DWA_03.4 Knowledge Check_DWA3.1

---

1. Please show how you applied a Markdown File to a piece of your code.

```python
# example.py

def analyze_data(data):
    """
    Perform some analysis on the given data.

    Parameters:
    - data (list): The input data for analysis.

    Returns:
    - result (float): The result of the analysis.
    """
    # Your analysis code here
    result = sum(data) / len(data)
    return result

if __name__ == "__main__":
    # Example usage
    data = [1, 2, 3, 4, 5]
    result = analyze_data(data)
    print(f"Analysis result: {result}")
```

````
# Data Analysis Documentation

## Overview

This script, `example.py`, performs data analysis on a given dataset.

## Functionality

The main function in this script is `analyze_data(data)`,
which takes a list of data as input and returns the result of the analysis.

### Usage

```python
data = [1, 2, 3, 4, 5]
result = analyze_data(data)
print(f"Analysis result: {result}")
````

````

2. Please show how you applied JSDoc Comments to a piece of your code.

```
/**
 * Represents a simple calculator class.
 *
 * @class
 */
class Calculator {
  /**
   * Adds two numbers.
   *
   * @param {number} a - The first number.
   * @param {number} b - The second number.
   * @returns {number} The sum of the two numbers.
   */
  add(a, b) {
    return a + b;
  }

  /**
   * Subtracts one number from another.
   *
   * @param {number} a - The minuend.
   * @param {number} b - The subtrahend.
   * @returns {number} The result of the subtraction.
   */
  subtract(a, b) {
    return a - b;
  }
```

```
  /**
   * Multiplies two numbers.
   *
   * @param {number} a - The first factor.
   * @param {number} b - The second factor.
   * @returns {number} The product of the multiplication.
   */
  multiply(a, b) {
    return a * b;
  }

  /**
   * Divides one number by another.
   *
   * @param {number} a - The dividend.
   * @param {number} b - The divisor.
   * @returns {number} The result of the division.
   */
  divide(a, b) {
    if (b === 0) {
      throw new Error("Division by zero is not allowed.");
    }
    return a / b;
  }
}

// Example usage
const calculator = new Calculator();
console.log(calculator.add(5, 3)); // Output: 8
console.log(calculator.subtract(10, 4)); // Output: 6
console.log(calculator.multiply(2, 6)); // Output: 12
console.log(calculator.divide(8, 2)); // Output: 4
```

JSDoc comments are used to document the class Calculator and its methods (add, subtract, multiply, divide). The @param tag is used to describe the parameters of each method, and the @returns tag is used to describe the return value.

JSDoc comments can be processed by various tools to generate documentation in different formats, such as HTML or Markdown. Popular tools for generating

documentation from JSDoc comments include JSDoc itself, as well as tools like ESDoc, documentation.js, and others.

---

3. Please show how you applied the @ts-check annotation to a piece of your code.

```
// @ts-check

/**
 * Represents a simple calculator class.
 *
 * @class
 */
class Calculator {
  /**
   * Adds two numbers.
   *
   * @param {number} a - The first number.
   * @param {number} b - The second number.
   * @returns {number} The sum of the two numbers.
   */
  add(a, b) {
    return a + b;
  }

  /**
   * Subtracts one number from another.
   *
   * @param {number} a - The minuend.
   * @param {number} b - The subtrahend.
   * @returns {number} The result of the subtraction.
   */
  subtract(a, b) {
    return a - b;
  }
```

```
  /**
   * Multiplies two numbers.
   *
   * @param {number} a - The first factor.
   * @param {number} b - The second factor.
   * @returns {number} The product of the multiplication.
   */
  multiply(a, b) {
    return a * b;
  }

  /**
   * Divides one number by another.
   *
   * @param {number} a - The dividend.
   * @param {number} b - The divisor.
   * @returns {number} The result of the division.
   */
  divide(a, b) {
    if (b === 0) {
      throw new Error("Division by zero is not allowed.");
    }
    return a / b;
  }
}

// Example usage
const calculator = new Calculator();
console.log(calculator.add(5, 3)); // Output: 8
console.log(calculator.subtract(10, 4)); // Output: 6
console.log(calculator.multiply(2, 6)); // Output: 12
console.log(calculator.divide(8, 2)); // Output: 4
```

the `@ts-check` comment at the top of the file instructs TypeScript to check the code for type errors. TypeScript will perform static type checking on the annotated JavaScript code, providing feedback on potential type-related issues.

Note that using `@ts-check` in a JavaScript file doesn't enforce strict typing, but it allows you to gradually introduce TypeScript features and benefit from static type checking in your existing JavaScript codebase.

---

4. As a BONUS, please show how you applied any other concept covered in the 'Documentation' module.

"Task" custom type

```typescript
// Define the Task type
type Task = {
  id: number;
  title: string;
  description: string;
  completed: boolean;
};

// Example usage
const myTask: Task = {
  id: 1,
  title: "Complete Assignment",
  description: "Finish the coding assignment by Friday.",
  completed: false,
};

// Function that takes a Task as a parameter
function markAsCompleted(task: Task): Task {
  return { ...task, completed: true };
}

// Using the function
const completedTask = markAsCompleted(myTask);
console.log(completedTask);
```

we've defined a Task type that represents an object with properties such as id, title, description, and completed. We then create an instance of a task (myTask) and pass it to a function (markAsCompleted) that marks the task as completed.