Assignment 6 – Future Value App

Setup

Each week you will be asked to create a new folder under web-330 following a naming convention of **<week>-number**. If we are on week two, the folder name should be **week-2**. All files associated with the weekly assignment will be added to the appropriate folder. All programs must be linked in the appropriate landing page. Projects will be linked under the "Projects" section of the index.html landing page.

The document title of all HTML files in this course must say "WEB 330 – Enterprise JavaScript II." And, all HTML and CSS files must be valid HTML/CSS, tested through the WC3 validator. The links were provided during WEB 200 and 231. As part of your submission, be sure to include screenshots of the results from the validation tests (HTML and CSS validators).

User interface styling and formatting requirements are located in the Web 330 HTML, CSS, and JavaScript Requirements document.

HTML: **<yourLastName>-future-value.html**
CSS: **<yourLastName>-future-value.css, flex-layout.css**
JS: **required-field.js, float-field.js, float-max-field.js, float-min-field.js, finance-calculator.js, validator.js**

Grading Reminders

A. (rubric) All code sources (.html, .css, .js) are measured against
    1. Code functionality: Does it work? Does it meet requirements?
    2. Adherence to standards and conventions. Are you using the appropriate data types, including proper indention, are variables named appropriate (variable x is an example of poor naming conventions), is there an appropriate use of whitespace, is the code organized, and are semicolons being used to terminate code sentences?

3. Efficiency: Use of language features. Are you practicing DRY (Don't-Repeat-Yourself?), are you leveraging built-in language features where appropriate, and are you using classes/functions to reduce code clutter?
4. Documentation: Code is maintainable by others
    i. Code comments are present in all blocks of code, written as full sentences, free of grammatical errors, and function/class parameters and data types have been identified.
    ii. Code attribution is present in all files and authorship is clearly annotated.
5. Error trapping/handling. Are there errors in the program? Is there evidence of coding best practices to reduce user errors?
6. Assignment Specific Compliance. Does the delivered solution follow the instructions, as they are written? Does the output match what was provided in the screenshots (including spaces, styling, etc.)?

Required Modifications

- Cite any sources in your opening programmer's comment
- Link the appropriate CSS, JavaScript, and Google fonts
- flex-layout.css

Additional Styling Requirements

a. Visit the courses GitHub repository and look for a file named flex-layout.css.
b. Add this file to the root of your web-330 GitHub repository and add a reference to this file in the <yourLastName>-future-value.html file.
c. **WARNING – CAUTION – DANGER – Do I have your attention?** Before you leave the courses, GitHub repository please make sure you review the flex-layout.html file and practice building HTML pages using the CSS classes in the flex-layout.css file. This week's assignment is using the flex-layout.css file to build the FutureValue form. At a high-level, the flex-layout.css file allows you to build HTML sections following a grid-like structure. Columns are created using col-<length> and rows are created using the row class. The max number of columns in a row is 12. In other words, if you want three

side by side divs, you will need to make sure the total column count does not exceed 12. This file is heavily influenced by Angular's Flex-Layout and Bootstraps grid system.

- required-field.js

Additional JavaScript Requirements

a. Create a class named RequiredField.
b. Create a constructor and supply it with two parameters: name and field.
c. Set the parameters to class properties.
d. Create a function named validate. In the body of the function return Boolean(this.field), which will return true if this.field is a string value and false if not.
e. Create a function named getMessage() and return the string message "<name> is a required field."
f. Export the class.

- float-field.js

Additional JavaScript Requirements

a. Create a class named FloatField.
b. Create a constructor and supply it with two parameters: name and field.
c. Set the parameters to class properties.
d. Create a function named validate. In the body of the function return true if you can parseFloat the value and false if not (hint: you will need to check for NaN values; you can do this by calling JavaScript's built-in isNaN() function).
e. Create a function named getMessage() and return the string message "<name> must be a float value. You entered <field>"
f. Export the class.

- float-max-field.js

Additional JavaScript Requirements

a. Create a class named FloatMaxField.

b. Create a constructor and supply it with three parameters: name, field, and max.
c. Set the parameters to class properties.
d. Create a function named validate. In the body of the function parseFloat the field value and return true if the value is less than the max parameter. Otherwise, return false.
e. Create a function named getMessage() and return the string message "<name> must be less than <max>. You entered <field>"
f. Export the class.

- float-min-field.js

Additional JavaScript Requirements

a. Create a class named FloatMinField.
b. Create a constructor and supply it three parameters: name, field, and min.
c. Set the parameters to class properties.
d. Create a function named validate. In the body of the function parseFloat the field value and return true if the value is greater than min. Otherwise, return false.
e. Create a function named getMessage() and return the string message "<name> must be more than <min>. You entered <field>"
f. Export the class.

- finance-calculator.js

Additional JavaScript Requirements

a. Create a class named FinanceCalculator.
b. Create a static property named MONTHS_IN_YEAR and assign it a default value of 12.
c. Create a static function named calculateFutureValue with three parameters: monthlyPayment, rate, and years.

Additional JavaScript Requirements

1) Create a variable named month and assign it the parameter years times the property MONTHS_IN_YEAR.

2) Create a variable named interestRate and assign it the calculation: **1 + rate / 100**
3) Create a variable named presentValue and assign it the monthlyPayment parameter times the months variable.
4) Create a variable named futureValue and assign it the calculation: presentValue * (Math.pow(interestRate, months))
5) Return the futureValue and set the precision to two decimal places.

d. Create a static function named convertToCurrency with a single parameter named field.

Additional JavaScript Requirements

1) Create a variable named currencyFormatter and use JavaScript's built-in International Number Formatter. Set the style to currency and set the currency to USD.
2) Return the currencyFormatter.format() function, passing-in the field parameter. See Exhibit A.
3) Export the class.

Exhibit A. JavaScript's International Currency Formatter

```
let currencyFormatter = new Intl.NumberFormat( locales: "en-US", options: {
    style: "currency",
    currency: "USD"
});

return currencyFormatter.format(field);
```

- validator.js

Additional JavaScript Requirements

a. Add import statements for RequiredField, FloatField, FloatMinField, FloatMaxField.
b. Create two class properties: validators and messages. Assign each of the properties an empty array.
c. Create a class constructor with two parameters: name and field.
d. Create a function named addRequiredField() and in the body push a new instance of the RequiredField class to the validators array using the name and field as parameters.

e. Create a function named addRequiredFloatField() and in the body of the function push a new instance of the FloatField class to the validators array using the name and field as parameters.

f. Create a function named addFloatMinField() with a single parameter for min. In the body of the function push a new instance of the FloatMinField class to the validators array using the name, field, and min as parameters.

g. Create a function named addFloatMaxField() with a single parameter for max. In the body of the function push a new instance of the FloatMaxField class to the validators array using the name, field, and max as parameters. See Exhibit B.

Exhibit B. addFloatMaxField(max)

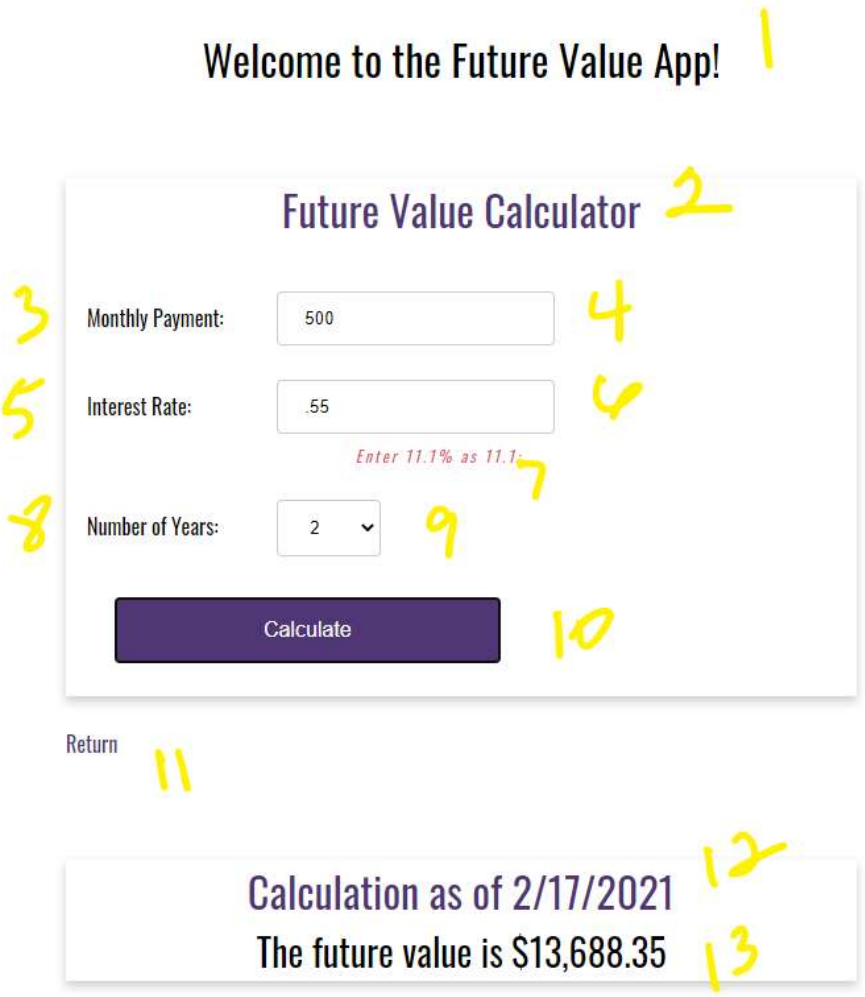
```
addFloatMaxField(max)
{
    this.validators.push(new FloatMaxField(this.name, this.field, max));
}
```

h. Create a function named validate()

Additional JavaScript Requirements

1) Using a for…of statement, iterate over the validators array and call the iterated objects validate function. If false, push the iterated objects getMessage() function to the class properties message array and return the value false.
2) Outside of the for…of statement return true.
3) Note: the validate() function should return a Boolean true or false. Each time you iterate over the validators array you will call the validate() function on the iterated object (the objects are the RequiredField, FloatField, FloatMinField, and FloatMaxField). If the call to the validate() function on the iterated object returns false, we add the iterated objects getMessage() to the class properties messages array. Otherwise, we return true, which lets the calling program know that the values validated correctly.

Exhibit C. User Interface (final solution)



1. h1 with a CSS class of app-header and a text value of "Welcome to the Future Value App!"
2. card-title with a text value of "Future Value Calculator"
    a. Below this section and above item 3 and 4 in Exhibit C you will need to create an empty row with a 12 column div and assign it the CSS danger and an id of errorLog.

Additional Styling Requirements

1) In <yourLastName>-future-value.css add a CSS class for danger. Float the div to the right, set the text color to df546a, set the font size to 12 pixels, and set the letter spacing to 2 pixels.
2) In <yourLastName>-future-value.css add a CSS class for .form-label. Set the display to inline block and give the top padding a value of 1rem.
3) Note: all form-field labels will need to be assigned the CSS class form-label (see Exhibit C, items 3, 5, and 8).

3. 5 column form-field with a text value of "Monthly Payment:"
4. 7 column form-field input with an id of txtMonthlyPayment
5. 5 column form-field with a text value of "Interest Rate:"
6. 7 column form-field input with an id of txtYearlyRate
7. HTML <i> with a CSS class of danger and a text value of "Enter 11% as 11.1:"
8. 5 column form-field with a text value of "Number of Years:"
9. 7 column HTML select element with and id of listNumOfYears a CSS class of drop-down-menu and options for years 1-10. Set the value 1-10 to match the displayable text.

Exhibit D. listNumOfYears

```
<option value="5">5</option>
```

10. form-field button with a text value of Calculate and an id of btnCalculate.
11. Anchor tag with a link back to the inde.html landing page.
12. card-title with a text value of "Calculation as of" and a span tag with an id of today (hint: we will bind the date to the span tag using JavaScript).
13. card-content with an id of futureValue.

<yourLastName>-future-value.html

a. Add import statements for the classes FinanceCalculator and Validator.
b. Find the div with the id today and set the innerHTML to a new JavaScript date object. Using the toLocaleDateString function and pass-in "en-US"
c. Register an onclick even for the btnCalculator

Additional JavaScript Requirements

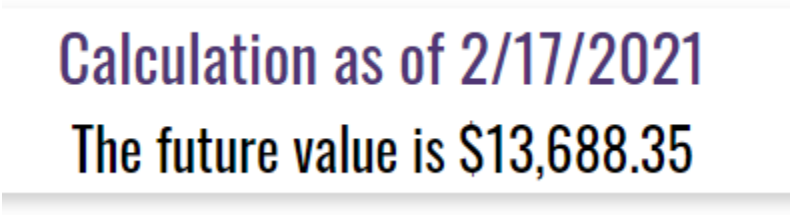
1) Create a variable named monthlyPayment and assign it the value from the txtMonthlyPayment input field.
2) Create a variable named rate and assign it the value from the txtYearlyRate input field.
3) Create a variable named years and assign it the value from the listNumOfYears select element.
4) Create a variable named errorLogEl and assign it the element errorLog.
5) Instantiate a new Validator class supplying it with the values "Monthly Payment" and the variable monthlyPayment (see item 1) and assign that to a new variable named monthlyPaymentValidator.
6) Call the addRequiredField() function on the monthlyPaymentValidator to make the field a required field.
7) Call the addRequiredFloatFIeld() function on the monthlyPaymentValidator to make the field a required float field.
8) Call the addFloatMinField() function on the monthlyPaymentValidator and supply it a value of 100 to make the field minimum value of 100.
9) Instantiate a new Validator class supplying it with the values "Interest Rate" and the rate variable (see item 2) and assign it to a new variable named rateValidator.
10) Call the addRequiredField() function on the rateValidator to make the field a required field.
11) Call the addRequiredFloatField() function on the rateValidator to make the field a required float field.
12) Call the addFlaotMaxField() function on the rateValidator and supply it a value of 100 to make the fields maximum value 100.
13) Create a variable named errorLog and set it to an empty array.
14) Using an if statement, call the monthlyValidator.validate() function and if the response is false, use a for…of statement to iterate over the messages fields and push each message to the errorLog variable. See Exhibit D.

Exhibit D. validation example

```
if (!monthlyPaymentValidator.validate()) {
    for (let msg of monthlyPaymentValidator.messages)
    {
        errorLog.push(msg);
    }
}
```

15) Using an if statement, call the rateValidator.validate() function and if the response is false, use a for…of statement to iterate over the messages field and push each message to the errorLog variable (refer to Exhibit D for an example).
16) Using an if…else statement, if monthlyPaymentValidator.validate() and rateValidator.validate() are true
    i. If true,
        - Clear the errorLogEl.innerHTML
        - Create a variable named futureValue and call the static calculateFutureValue off of the FinanceCalculator supplying it with the variables monthlyRate, rate, and years.
        - Bind the futrueValue results to the innerHTML of the futureValue div. See exhibit E. for the output expectations.

Exhibit E. FutureValue message



    ii. If false,
        - Clear the errorLogEl.innerHTML
        - Create a variable named errorLogMessage and build an unordered list of the messages in the errorLog array (hint: you will need to use a for…of statement to iterate over the array of error messages).

- Bind the string errorLogMessage to the errorLogEl.innerHTML. See Exhibit F.

Exhibit F. Required field error messages

Exhibit G. Required float field error messages

Exhibit H. Min/Max field error messages