DWA_03.4 Knowledge Check_DWA3.1

---

1. Please show how you applied a Markdown File to a piece of your code.
MD files has its own Syntax:
Using headers:(hashtag)

```
# header 1 #
## header 2 ##
### header 3 ### etc
```

There are different text styles, the most common would be:
Using normal text:

```
Just a normal text.
```

You can also nest bullet/dash points:

```
* Bullet list
      * Nested bullet
      * Sub-nested bullet etc
```

Can use Tables:

```
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

Tables Aligned:

```
Left aligned Header | Right aligned Header | Center aligned Header
| :--- | ---: | :---:
Content Cell  | Content Cell | Content Cell
Content Cell  | Content Cell | Content Cell
```

Can also use emojis:
[Emoji cheat sheet for GitHub, Basecamp, Slack & more (webfx.com)]

```
😀😄🤣🤩😔🤕
```

Can also add Pictures:

```
![picture alt](http://via.placeholder.com/200x150 "Title is optional")
```

---

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

```
/**
 * @description The first name of a person.
 * @type {string}
 */
const firstName = 'John';

/**
 * @description The age of a person.
 * @type {number}
 */
const age = 35;

/**
 * @description The hobby of a person.
 * @type {string}
 */
const hobby = 'Coding';

/**
 * @description Logs the given parameter twice to the console.
 * @param {any} parameter - The value to be logged.
 * @returns {void}
 */
const logTwice = (parameter) => {
  console.log(parameter);
  console.log(parameter);
};

/**
 * @description Calls the logTwice function with a formatted message.
 * @returns {void}
 */
const iLoveCoding = () => {
  logTwice(`Hello, ${firstName} (${age}). I love ${hobby}!`);
};

// Call the iLoveCoding function.
iLoveCoding();
```

---

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

```
// @ts-check

/**
 * A simple function to add two numbers.
 * @param {number} a - The first number.
 * @param {number} b - The second number.
 * @returns {number} The sum of the two numbers.
 */
function addNumbers(a, b) {
  return a + b;
}

// Test the function
const result = addNumbers(5, 10);
console.log(result); // Output: 15
```

---

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

```
/**
 * @typedef {object} Tasks
 * @prop {string} id - The unique identifier of the task.
 * @prop {string} title - The title of the task.
 * @prop {boolean} completed - Indicates whether the task is completed or
not.
 * @prop {Date} created - The date when the task was created.
 * @prop {null | Date} due - The date when the task is due. Can be null if
no due date is set.
 * @prop {'high' | 'medium' | 'low'} urgency - The urgency level of the
task, which can be 'high', 'medium', or 'low'.
 */

// Example usage of the Tasks typedef
const task1 = {
  id: '1',
  title: 'Buy groceries',
```

```
  completed: false,
  created: new Date('2023-07-19'),
  due: new Date('2023-07-25'),
  urgency: 'medium',
};

const task2 = {
  id: '2',
  title: 'Pay bills',
  completed: false,
  created: new Date('2023-07-19'),
  due: null, // No due date set for this task
  urgency: 'high',
};

// Function that takes a Tasks object as an argument
function displayTask(task) {
  console.log(`Task: ${task.title}`);
  console.log(`Status: ${task.completed ? 'Completed' : 'Incomplete'}`);
  console.log(`Created: ${task.created.toDateString()}`);
  if (task.due !== null) {
    console.log(`Due: ${task.due.toDateString()}`);
  } else {
    console.log('No due date set.');
  }
  console.log(`Urgency: ${task.urgency}`);
}

// Output information about the tasks
console.log('Task 1:');
displayTask(task1);
console.log('Task 2:');
displayTask(task2);
```
