Documentation Tool: JSDoc

Introduction:

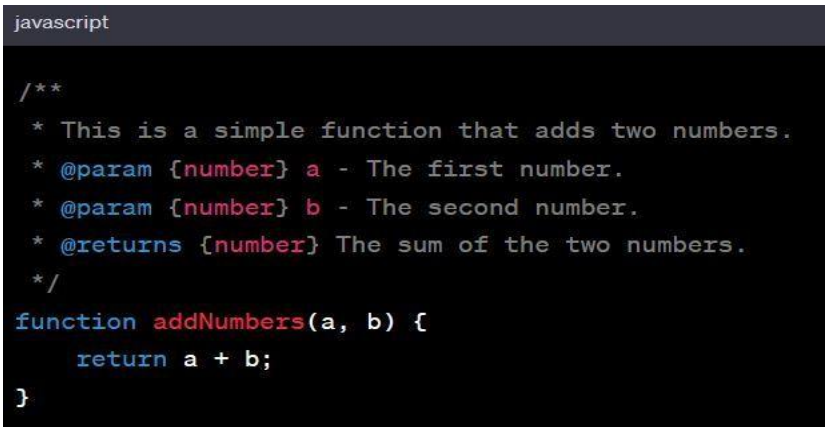
1. What is JSDoc?

JSDoc is a documentation tool for JavaScript that lets developers write structured comments directly inside their code. These comments describe functions, parameters, return values, classes, and other elements in a standardized format. From those comments, JSDoc can automatically generate clear, formatted documentation (usually in HTML) that explains how the code works and how to use it.

It also integrates with many IDEs, so while coding, developers get benefits like autocompletion, inline hints, and IntelliSense based on the JSDoc comments. In short, JSDoc makes JavaScript code easier to understand, maintain, and share by turning in-code annotations into professional documentation.

2. Basic Syntax:

JSDoc uses a syntax similar to JavaDoc and other documentation tools. It involves adding special comments to your code using a specific format. Here's a basic example:



3. JSDoc Tags and Their Purpose

a) **@param** – Documents a function parameter (name, type, description).
b) **@returns / @return** – Describes the return value of a function.
c) **@type** – Declares the data type of a variable, property, or constant.
d) **@description** – Adds a detailed description for functions, methods, or classes.
e) **@example** – Provides usage examples of the documented code.
f) **@see** – References related documentation or resources.
g) **@author** – States the author of the code.
h) **@version** – Indicates the version of a function, module, or file.
i) **@deprecated** – Marks a function or feature as outdated and warns developers not to use it.
j) **@throws / @exception** – Describes errors or exceptions a function might throw.
k) **@since** – Specifies when a function/class/module was introduced .
l) **@license** – Provides license details for the code.
m) **@file / @fileoverview** – Describes the overall purpose of the file or module.
n) **@constant** – Indicates that a variable should be treated as a constant.
o) **@readonly** – Marks a property as read-only.
p) **@private** – Marks a property or function as private (intended for internal use only).
q) **@protected** – Marks a property or function as protected (accessible to subclasses).
r) **@public** – Explicitly marks something as public.
s) **@class** – Defines a class.
t) **@constructor** – Marks a function as a constructor.
u) **@extends** – Indicates that a class inherits from another class.
v) **@implements** – States that a class implements an interface.
w) **@interface** – Defines an interface.

x) **@typedef** – Creates a custom type definition.
y) **@namespace** – Groups related items into a namespace.
z) **@module** – Describes a module (file or library).
aa) **@requires** – Indicates a dependency on another module or file.
bb) **@async** – Marks a function as asynchronous.
cc) **@callback** – Documents the signature of a callback function.

Installation:

1. How to install JSDoc?

Install JSDoc globally using this command:

```
npm install -g jsdoc
```

Or use the following command to install it for a single project:

```
npm install --save-dev jsdoc
```

Usage:

Video link:

1. Document:

To start documenting code, a comment has to be added starting with /** over each block of code the user wants to document: Modules, methods, classes, functions, etc.

It can be kept simple by just adding a description:

```
/**
 * Retrieves a user by email.
 */
const getByEmail = async (email) => {
    // ...
}
```

Or you can take full advantage of JSDoc using tags:

```
/**
 * Retrieves a user by email.
 * @async
 * @method
 * @param {String} email - User email
 * @returns {User} User object
 * @throws {NotFoundError} When the user is not found.
 */
const getByEmail = async (email) => {
    // ...
}
```

Remember, the more info you add to your comments, the more detailed your API documentation will be. But also, find the amount of detail that feels right to you**.** It's better to have all your code documented with only a few tags than to have only a few methods fully documented using all the tags because it was too much and you couldn't keep up.

2. Export

After adding the comments all that's left to do is generate your documentation website: **Export files or folders**

Simply call jsdoc and add the path to the file or folder.

```
jsdoc path/to/my/file.js
```

Advantages and Disadvantages of JSDoc:

1. Advantages:

Self-Documentation:

a. *Advantage:* JSDoc allows developers to embed documentation directly within the code. This makes the code self-documenting, providing details about functions, parameters, return types, and more.
b. *Why it matters:* Self-documenting code improves code readability and helps developers understand the purpose and usage of various elements without referring to external documentation.

Automated Documentation Generation:

c. *Advantage:* JSDoc facilitates the automatic generation of documentation using tools like the JSDoc command-line interface. This ensures that documentation stays up-to-date with the codebase.
d. *Why it matters:* Automated documentation saves time and reduces the risk of inconsistencies between code and documentation. It also encourages developers to keep documentation current.

IDE Integration:

e. *Advantage:* Many integrated development environments (IDEs), such as Visual Studio Code, support JSDoc. This integration provides features like autocompletion, inline documentation, and tooltips based on the JSDoc comments.
f. *Why it matters:* IDE support enhances the development experience, making it easier for developers to explore and use APIs while writing code.

Type Checking and IntelliSense:

g. *Advantage:* JSDoc includes support for specifying data types using @type tags. This information can be used by tools for static type checking and to improve IntelliSense features in IDEs.
h. *Why it matters:* Type annotations help catch potential errors early in the development process and improve the accuracy of autocompletion suggestions.

2. Disadvantages:

Overhead in Code Maintenance:

a. *Disadvantage:* Writing and maintaining JSDoc comments requires additional effort, especially in large codebases. Developers need to ensure that comments accurately reflect changes to the code.
b. *Consideration:* The benefits of self-documenting code need to be weighed against the time and effort spent on maintaining the documentation.

Potential for Outdated Documentation:

c. *Disadvantage:* If developers do not update JSDoc comments alongside code changes, the generated documentation may become outdated, leading to confusion.
d. *Consideration:* Establishing a process to review and update documentation during code reviews can help mitigate this risk.

Learning Curve for New Developers:

e. *Disadvantage:* For developers unfamiliar with JSDoc syntax and conventions, there may be a learning curve to effectively use and understand the documentation.
f. *Consideration:* Providing documentation or training on JSDoc usage within the development team can help new members get up to speed.

Potential for Redundancy:

g. *Disadvantage:* In some cases, JSDoc comments may duplicate information already present in the code, potentially leading to redundancy.
h. *Consideration:* Striking a balance between concise code and comprehensive documentation is important. Developers should avoid unnecessary repetition.

In summary, while JSDoc offers several advantages in terms of code documentation and automation, its effective use requires careful consideration of the potential drawbacks. Balancing the benefits and costs ensures that JSDoc enhances the development process rather than introducing unnecessary complexity.