Documentation Tool: JSDoc

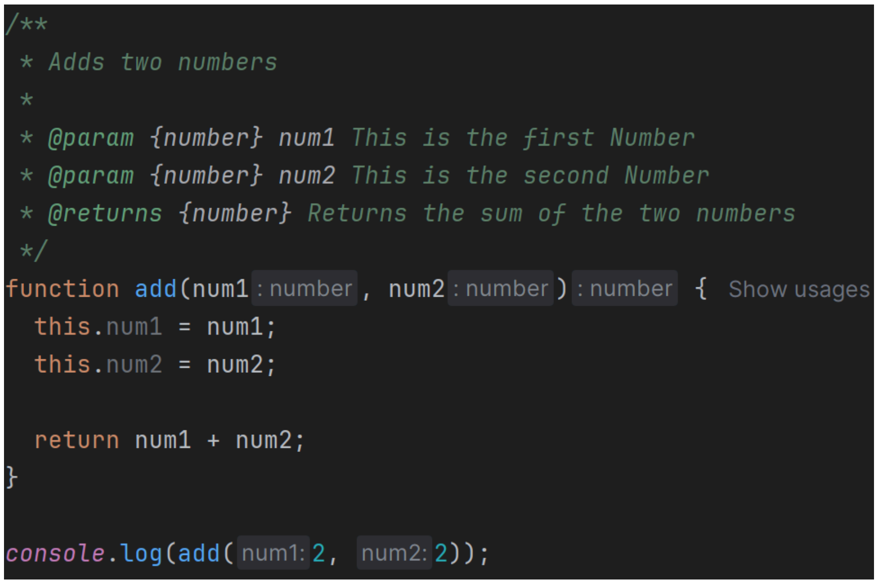
Introduction:

1. What is JSDoc?

JSDoc is a documentation syntax for JavaScript, similar to JavaDoc for Java or PyDoc for Python, enabling developers to annotate their code with comments that can later be transformed into a detailed documentation website. This tool not only facilitates better code comprehension but also fosters a culture of documentation-first, which can significantly reduce the learning curve for new team members and streamline the code review process. It also integrates with popular editors like VS Code and JetBrains, displaying documentation in-editor when developers hover over a specific field or method.

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:



```
/**
 * Adds two numbers
 *
 * @param {number} num1 This is the first Number
 * @param {number} num2 This is the second Number
 * @returns {number} Returns the sum of the two numbers
 */
function add(num1 : number , num2 : number ) : number  {  Show usages
  this.num1 = num1;
  this.num2 = num2;

  return num1 + num2;
}

console.log(add( num1: 2,  num2: 2));
```

3. Tags and Types:

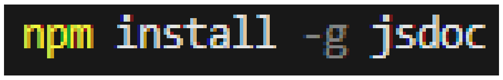
JSDoc supports a variety of tags to document different aspects of your code. Some common tags include:

- @param: Documents a function parameter.
- @returns: Documents the return value of a function.
- @type: Specifies the data type of a variable.
- @description: Provides a detailed description of the code.
- @example: Includes example code.
- @see: Adds references to other resources or related documentation.

Installation:

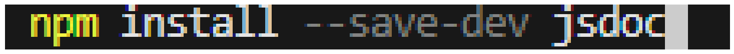
1. How to install JSDoc?

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
```

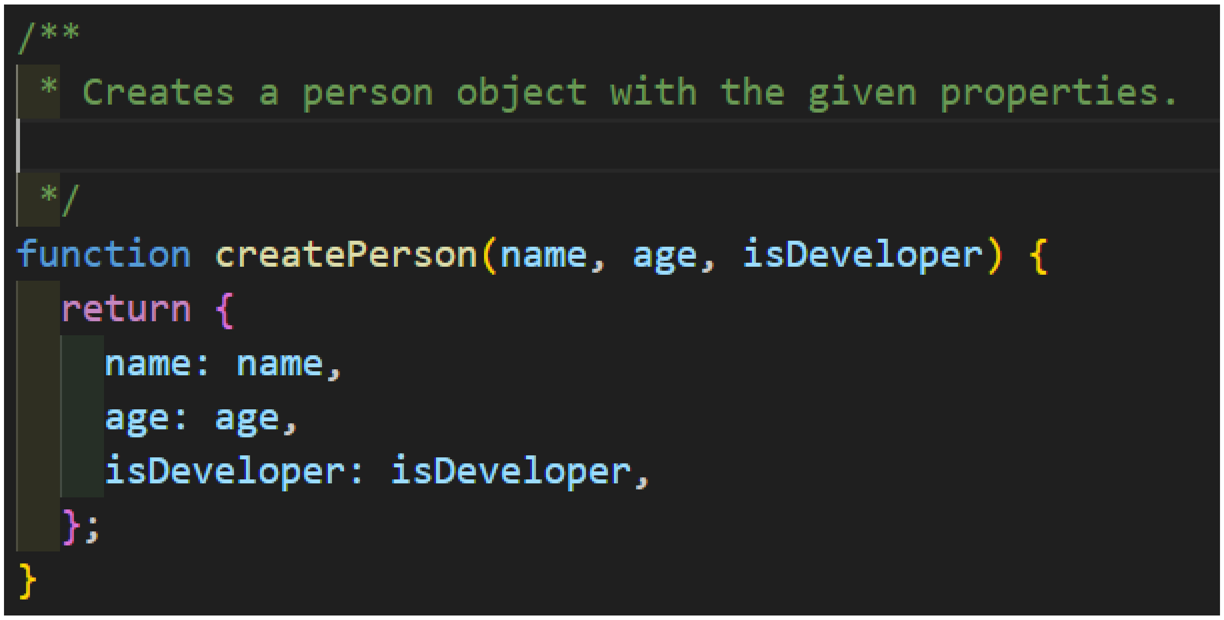
Usage:

Video link:

1. Document:

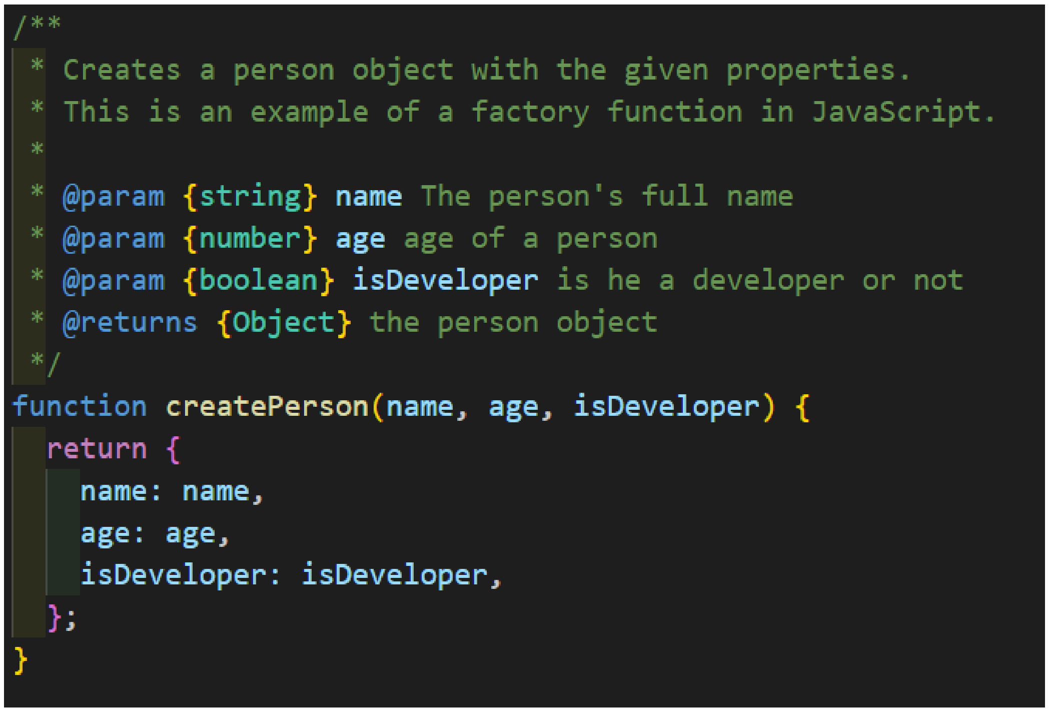
To start documenting code, a comment has to be added starting with /** over each block of code the user wants to document: Modules, methods, classes, functions, etc.

It can be kept simple by just adding a description:



```
/**
 * Creates a person object with the given properties.

 */
function createPerson(name, age, isDeveloper) {
  return {
    name: name,
    age: age,
    isDeveloper: isDeveloper,
  };
}
```

Or you can take full advantage of JSDoc using tags:



```
/**
 * Creates a person object with the given properties.
 * This is an example of a factory function in JavaScript.
 *
 * @param {string} name The person's full name
 * @param {number} age age of a person
 * @param {boolean} isDeveloper is he a developer or not
 * @returns {Object} the person object
 */
function createPerson(name, age, isDeveloper) {
  return {
    name: name,
    age: age,
    isDeveloper: isDeveloper,
  };
}
```

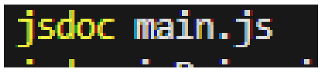
Remember, the more info you add to your comments, the more detailed your API documentation will be. But also, find the amount of detail that feels right to you**.** It's better to have all your code documented with only a few tags than to have only a few methods fully documented using all the tags because 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**

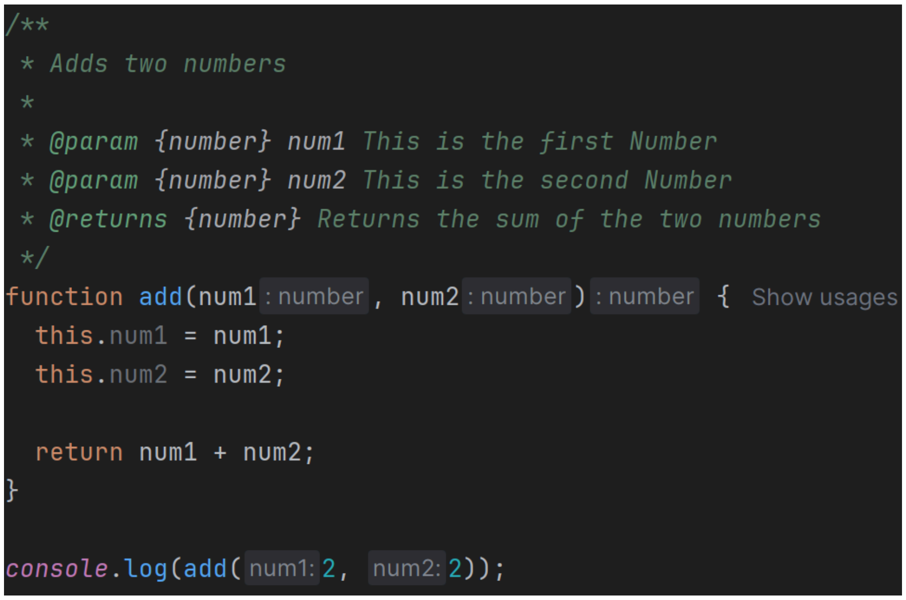
Simply call jsdoc and add the path to the file or folder.



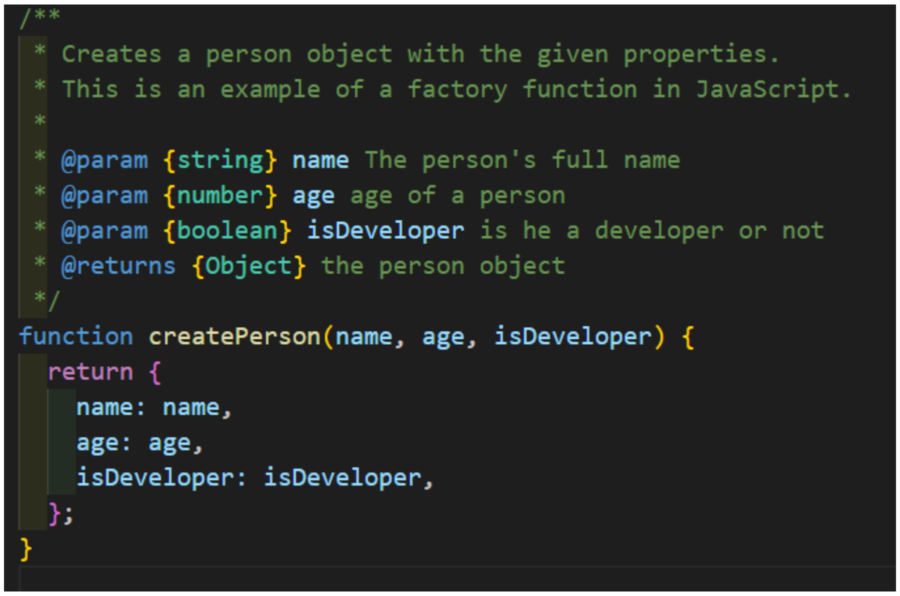
```
jsdoc main.js
```

3. Source Code:

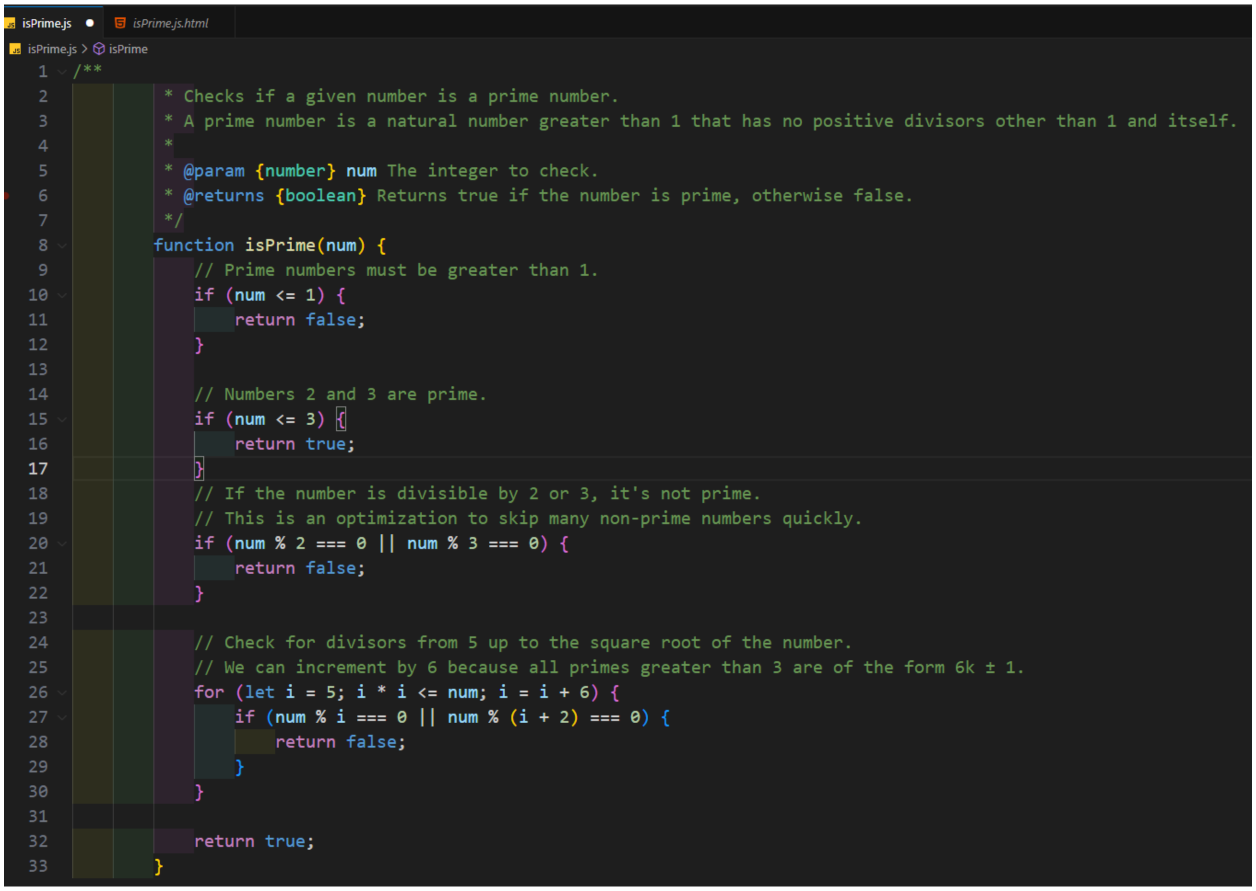
Here are the source codes of three javascript files that we've created: add.js, main.js, and isPrime.js .



```
/**
 * Adds two numbers
 *
 * @param {number} num1 This is the first Number
 * @param {number} num2 This is the second Number
 * @returns {number} Returns the sum of two numbers
 */
function add(num1 : number , num2 : number ) : number  {  Show usages
  this.num1 = num1;
  this.num2 = num2;

  return num1 + num2;
}

console.log(add( num1: 2,  num2: 2));
```



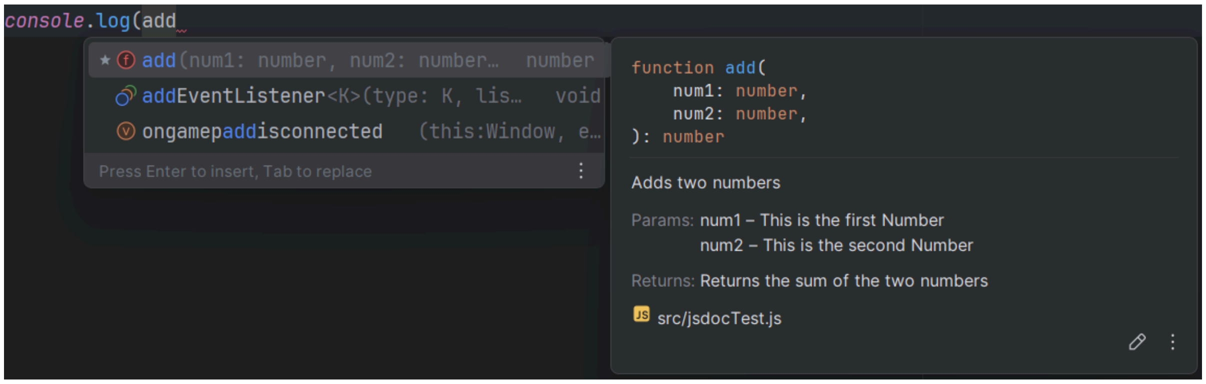
```javascript
/**
 * Creates a person object with the given properties.
 * This is an example of a factory function in JavaScript.
 *
 * @param {string} name The person's full name
 * @param {number} age age of a person
 * @param {boolean} isDeveloper is he a developer or not
 * @returns {Object} the person object
 */
function createPerson(name, age, isDeveloper) {
  return {
    name: name,
    age: age,
    isDeveloper: isDeveloper,
  };
}
```



```javascript
/**
     * Checks if a given number is a prime number.
     * A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.
     *
     * @param {number} num The integer to check.
     * @returns {boolean} Returns true if the number is prime, otherwise false.
     */
    function isPrime(num) {
      // Prime numbers must be greater than 1.
      if (num <= 1) {
        return false;
      }

      // Numbers 2 and 3 are prime.
      if (num <= 3) {
        return true;
      }
      // If the number is divisible by 2 or 3, it's not prime.
      // This is an optimization to skip many non-prime numbers quickly.
      if (num % 2 === 0 || num % 3 === 0) {
        return false;
      }

      // Check for divisors from 5 up to the square root of the number.
      // We can increment by 6 because all primes greater than 3 are of the form 6k ± 1.
      for (let i = 5; i * i <= num; i = i + 6) {
        if (num % i === 0 || num % (i + 2) === 0) {
          return false;
        }
      }

      return true;
    }
```

4. Produced documentation:

Here is the documentation created for those JavaScript files.



```
console.log(add
```

add(num1: number, num2: number… number
addEventListener<K>(type: K, lis… void
ongamepaddisconnected (this:Window, e…
Press Enter to insert, Tab to replace

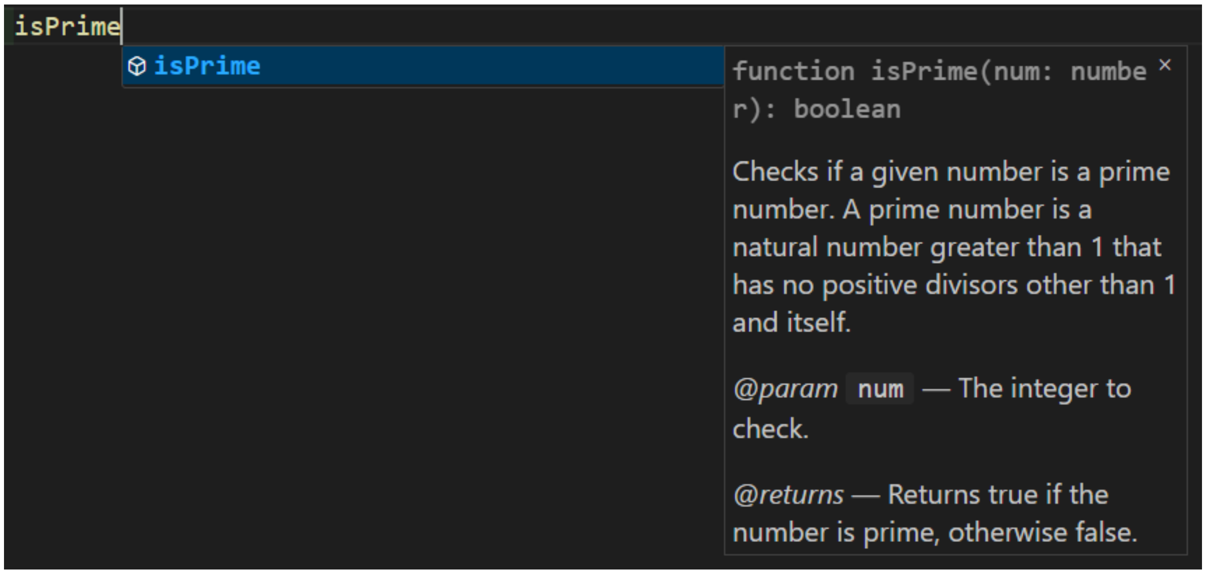
```
function add(
    num1: number,
    num2: number,
): number
```

Adds two numbers

Params: num1 – This is the first Number
num2 – This is the second Number

Returns: Returns the sum of the two numbers

src/jsdocTest.js



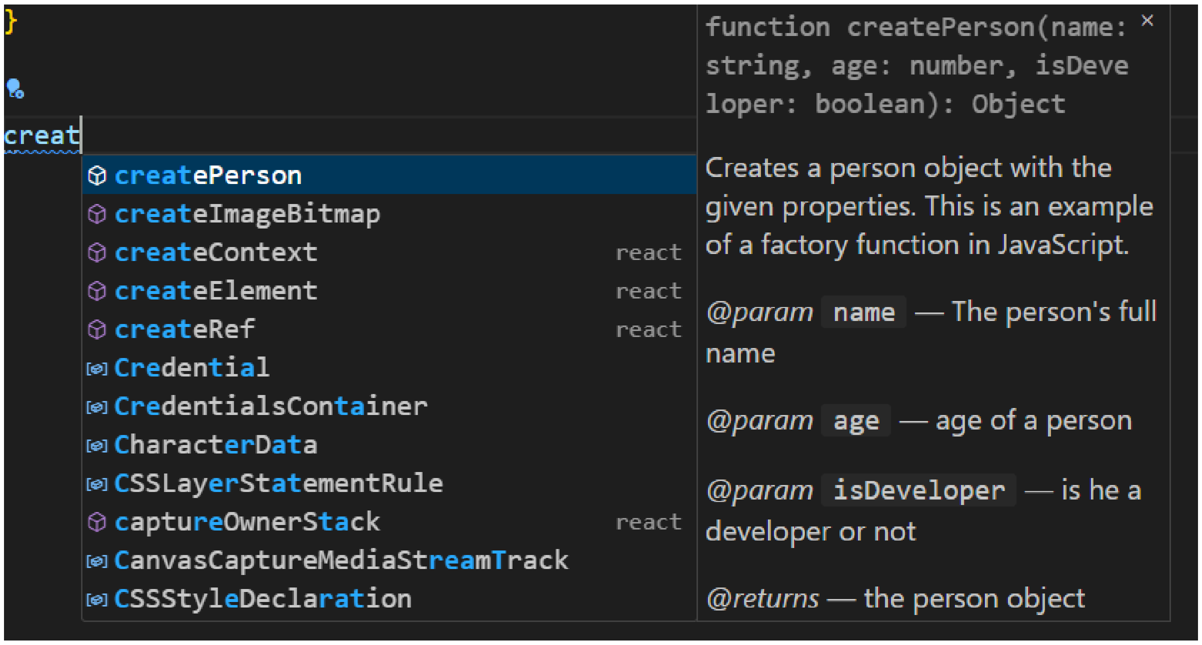
```
isPrime
```

```
function isPrime(num: number): boolean
```

Checks if a given number is a prime number. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

@param `num` — The integer to check.

@returns — Returns true if the number is prime, otherwise false.



```
}

creat
createPerson
createImageBitmap
createContext                react
createElement                react
createRef                    react
Credential
CredentialsContainer
CharacterData
CSSLayerStatementRule
captureOwnerStack            react
CanvasCaptureMediaStreamTrack
CSSStyleDeclaration
```

```
function createPerson(name: string, age: number, isDeveloper: boolean): Object
```

Creates a person object with the given properties. This is an example of a factory function in JavaScript.

@param `name` — The person's full name

@param `age` — age of a person

@param `isDeveloper` — is he a developer or not

@returns — the person object

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 use and understand the documentation effectively.

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.