Documenting JavaScript Code with JSDocs JSDoc is a documentation generator for JavaScript. It allows you to add documentation comments to your code, which can then be used to generate API documentation. It extracts comments written in the JSDoc format from the source code and generates HTML or other output formats. JSDoc is similar to Javadoc and phpDocumentor.

Why you need JSDocs

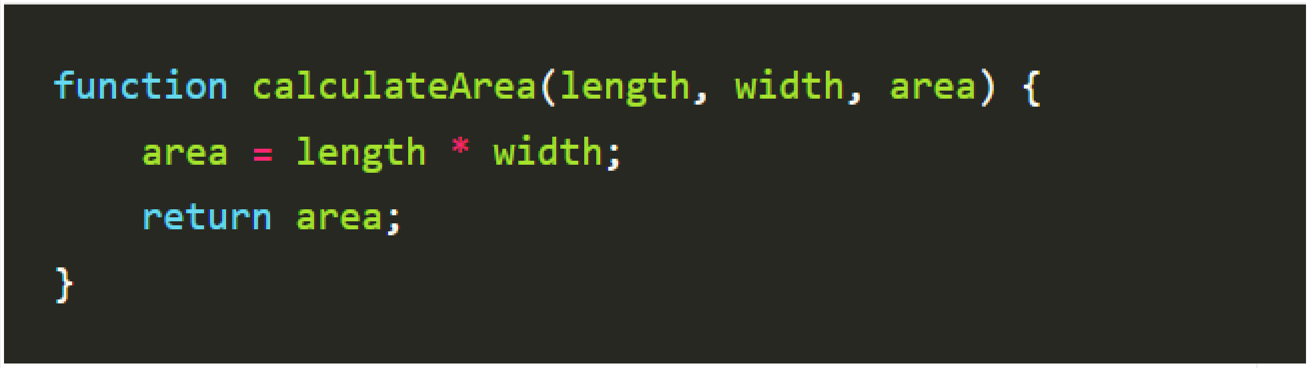
As we all know, writing code documentation can be very tedious. Therefore, we need something to cut time. The main idea behind JSDocs is to generate documentation for functions, classes, and methods.
The benefit of using JSDocs for your code is how easy it is to export to HTML. Moreover, it integrates well with IDE’s and code editors such as VS Code which has an extension for it.

Goal

By the end of this tutorial, the reader should have learned how to initialize JSDocs in a JavaScript program and use it in a real-life programming scenario. We will write code snippets that are documented using JSDocs to demonstrate this concept.

Insight

If you take a look at the function below, it is easy to tell what the function does as it speaks for itself.
From the function name to its parameters, we can tell that this function calculates the area of a rectangle by taking the length and width as parameters.



```
function calculateArea(length, width, area) {
    area = length * width;
    return area;
}
```

If we change the function and parameter names, this function could mean something else. We can use the JSDocs documentation API to help us describe the function better than it speaks for itself.
You add a line of comment starting with an asterisk * before the function name to

document it:

```
/**
* function to  calculate the area of a rectangle
*/
function calculateArea(length, width, area){
    area = length * width;
    return area;
}}
```

JSDocs Annotations

Writing the function name alone is not enough. With JSDocs annotation, we can make things more interesting by documenting parameters as well.
For each parameter, we take note of its type and the description of what it does in the code. The syntax for a function parameter is shown below.

```
/**
 *
 * @param {parameter type} parameter name - parameter descripti
 *
 */
```

- Type: parameter type may be a string, integer, array, floating-point, etc.
- Name: every parameter must have a name for referencing purposes in the code and when the function is called.
- Description: an explanation of what the parameter is about.

Getting Started/ Installation

To get started with JSDoc, you will need to install the JSDoc package. Create a folder and name it.

Head over to your terminal and run the following commands.

```
npm init -y
npm i- dev jsdocs
```

We need a config file to use JSDocs. In the root folder of the application, create a file named `jsodc.json` and paste the following snippet.

```
{
    "source": {
        "include": ["source"],
        "includePattern": ".js$",
        "excludePattern": "(node_modules/|docs)"
    },
    "plugins": ["plugins/markdown"],
    "templates": {
        "cleverLinks": true,
        "monospaceLinks": true
    },
    "opts": {
        "recurse": true,
        "destination": "./documentation/"
    }
}
```

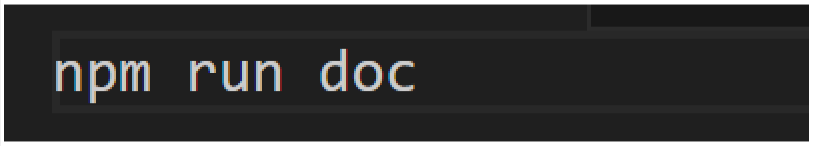
In the root folder, create a new folder named source and add a new file named `index.js` inside it. Here is where we are going to write the code whose documentation is to be generated.

Creating the Startup Script

Add the snippets below in the Script Object in the `package.json` file:

```
"doc": "jsdoc -c jsdoc.json"
```

Now run the command.

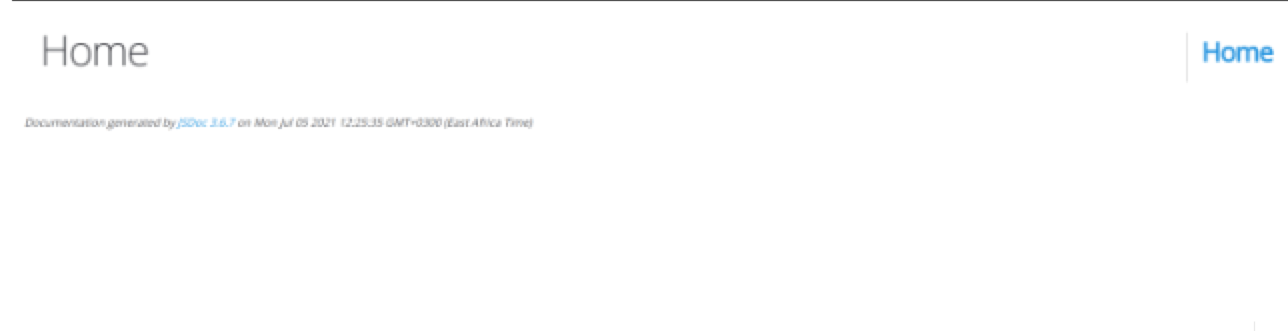


```
npm run doc
```

After which you'll see a folder called documentation in your root folder. At this time, your folder structure should be as shown below:

```
|-- jsdocs.json
|-- package-lock.json
|-- package.json
|-- documentation
|    |-- index.html
|    |-- fonts
|    |-- scripts
|    |-- style
|-- source
     |-- index.js
```

If you open your `index.html` file in the `documentation` folder, you should see the auto generated page where your documentation will go.



Home

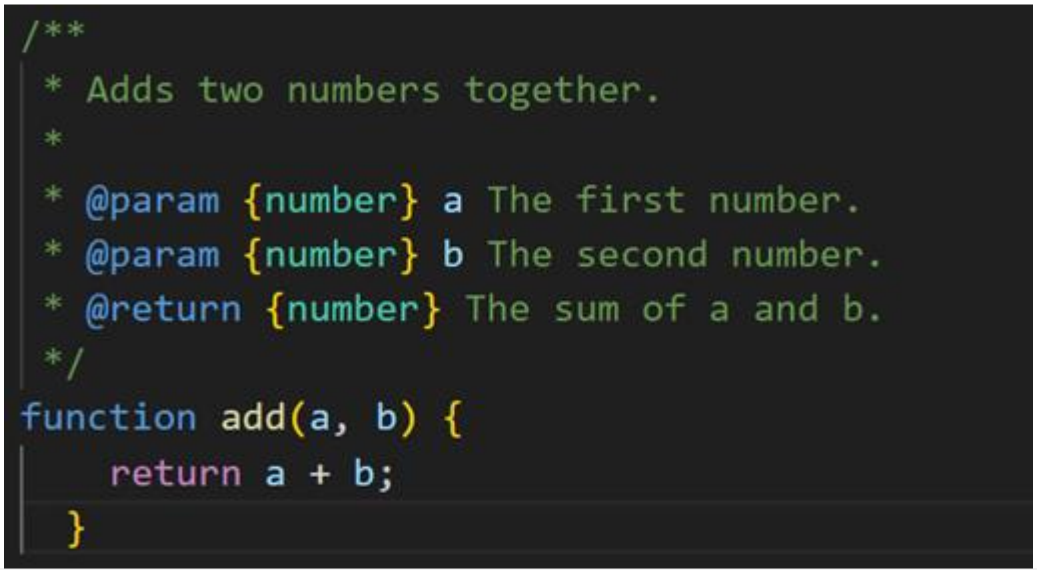
Home

Documentation generated by JSDoc 3.6.7 on Mon Jul 05 2021 12:25:35 GMT+0300 (East Africa Time)

Basic Tags

JSDoc uses a variety of tags to document your code. The most basic tags are:

- @name: This tag specifies the name of the item being documented.
- @description: This tag provides a description of the item.
- @return: This tag specifies the return type of a function.
- @param: This tag specifies the parameters of a function.
- @type: This tag specifies the type of a variable or property. For example, the following code documents a function called add:



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

Advanced Tags

JSDoc also supports a variety of advanced tags, such as:

- @abstract: This tag indicates that an item is abstract and must be implemented by a subclass.

- @access: This tag specifies the access level of an item (public, private, protected, or package-private).
- @alias: This tag creates an alias for an item.
- @async: This tag indicates that a function is asynchronous.
- @augments: This tag indicates that an item inherits from another item.
- @author: This tag specifies the author of an item.
- @borrows: This tag indicates that an item borrows something from another item.
- @class: This tag indicates that a function is a constructor.
- @classdesc: This tag provides a description of a class.
- @constant: This tag indicates that a variable is a constant.
- @constructs: This tag specifies the constructor for a class.
- @deprecated: This tag indicates that an item is deprecated and should not be used.
- @enum: This tag declares an enum.
- @event: This tag documents an event.
- @example: This tag provides an example of how to use an item.
- @extends: This tag indicates that an item inherits from another item.
- @external: This tag indicates that an item is defined in an external library