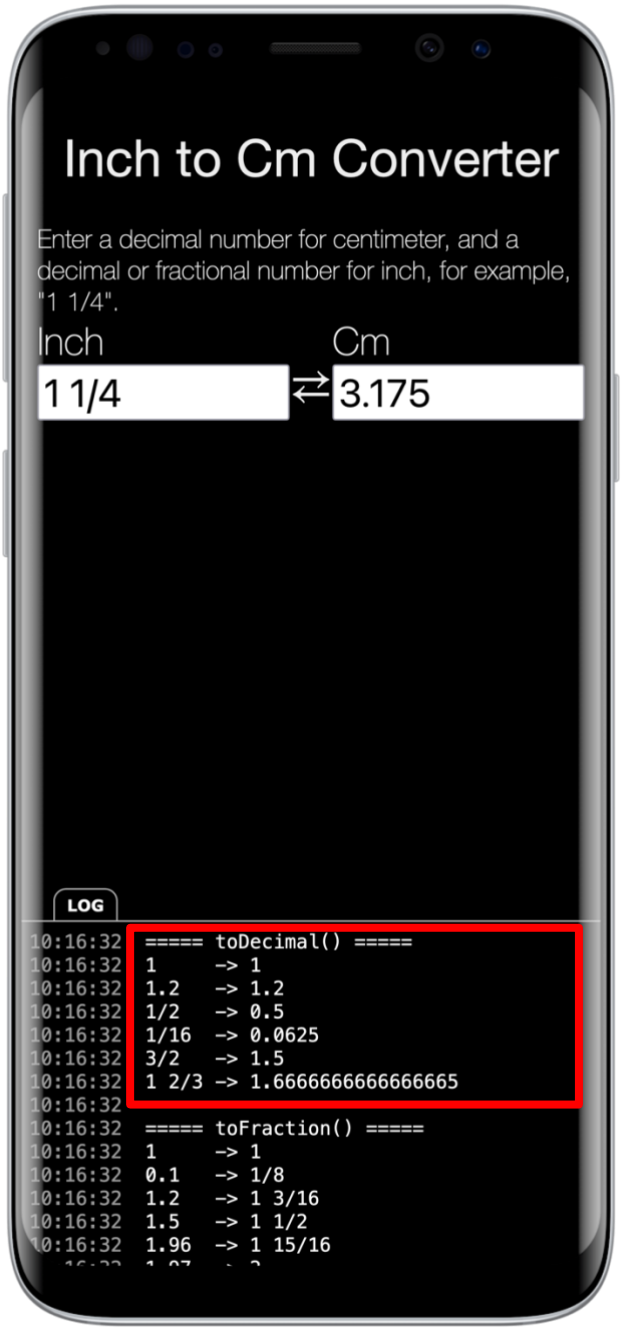
Assignment 3: Inch-To-Cm (Cm-To-Inch) Converter

Description

Write an Angular mobile application to convert inches to centimetres and vice versa. Both values in the input text fields should be automatically synchronized (bi-directional data binding) using Angular data binding technique. For the inch <input>, it allows to enter a decimal or fractional number, for example, “1.25” or “1 1/4”. For centimeter <input>, it only allows a decimal number, for example, “1.23”.

The application must be responsive to fit in any mobile device width for either portrait or landscape mode, and to place 2 input boxes in a row without clipping.

Option A: (max 80 points)



Option B: (max 100 points)

Option A (max 80 points)

- Must use Angular framework (NOT AngularJS).
- Must be responsive (the page must fit in 360 pixel-wide device width).
- Must use Angular data bindings **ngModel** to sync the values of both <input> elements.
- Must label clearly "inch" and "cm" for the <input> elements.
- Must accept either a decimal or a fractional number, such as "1.23" or "1 2/3" for the "inch" <input> element.
- Must accept only decimal number, such as "1.23" for the "cm" <input> element.
- Must define a constant for the conversion formula, for example FACT = 2.54
- Must define **toDecimal(***fractionString***)** to convert a fractional string to a decimal number (up to 4-digit).
- Must verify your **toDecimal(***fractionString***)** in your constructor() or ngOnInit() with the following decimal inches using console.log() or Logger.log().
    - toDecimal("1") → 1 (2.54 cm)
    - toDecimal("1.2") → 1.2 (3.048 cm)
    - toDecimal("1/2") → 0.5 (1.27 cm)
    - toDecimal("1/16") → 0.0625 (0.1588 cm)
    - toDecimal("3/2") → 1.5 (3.81 cm)
    - toDecimal("1 2/3") → 1.6667 (4.2333 cm)

Option B (max 100 points)

- Implement all the requirements in Option A.
- Must have a checkbox to toggle the fractional numbers (It is used to display "inch" <input> as fractional. For example, if a user enters 1.23 in the "cm" <input>, then it displays "8/16" in the "inch" <input>.)
- Must define **toFraction(***inchDecimal***)** to convert a decimal number to the closest fractional number as string.
- Must verify your **toFraction()** in the constructor() or ngOnInit() with the following numbers using console.log() or Logger.log(),

Examples: Test your **toFraction(***number***)** function with the following examples

- toFraction(1) → "1"
- toFraction(0.1) → "2/16"
- toFraction(1.2) → "1 3/16"
- toFraction(1.5) → "1 8/16"
- toFraction(1.96) → "1 15/16"
- toFraction(1.97) → "2"

References

Note 09. Angular Component & Template and Lab09

Deliverables

An archive file, **Assignment3_<*yourname*>.zip**, which contains your project distribution folder only under **dist** subdirectory. It should be runnable on any system after download it.

NOTE: *You must build your project with "***`ng build --base-ref ./ --configuration development`***" at the terminal. (An incorrect build/submission will be deducted by 50 points)*

NOTE*: You must include the file header at the beginning of each file. The header must contain a short description, your name, email, date, etc.*

Submission and Due Date

Submit your deliverables to SLATE/Assignments/Assignment3 by **Saturday, July 29, 11:59 PM**.
You may submit multiple versions, but only the latest version will be evaluated.
***NOTE**: Late submission will be deducted 10% per day. (max. 3 days)*
***NOTE**: Partial implementation will be accepted.*
***NOTE**: This assignment is individual work and subject to Sheridan Academic Integrity Policy.*

Tips

- Use flex layout to dynamically resize and center the DOM elements
- Implement the optionA first, then add the optionB task later
- Use the pseudo code blocks in Q&A section below

Evaluation for Option A (Total 80 points)

Task 1: Template & CSS (30 points)

1. Implement **app.component.html**
    a. Heading for the title, "**Inch To Cm Converter**"
    b. 2 <Input> elements with labels
    c. A separator between 2 <input> elements; "=" or arrow character (#8644, #8660)
    d. Angular data binding syntax; 2-way bindings, event bindings for Angular FormControl

2. Implement **app.component.css** and/or **styles.css**
    a. Dynamically resize the whole page to fit in any mobile device, and 2 <input> elements in a row
    b. Clearly label each <input> element, "inch" and "cm"
    c. Make bigger font size for input texts

Task 2: app.component.ts (50 points)

1. Define 2 properties; **inch** as a string and **cm** as a number
2. Define a constant for the conversion formula; inch = 2.54 * cm (Must define it outside of the class)
3. Display the decimal numbers with upto 4 decimal places
4. Define **convertToInch()** member function to convert **this.cm** to **this.inch** (*Note that this.inch is a string*)
5. Define **convertToCm()** member function to convert **this.inch** to **this.cm** using toDecimal()
6. Define **toDecimal(***fractionString : string***)** function to convert a fractional string to a decimal number. This function is used to convert the fractional user input to a decimal number before converting the inch to cm. The programming logics of this functions are; (Also see **Q2** page 5)

    a. Split *fractionString* by spaces into 2 tokens
    b. If there is only one token, *fractionString* contains either a decimal number or fraction number, for example, 1.2, or 1/2.
        i. Split the token by "/" again into 2 parts
        ii. If there is only 1 part, the string is a decimal number. Use parseFloat() to convert it to a number
        iii. If there are 2 parts, the string is a fraction number. Parse each part as number, then divide the first part (numerator) by the second (denominator)
    c. If there are 2 tokens, it contains an integer and fraction, for example, 1 2/3.
        i. Parse the first token to a integer and store it first the returning variable, *decimal*
        ii. Split the second token by "/" into 2 parts again (If it cannot be split to 2 parts, it is not a valid fraction format. Skip the following steps, and return only *decimal* or NaN.)
        iii. Parse 2 parts to integers and divide them
        iv. Add the division to the returning variable, *decimal* then return it

Examples: Test your toDecimal(fractionString : string) function with the following examples

"1"	→	1	(2.54 cm)
"1.2"	→	1.2	(3.048 cm)
"1/2"	→	0.5	(1.27 cm)
"1/16"	→	0.0625	(0.1588 cm)
"3/2"	→	1.5	(3.81 cm)
"1 2/3"	→	1.6667	(4.2333 cm)

Evaluation for Option B (Total 100 points)

Task 1: Template & CSS (40 points)

1. Implement **app.component.html**
    a. Implement Option A
    b. Add a checkbox <input> "Use Fraction" inside of a <fieldset>
    d. Angular data binding for the checkbox <input>

Task 2: app.component.ts (60 points)

1. Implement Option A
2. Add **fractionUsed** property as boolean
3. Define **toggleFraction()** member function to switch the display mode of the inch <input>
4. Modify **convertToInch()** function to use toFraction() function
5. Define **toFraction(***number: number***)** function to convert to a fractional number to the closest fractional format with the base 16. The programming logics of **toFraction(***number***)** are; (Also see **Q3** page 6)

    a. Check if the number is **NaN** first using isNaN(). If so, do nothing and return "NaN"
    b. Extract the integer part using Math.trunc() to a local variable *integer*
    c. Extract the *numerator* from the decimal part (*number* - *integer*) by multiplying the base 16. The *numerator* should be an integer as well.
    d. If the *numerator* is 16, then round up the *integer* by one and set the *numerator* to 0
    e. If the *numerator* is 0, then return the *integer* only after converting it to a string
    f. If the *numerator* is not 0, then construct the fraction format below
        a. If the *integer* is 0, then the return string will be ***numerator*** `+ "/" +` `16`
        b. If the *integer* is not zero, then the string will be
        integer `+ " " +` ***numerator*** `+ "/" +` `16`

Examples: Test your **toFraction(***number : number***)** function with the following examples

1	→	1
0.1	→	2/16
1.2	→	1 3/16
1.5	→	1 8/16
1.96	→	1 15/16
1.97	→	2

Bonus (20 points max)

- Simplify the fraction form, so 2/16=1/8, 4/16=1/4, 8/16=1/2 ... (5 points)
  HINT: Must modify your existing **toFraction**(number, **simplified=false**) with additional parameter
- Allow user to choose a different base 4, 8, 16, 32 or 64 with a combobox (5 points)
  HINT: Must modify your existing **toFraction**(number, **base=16**, simplified=false)
- Display fraction to decimal conversion chart (5 points)
- Add additional features/styles to enhance your application (5 points)

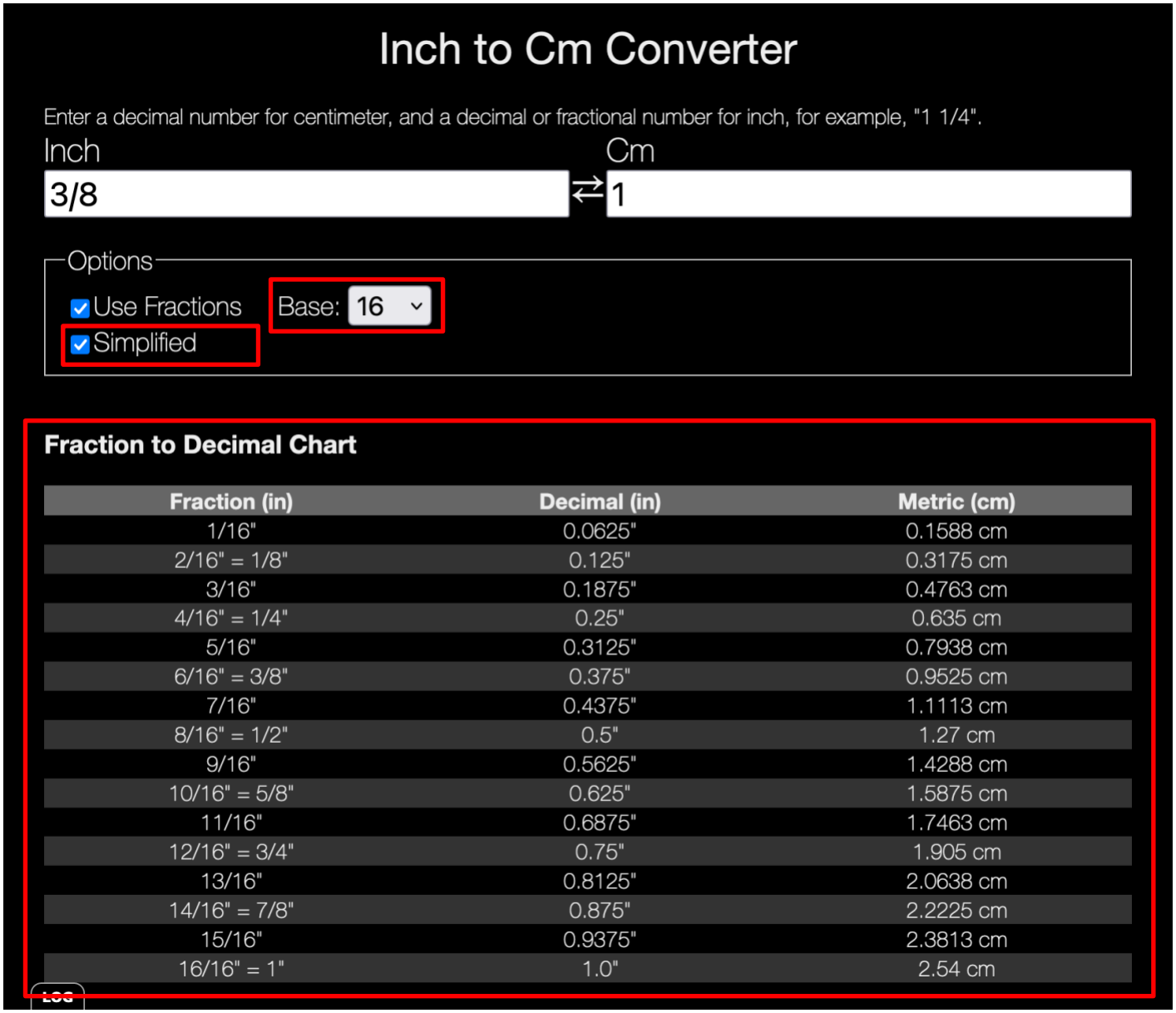


Figure 1: Example of Bonus Implementation

Q & A (Extra Notes)

Q1. Why "inch" variable (property) is a string type (not a number)?

We accept a fractional inch value from the <input> element, and the fractional format contains non-numeric character, such as '/' and 'space'.

Q2. How to implement toDecimal(*fractionString : string*)?

```
// Convert fraction inch to decimal inch
// The possible input are: 1.2, 1/2, 1 2/3
function toDecimal(fractionString: string) : number
{
    let decimal = 0; // return value

    // Split fractionString by spaces into 2 tokens using regex
    let tokens = fractionString.trim().split(/\s+/);

    // If there is only one token, it contains either a decimal number or
    // fraction number, for example, 1.2, or 1/2
    if(tokens.length == 1)
    {
        // Split the token by "/" again into 2 parts
        let parts = ...

        // If there is only 1 part, the string is a decimal number (1.2)
        // Use parseFloat() to convert it to a number
        if(parts.length == 1)
        {
            ...
        }
        // If there are 2 parts, the string is a fraction number (1/2)
        // Parse each part as number, then divide the first part by the second
        else if(parts.length == 2)
        {
            ...
        }
    }
    // If there are 2 tokens, it contains an integer and fraction, e.g. 1 2/3
    else if(tokens.length == 2)
    {
        // Parse the first token to an int and store it to the return var, decimal
        ...

        // Split the second token by "/" into 2 parts again
        // If it cannot be split to 2 parts, it is not a valid fraction format.
        // Skip the following steps, and return only decimal part or NaN.
        let parts = ...
        if(parts.length == 2)
        {
            // Parse 2 parts to integers and divide them
            // Add the division to number then return it
            ...
        }
    }
    return decimal;
}
```

Q3. How to implement toFraction(*number : number*)?

```
// Convert decimal number to fractional inch with base 16 as string
// The fraction number may have 3 parts: integer, numerator and denominator, "1 2/16"
function toFraction(number: number) : string
{
    // Check if the number is NaN first using isNaN().
    // If so, do nothing and return "NaN"
    if(isNaN(number))
        return "NaN";

    // Extract the integer part using Math.trunc() to a local variable integer
    let integer = ...

    // Extract the numerator from the decimal part, (number - integer)
    // by multiplying the base 16. The numerator should be an integer as well.
    let numerator = ...

    // If the numerator is 16, then round up the integer by one and
    // set the numerator to 0
    if(numerator == 16)
    {
        ...
    }

    // If the numerator is 0, then return the integer only after converting it
    // to a string
    if(numerator == 0)
    {
        return ...
    }
    // If the numerator is not 0, then construct the fraction format below
    else
    {
        // If the integer is 0, then the return string will be numerator + "/" + 16
        if(integer == 0)
        {
            return ...
        }
        // If the integer is not zero, then the string will be
        // integer + " " + numerator + "/" + 16
        else
        {
            return ...
        }
    }
}
```

Q4. How to include Logger.js (or other JS libraries) to an Angular project?

STEP1: Add Logger.js file to **src/assets/** folder

STEP2: Modify **angular.json** file

```
"projects": {
    "assignment3": {
        ...
        "scripts": [ "src/assets/Logger.js" ]
```

STEP3: Declare the class and function names in **src/app/app.component.ts** before the class definition

```
declare let Logger : any
declare let log : any
```