



Function Documentation

Basic Function Documentation

Use JSDoc comments `/** ... */` to document your functions.

```
/**
 * Description of the function.
 * @param {Type} paramName - Description of the parameter.
 * @returns {Type} Description of the return value.
 */
function myFunction(paramName) {
  // Function body
}
```

Example:

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

Optional Parameters

| | |
|---|--|
| <code>@param {string=} name</code> or <code>@param {string} [name]</code> | An optional parameter of type string. |
| <code>@param {number} [age=18]</code> | An optional parameter of type number with a default value of 18. |

Parameter Types

| | |
|---|--|
| <code>@param {string} name</code> | A parameter of type string. |
| <code>@param {number} age</code> | A parameter of type number. |
| <code>@param {boolean} isValid</code> | A parameter of type boolean. |
| <code>@param {object} options</code> | A parameter of type object. |
| <code>@param {Array<string>} names</code> | A parameter that is an array of strings. |

Type Definitions & Variables

Type Definitions (@typedef)

Use `@typedef` to define custom types for complex objects.

```
/**
 * Represents a point in 2D space.
 * @typedef {object} Point
 * @property {number} x - The x coordinate.
 * @property {number} y - The y coordinate.
 */

/**
 * Moves the point to a new location.
 * @param {Point} newPoint - The new coordinates.
 */
function movePoint(newPoint) {
  // Implementation
}
```

Shorthand Syntax:

```
/**
 * @typedef {{name: string, age: number}}
Person
 */

/**
 * Greets a person.
 * @param {Person} person - The person to greet.
 */
function greet(person) {
  // Implementation
}
```

Documenting Variables

| | |
|-----------------------|--|
| <code>@type</code> | Specifies the type of a variable. |
| <code>{number}</code> | |
| <code>@const</code> | Specifies the type of a constant variable. |
| <code>{string}</code> | |

```
/**
 * Examples of documenting variables with @type and @const.
 */
var message
= 'Hello';

/**
 * @const
 {number}
 */
const PI =
3.14;
```

Importing Types

You can import types from other modules using the `@typedef` and `@import` tags. (Note: `@import` may be TypeScript-specific).

```
/**
 * @typedef {import('./module').MyType} MyType
 */

/**
 * @param {MyType} value - A value of the imported type.
 */
function processValue(value) {
  // Implementation
}
```

Advanced JSDoc Tags

Other Useful Tags

| | |
|---|---|
| <code>@throws</code> <code>{Error}</code> | Documents that a function may throw an error. |
| <code>@async</code> | Indicates that a function is asynchronous. |
| <code>@private</code> | Marks a property or method as private. |
| <code>@deprecated</code> | Indicates that a function or property is deprecated. |
| <code>@see</code> | References another function, class, or external resource. |
| <code>@function</code> or <code>@method</code> | Explicitly defines a function or method. |

Code Examples with @example

Use the `@example` tag to provide usage examples for your functions or classes.

```
/**
 * Converts a string to uppercase.
 *
 * @param {string} str - The string to
convert.
 * @returns {string} The uppercase string.
 *
 * @example
 * toUpperCase('hello'); // returns 'HELLO'
 */
function toUpperCase(str) {
  return str.toUpperCase();
}
```

Linking to Other Documentation

You can create links to other parts of your documentation using the `{@link}` tag.

```
/**
 * A {@link Point} represents a coordinate in
2D space.
 *
 * @param {Point} point - The point to
process.
 */
function processPoint(point) {
  // Implementation
}
```

Class and Namespace Documentation

Class Documentation

Use the `@class` tag to document JavaScript classes.

```
/**
 * Represents a circle.
 * @class
 */
class Circle {
  /**
   * Creates a new Circle.
   * @param {number} radius - The radius of
the circle.
   */
  constructor(radius) {
    this.radius = radius;
  }

  /**
   * Calculates the area of the circle.
   * @returns {number} The area of the circle.
   */
  area() {
    return Math.PI * this.radius *
this.radius;
  }
}
```

Namespace Documentation

Use the `@namespace` tag to group related functions and classes.

```
/**
 * @namespace Geometry
 */
var Geometry = {};

/**
 * Calculates the distance between two points.
 * @param {Point} p1 - The first point.
 * @param {Point} p2 - The second point.
 * @returns {number} The distance between the
points.
 */
Geometry.distance = function(p1, p2) {
  // Implementation
};
```

Renaming with @alias

Use the `@alias` tag to rename a symbol. Prefer `@alias` over `@name`.

```
/**
 * @alias MyModule.myFunction
 */
function realFunctionName() {
  // Implementation
}
```