

CODE DOCUMENTATION

Content

Introduction:	2
1. Auto caps	3
Functionality	4
How it works	4
2. Character converter	5
Functionality	6
How it works	6
3. SVG coloring scoring system	7
Functionality	8
How it works	8
4. Braid wheels home page	9
Instructions	9
5. Basic Unit Converter	9
Instructions	10
6. Display JSON data React	11
Instructions	11
7. Rotiform wheels landing page	12
Instructions	12

Introduction:

The code samples do not contain corporate code, they are drafts illustrating personal day by day work without exposing corporate technology.

All provided solutions run locally and the only dependency is the 'jQueryV3_2_1.js' file and will be provided and mentioned accordingly.

1. Auto caps

Files required:

- AutoCaps.html
- AutoCaps sample text.txt (content to copy and paste in the solution in order to see the output)
- Dependency: locally executed 'jQueryV3_2_1.js' file (Provided)

Description In typing exercises where the student should type the correct answers, if the answer was a word (not phrase or sentence), case sensitivity was not mandatory. Typing the answer with lowercase letters was acknowledged to the student, which also was the default behavior of the software.

Example:

Question: "What is the opposite of black?"

Answer: "white" (Notice: the answer was placed in field called "answer box").

Problem In IOS operating system when a user typed an answer, in an answer box, the first letter was automatically capitalized. The software's scoring system used an unordered list (we will call it from now on "answer list") with the answers populated as lowercased. So even if the student typed the correct answer i.e. "white" because it was capitalized ("White") was considered wrong. This problem affected not only the eBooks currently in development or published, but all the previously released.

Until a permanent solution was provided by the software vendor, a temporary solution should be provided at least for the recently released books and the ones on development. The temporary solution was to populate the answer list with the answers both lowercased and capitalized. The so called "teachers' book" from where the answers were copied were written lowercase. So, typing all the answers capitalized was cumbersome and unproductive.

Solution I created a script where the copied answers from the "teachers book" will be extracted both lowercased and capitalized in the desired format ("answer", "Answer") in order to paste them in the "answer list" of the eBook software.

Preview

The screenshot shows a web application interface with the following elements:

- 1. Insert answers:** A text input field containing "white blue red".
- 2. Process answers** A button with a blue border.
- 3. Result:** A text area displaying the output: `"white", "White"`, `"blue", "Blue"`, and `"red", "Red"`.
- 4. Copy answers** A button.
- 5. Reset** A button.

Functionality

- Copy sample text from "AutoCaps sample text.txt" or write words space delimited (as formatted in "Teachers book") in "1. Insert answers".
- Press "2. Process answers"
- "3. Result" field populated with answers.
- Press "4. Copy answers" in order to paste them to the software's "answer list".
- Press "5. Reset" to start again
- The interface was plain simple, the priority was to be able to replace as most answer lists as possible, fast.

How it works

Two containers, one for input ("1. Insert answers" text area) and one for output div element with id "testResults"

1. After inserting the answers in the textarea with id "string" they are tested against 2 regular expressions, "findFirstLetter" which finds the first letter of every word and "findFirstSpaceLetter" which finds the first letter of every word that is in the lead of a space character.
2. If evaluation fails a "check input" message appears.
3. If evaluation succeeds the first letter of every word is capitalized.
4. Both strings, input (stored in the variable "resultBeforeEvaluation") and capitalized strings (stored in the "resultAfterEvaluation" variable) splits in substrings with String split() method.
5. Those 2 variables are concatenated by looping through their values (with the Array.length property) stored in the "listContents" and "listAddContents" respectively and append the output to the div element with id "testResults"
6. When the "Reset" button is clicked, the "clearInput" function checks if input and output is populated and after pressing "OK" in the confirmation dialog erases the content in order to insert new answers to process.
7. The "Copy answers" function "copyToClipboard" derived from [www.codepen.io \(https://codepen.io/dejanstojanovic/pen/PZJVRr/\)](https://codepen.io/dejanstojanovic/pen/PZJVRr/) customized accordingly. Investing time in creating a copy to clipboard function was neither a priority nor relevant to the urgent requirement for a solution to the problem described above.

2. Character converter

Files required:

- characterConverter.html
- character Converter sample text.txt (content to copy and paste in the solution in order to see the output)
- Dependency: locally executed 'jQueryV3_2_1.js' file (Provided)

Description The content for the eBooks, exercises, grammar and syntax rules, texts, where copied from pdf formatted documents, and pasted to the software in order to manipulate accordingly and provide the desired result.

Problem In IOS operating system when a user selected any exercise to complete and generally processing any kind of content that contained “PRIME (’), Unicode format: \u2032”, “RIGHT SINGLE QUOTATION MARK (’), Unicode format: \u2019”, “HYPHEN-MINUS (-), Unicode format: \u002D”, “NON-BREAKING HYPHEN (-), Unicode format: \u2011”, “EM DASH (—), Unicode format: \u2014”. The text was unreadable because these punctuation characters were shown in an unrecognized “format”. This problem affected not only the eBooks currently in development or published but all the previously released.

Until a permanent solution was provided by the software vendor, a temporary solution should be provided at least for the recently released books and the ones on development. The temporary solution was to manually replace these characters with “HYPHEN (-), Unicode format: \u2010” and “APOSTROPHE (’), Unicode format: \u0027” because they were not creating compatibility issues and/or find a way to automatically detect and replace these characters with “HYPHEN” and “APOSTROPHE”.

Solution I created a script where the copied content will be extracted in a compatible format (HYPHEN (-), Unicode format: \u2010 for dash and APOSTROPHE (’), Unicode format: \u0027 for single quotation) before embedding it to the eBook software.

Preview

1. Insert text:

2.

3. Result:

4. 5.

Now it's time to set up our environment.
Table 16-1 working "things" is what floats Ed's
boat.developing mobile-first web started-in fact

Now it's time to set up our environment. Table 16-1 working "things"
is what floats Ed's boat.developing mobile-first web started-in fact

Functionality

- Copy sample text from "character Converter sample text.txt" in "1. Insert text".
- Press "2. Process text"
- "3. Result" field populated with compatible text; punctuation characters replaced are highlighted with red color.
- Press "4. Copy text" in order to paste them to the eBook software .
- Press "5. Reset" to start again
- The interface was plain simple, the priority was to be able to replace as most content as possible, fast.

How it works

Like autoCaps solution, two containers, one for input ("1. Insert text" text area) and one for output div element with id "testResults"

1. After inserting the content in the textarea with id "string" it is tested against 2 regular expressions, "singleQuotation" and "hyphen" which find the incompatible punctuation characters.
2. If evaluation fails a "No match!" message appears.
3. If evaluation succeeds the incompatible punctuation characters are replaced and append the output to the div element with id "testResults".
4. When the "Reset" button is clicked ,the "clearInput" function checks if input and output is populated and after pressing "OK" in the confirmation dialog erases the content in order to insert new content to process.
5. The "Copy text" function "copyToClipboard" derived from [www.codepen.io](https://www.codepen.io/dejanstojanovic/pen/PZJVRr/) (<https://codepen.io/dejanstojanovic/pen/PZJVRr/>) customized accordingly. Investing time in creating a copy to clipboard function was neither a priority nor relevant to the urgent requirement for a solution to the problem described above.

3. SVG coloring scoring system

Files required:

- svgColoring.html
- Dependencies: none

Description SVG Coloring is a type of exercise where the user can drag and drop colors from the “color palette”, to the drawing in order to colorize the drawing.

Requirement Embed existing scoring system from the software vendors solution to this exercise.

Problem The existing scoring system (by definition), couldn’t read the SVG “fill” attribute. Manipulating the software’s vendor scoring system was not an option.

Solution I created a dedicated scoring system with no dependencies from the original scoring system. The only requirements are (in order to be embedded in the software), that it must be Internet Explorer 11 compatible and the colors in hex format.

Preview

Hypothetical color palette. User can drag and drop colors ONLY to svg paths with the class="setColor" to change the default "fill" color values



Hypothetical SVG that must be colored correctly. The fill color values are the ones that the user placed to svg paths with the class="setColor". Default fill color is white.

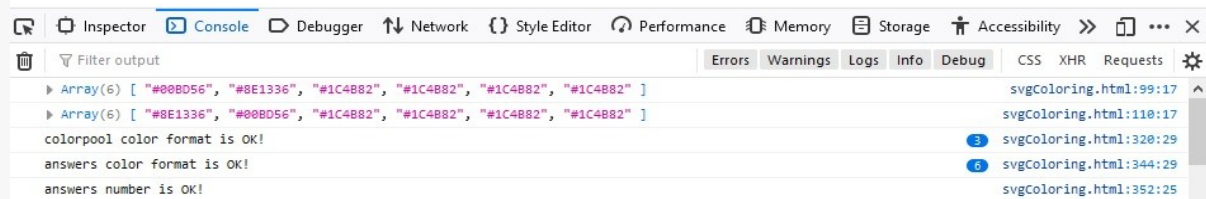


X

Score 66% (4/6)

Correct 4

Wrong 2



Functionality

- The exercise is not showing the drag and drop functionality (already implemented by the software vendor) but it is in a state where the user hypothetically has already colored the appropriate svg paths and then checks his score.
- User can drag and drop colours ONLY to svg paths with the class “setColor” to change the default "fill" colour values. The other svg paths are presented as white in the example.
- User has made the following choices: green, red, blue (4 times).
- Correct coloring: red, green, blue (4). The mistake is on purpose. To see the exercise correct go to SVG paths with id “aSVG”, “bSVG” respectively and swap the “fill” attribute hex values from #00BD56 and #8E1336 to #8E1336 and #00BD56.

How it works

1. We are looping through the “setColor” (students answers) and “t1” (correct answers list) classes, push their values in an array respectively and compare them with the helper function “isEqual” (derived from <https://vanillajstoolkit.com/helpers/isequal/>) in order to check answers validity and add the class “wrongSvg” to the wrong path(s) when the “Check” button is clicked.
2. The “scoreFunction” generates the score after the evaluation which presents the score in percentage and number of correct/wrong answers.
3. “Try again” button. After “Try-again” button is clicked, total score (percentage) and wrong answers must change, because only wrong answers (with the class “wrongSvg”) will be removed.
4. “Reset” button. After “Reset” button is clicked total score (percentage) and wrong/correct answers must reset to zero.
5. A self-invoking function “debug” is running to help correct mistakes during the creation of the exercise. Correct hex colour formatting (with regular expression) and the number of students’ answers compared to correct answers list is validated. The output is visible in developer tools=> console.

4. Braid wheels home page

Description The home page of a motorsport's wheel manufacturer, showcasing UI, UX design.

Preview




Instructions

- Download the folder “Braid Wheels Home page sample”
- In the main folder “Braid Wheels Home page sample” double click the “index.html” file to see the home page.

5. Basic Unit Converter

Description A simple UI of a unit converter made with react.js based on the reactjs.org tutorial. It is used to experiment with react.js framework and single page application methodology.

Preview



The screenshot shows a web application titled "Basic Unit Converter" with a dark blue header. In the top right corner of the header is a logo with a circular arrow and a lightning bolt, labeled "UNIT CONVERTER". Below the header, the text "Select category:" is followed by three expandable category panels. The first panel, "Temperature", is expanded and shows a thermometer icon. The second panel, "Weight", is expanded and shows a scale icon; it contains two input fields: "Enter weight in kilos" with the value "15" and "Enter weight in lbs" with the value "33.069". The third panel, "Length", is collapsed and shows a ruler icon. Each panel has a "+" or "-" icon on its right side to toggle its state.

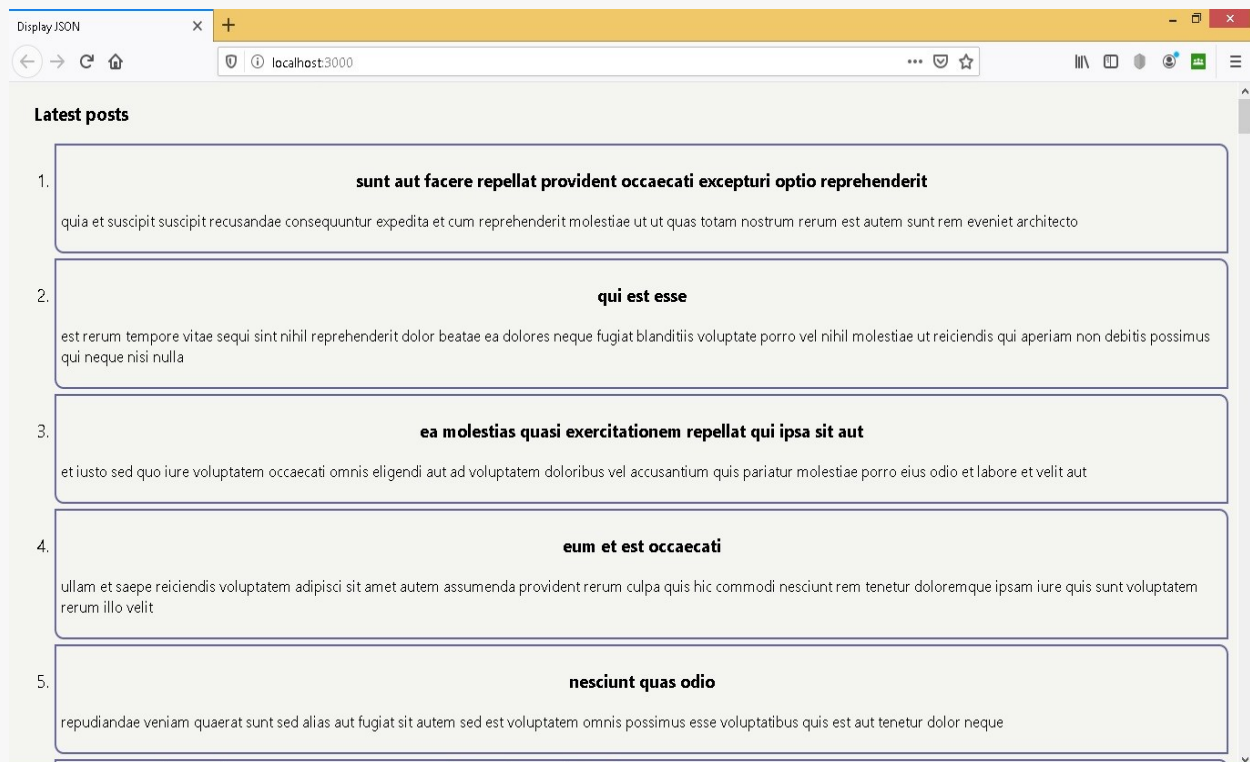
Instructions

- Download and extract the compressed folder "unitconverter.zip" in your *username* folder.
- Run this command to move to the "unitconverter" directory "C:\Users*Your Name*>cd unitconverter"
- Run this command to execute the React application "C:\Users*Your Name*\unitconverter>npm start".

6. Display JSON data React

Description A simple data retrieval with fetch API and react. It is used to experiment with react.js framework and single page application methodology.

Preview



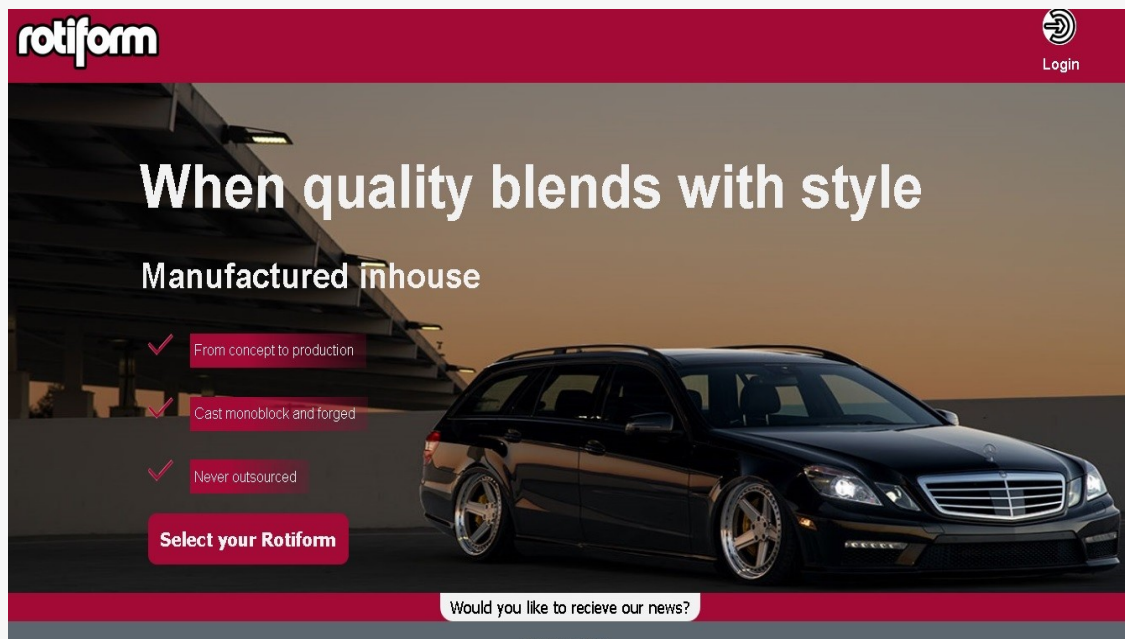
Instructions

- Download and extract the compressed folder "displayjsondata_react.7z" in your [username](#) folder.
- Run this command to move to the "unitconverter" directory "C:\Users\Your Name>cd displayjsondata_react\displayjsondata"
- Run this command to execute the React application "C:\Users\Your Name\displayjsondata_react\displayjsondata>npm start".

7. Rotiform wheels landing page

Description The landing page of a motorsport's wheel manufacturer, showcasing UI, UX design and simple newsletter form evaluation.

Preview



Instructions

- Download the folder "Rotiform wheels landing page"
- In the main folder "Rotiform wheels landing page" double click the "index.html" file to see the landing page.