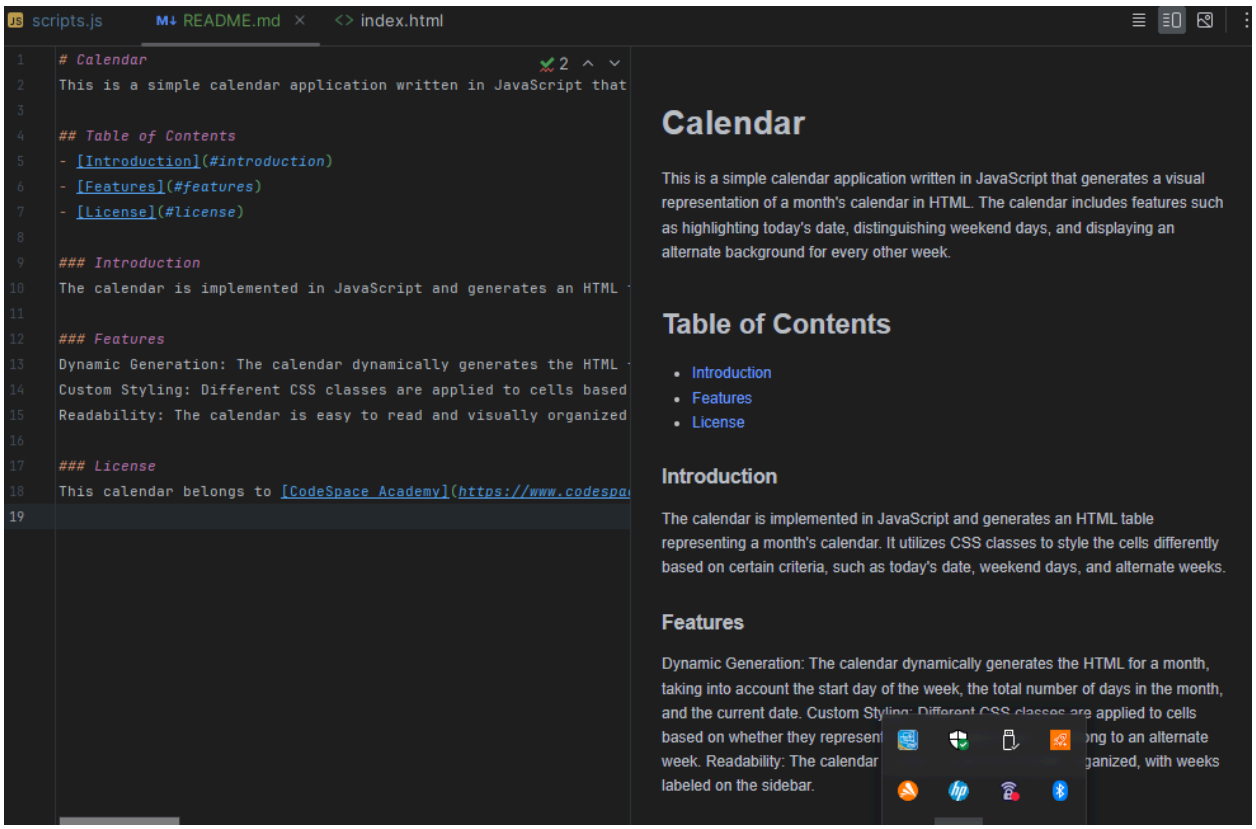
DWA_03.4 Knowledge Check_DWA3.1

---

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



I created a new file with the '.md' extension. I then gave it a description after the first and main header. I then added a table of contents linking all the sections. In the features section I elaborated on the project.

---

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

I've added JSDoc comments to document the purpose and usage of my JavaScript code.

```ts
// @ts-check

/**
 * Array that contains the months.
 * @type {Array<string>}
 */
const MONTHS = [
    'January',
    'February',
    'March',
    'April',
    'May',
    'June',
    'July',
    'August',
    'September',
    'October',
    'November',
    'December',
]

/**
 * Gets the number of days in a month.
 *
 * @param {Date} date - The date object representing the month.
 * @returns {number} - The number of days in the month.
 */
1 usage
const getDaysInMonth = (date) => new Date(date.getFullYear(), monthIndex: date.getMonth() + 1, date: 0).getDate()
```

First I documented the purpose of ‘MONTHS’ and specified its type using ‘@type’

```ts

/**
 * Gets the number of days in a month.
 *
 * @param {Date} date - The date object representing the month.
 * @returns {number} - The number of days in the month.
 */
1 usage
const getDaysInMonth = (date) => new Date(date.getFullYear(), monthIndex: date.getMonth() + 1, date: 0).getDate()

// Only edit below

/**
 * Generates an array of objects representing the weeks and days in a month.
 * It initializes a date object to the current date with the day set to 1.
 * startDay: The day of the week on which the month starts.
 * daysInMonth: The total number of days in the month.
 * Uses nested loops to populate array - result
 * @returns {Array<{week: number, days: Array<{ dayOfWeek: number, value: number | string }> }>} - An array of weeks and days.
 */
1 usage
const createData = () => {
    /**
     * The current date object with the day set to 1.
     * @type {Date}
     */
    const current = new Date()
    current.setDate(1)
```

In this second picture I've documented the purpose of the 'getDaysInMonth' function, its parameters('date') and the return type.
I also provided an overview of what the 'createData' function does and the return value.

```
    /**
     * The day of the week on which the month starts.
     * @type {number}
     */
    const startDay = current.getDay()

    /**
     * The total number of days in the month.
     * @type {number}
     */
    const daysInMonth = getDaysInMonth(current)

    /**
     * The number of weeks needed to represent the month.
     * @type {number}
     */
    const weeks = Math.ceil( x: (startDay + daysInMonth) / 7)

    /**
     * The number of days in each week.
     * @type {number}
     */
    const days = 7

    /**
     * The resulting array containing weeks and days in month.
     * @type {Array<{ week: number, days: Array<{ dayOfWeek: number, value: number | string }> }>}
```

```
    /**
     * The resulting array containing weeks and days in month.
     * @type {Array<{ week: number, days: Array<{ dayOfWeek: number, value: number | string }> }>}
     */
    const result = []

    for (let weekIndex = 0; weekIndex < weeks; weekIndex++) {
    result.push({
        week: weekIndex + 1,
        days: []
    })

    for (let dayIndex = 0; dayIndex < days; dayIndex++) {
        /**
         * The day of the month represented by current cell.
         * @type {number}
         */
        const day = weekIndex * days + dayIndex - startDay + 1;

        /**
         * Indicates whether current day is a valid day in the month.
         * @type {boolean}
         */
        const isValid = day > 0 && day <= daysInMonth
```

```
        /**
         * Add information about the day to the result array.
         */
        result[weekIndex].days.push({
            dayOfWeek: dayIndex + 1,
            value: isValid ? day : '',
        })
    }
}
return result;
};

/**
 * Adds a table cell to an existing HTML string.
 * Takes three parameters
 * @param {string} existing - The existing HTML string.
 * @param {string} classString - String representing css classes.
 * @param {number | string } value - Content of table cell.
 * @returns {string} - The updated HTML string.
 */
2 usages
const addCell = (existing, classString, value) => {
    const result = /* html */ `
        ${existing}

        <td class="${classString}">
```

Here my comments explain the purpose of the functions, their parameters and return type.

```

/**
 * Converts data generated by createData into an HTML string for rendering.
 * Iterates over each week and days, creating table cells for each day.
 * Applies different CSS classes based on whether the day is today,
 * a weekend day, or part of an alternate week.
 *
 * @param {Array<{ week: number, days: Array<{ dayOfWeek: number, value: number | string }> }>} data - The data to convert.
 * @returns {string} - The HTML string.
 */
1 usage
const createHtml = (data) => {
    /**
     * The Html string representing the rendered table.
     * @type {string}
     */
    let result = '';

    for (const { week, days } of data ) {
        /**
         * The Html string representing the current week..
         * @type {string}
         */
        let inner = "";

```

Here I added documentation for the code that converts my data into a HTML string, and for the loop as well as the params and what it returns.

```
        /**
         * Add a cell for the week number.
         * @type {string}
         */
        inner = addCell(inner, classString: 'table__cell table__cell_sidebar', value: `Week ${week}`);

    for (const { dayOfWeek, value } of days) {
            /**
             * Indicates whether the current day is today.
             * @type {boolean}
             */
            const isToday = new Date().getDate() === value

            /**
             * Indicates whether the current day is a weekend
             * @type {boolean}
             */
            const isWeekend = dayOfWeek === 1 || dayOfWeek === 7;

            /**
             * Indicates whether the current week is an alternate week.
             * @type {boolean}
             */
            const isAlternate = week % 2 === 0;
```

```
        /**
         * The CSS classes to apply to the current table cell.
         * @type {string}
         */
        let classString = 'table__cell'

            if (isToday) classString = `${classString} table__cell_today`
            if (isWeekend) classString = `${classString} table__cell_weekend`
            if (isAlternate) classString = `${classString} table__cell_alternate`

        /**
         * Add a cell for the current day.
         * @type {string}
         */
        inner = addCell(inner, classString, value)
        }

        /**
         * Add a row for the current week.
         * @type {`${string}<tr>${string}</tr>`}
         */
        result = `
            ${result}
            <tr>${inner}</tr>
```

```
/**
 * The current year.
 * @type {Date}
 */
const current = new Date()

/**
 * Updates the content of an HTML element with the current month and year.
 * @param {Date} currentDate - The current date.
 * @param {string} selector - The selector of the Html element to update.
 */
document.querySelector( selectors: '[data-title]').innerText = `${MONTHS[current.getMonth()]} ${current.getFullYear()}`

/**
 * Represents the data generated for rendering.
 * @type {Array<{ week: number, days: Array<{ dayOfWeek: number, value: number | string }> }>}
 */
const data = createData()
/**
 * Updates the content of an HTML element with the data generated from createData and formatted using createHtml.
 *
 * @param {Array<{ week: number, days: Array<{ dayOfWeek: number, value: number | string }> }>} data - The data to display.
 * @param {string} selector - The selector of the HTML element to update.
 * @returns {void}
 */
document.querySelector( selectors: '[data-content]').innerHTML = createHtml(data)
```

These comments explain the purpose of the variables and the code that updates the content of the HTML elements with the rendered data.

---

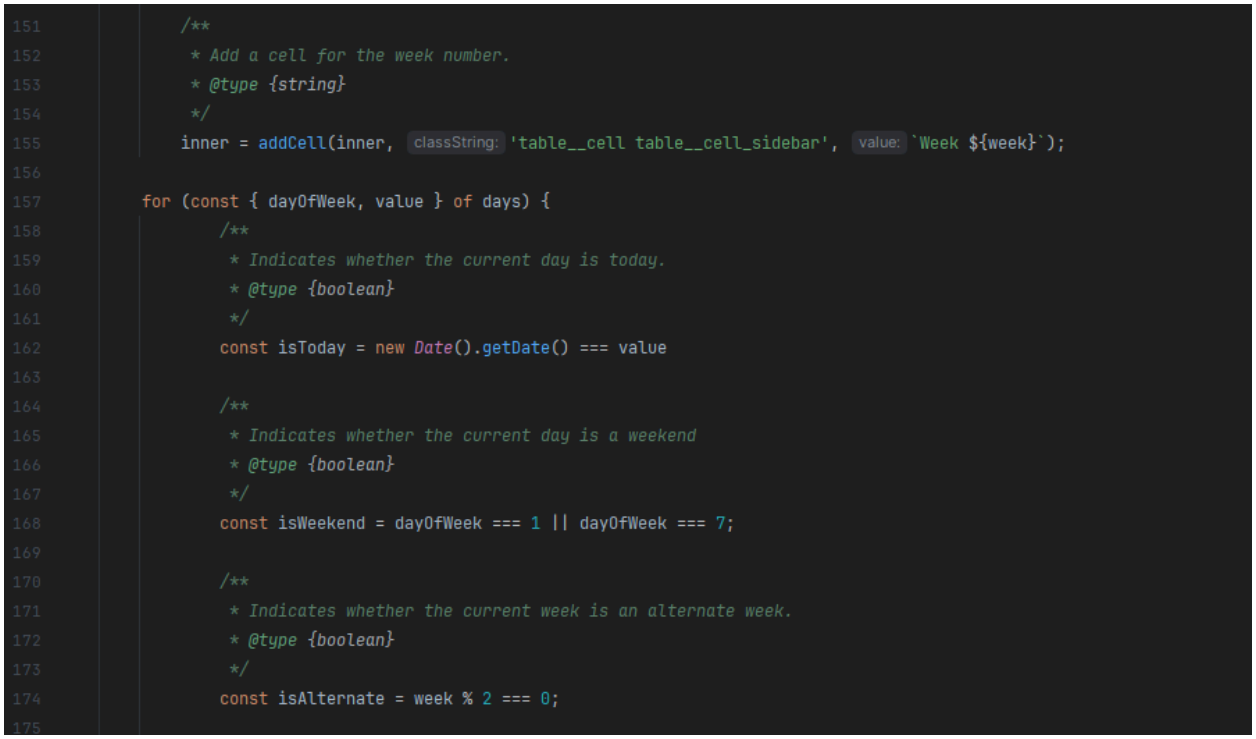
3. Please show how you applied the @ts-check annotation to a piece of your code.

```
// @ts-check

/**
 * Array that contains the months.
 * @type {Array<string>}
 */
const MONTHS = [
    'January',
    'February',
    'March',
    'April',
    'May',
    'June',
    'July',
    'August',
    'September',
    'October',
    'November',
    'December',
]

/**
 * Gets the number of days in a month.
 *
 * @param {Date} date - The date object representing the month.
 * @returns {number} - The number of days in the month.
 */
1 usage
const getDaysInMonth = (date) => new Date(date.getFullYear(), monthIndex: date.getMonth() + 1, date: 0).getDate()
```

On the very first line in my js doc I added the '@ts-check' annotation, to enable TypeScript's type checking. It instructs TypeScript to check types within the file, helping to catch type-related errors.

---

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



```
        /**
         * Add a cell for the week number.
         * @type {string}
         */
        inner = addCell(inner, classString: 'table__cell table__cell_sidebar', value: `Week ${week}`);

    for (const { dayOfWeek, value } of days) {
            /**
             * Indicates whether the current day is today.
             * @type {boolean}
             */
            const isToday = new Date().getDate() === value

            /**
             * Indicates whether the current day is a weekend
             * @type {boolean}
             */
            const isWeekend = dayOfWeek === 1 || dayOfWeek === 7;

            /**
             * Indicates whether the current week is an alternate week.
             * @type {boolean}
             */
            const isAlternate = week % 2 === 0;
```

I also added JSDocs inside my code block to explain different variables.
