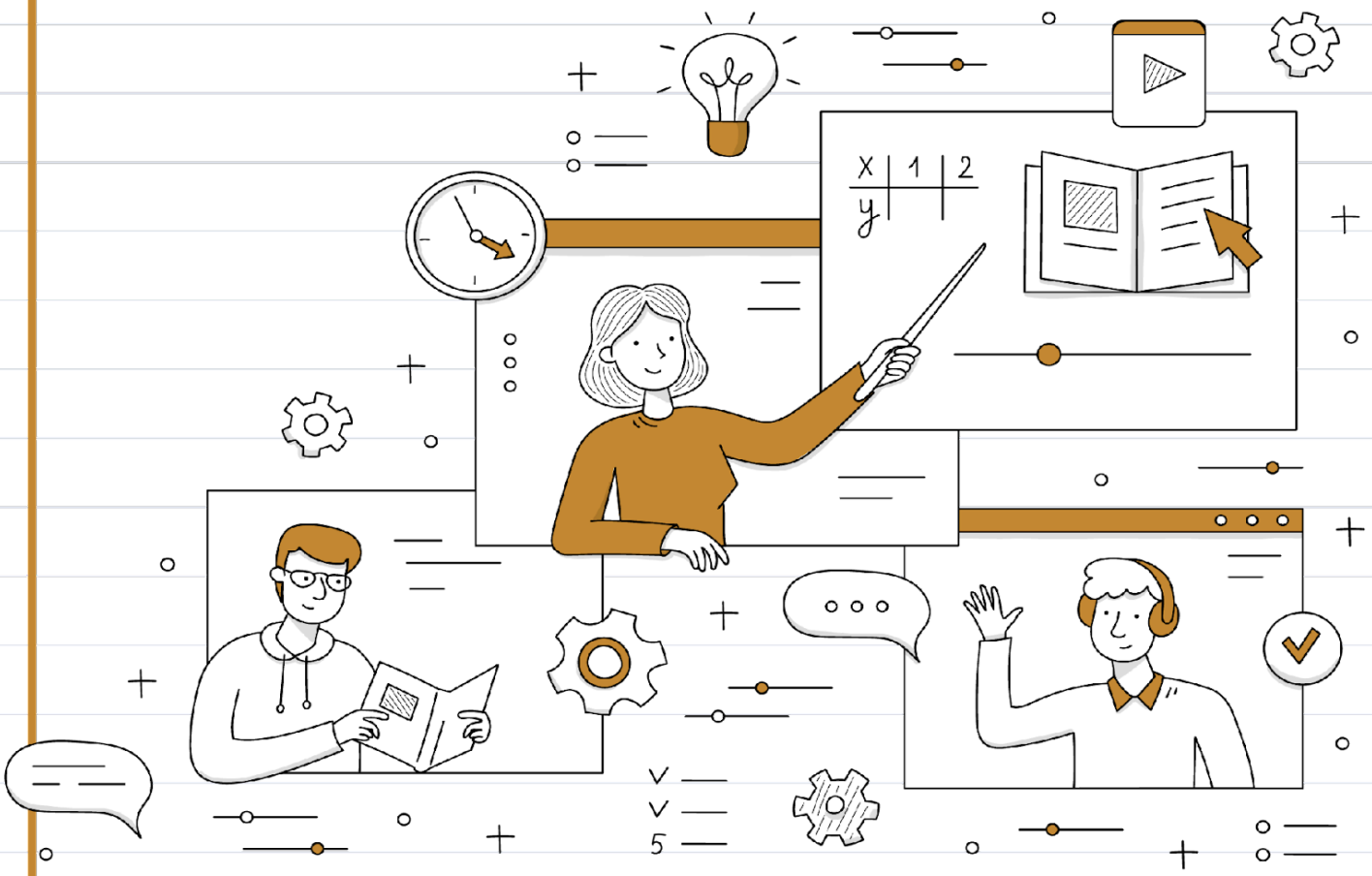
SUNSTONE

JavaScript

LECTURE 14 NOTES

JavaScript Subsets and Extensions



- **ECMAScript and its Versions**
  - ECMAScript (also known as ES) is a standardized scripting language specification developed by ECMA International. It defines the syntax, semantics, and features of scripting languages like JavaScript.
  - Here are the major versions of ECMAScript that have been released:
    1. **ECMAScript 1 (ES1):** The initial version was released in June 1997. It laid the foundation for the ECMAScript language.
    2. **ECMAScript 2 (ES2):** Released in June 1998, this version introduced minor changes to the language specification.
    3. **ECMAScript 3 (ES3):** Released in December 1999, ES3 brought significant improvements and became widely supported in web browsers. It introduced features such as regular expressions, try/catch exception handling, and more.
    4. **ECMAScript 4 (ES4):** This version was originally planned to be a major update with significant changes and new features. However, it was abandoned in favor of a more incremental approach due to disagreements within the ECMAScript committee.
    5. **ECMAScript 5 (ES5):** Released in December 2009, ES5 added several important features like strict mode, JSON support, and improved array manipulation.
    6. **ECMAScript 6 (ES6) / ECMAScript 2015:** Released in June 2015, ES6 introduced major enhancements to the language, including arrow functions, classes, modules, promises, and template literals.
    7. **ECMAScript 2016 (ES2016) / ECMAScript 7:** This version, released in June 2016, introduced a few new features such as the exponentiation operator (**), `Array.prototype.includes()`, and the async and await keywords for asynchronous programming.
    8. **ECMAScript 2017 (ES2017) / ECMAScript 8:** Released in June 2017, ES2017 introduced features such as async functions with shared memory, `Object.values()`, `Object.entries()`, String padding methods, and more.
    9. **ECMAScript 2018 (ES2018) / ECMAScript 9:** This version, released in June 2018, added new features such as asynchronous iteration, rest/spread properties, `Promise.prototype.finally()`, and more.

10. **ECMAScript 2019 (ES2019) / ECMAScript 10:** Released in June 2019, ES2019 introduced features like `Array.prototype.flat()`, `Array.prototype.flatMap()`, `Object.fromEntries()`, optional catch binding, and more.
11. **ECMAScript 2020 (ES2020) / ECMAScript 11:** Released in June 2020, ES2020 introduced features such as optional chaining (?.), nullish coalescing operator (??), BigInt type, dynamic import(), and more.
12. **ECMAScript 2021 (ES2021) / ECMAScript 12:** Released in June 2021, ES2021 introduced features such as `String.prototype.replaceAll()`, `Promise.any()`, logical assignment operators (||=, ??=, &&=), and more.

- **Browser-specific Extensions**
  - Different web browsers have implemented their own browser-specific extensions to provide additional functionality or experimental features. These extensions are typically not part of the ECMAScript specification and may not be supported across all browsers.
  - Here are a few examples of browser-specific extensions:
    1. **Document Object Model (DOM) extensions:** Browsers have implemented various DOM extensions to interact with web pages.

       For example,

       Internet Explorer introduced the `attachEvent` and `detachEvent` methods for event handling, while other browsers standardized the `addEventListener` and `removeEventListener` methods.
    2. **Internet Explorer (IE) specific features:** Internet Explorer had its own set of extensions, such as the `innerText` property for manipulating the text content of an element (as an alternative to the standard-compliant `textContent` property) and the `document.all` collections for accessing elements by their id attributes.
    3. **Vendor-specific CSS properties:** Browsers sometimes introduce vendor-specific CSS properties to experiment with new styling options before they become

standardized. For example, `-webkit-` prefix is used by WebKit-based browsers (such as Safari) and Blink-based browsers (such as Chrome) for experimental CSS features.

4. **Microsoft-specific features:** Microsoft has introduced several JavaScript extensions for their Edge browser, such as the `msRequestAnimationFrame` method for efficient animations, `msSaveBlob` for saving files, and `msCrypto` for cryptographic operations.

5. **Mozilla-specific features:** Mozilla Firefox has introduced its own set of JavaScript extensions, including the `MozOrientation` API for accessing device orientation information and the `mozRequestAnimationFrame` method for animation timing.

6. **Web APIs:** Browsers often provide additional web APIs beyond the standard JavaScript APIs. These APIs enable interactions with specific browser functionalities, such as geolocation, notifications, local storage, and more. These APIs are typically prefixed with the browser vendor's name, like `navigator.geolocation` or Notification.

- **Note:** Browser-specific extensions can limit the cross-browser compatibility of your code. When developing for the web, it's generally recommended to use standard ECMAScript features and check for browser support or use `polyfills` to ensure compatibility across different browsers.

- **Server-side JavaScript and Node.js**

  - Server-side JavaScript refers to the execution of JavaScript code on the server, as opposed to running it on the client's web browser. Traditionally, JavaScript was primarily used for client-side scripting, enabling interactive web pages. However, with the advent of server-side JavaScript, JavaScript can now be executed on the server, enabling developers to build full-stack web applications using a single language.

  - Node.js is a popular runtime environment for executing server-side JavaScript. It allows developers to run JavaScript code on the server, providing a range of features and capabilities for building scalable and high-performance web applications. Node.js uses an event-driven, non-blocking I/O model, which makes it efficient and suitable for handling concurrent requests.

- Here are some key features and benefits of using server-side JavaScript with Node.js:

    1. **Single language:** Node.js allows developers to use JavaScript for both client-side and server-side development, eliminating the need to learn a separate programming language for server-side tasks. This can lead to increased productivity and code reuse.

    2. **Asynchronous programming:** Node.js uses asynchronous programming techniques, such as callbacks, promises, and async/await, which allow for efficient handling of I/O operations. This makes it well-suited for building scalable and responsive web applications that can handle a large number of concurrent requests.

    3. **NPM ecosystem:** Node.js has a vast ecosystem of open-source libraries and modules available through the Node Package Manager (NPM). These modules cover a wide range of functionalities, including web frameworks, database drivers, authentication systems, and more, which can significantly speed up development.

    4. **Event-driven architecture:** Node.js follows an event-driven architecture, where events trigger the execution of specific code. This approach enables developers to build highly scalable applications that can handle many concurrent connections without incurring the overhead of thread-based concurrency.

    5. **Real-time applications:** Node.js is well-suited for building real-time applications, such as chat applications, collaborative tools, or live data streaming systems. Its event-driven nature and support for `WebSockets` enable bidirectional communication between clients and servers in real-time.

- Server-side JavaScript with Node.js provides developers with a powerful and unified platform for building web applications. It combines the flexibility and familiarity of JavaScript with a robust and efficient server-side runtime, making it a popular choice for web developers.

➢ Server-side JavaScript (SSJS)

- SSJS refers to the execution of JavaScript code on the server rather than in the client's web browser. It allows developers to use JavaScript as a programming language for server-side tasks, such as handling HTTP requests, interacting with databases, performing server-side rendering, and implementing business logic.

- There are several environments and platforms that support server-side JavaScript.

The most prominent one is Node.js, which is built on the V8 JavaScript engine. Node.js provides a runtime environment for executing JavaScript code outside the browser, making it possible to build scalable and efficient server applications.

- Here are some common use cases and benefits of server-side JavaScript:

    1. **Web application development:** With server-side JavaScript, developers can build full-stack web applications using a unified language. JavaScript can be used for both client-side and server-side tasks, resulting in better code reusability and a consistent development experience.

    2. **API development:** Server-side JavaScript is widely used for building APIs (Application Programming Interfaces). APIs allow different software systems to communicate and exchange data. JavaScript's versatility and ease of use make it a popular choice for creating robust and scalable API endpoints.

    3. **Serverless computing:** Serverless architecture, where developers can focus on writing code without managing server infrastructure, has gained popularity. Serverless platforms like AWS Lambda and Azure Functions support server-side JavaScript, allowing developers to build and deploy functions that can be triggered by events or API requests.

    4. **Data manipulation and processing:** JavaScript provides powerful tools for working with data. Server-side JavaScript can be used to process and manipulate data, perform computations, transform data formats, and implement complex algorithms, making it suitable for tasks such as data processing pipelines or data analytics.

    5. **Automation and scripting:** JavaScript's simplicity and versatility make it a great choice for automating repetitive tasks and creating scripts. With server-side JavaScript, you can automate server maintenance, perform batch processing, or create custom scripts for administrative tasks.

- **Note:** Server-side JavaScript is not limited to Node.js. There are other server-side environments and frameworks like Deno, which provide alternatives for executing JavaScript on the server. Additionally, some web servers, such as Apache and Nginx, offer plugins or modules that enables executing JavaScript code on the server side.

- Overall, server-side JavaScript opens up a wide range of possibilities for web developers, enabling them to leverage their existing JavaScript skills and create powerful, scalable, and efficient server applications.

➢ Server-side Node.js

- Server-side Node.js refers to using the Node.js runtime environment to execute JavaScript code on the server. Node.js is built on the V8 JavaScript engine and provides a powerful platform for building server applications.
- Here are some key aspects and benefits of using Node.js for server-side development:

1. **Asynchronous and Non-blocking I/O:** One of the key features of Node.js is its asynchronous, non-blocking I/O model. This means that Node.js can handle multiple concurrent connections without getting blocked, making it highly efficient for handling I/O-intensive operations, such as network requests or interacting with databases. It utilizes event-driven programming and callback functions to achieve this, allowing for scalable and responsive server applications.
2. **Vast Ecosystem:** Node.js has a rich ecosystem of packages and modules available through the Node Package Manager (NPM). These modules cover a wide range of functionalities, including web frameworks (e.g., Express.js, Koa.js), database connectors, authentication libraries, template engines, and more. The extensive NPM ecosystem makes it easy to find and integrate third-party libraries into your Node.js applications, saving development time and effort.
3. **Full-stack JavaScript:** With Node.js, you can use JavaScript as a programming language both on the client-side (in web browsers) and the server-side. This allows developers to use a single language throughout the entire web application stack, promoting code reuse and reducing the cognitive overhead of switching between different languages.
4. **Scalability:** Node.js is known for its ability to handle a large number of concurrent connections efficiently. The event-driven, non-blocking architecture of Node.js makes it well-suited for building scalable applications that can handle high traffic loads. It also supports clustering, which enables the utilization of multiple CPU cores for better performance and scalability.
5. **Real-time Applications:** Node.js is an excellent choice for building real-time applications and applications that require bidirectional communication between the server and clients. Its event-driven nature, along with libraries like Socket.IO, enables developers to create real-time chat applications, collaborative tools, multiplayer games, and other applications that rely on instant data updates.

6. **Microservices and API Development:** Node.js, with its lightweight and modular nature, is well-suited for building microservices architectures and developing APIs. Its ability to handle asynchronous I/O and its support for building RESTful APIs or GraphQL APIs make it a popular choice for creating scalable and flexible back-end services.

7. **Serverless Computing:** Node.js is also widely used in serverless computing platforms, such as AWS Lambda or Google Cloud Functions. With serverless architectures, developers can focus on writing functions without the need to manage server infrastructure. Node.js offers a lightweight and efficient runtime for serverless functions, making it a popular choice for building serverless applications.

- Node.js has a vibrant and active community, which provides ongoing support, tutorials, and resources for developers. Its popularity and widespread adoption have led to its use by many major companies and organizations, further solidifying its position as a leading server-side technology.

- **TypeScript and Other Superset Languages**

  - TypeScript is designed to enhance JavaScript's capabilities by providing optional type annotations, improved tooling, and support for modern ECMAScript features.

  - When using TypeScript, developers can write code using JavaScript syntax and take advantage of additional features that TypeScript offers. Some key benefits of TypeScript include:

    1. **Static Typing:** TypeScript introduces static typing to JavaScript, allowing developers to specify types for variables, function parameters, return values, and more. This helps catch type-related errors during development, provides better code documentation, and enables advanced code analysis and refactoring tools.

    2. **Improved Tooling:** TypeScript comes with a rich set of tooling support, including code editors with intelligent autocompletion, code navigation, and refactoring capabilities. These tools enhance the developer experience, improve code quality, and help identify potential issues early on.

3. **Early Error Detection:** With TypeScript's static typing, many common programming errors, such as type mismatches, undefined variables, or incorrect function usage, are caught at compile-time rather than at runtime. This helps prevent bugs and improves the overall robustness of the codebase.

4. **ECMAScript Support:** TypeScript supports the latest ECMAScript standards and provides features like classes, modules, arrow functions, async/await, and more. It allows developers to write modern JavaScript code while targeting specific ECMAScript versions for compatibility with different environments and browsers.

5. **Code Maintainability and Scalability:** TypeScript's static typing and compile-time checks contribute to better code maintainability and scalability. As projects grow in size and complexity, TypeScript helps in managing dependencies, enforcing code contracts, and providing a clear understanding of the codebase.

➢ TypeScript

- TypeScript is a superset of JavaScript, which means that any valid JavaScript code is also valid TypeScript code. TypeScript extends JavaScript by adding static typing, advanced type checking, and additional language features.
- Here are a few notable examples:

1. **Flow:** Flow is a static type checker for JavaScript developed by Facebook. It aims to provide type safety and error detection similar to TypeScript. Flow is opt-in, meaning you can gradually introduce type annotations to existing JavaScript code without requiring a complete rewrite.

2. **CoffeeScript:** CoffeeScript is a language that compiles into JavaScript. It offers a cleaner and more concise syntax compared to JavaScript, emphasizing readability and ease of use. CoffeeScript provides features like comprehensions, implicit returns, and class definitions with simpler syntax.

3. **Babel:** Babel is a JavaScript compiler that enables developers to write code in the latest ECMAScript version and compile it into older versions of JavaScript for wider browser compatibility. While not strictly a superset language, Babel supports using plugins and presets to transform and extend JavaScript syntax.

- These superset languages provide additional tools, features, and syntax options to enhance JavaScript development. The choice of which superset language to use depends on factors such as project requirements, team preferences, and existing codebase considerations. TypeScript, being widely adopted and supported, is particularly popular for its robust type system and comprehensive tooling ecosystem.

  How TypeScript can be used in a JavaScript project:

  1. **Setting up TypeScript:** To use TypeScript, you need to set up a TypeScript compiler in your project. You can install TypeScript globally on your machine or include it as a project dependency using npm or yarn.

  2. **Creating TypeScript Files:** TypeScript files have the .ts extension. You can start by renaming your existing JavaScript files with .ts extensions or create new TypeScript files. TypeScript files can contain both TypeScript-specific syntax and regular JavaScript code.

  3. **Type Annotations:** One of the main features of TypeScript is static typing. You can add type annotations to variables, function parameters, return types, and more. TypeScript supports built-in types like string, number, boolean, as well as complex types such as arrays, objects, and custom types/interfaces.

  4. **Type Checking:** The TypeScript compiler performs static type checking based on the type annotations in your code. It helps catch potential type-related errors during development and provides feedback on type mismatches or potential issues.

  5. **Compilation:** To convert TypeScript code to regular JavaScript code, you need to compile your TypeScript files. The TypeScript compiler (tsc) will transpile TypeScript into JavaScript code that can be understood and executed by JavaScript engines. The resulting JavaScript files can be used in your project.

  6. **JavaScript Interoperability:** TypeScript can work seamlessly with existing JavaScript code. You can include JavaScript files in your TypeScript project, and TypeScript will recognize and allow you to use the JavaScript code without any modifications. This allows you to gradually introduce TypeScript into an existing JavaScript codebase.

  7. **TypeScript-specific Features:** TypeScript provides additional language features beyond JavaScript, such as decorators, generics, enums, namespaces, and more. These features can enhance your development experience and help write more expressive and maintainable code.