WRITING READABLE CODE

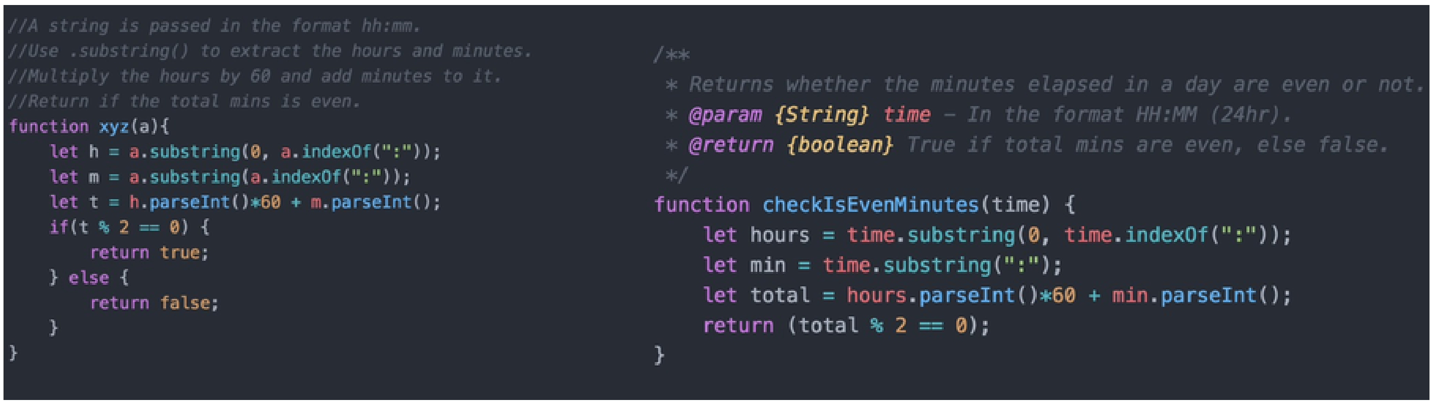
"Any fool can write a code that a computer can understand.
Good programmers write code that humans can understand."

- Martin Fowler

GENERAL GUIDELINES

Writing clean and readable code is not hard, but it is very important toward the maintainability of your program. Readability is achieved by through experience and practiced intuition, but can be guided by key principles including:

1. **Good variable names**. Every variable should have an identifier (name) that directly reflects what it represents. A variable that is describing the number of ducks in a pond might be called ducks or count but it shouldn't be called something like pancakes. Never name your variables as single letters (except for counters in for loops) or anything too abstract.
2. **Function naming and responsibility**. Make sure your functions also are named well represent what the function does. Every function should also have a single responsibility, something you could describe in a sentence or two. (This is called the SINGLE RESPONSIBILITY PRINCIPLE).
3. **Proper formatting**. Always maintain correct and consistent indentation, spacing, and location of curly braces, if required by the language. One of the most helpful motivations for clean and consistent formatting is to consider the perspective of the person trying to understand your code, making sure one could painlessly read and interpret your code's purpose through a consistent organization and hierarchal structure.
4. **Clear documentation**.You should provide clear documentation in all of your web development work, describing functions, key variables or constants, and any particularly complex sections of code. At the same time, your documentation should be client-focused, avoiding implementation details about how your programs are implemented (this latter point is especially relevant when working with function-based programming languages like JavaScript and PHP).



```
//A string is passed in the format hh:mm.
//Use .substring() to extract the hours and minutes.
//Multiply the hours by 60 and add minutes to it.
//Return if the total mins is even.
function xyz(a){
    let h = a.substring(0, a.indexOf(":"));
    let m = a.substring(a.indexOf(":"));
    let t = h.parseInt()*60 + m.parseInt();
    if(t % 2 == 0) {
        return true;
    } else {
        return false;
    }
}
```

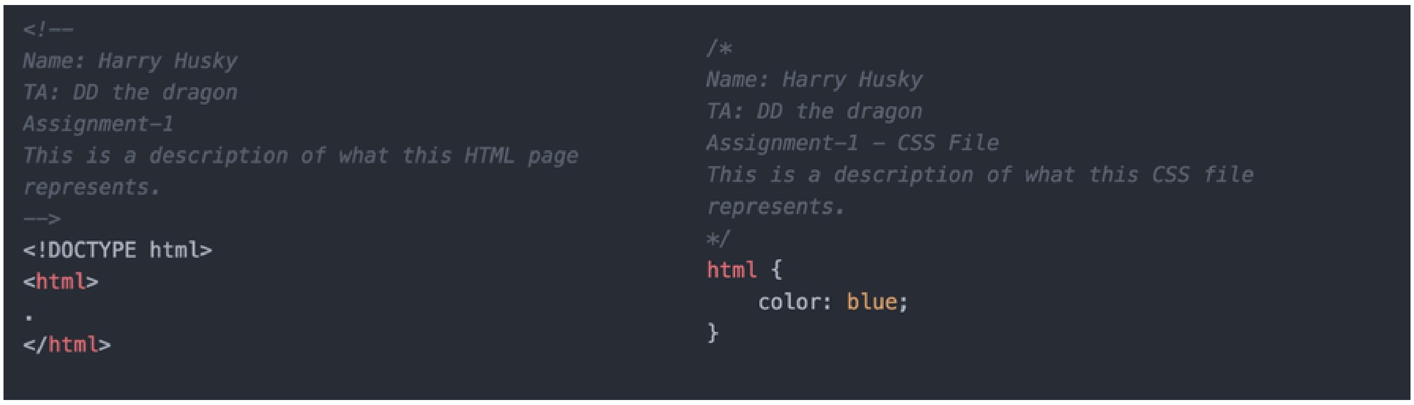
```
/**
 * Returns whether the minutes elapsed in a day are even or not.
 * @param {String} time - In the format HH:MM (24hr).
 * @return {boolean} True if total mins are even, else false.
 */
function checkIsEvenMinutes(time) {
    let hours = time.substring(0, time.indexOf(":"));
    let min = time.substring(":");
    let total = hours.parseInt()*60 + min.parseInt();
    return (total % 2 == 0);
}
```

COMMENTING FOR CSE 154

COMMENTING YOUR HTML AND CSS CODE

The comment tag is used to insert comments in your HTML source code. Text written inside of comment tags are not displayed by the browser. The comment tag is `<!-- your comment here -->`.

The comments in CSS are wrapped between `/* your comment here */`. Comments in HTML and CSS can also span multiple lines.



```
<!--
Name: Harry Husky
TA: DD the dragon
Assignment-1
This is a description of what this HTML page
represents.
-->
<!DOCTYPE html>
<html>
.
</html>
```

```
/*
Name: Harry Husky
TA: DD the dragon
Assignment-1 - CSS File
This is a description of what this CSS file
represents.
*/
html {
    color: blue;
}
```

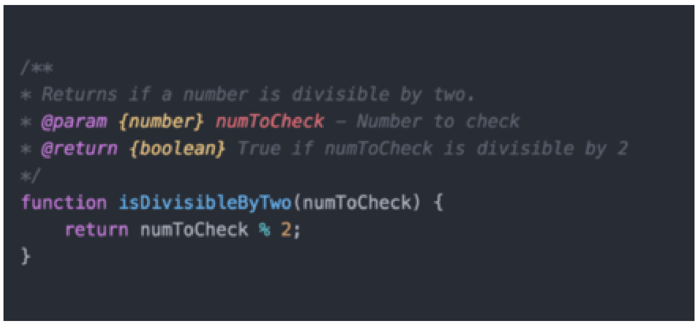
COMMENTING YOUR JAVASCRIPT CODE

There are two different ways to comment in JavaScript code:

1. **Single-line comments:** Including `//` in front of the single line of code that you want to comment out.
2. **Multi-line comments:** This method of commenting is exactly the same as commenting CSS files. You insert multiple lines of code between the following: `/* your comment here */`

JavaScript is used **boundlessly** to develop huge applications and Application Program Interfaces (APIs). Because of this, good documentation of the application is necessary. To document our API's and applications, we use a documentation generator called JSDoc.
Some of the important JSDoc rules:

a. Each comment must start with a `/**` sequence in order to be recognized by the JSDoc parser. Comments beginning with `/*`, `/***`, or more than 3 stars will be ignored. This is to suppress parsing of comment blocks.
b. Use JSDoc tags to provide more information about the code:
    - If the function to be commented is a constructor, then add a `@constructor` tag to it.
    - Use `@param {data type}` to tag the parameters. Each parameter gets a different tag and on a different line.
    - Use `@return {data type}` to tag the return type for the function. Explain what it represents in a sentence.
c. The comment should also include a single sentence description for the function as well as for each parameter.



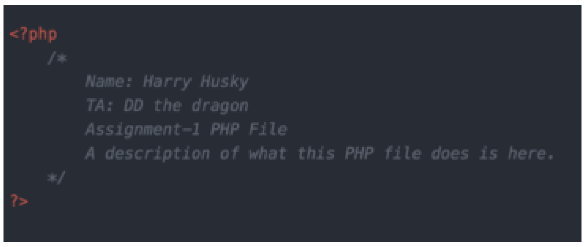
```
/**
* Returns if a number is divisible by two.
* @param {number} numToCheck - Number to check
* @return {boolean} True if numToCheck is divisible by 2
*/
function isDivisibleByTwo(numToCheck) {
    return numToCheck % 2;
}
```

Though you are not required to for this class, you can generate the documentation webpage, run the following command in a terminal window: `jsdoc relative-path/book.js`
This command will create a directory named `out/` in the current working directory. Within that directory, you will find the generated HTML pages.

COMMENTING YOUR PHP CODE

Commenting in PHP is exactly the same as commenting in JavaScript by using `//` for single line comments and `/* … */` for multi-line comments.

IMPORTANT NOTE: Make sure to add comment after the opening PHP tag as shown below.



```
<?php
    /*
        Name: Harry Husky
        TA: DD the dragon
        Assignment-1 PHP File
        A description of what this PHP file does is here.
    */
?>
```

COMMENTING YOUR SQL

We're going to cut you some slack here… we don't require you to comment your SQL code (It is pretty much self-explanatory, isn't it?).

Now that you have read this document, make this quote your mantra:

*"Writing well-documented code is not only important for communicating with others in your team.
It's also for communicating with yourself in the future."*

– Dr. Lauren Bricker