Documentation Tool: JSDoc

Introduction:

1. **What is JSDoc?**

   JSDoc is a documentation tool and standard syntax for describing JavaScript code. It works by adding specially formatted comments (called JSDoc comments) directly above functions, classes, variables, or methods. These comments help both developers (by making the code easier to understand) and tools (to generate clean documentation websites or IDE hints).

2. **Basic Syntax:**

   JSDoc uses a syntax similar to JavaDoc and other documentation tools. It works by placing special block comments directly above your code, written in a specific format. These comments describe what the code does, its parameters, and its return values.

   Here's a basic example:

javascript

```javascript
/**
 * This is a simple function that adds two numbers.
 * @param {number} a - The first number.
 * @param {number} b - The second number.
 * @returns {number} The sum of the two numbers.
 */
function addNumbers(a, b) {
    return a + b;
}
```

3. **Tags and Types:**

JSDoc supports a variety of tags to document different aspects of your code. Some common tags include:

Function & Parameter Related

➔ @param {type} name – Documents a function parameter.

➔ @returns {type} – Documents the return value of a function.

➔ @throws {type} – Describes an error or exception a function might throw.

➔ @async – Marks a function as asynchronous.

➔ @callback – Describes a callback function type.

➔ @yields {type} – For generator functions, documents yielded values.

Variable & Type Related

➢ @type {type} – Specifies the data type of a variable or property.

➢ @const – Marks a variable as constant.

➢ @enum {type} – Documents an enumeration.

➢ @typedef {type} – Defines a custom type alias.

➢ @property {type} name – Documents an object's property.

Class & Object Related

❖ @class – Marks a function or definition as a class.

❖ @constructor – Marks a function as a constructor.

❖ @this {type} – Specifies the type of this inside a function.

❖ @extends {class} – Indicates inheritance from another class.

❖ @implements {interface} – Indicates that a class implements an interface.

❖ @override – Marks a method as overriding a superclass method.

Installation:

Prerequisites:

➔ Node.js and npm/yarn installed.
➔ Basic knowledge of JavaScript or TypeScript.

Installation Steps:

1. Navigate to your project: bash

```
cd my-jsdoc-project
```

2. Initialize package.json: bash

```
npm init -y
```

3. Install JSDoc: bash

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

4. Generate Documentation: bash

```
npx jsdoc example.js
```

5. View Documentation: Open out/index.html in a browser.

Usage:

Video link:

1. Documenting Functions JSDoc helps document function parameters, return values, and descriptions, improving code readability.

```
/**
* Adds two numbers.
*
* @param {number} a - The first number.
* @param {number} b - The second number.
* @returns {number} The sum of a and b.
*/
function add(a, b) {
return a + b;
}
```

2. Documenting Classes JSDoc can be used to document class definitions and their methods, including constructors.

```
/**
 * Represents a person.
 *
 * @class
 */
class Person {
    /**
     * Creates a person object.
     *
     * @param {string} name - The person's name.
     * @param {number} age - The person's age.
     */
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
}
```

3. Documenting Objects and Properties. One can describe objects and their properties with JSDoc annotations.

```
/**
 * Represents a car object.
 *
 * @typedef {Object} Car
 * @property {string} make - The manufacturer.
 * @property {string} model - The model name.
 * @property {number} year - The year of production.
 */

 /**
  * Displays information about a car.
  *
  * @param {Car} car - The car object.
  */
function showCarInfo(car) {
    console.log(`${car.make} ${car.model}, ${car.year}`);
}
```

4. Documenting Asynchronous Functions: Mark async functions with JSDoc annotations to clarify their behavior.

```
/**
 * Fetches data from an API.
 *
 * @async
 * @returns {Promise<Object>} The fetched data.
 */
async function fetchData() {
    const response = await fetch('https://api.example.com/data');
    return response.json();
}
```

5. Generating Documentation Once JSDoc comments have been written, one can use the JSDoc tool to generate HTML documentation from JavaScript files: bash

npm jsdoc your-file.js

So, in a summary, 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.

For 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
```

Outputs

Function

Global

Home

Global

add

Methods

`add(a, b) → {number}`

Adds two numbers.

Parameters:

Name	Type	Description
a	number	The first number.
b	number	The second number.

Source: index.js, line 8

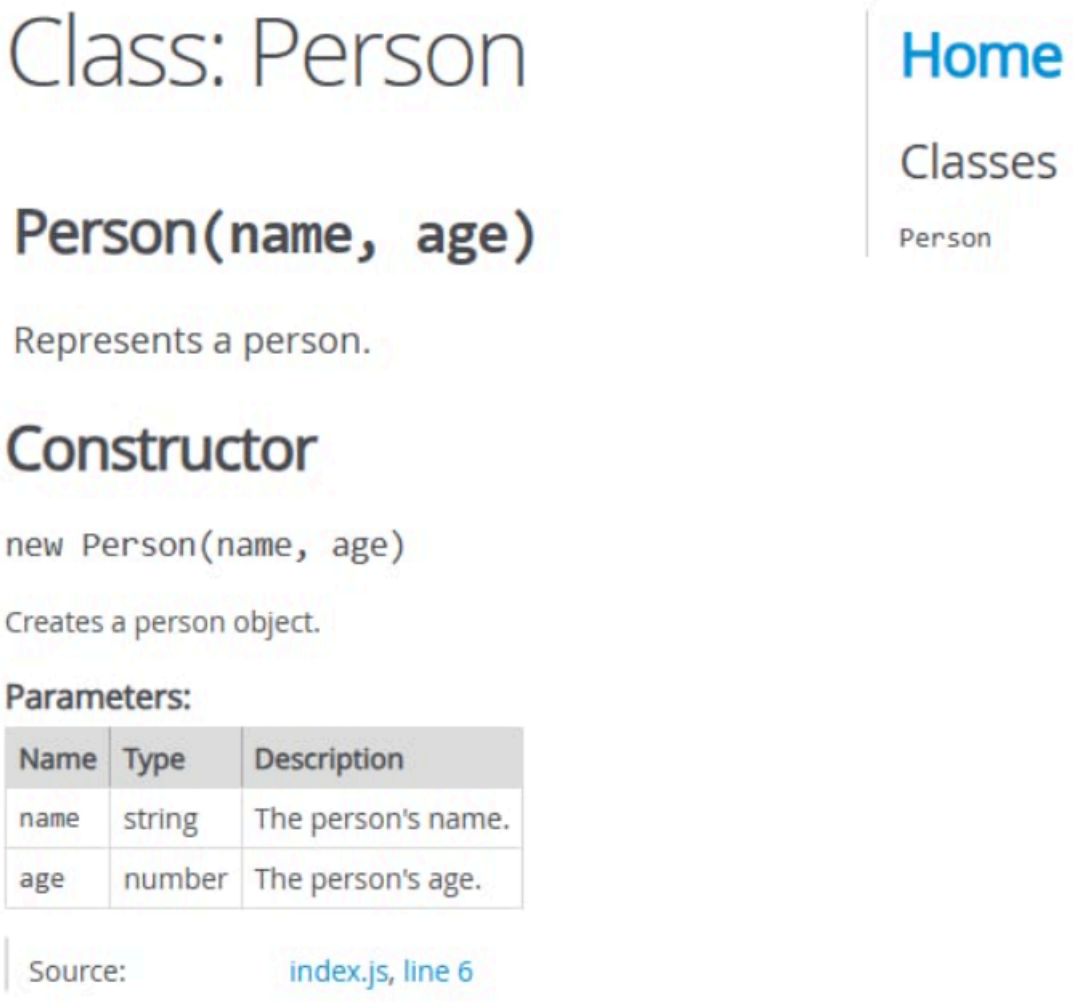
Returns:

The sum of a and b.

Type

number

Classes



Class: Person

Home

Classes

Person

Person(name, age)

Represents a person.

Constructor

new Person(name, age)

Creates a person object.

Parameters:

Name	Type	Description
name	string	The person's name.
age	number	The person's age.

Source: index.js, line 6

Objects

Global

showCarInfo

Methods

showCarInfo(car)

Displays information about a car.

Parameters:

Name	Type	Description
car	Car	The car object.

Source: index.js, line 15

Type Definitions

Car

Represents a car object.

Type:

- Object

Properties:

Name	Type	Description
make	string	The manufacturer.
model	string	The model name.
year	number	The year of production.

Source: index.js, line 1

Advantages and Disadvantages of JSDoc:

---

1. **Advantages of JSDoc**

   ➔ **Improved Code Readability**

   a) Makes code easier to understand for other developers (and even your future self).

   b) Provides clear descriptions of parameters, return values, and usage examples.

   ➔ **Better IDE Support**

   c) Editors like VS Code, WebStorm, IntelliJ use JSDoc to show tooltips, auto-completion, and type hints.

   d) Reduces the chances of calling functions incorrectly.

   ➔ **Automatic Documentation Generation**

   e) JSDoc can generate HTML documentation websites directly from comments.

   f) Saves time compared to writing external documentation manually.

   ➔ **Lightweight Type Checking**

   g) Without needing TypeScript, JSDoc lets you specify types (@param {number} etc.).

   h) Helps catch errors early and adds a form of type safety in plain JavaScript.

2. Disadvantages of JSDoc

   ➔ Extra Effort Required

a) Writing and maintaining JSDoc comments adds extra workload, especially for large projects.

b) Developers may skip updating docs when code changes.

➔ Can Become Outdated

c) If documentation is not maintained, JSDoc comments may mismatch the actual code, causing confusion.

➔ Limited Compared to TypeScript

d) JSDoc adds type hints but does not enforce strict typing like TypeScript.

e) Errors can still slip through at runtime.

➔ Learning Curve for Beginners

f) New developers may find tags and syntax (like @typedef, @property, @implements) overwhelming at first.

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.