# JSDoc Cheatsheet

A quick reference guide for JSDoc, the industry standard for documenting JavaScript code. This cheatsheet provides syntax and examples for documenting functions, variables, types, and more.

## 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

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

### Parameter Types

| | |
|---|---|
| `@param {string} name` | A parameter of type string. |
| `@param {number} age` | A parameter of type number. |
| `@param {boolean} isValid` | A parameter of type boolean. |
| `@param {object} options` | A parameter of type object. |
| `@param {Array<string>} names` | 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

| | |
|---|---|
| `@type {number}` | Specifies the type of a variable. |
| `@const {string}` | Specifies the type of a constant variable. |

```
/**
 * @type {string}
 */
var message = 'Hello';

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

Examples of documenting variables with `@type` and `@const`.

### 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

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