a try...catch statement when the caught error is not needed.
- **Array.prototype.sortStable():** A method for performing a stable sort (i.e., maintaining the order of equal elements) on an array, ensuring predictable sorting behavior.
- **Temporal:** A new global object for working with dates, times, and time zones in a modern and standardized way.

- **Private methods and accessors in classes:** Syntax for declaring private methods and accessors in class definitions, providing encapsulation and data privacy.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **Array.prototype.groupBy():** A method for grouping elements of an array based on a specified criterion, returning an object where keys are the result of applying the criterion and values are arrays of corresponding elements.
- **WeakRefs and FinalizationRegistry:** New features for managing weak references to objects and registering finalization callbacks for objects that are garbage-collected.
- **GlobalThis:** A standardized way to access the global object (window in browsers, global in Node.js, etc.) across different JavaScript environments.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- ****Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings.
- **String.prototype.codePoints():** A method that returns an iterator of the Unicode code points of a string, allowing for more accurate and reliable string manipulation.
- **String.prototype.trimStart() and String.prototype.trimEnd():** Methods for trimming whitespace from the beginning (trimStart()) or end (trimEnd()) of a string.
- **String.prototype.matchAll():** A method that returns an iterator of all matches for a regular expression within a string, providing detailed information about each match.

- **Array.prototype.flat() and Array.prototype.flatMap():** Methods for flattening nested arrays (flat()) and mapping and flattening arrays in one step (flatMap()).
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **String.prototype.matchAll():** A method that returns an iterator of all matches for a regular expression within a string, providing detailed information about each match.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified substring within a string with another substring.
- **Array.prototype.at():** A method for accessing an element at a specified index in an array, providing a more concise and readable alternative to array indexing.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Optional catch binding:** Syntax for omitting the catch parameter in a try...catch statement when the caught error is not needed.
- **Array.prototype.sortStable():** A method for performing a stable sort (i.e., maintaining the order of equal elements) on an array, ensuring predictable sorting behavior.
- **Temporal:** A new global object for working with dates, times, and time zones in a modern and standardized way.
- **Private methods and accessors in classes:** Syntax for declaring private methods and accessors in class definitions, providing encapsulation and data privacy.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **Nullish coalescing operator (??):** An operator that provides a default value when encountering null or undefined, unlike the logical OR operator which also considers falsy values like 0 or an empty string.
- **BigInt support in Number methods:** Additional methods added to the Number object that support BigInt values, such as Number.parseInt and Number.parseFloat.

- **Optional chaining (?.):** An operator that allows you to safely access nested properties of an object without worrying about null or undefined values causing errors.
- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).
- **Named capture groups in regular expressions:** Syntax for naming capturing groups in regular expressions, making it easier to extract specific pieces of matched text.
- **Intl.Locale:** A constructor for creating locale objects that represent a specific language, region, and other locale-related information, providing a standardized way to work with locales and perform locale-sensitive operations.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Logical assignment operators:** Compound assignment operators combined with logical operators, such as &&= and ||=, providing a concise way to update variables based on conditions.
- **Top-level await:** Support for using the await keyword at the top level of module code, allowing modules to asynchronously initialize their dependencies.

- **Optional chaining (?.):** An operator that allows you to safely access nested properties of an object without worrying about null or undefined values causing errors.

- **Nullish coalescing operator (??):** An operator that provides a default value when encountering null or undefined, unlike the logical OR operator which also considers falsy values like 0 or an empty string.

- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).

- **BigInt support in Number methods:** Additional methods added to the Number object that support BigInt values, such as Number.parseInt and Number.parseFloat.

- **Named capture groups in regular expressions:** Syntax for naming capturing groups in regular expressions, making it easier to extract specific pieces of matched text.

- **Intl.Locale:** A constructor for creating locale objects that represent a specific language, region, and other locale-related information, providing a standardized way to work with locales and perform locale-sensitive operations.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Logical assignment operators:** Compound assignment operators combined with logical operators, such as &&= and ||=, providing a concise way to update variables based on conditions.
- **Top-level await:** Support for using the await keyword at the top level of module code, allowing modules to asynchronously initialize their dependencies.
- **String.prototype.trimStart() and String.prototype.trimEnd():** Methods for trimming whitespace from the beginning (trimStart()) or end (trimEnd()) of a string.
- **Array.prototype.flat() and Array.prototype.flatMap():** Methods for flattening nested arrays (flat()) and mapping and flattening arrays in one step (flatMap()).
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- **Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings.

- **WeakRefs and FinalizationRegistry:** New features for managing weak references to objects and registering finalization callbacks for objects that are garbage-collected.
- **String.prototype.codePoints():** A method that returns an iterator of the Unicode code points of a string, allowing for more accurate and reliable string manipulation.
- **Array.prototype.groupBy():** A method for grouping elements of an array based on a specified criterion, returning an object where keys are the result of applying the criterion and values are arrays of corresponding elements.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale.
- **Temporal:** A new global object for working with dates, times, and time zones in a modern and standardized way.
- **GlobalThis:** A standardized way to access the global object (window in browsers, global in Node.js, etc.) across different JavaScript environments.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified substring within a string with another substring.
- **String.prototype.matchAll():** A method that returns an iterator of all matches for a regular expression within a string, providing detailed information about each match.
- **Array.prototype.at():** A method for accessing an element at a specified index in an array, providing a more concise and readable alternative to array indexing.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Optional catch binding:** Syntax for omitting the catch parameter in a try...catch statement when the caught error is not needed.
- **Array.prototype.sortStable():** A method for performing a stable sort (i.e., maintaining the order of equal elements) on an array, ensuring predictable sorting behavior.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings.

- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **BigInt support in Number methods:** Additional methods added to the Number object that support BigInt values, such as Number.parseInt and Number.parseFloat.
- **Named capture groups in regular expressions:** Syntax for naming capturing groups in regular expressions, making it easier to extract specific pieces of matched text.
- **Nullish coalescing operator (??):** An operator that provides a default value when encountering null or undefined, unlike the logical OR operator which also considers falsy values like 0 or an empty string.
- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Top-level await:** Support for using the await keyword at the top level of module code, allowing modules to asynchronously initialize their dependencies.
- **Intl.DateTimeFormat.formatRange():** A method for formatting date and time ranges in a locale-specific manner, providing a standardized way to format intervals between two dates or times.
- **String.prototype.codePoints():** A method that returns an iterator of the Unicode code points of a string, allowing for more accurate and reliable string manipulation.
- **WeakRefs and FinalizationRegistry:** New features for managing weak references to objects and registering finalization callbacks for objects that are garbage-collected.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale.

- **Array.prototype.filterMap():** A method combining filtering and mapping operations on an array, allowing for concise and efficient transformations of array elements.
- **Array.prototype.unique():** A method for removing duplicate elements from an array, preserving the order of unique elements.
- **Intl.NumberFormat.formatToParts():** A method for formatting numbers into an array of objects representing parts of the formatted number, such as integer, fraction, and exponent.
- **Promise.prototype.finally():** A method for specifying a callback to be executed when a promise is settled (either resolved or rejected), regardless of its outcome.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified substring within a string with another substring.
- **String.prototype.trimStart() and String.prototype.trimEnd():** Methods for trimming whitespace from the beginning (trimStart()) or end (trimEnd()) of a string.
- **Array.prototype.flat() and Array.prototype.flatMap():** Methods for flattening nested arrays (flat()) and mapping and flattening arrays in one step (flatMap()).
- **Array.prototype.at():** A method for accessing an element at a specified index in an array, providing a more concise and readable alternative to array indexing.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Optional catch binding:** Syntax for omitting the catch parameter in a try...catch statement when the caught error is not needed.

- **Array.prototype.sortStable():** A method for performing a stable sort (i.e., maintaining the order of equal elements) on an array, ensuring predictable sorting behavior.
- **Temporal:** A new global object for working with dates, times, and time zones in a modern and standardized way.
- **GlobalThis:** A standardized way to access the global object (window in browsers, global in Node.js, etc.) across different JavaScript environments.
- **String.prototype.codePoints():** A method that returns an iterator of the Unicode code points of a string, allowing for more accurate and reliable string manipulation.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **BigInt support in Number methods:** Additional methods added to the Number object that support BigInt values, such as Number.parseInt and Number.parseFloat.
- **Named capture groups in regular expressions:** Syntax for naming capturing groups in regular expressions, making it easier to extract specific pieces of matched text.
- **Nullish coalescing operator (??):** An operator that provides a default value when encountering null or undefined, unlike the logical OR operator which also considers falsy values like 0 or an empty string.
- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale.
- **Array.prototype.filterMap():** A method combining filtering and mapping operations on an array, allowing for concise and efficient transformations of array elements.

- **Intl.DateTimeFormat.formatRange():** A method for formatting date and time ranges in a locale-specific manner, providing a standardized way to format intervals between two dates or times.
- **WeakRefs and FinalizationRegistry:** New features for managing weak references to objects and registering finalization callbacks for objects that are garbage-collected.
- **Intl.NumberFormat.formatToParts():** A method for formatting numbers into an array of objects representing parts of the formatted number, such as integer, fraction, and exponent.
- **String.prototype.padStart() and String.prototype.padEnd():** Methods for padding a string with spaces or other characters to reach a desired length from the start (padStart()) or end (padEnd()).
- **Array.prototype.unique():** A method for removing duplicate elements from an array, preserving the order of unique elements.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **String.prototype.padStart() and String.prototype.padEnd():** Methods for padding a string with spaces or other characters to reach a desired length from the start (padStart()) or end (padEnd()).
- **Intl.NumberFormat.formatToParts():** A method for formatting numbers into an array of objects representing parts of the formatted number, such as integer, fraction, and exponent.
- **Array.prototype.unique():** A method for removing duplicate elements from an array, preserving the order of unique elements.

- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale.
- **Array.prototype.filterMap():** A method combining filtering and mapping operations on an array, allowing for concise and efficient transformations of array elements.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- **WeakRefs and FinalizationRegistry:** New features for managing weak references to objects and registering finalization callbacks for objects that are garbage-collected.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale.
- **Intl.DateTimeFormat:** An object for formatting dates and times according to the conventions of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale.
- **Intl.Collator:** An object for performing locale-sensitive string comparison.
- **Intl.NumberFormat.formatToParts():** A method for formatting numbers into an array of objects representing parts of the formatted number, such as integer, fraction, and exponent.
- **Intl.DateTimeFormat.formatToParts():** A method for formatting dates and times into an array of objects representing parts of the formatted date or time, such as year, month, day, hour, minute, and second.

- **Intl.RelativeTimeFormat.formatToParts():** A method for formatting relative time into an array of objects representing parts of the formatted relative time, such as value and unit.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale.
- **Intl.DateTimeFormat:** An object for formatting dates and times according to the conventions of a specified locale.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale.
- **Intl.Collator:** An object for performing locale-sensitive string comparison.
- **Intl.NumberFormat.formatToParts():** A method for formatting numbers into an array of objects representing parts of the formatted number, such as integer, fraction, and exponent.
- **Intl.DateTimeFormat.formatToParts():** A method for formatting dates and times into an array of objects representing parts of the formatted date or time, such as year, month, day, hour, minute, and second.
- **Intl.RelativeTimeFormat.formatToParts():** A method for formatting relative time into an array of objects representing parts of the formatted relative time, such as value and unit.
- **Intl.Collator:** An object for performing locale-sensitive string comparison, allowing for sorting strings based on the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (e.g., "in 5 minutes", "2 days ago") according to the conventions of a specified locale.

- **Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale, allowing for localized representation of numeric values.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale, allowing for localized selection of plural forms based on numeric values.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale, allowing for localized formatting of lists with proper punctuation and conjunctions.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale, allowing for localized display of language and region names.
- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings, allowing for programmatic manipulation of locale information.
- **Intl.Collator.compare():** A method for comparing two strings based on the rules of a specified locale, returning a numerical value indicating the relative order of the strings.
- **Intl.Segmenter:** An object for breaking text into segments (such as words or sentences) according to the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Intl.Collator:** An object for performing locale-sensitive string comparison, allowing for sorting strings based on the rules of a specified locale.
- **Intl.RelativeTimeFormat.formatToParts():** A method for formatting relative time into an array of objects representing parts of the formatted relative time, such as value and unit.
- **Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale, allowing for localized representation of numeric values.
- **Intl.DateTimeFormat:** An object for formatting dates and times according to the conventions of a specified locale, allowing for localized display of date and time information.

- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale, allowing for localized selection of plural forms based on numeric values.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale, allowing for localized formatting of lists with proper punctuation and conjunctions.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale, allowing for localized display of language and region names.
- **Intl.Collator.compare():** A method for comparing two strings based on the rules of a specified locale, returning a numerical value indicating the relative order of the strings.
- **Intl.DateTimeFormat.formatToParts():** A method for formatting dates and times into an array of objects representing parts of the formatted date or time, such as year, month, day, hour, minute, and second.
- **Intl.RelativeTimeFormat.formatToParts():** A method for formatting relative time into an array of objects representing parts of the formatted relative time, such as value and unit.
- **Intl.Collator:** An object for performing locale-sensitive string comparison, allowing for sorting strings based on the rules of a specified locale.
- **Intl.RelativeTimeFormat:** An object for formatting relative time (e.g., "in 5 minutes", "2 days ago") according to the conventions of a specified locale.
- **Intl.NumberFormat:** An object for formatting numbers according to the conventions of a specified locale, allowing for localized representation of numeric values.
- **Intl.PluralRules:** An object for handling pluralization according to the rules of a specified locale, allowing for localized selection of plural forms based on numeric values.
- **Intl.ListFormat:** An object for formatting lists of items according to the conventions of a specified locale, allowing for localized formatting of lists with proper punctuation and conjunctions.
- **Intl.DisplayNames:** An object for formatting display names of languages, regions, scripts, and currencies according to the conventions of a specified locale, allowing for localized display of language and region names.

- **Intl.Locale:** An object representing a locale, providing information about its language, region, and other settings, allowing for programmatic manipulation of locale information.
- **Intl.Collator.compare():** A method for comparing two strings based on the rules of a specified locale, returning a numerical value indicating the relative order of the strings.
- **Intl.RelativeTimeFormat.formatRange():** A method for formatting ranges of relative time (e.g., "5 minutes ago - 10 minutes ago") according to the conventions of a specified locale.
- **Intl.PluralRules.select():** A method for selecting the appropriate plural category (e.g., "one", "few", "many") based on a numeric value and the rules of a specified locale.
- **Intl.DateTimeFormat.formatRange():** A method for formatting ranges of dates and times (e.g., "January 1, 2022 - January 5, 2022") according to the conventions of a specified locale.
- **Intl.ListFormat.formatToParts():** A method for formatting lists of items into an array of objects representing parts of the formatted list, such as items, punctuation, and conjunctions.
- **Intl.NumberFormat.formatRange():** A method for formatting ranges of numbers (e.g., "1,000 - 5,000") according to the conventions of a specified locale.
- **Intl.DisplayNames.of():** A method for retrieving the display name of a language, region, script, or currency for a given code or key.
- **Intl.NumberFormat.resolvedOptions():** A method for retrieving an object containing the resolved options used by an Intl.NumberFormat instance, such as the locale, numbering system, and formatting options.
- **Intl.DateTimeFormat.resolvedOptions():** A method for retrieving an object containing the resolved options used by an Intl.DateTimeFormat instance, such as the locale, calendar, numbering system, and formatting options.
- **Intl.ListFormat.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.ListFormat constructor is supported by the JavaScript engine.
- **Intl.RelativeTimeFormat.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.RelativeTimeFormat constructor is supported by the JavaScript engine.

- **Intl.Collator.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.Collator constructor is supported by the JavaScript engine.
- **Intl.PluralRules.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.PluralRules constructor is supported by the JavaScript engine.
- **Intl.NumberFormat.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.NumberFormat constructor is supported by the JavaScript engine.
- **Intl.DateTimeFormat.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.DateTimeFormat constructor is supported by the JavaScript engine.
- **Intl.DisplayNames.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.DisplayNames constructor is supported by the JavaScript engine.
- **Intl.ListFormat.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.ListFormat constructor is supported by the JavaScript engine.
- **Intl.RelativeTimeFormat.supportedLocalesOf():** A method for retrieving an array of locales for which the Intl.RelativeTimeFormat constructor is supported by the JavaScript engine.
- **Array.prototype.forEach():** A method for iterating over elements of an array and executing a callback function for each element.
- **Array.prototype.filter():** A method for creating a new array with all elements that pass a test implemented by a provided function.
- **Array.prototype.map():** A method for creating a new array populated with the results of calling a provided function on every element in the calling array.
- **Array.prototype.reduce():** A method for reducing the elements of an array to a single value, using a callback function and an initial value.
- **Array.prototype.reduceRight():** Similar to reduce(), but processes the array from right to left.
- **Array.prototype.some():** A method that tests whether at least one element in the array passes the test implemented by the provided function.
- **Array.prototype.every():** A method that tests whether all elements in the array pass the test implemented by the provided function.

- **Array.prototype.find():** A method that returns the first element in the array that satisfies the provided testing function, or undefined if no such element is found.
- **Array.prototype.findIndex():** A method that returns the index of the first element in the array that satisfies the provided testing function, or -1 if no such element is found.
- **Array.prototype.flat():** A method that creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.
- **Array.prototype.flatMap():** A method that first maps each element using a mapping function, then flattens the result into a new array.
- **Array.prototype.includes():** A method that determines whether an array includes a certain value among its elements, returning true or false as appropriate.
- **String.prototype.trim():** A method that removes whitespace from both ends of a string.
- **String.prototype.trimStart() and String.prototype.trimEnd():** Methods for trimming whitespace from the beginning (trimStart()) or end (trimEnd()) of a string.
- **Array.prototype.flat() and Array.prototype.flatMap():** Methods for flattening nested arrays (flat()) and mapping and flattening arrays in one step (flatMap()).
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Promise.any():** A method that returns a promise that resolves as soon as one of the provided promises in the iterable resolves, or rejects if all of the promises are rejected.
- **BigInt support in Number methods:** Additional methods added to the Number object that support BigInt values, such as Number.parseInt and Number.parseFloat.
- **Named capture groups in regular expressions:** Syntax for naming capturing groups in regular expressions, making it easier to extract specific pieces of matched text.
- **Nullish coalescing operator (??):** An operator that provides a default value when encountering null or undefined, unlike the logical OR operator which also considers falsy values like 0 or an empty string.
- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).

- **Optional chaining (?.):** An operator that allows you to safely access nested properties of an object without worrying about null or undefined values causing errors.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Object.entries():** A method that returns an array of a given object's own enumerable string-keyed property [key, value] pairs.
- **Object.fromEntries():** A method that transforms a list of key-value pairs into an object.
- **Object.getOwnPropertyDescriptors():** A method that returns all own property descriptors of a given object.
- **Object.values():** A method that returns an array of a given object's own enumerable property values.
- **Promise.prototype.finally():** A method that allows you to specify a function to be called when the promise is settled (either fulfilled or rejected).
- **String.prototype.matchAll():** A method that returns an iterator of all matches of a regular expression within a string.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified string with another string.
- **Dynamic import:** An ECMAScript proposal that allows for importing modules asynchronously at runtime.
- **Private class fields:** An ECMAScript proposal that allows class fields to be private, accessible only within the class.
- **Static class fields:** An ECMAScript proposal that allows for defining static properties directly on a class.
- **BigInt:** A built-in object that provides a way to represent whole numbers larger than 2^53 - 1, which is the largest number JavaScript can reliably represent with the Number primitive.
- **Optional chaining (?.):** An operator that allows you to safely access deeply nested properties of an object without causing errors if a property is null or undefined.

- **Nullish coalescing operator (??):** An operator that provides a default value if the left-hand side of the expression evaluates to null or undefined, but not for other falsy values like 0 or an empty string.
- **Logical assignment operators (||=, &&=):** Compound assignment operators combined with logical operators (|| and &&), allowing for concise conditional assignment.
- **Promise.allSettled():** A method that returns a promise that resolves after all of the provided promises have settled (either resolved or rejected), providing information about each promise's outcome.
- **Promise.race():** A method that returns a promise that fulfills or rejects as soon as one of the promises in an iterable fulfills or rejects, with the value or reason from that promise.
- **Array.prototype.at():** A proposed method that allows you to access an element at a specified index in an array, providing a safer alternative to array indexing, especially for negative indices.
- **String.prototype.replaceAll():** A method that replaces all occurrences of a specified substring with another substring, providing a more convenient and efficient way to perform global string replacement.
- **Logical nullish assignment (??=):** An operator for assigning a default value to a variable only if it is nullish (null or undefined).
- **Intl.RelativeTimeFormat:** An object for formatting relative time (such as "in 5 minutes" or "2 days ago") according to the conventions of a specified locale.
- **Array.prototype.copyWithin():** A method that shallow copies part of an array to another location in the same array and returns it without modifying its length.
- **Array.prototype.includes():** A method that determines whether an array includes a certain element, returning true or false as appropriate.
- **String.prototype.startsWith() and String.prototype.endsWith():** Methods that determine whether a string begins or ends with the characters of another string, returning true or false as appropriate.
- **String.prototype.repeat():** A method that constructs and returns a new string which contains the specified number of copies of the string on which it was called, concatenated together.
- **String.prototype.padStart() and String.prototype.padEnd():** Methods that pad the current string with a given string (repeated, if needed) so that the resulting string reaches a given length.

- **Array.prototype.find():** A method that returns the value of the first element in the array that satisfies the provided testing function. Otherwise, undefined is returned.
- **Array.prototype.findIndex():** A method that returns the index of the first element in the array that satisfies the provided testing function. Otherwise, -1 is returned.
- **Array.prototype.fill():** A method that fills all the elements of an array from a start index to an end index with a static value.
- **Promise.prototype.catch():** A method that specifies a function to be called when a promise is rejected.
- **Promise.prototype.finally():** A method that specifies a function to be called when a promise is settled (either fulfilled or rejected).
- **Object.seal():** A method that seals an object, preventing new properties from being added to it and marking all existing properties as non-configurable.
- **Object.freeze():** A method that freezes an object, preventing new properties from being added to it, existing properties from being removed, and all properties (including nested objects) from being modified.
- **Object.preventExtensions():** A method that prevents new properties from being added to an object, but allows existing properties to be modified or removed.
- **Object.isSealed():** A method that determines if an object is sealed (i.e., if new properties cannot be added and all existing properties are non-configurable).
- **Object.isFrozen():** A method that determines if an object is frozen (i.e., if it cannot be modified in any way).
- **Object.isExtensible():** A method that determines if an object is extensible (i.e., if new properties can be added to it).
- **Object.getOwnPropertyNames():** A method that returns an array of all property names (including non-enumerable properties) of a given object.
- **Object.getOwnPropertySymbols():** A method that returns an array of all symbol properties of a given object.
- **Object.getOwnPropertyDescriptor():** A method that returns the descriptor for a given property of an object.
- **Object.entries():** A method that returns an array of a given object's own enumerable string-keyed property [key, value] pairs.

- **Array.prototype.keys():** A method that returns an iterator of keys in the array, including both numeric and non-numeric keys.
- **Array.prototype.values():** A method that returns an iterator of values in the array.
- **Array.prototype.entries():** A method that returns an iterator of key/value pairs in the array.
- **Array.prototype@@iterator:** A method that returns a new Array Iterator object that contains the keys for each index in the array.
- **Object.keys():** A method that returns an array of a given object's own enumerable property names.
- **Object.values():** A method that returns an array of a given object's own enumerable property values.
- **Object.entries():** A method that returns an array of a given object's own enumerable string-keyed property [key, value] pairs.
- **Object.fromEntries():** A method that transforms a list of key-value pairs into an object.
- **Array.prototype.sort():** A method that sorts the elements of an array in place and returns the sorted array.
- **Array.prototype.reverse():** A method that reverses the elements of an array in place and returns the reversed array.
- **String.prototype.split():** A method that splits a string into an array of substrings based on a specified separator and returns the array.
- **String.prototype.slice():** A method that extracts a section of a string and returns it as a new string, without modifying the original string.
- **String.prototype.substring():** A method similar to slice(), but with differences in how negative arguments are handled.
- **String.prototype.substr():** A method that returns the characters in a string beginning at the specified location through the specified number of characters.
- **String.prototype.charAt():** A method that returns the character at the specified index in a string.
- **String.prototype.charCodeAt():** A method that returns the Unicode value of the character at the specified index in a string.
- **String.prototype.codePointAt():** A method that returns a Unicode code point value for the character at the specified index in a string.

- **String.prototype.endsWith():** A method that determines whether a string ends with the characters of a specified string.
- **String.prototype.startsWith():** A method that determines whether a string begins with the characters of a specified string.
- **String.prototype.includes():** A method that determines whether one string may be found within another string, returning true or false as appropriate.
- **String.prototype.trimStart() and String.prototype.trimEnd():** Methods that remove whitespace from the beginning (trimStart()) or end (trimEnd()) of a string.
- **String.prototype.repeat():** A method that constructs and returns a new string which contains the specified number of copies of the string on which it was called, concatenated together.
- **String.prototype.match():** A method that retrieves the matches when matching a string against a regular expression.
- **String.prototype.search():** A method that searches a string for a specified value, and returns the position of the first occurrence in the string.
- **String.prototype.toLocaleLowerCase():** A method that returns the calling string converted to lowercase, according to any locale-specific case mappings.
- **String.prototype.toLocaleUpperCase():** A method that returns the calling string converted to uppercase, according to any locale-specific case mappings.
- **String.prototype.normalize():** A method that returns the Unicode Normalization Form of the calling string value.
- **Array.prototype.reduceRight():** A method that applies a function against an accumulator and each value of the array (from right-to-left) to reduce it to a single value.
- **Array.prototype.some():** A method that tests whether at least one element in the array passes the test implemented by the provided function.
- **Array.prototype.every():** A method that tests whether all elements in the array pass the test implemented by the provided function.
- **Array.prototype.join():** A method that joins all elements of an array into a string, optionally separating them with a specified separator string.
- **Array.prototype.lastIndexOf():** A method that returns the last index at which a given element can be found in the array, or -1 if it is not present.

- **Array.prototype.fill():** A method that fills all the elements of an array from a start index to an end index with a static value.
- **Array.prototype.flat():** A method that creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.
- **Array.prototype.flatMap():** A method that first maps each element using a mapping function, then flattens the result into a new array.
- **Array.prototype.unshift():** A method that adds one or more elements to the beginning of an array and returns the new length of the array.
- **Array.prototype.shift():** A method that removes the first element from an array and returns that removed element.
- **Array.prototype.splice():** A method that changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.
- **Array.prototype.concat():** A method that returns a new array comprised of this array joined with other array(s) and/or value(s).
- **Array.prototype.includes():** A method that determines whether an array includes a certain element, returning true or false as appropriate.
- **Array.prototype.keys():** A method that returns an iterator of keys in the array, including both numeric and non-numeric keys.
- **Array.prototype.values():** A method that returns an iterator of values in the array.
- **Array.prototype.entries():** A method that returns an iterator of key/value pairs in the array.
- **Array.prototype@@iterator:** A method that returns a new Array Iterator object that contains the keys for each index in the array.
- **Object.keys():** A method that returns an array of a given object's own enumerable property names.
- **Object.values():** A method that returns an array of a given object's own enumerable property values.
- **Object.entries():** A method that returns an array of a given object's own enumerable string-keyed property [key, value] pairs.
- **Object.fromEntries():** A method that transforms a list of key-value pairs into an object.
- **Array.prototype.sort():** A method that sorts the elements of an array in place and returns the sorted array.

- **Array.prototype.reverse():** A method that reverses the elements of an array in place and returns the reversed array.
- **Typed Arrays:** Arrays where each element is of the same data type, such as Int8Array, Uint8Array, Float32Array, etc., providing efficient storage and manipulation of binary data.
- **Set:** A built-in object that allows you to store unique values of any type, whether primitive values or object references.
- **Map:** A built-in object that allows you to store key-value pairs where keys can be of any type, providing efficient lookups and iteration.
- **WeakSet:** A built-in object that allows you to store weakly held objects in a collection, allowing objects to be garbage collected if no other references to them exist.
- **WeakMap:** A built-in object that allows you to store key-value pairs where keys are weakly held objects, allowing keys to be garbage collected if no other references to them exist.
- **Proxy:** A built-in object that enables you to intercept and customize operations performed on objects, such as property access, assignment, enumeration, etc.
- **Reflect:** A built-in object that provides methods for interceptable JavaScript operations, such as property manipulation, function invocation, etc., allowing you to perform these operations in a more controlled manner.
- **Error Handling:** Techniques for handling errors in JavaScript, such as try-catch blocks, throwing custom errors, handling asynchronous errors, etc.
- **Asynchronous Programming:** Techniques for writing asynchronous code in JavaScript, such as using callbacks, promises, async/await, etc., to handle asynchronous operations effectively.
- **Event Loop:** The mechanism in JavaScript that allows for non-blocking asynchronous execution, ensuring that tasks are queued and processed efficiently.
- **Generator Functions:** Functions that can be paused and resumed, allowing you to generate a sequence of values lazily.
- **Iterators:** Objects that implement the Iterator protocol, allowing you to iterate over a sequence of values.
- **Async Iterators:** Iterators that work asynchronously, allowing you to iterate over a sequence of values produced asynchronously.

- **Async Generators:** Generator functions that produce values asynchronously, combining the features of generators and async functions.
- **Concurrency Models:** Different approaches to managing concurrent execution in JavaScript, such as using web workers, shared memory, or message passing.
- **Service Workers:** Scripts that run in the background of web applications, enabling features such as push notifications, background sync, and offline capabilities.
- **IndexedDB:** A low-level API for client-side storage of significant amounts of structured data, allowing you to store and retrieve objects using indexed keys.
- **WebSockets:** A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling real-time communication between a client and a server.
- **WebRTC:** A collection of protocols and APIs that allow browsers to communicate in real-time via peer-to-peer connections, enabling features such as video chat and file sharing.
- **WebAssembly:** A binary instruction format for a stack-based virtual machine, enabling you to run high-performance code written in languages such as C, C++, and Rust in the browser.
- **Blob:** An object representing a file-like object of immutable, raw data. Blobs can represent data that isn't necessarily in a JavaScript-native format.
- **URL:** A utility object providing methods for working with URLs, such as parsing, constructing, and resolving URLs.
- **FormData:** An object that allows you to easily construct a set of key/value pairs representing form fields and their values, which can then be sent via XMLHttpRequest or fetch.
- **Intersection Observer API:** A browser API that allows you to asynchronously observe changes in the intersection of a target element with an ancestor element or with a top-level document's viewport.
- **Resize Observer API:** A browser API that allows you to asynchronously observe changes to the size of a target element's content area or border box.
- **BroadcastChannel API:** An API that allows communication between browsing contexts (windows, tabs, iframes) that are at the same origin, enabling real-time communication between different parts of an application.
- **IndexedDB:** A low-level API for client-side storage of significant amounts of structured data, allowing you to store and retrieve objects using indexed keys.

- **Web Authentication API (WebAuthn):** An API that allows web applications to use hardware authenticators, such as fingerprint scanners or security keys, for user authentication.
- **Web Storage API:** An API that provides mechanisms by which browsers can store key/value pairs, similar to cookies but with a greatly enhanced capacity and improved user privacy.
- **Web Workers:** A browser API that allows you to run JavaScript code in background threads, enabling concurrent execution and improving performance for CPU-intensive tasks.
- **OffscreenCanvas:** An interface representing a canvas rendering context that can be used to draw graphics offscreen, allowing for efficient rendering in web workers or other background threads.
- **MediaDevices API:** An API that provides access to connected media input devices, such as cameras and microphones, allowing you to capture audio and video from the user's device.
- **WebRTC Data Channels:** A feature of the WebRTC API that allows for peer-to-peer communication of arbitrary data, enabling real-time data exchange between browsers without the need for a server.
- **HTML Drag and Drop API:** An API that allows you to implement drag and drop functionality in web applications, enabling users to drag elements and drop them onto target elements.
- **Pointer Events API:** An API that provides a unified way of handling input from pointing devices, such as mouse, pen, and touch input, allowing for more consistent and efficient event handling across different devices.
- **File System Access API:** An API that allows web applications to read and write files on the user's local file system, providing access to files and directories with the user's consent.
- **Clipboard API:** An API that allows web applications to interact with the system clipboard, enabling copy and paste operations between web applications and other applications on the user's device.
- **Network Information API:** An API that provides information about the user's network connection, such as connection type, speed, and whether the user is online or offline, enabling developers to optimize their applications based on network conditions.

- **Battery Status API:** An API that provides information about the user's device battery, such as charge level, charging status, and time until fully charged or discharged, enabling developers to optimize their applications to conserve battery life.
- **Device Orientation and Motion Sensors API:** An API that provides access to the device's orientation and motion sensors, such as accelerometer and gyroscope, enabling developers to create immersive experiences that respond to the user's movements.
- **Web Audio API:** An API that allows developers to create and manipulate audio content directly in the browser, enabling the creation of interactive audio applications, games, and multimedia experiences.
- **CanvasRenderingContext2D:** The 2D drawing context for the HTML Canvas element, which provides methods and properties for drawing shapes, text, and images on a canvas.
- **WebGL:** A JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins, providing high-performance graphics rendering capabilities.
- **WebVR and WebXR:** APIs that enable the creation of immersive virtual reality (VR) and augmented reality (AR) experiences directly in the browser, allowing users to explore virtual environments using VR headsets or AR-enabled devices.
- **Web MIDI API:** An API that enables web applications to communicate with MIDI devices, such as electronic musical instruments and MIDI controllers, enabling the creation of web-based music production and performance tools.
- **Payment Request API:** An API that allows web developers to request payment information from users through a standardized user interface, facilitating seamless and secure online payments in web applications.
- **Push API:** An API that allows web applications to receive push notifications from servers even when the application is not actively running, enabling real-time communication and engagement with users.
- **Screen Capture API:** An API that allows web applications to capture the contents of the user's screen or specific application windows, enabling features such as screen sharing and remote assistance.
- **Gamepad API:** An API that provides access to gamepad devices connected to the user's device, enabling developers to create web-based games that can be played using game controllers.

- **Device APIs:** APIs that provide access to various device capabilities and sensors, such as the Geolocation API for accessing the user's location, the DeviceOrientation API for accessing the device's orientation, and the Ambient Light Sensor API for accessing ambient light levels.
- **Animation API:** Libraries and frameworks for creating animations and transitions in web applications, such as CSS animations, GSAP (GreenSock Animation Platform), and Three.js for 3D animations.
- **Error Monitoring and Reporting:** Tools and services for monitoring and reporting errors and exceptions in web applications, such as Sentry, Rollbar, and Bugsnag.
- **Performance Optimization:** Techniques and tools for optimizing the performance of web applications, including code minification, bundle splitting, lazy loading, image optimization, and caching strategies.
- **Accessibility:** Best practices and guidelines for designing and developing accessible web applications that are usable by people with disabilities, including semantic HTML, proper use of ARIA attributes, keyboard navigation, and screen reader compatibility.
- **Internationalization and Localization:** Techniques for internationalizing and localizing web applications to support multiple languages and cultural preferences, including using language tags, date and number formatting, and translation libraries.
- **Progressive Web Apps (PWAs):** Web applications that use modern web technologies to provide a native app-like experience, including offline support, push notifications, and installation to the home screen.
- **Serverless Architecture:** Architectural patterns and frameworks for building web applications without managing server infrastructure, such as AWS Lambda, Azure Functions, and Google Cloud Functions.
- **GraphQL:** A query language and runtime for APIs that enables clients to request only the data they need, reducing over-fetching and under-fetching of data in web applications.
- **Web Components:** A set of standards for creating reusable and encapsulated custom HTML elements, including the Shadow DOM, Custom Elements, and HTML Templates.
- **Progressive Enhancement and Graceful Degradation:** Strategies for building web applications that work across a range of devices and browsers, including older browsers and devices with limited capabilities.

- **Microservices Architecture:** An architectural style that structures an application as a collection of loosely coupled services, each focused on a specific business function and communicating via lightweight protocols.
- **WebSockets:** A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling real-time communication between a client and a server.
- **Server-Side Rendering (SSR):** A technique for rendering web pages on the server side and sending the fully rendered HTML to the client, improving initial page load times and search engine optimization (SEO).
- **Content Delivery Networks (CDNs):** Distributed networks of servers that deliver web content to users based on their geographic location, improving performance and reliability by caching content closer to the user.
- **Web Authentication (WebAuthn):** An API that enables web applications to use hardware authenticators, such as fingerprint scanners or security keys, for user authentication, enhancing security and usability.
- **JSON Web Tokens (JWT):** A compact, URL-safe means of representing claims to be transferred between two parties, commonly used for authentication and authorization in web applications.
- **OAuth 2.0 and OpenID Connect:** Protocols for authentication and authorization, allowing users to grant third-party applications limited access to their resources without sharing their credentials.
- **Single Sign-On (SSO):** A mechanism that allows users to authenticate once and access multiple applications or services without having to authenticate again, improving user experience and security.
- **Containerization:** The practice of packaging an application and its dependencies into lightweight, portable containers that can run consistently across different computing environments, such as Docker containers.
- **Continuous Integration/Continuous Deployment (CI/CD):** Practices and tools for automating the process of integrating code changes, running tests, and deploying applications to production environments, improving development speed and reliability.
- **Server-Side JavaScript:** Using JavaScript on the server side with platforms like Node.js to build web servers, APIs, and other server-side applications.
- **Real-Time Web Applications:** Building applications that provide real-time updates to users without the need to refresh the page, often using technologies like WebSockets or Server-Sent Events.

- **Web Scraping and Automation:** Writing scripts to extract data from websites or automate tasks such as form filling and interaction with web applications.
- **Web Assembly (Wasm):** A binary instruction format for a stack-based virtual machine, enabling high-performance execution of code written in languages like C, C++, and Rust in web browsers.
- **Headless Browsers:** Browsers without a graphical user interface, often used for automated testing, web scraping, and other tasks where interaction with web pages is required.
- **Web Performance Optimization:** Techniques for improving the performance of web applications, such as optimizing images, minimizing HTTP requests, and implementing caching strategies.
- **Web Security:** Best practices for securing web applications against common threats, such as cross-site scripting (XSS), cross-site request forgery (CSRF), and injection attacks.
- **Web Analytics and Tracking:** Using tools like Google Analytics to collect data on user behavior and interactions with web applications, enabling insights into user engagement and website performance.
- **Web Accessibility Testing:** Evaluating web applications for compliance with accessibility standards like WCAG (Web Content Accessibility Guidelines), ensuring they are usable by people with disabilities.
- **Serverless Functions:** Writing and deploying functions that run in response to events triggered by cloud services, enabling event-driven architectures and reducing the need for server management.
- **Web Components:** A set of web platform APIs that allow you to create reusable custom elements with encapsulated functionality, enhancing modularity and reusability in web development.
- **Web Authentication (WebAuthn):** An API that enables web applications to use hardware authenticators, such as fingerprint scanners or security keys, for user authentication, improving security and usability.
- **Progressive Web Apps (PWAs):** Web applications that use modern web technologies to provide a native app-like experience, including offline support, push notifications, and installation to the home screen.
- **Content Security Policy (CSP):** A security feature that helps prevent cross-site scripting (XSS) attacks by restricting the sources from which resources like scripts, styles, and fonts can be loaded.

- **Subresource Integrity (SRI):** A security feature that allows you to ensure that the resources loaded by your web application have not been tampered with, by verifying their integrity using cryptographic hashes.
- **Web Workers:** A browser API that allows you to run JavaScript code in background threads, enabling concurrent execution and improving performance for CPU-intensive tasks.
- **IndexedDB:** A low-level API for client-side storage of significant amounts of structured data, allowing you to store and retrieve objects using indexed keys.
- **Background Sync API:** An API that allows web applications to defer actions until the user has a stable internet connection, enabling features like offline data synchronization and background updates.
- **Font Loading API:** An API that allows web developers to control the loading and rendering of web fonts, enabling better performance and user experience for typography on the web.
- **Media Capture and Streams API:** An API that allows web applications to access the user's media devices, such as cameras and microphones, enabling features like video conferencing and live streaming.
- **Graphics and Animation Libraries:** Libraries like D3.js for data visualization, PIXI.js for 2D rendering, and Three.js for 3D rendering, enabling developers to create complex graphics and animations in web applications.
- **Functional Programming:** A programming paradigm focused on writing functions that avoid changing state and mutable data, promoting immutability, and higher-order functions for building reusable and composable code.
- **Design Patterns:** Reusable solutions to common problems in software design, such as the Singleton pattern, Factory pattern, Observer pattern, and Module pattern, promoting maintainability and scalability in codebases.
- **Cryptography and Encryption:** Techniques for securing data and communications in web applications using algorithms like AES, RSA, and HMAC, and libraries like CryptoJS and ForgeJS for encryption and decryption.
- **Geolocation and Mapping:** APIs like the Geolocation API for accessing the user's location and mapping libraries like Leaflet and Mapbox for displaying maps and adding interactive features like markers and overlays.
- **WebRTC:** A set of APIs for real-time communication between web browsers, enabling features like video conferencing, voice calling, and peer-to-peer file sharing directly in web applications.

- **Performance Monitoring and Optimization:** Tools like Google Lighthouse, WebPageTest, and Chrome DevTools for measuring and optimizing the performance of web applications, including page load times, rendering performance, and network latency.
- **Code Quality and Testing:** Techniques for ensuring the quality of JavaScript code through practices like unit testing, integration testing, and static code analysis tools like ESLint and JSHint.
- **Package Managers and Build Tools:** Tools like npm and Yarn for managing dependencies and build tools like webpack, Parcel, and Rollup for bundling, minification, and optimization of JavaScript code.
- **Documentation and APIs:** Best practices for documenting JavaScript code using tools like JSDoc and Swagger, and publishing APIs for consumption by other developers or third-party applications.
- **Concurrency Models:** Different approaches to managing concurrent execution in JavaScript, such as using web workers, shared memory, or message passing.
- **Web Authentication API (WebAuthn):** An API that allows web applications to use hardware authenticators, such as fingerprint scanners or security keys, for user authentication.
- **Web Components:** A set of web platform APIs that allow you to create reusable custom elements with encapsulated functionality, enhancing modularity and reusability in web development.
- **Progressive Web Apps (PWAs):** Web applications that use modern web technologies to provide a native app-like experience, including offline support, push notifications, and installation to the home screen.
- **Web Assembly (Wasm):** A binary instruction format for a stack-based virtual machine, enabling high-performance execution of code written in languages like C, C++, and Rust in web browsers.
- **Headless Browsers:** Browsers without a graphical user interface, often used for automated testing, web scraping, and other tasks where interaction with web pages is required.
- **WebGL:** A JavaScript API for rendering interactive 2D and 3D graphics within any compatible web browser without the use of plug-ins, providing high-performance graphics rendering capabilities.
- **Service Workers:** Scripts that run in the background of web applications, enabling features such as push notifications, background sync, and offline capabilities.

- **IndexedDB:** A low-level API for client-side storage of significant amounts of structured data, allowing you to store and retrieve objects using indexed keys.
- **Serverless Architecture:** Architectural patterns and frameworks for building web applications without managing server infrastructure, such as AWS Lambda, Azure Functions, and Google Cloud Functions.
- **WebRTC (Web Real-Time Communication):** A set of APIs that enable real-time communication between web browsers, allowing for peer-to-peer audio, video, and data transfer without the need for plugins or additional software.
- **WebSockets:** A communication protocol that provides full-duplex communication channels over a single TCP connection, enabling real-time communication between a client and a server.
- **Server-Sent Events (SSE):** A standard allowing servers to push data to web clients over HTTP, enabling server-initiated updates to be sent to the browser in real-time.
- **Cross-Origin Resource Sharing (CORS):** A mechanism that allows resources on a web page to be requested from another domain outside the domain from which the first resource was served, enabling cross-origin communication securely.
- **Web Workers:** A browser API that allows you to run JavaScript code in background threads, enabling concurrent execution and improving performance for CPU-intensive tasks.
- **Fetch API:** A modern interface for fetching resources (such as JSON, text, or binary data) across the network, providing a more powerful and flexible replacement for the XMLHttpRequest (XHR) object.
- **Progressive Web Apps (PWAs):** Web applications that use modern web technologies to provide a native app-like experience, including offline support, push notifications, and installation to the home screen.
- **IndexedDB:** A low-level API for client-side storage of significant amounts of structured data, allowing you to store and retrieve objects using indexed keys.
- **Service Workers:** Scripts that run in the background of web applications, enabling features such as push notifications, background sync, and offline capabilities.
- **Web Components:** A set of web platform APIs that allow you to create reusable custom elements with encapsulated functionality, enhancing modularity and reusability in web development.

- **Machine Learning in JavaScript:** Libraries and frameworks like TensorFlow.js and ML5.js that enable machine learning and deep learning in the browser, allowing developers to build AI-powered web applications.
- **Progressive Enhancement and Graceful Degradation:** Strategies for building web applications that work across a range of devices and browsers, including older browsers and devices with limited capabilities.
- **Web Performance Optimization:** Techniques for improving the performance of web applications, such as optimizing images, minimizing HTTP requests, and implementing caching strategies.
- **Web Security:** Best practices for securing web applications against common threats, such as cross-site scripting (XSS), cross-site request forgery (CSRF), and injection attacks.
- **Web Accessibility (a11y):** Best practices and guidelines for designing and developing accessible web applications that are usable by people with disabilities, including semantic HTML, proper use of ARIA attributes, keyboard navigation, and screen reader compatibility.
- **Serverless Architecture:** Architectural patterns and frameworks for building web applications without managing server infrastructure, such as AWS Lambda, Azure Functions, and Google Cloud Functions.
- **Progressive Web Apps (PWAs):** Web applications that use modern web technologies to provide a native app-like experience, including offline support, push notifications, and installation to the home screen.
- **Web Assembly (Wasm):** A binary instruction format for a stack-based virtual machine, enabling high-performance execution of code written in languages like C, C++, and Rust in web browsers.
- **Headless Browsers:** Browsers without a graphical user interface, often used for automated testing, web scraping, and other tasks where interaction with web pages is required.
- **Web Analytics and Tracking:** Using tools like Google Analytics to collect data on user behavior and interactions with web applications, enabling insights into user engagement and website performance.
- **Augmented Reality (AR):** Utilizing libraries and frameworks like AR.js or WebXR to create immersive augmented reality experiences directly in the browser, allowing users to interact with virtual objects overlaid onto the real world.

- **Voice Recognition and Speech Synthesis:** Integrating speech recognition and synthesis APIs, such as Web Speech API, to enable voice-controlled interactions and text-to-speech capabilities in web applications.
- **WebXR:** A set of APIs that enable developers to create immersive virtual reality (VR) and augmented reality (AR) experiences directly in the browser, providing support for VR headsets and AR-enabled devices.
- **Natural Language Processing (NLP):** Leveraging NLP libraries and APIs, such as Natural or TensorFlow.js, to analyze and understand human language, enabling features like sentiment analysis, language translation, and chatbots.
- **Blockchain and Cryptocurrency:** Integrating blockchain technology and cryptocurrency APIs, such as Ethereum or Bitcoin APIs, to build decentralized applications (DApps) and facilitate secure transactions on the web.
- **Decentralized Identity (DID):** Implementing decentralized identity protocols, such as Decentralized Identifiers (DIDs) and Verifiable Credentials, to enable self-sovereign identity management and authentication on the web.
- **Data Visualization:** Utilizing libraries like Chart.js, D3.js, or Plotly.js to create interactive and visually engaging charts, graphs, and data visualizations for displaying complex datasets in web applications.
- **Cloud Computing and Serverless Architecture:** Leveraging cloud platforms like AWS, Azure, or Google Cloud to deploy and scale web applications, and utilizing serverless computing services like AWS Lambda or Azure Functions for backend logic.
- **Progressive Enhancement and Graceful Degradation:** Strategies for building web applications that provide a consistent user experience across a range of devices and browsers, including older browsers and devices with limited capabilities.
- **Localization and Internationalization:** Implementing localization and internationalization features in web applications to support multiple languages, regions, and cultural preferences, including language detection, translation services, and date/time formatting.