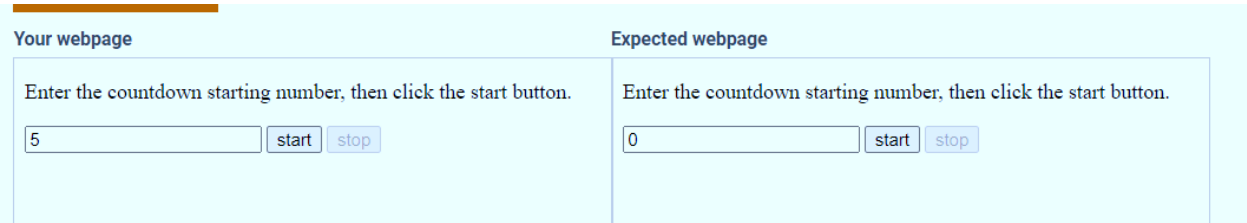
ASSIGNMENT DOM EVENTS

1. Write the JavaScript code to create a countdown timer. **10 points**
    1. Add code to startbutton's click event handler to start an interval that calls the countdown() function every second.
    2. Store the unique identifier returned by setInterval() in countdownTimerId so the interval can be canceled.
    3. Add code to countdown() to clear the countdown interval.
    4. Add code to stopbutton's click event handler to clear the countdown interval.



2. The webpage below displays a menu of food items with 3 buttons underneath: **10 points**

Insert Rule button - Calls insertRule() to add a new paragraph rule that turns the menu items' font color blue.

Change Rule button - Calls changeRule() to change the paragraph rule's color to red.

Delete Rule button - Calls deleteRule() to delete the paragraph rule, which turns the font color back to green.

Click the three buttons in order to watch the font color change from green to blue, blue to red, and finally back to green.

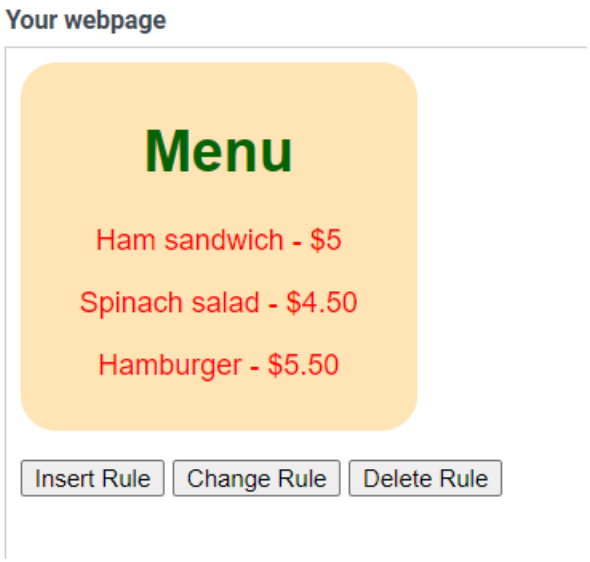
Make the following modifications:

Add code to insertRule() that inserts the rule .price { font-weight: bold; } so the prices appear bold.

Add code to changeRule() that changes the .price rule to include the property font-style set to italic so the prices appear bold and italic.

Add code to deleteRule() that deletes the .price rule so the font weight and style returns to normal.

After making the modifications, click the 3 buttons in order to verify the price font changes as expected.



3. The webpage below asks the user to enter a strong password that meets 3 criteria. When the user clicks the Submit button, the isStrongPassword() is called with the password entered. **10 points**

- If the password does not meet all 3 criteria, isStrongPassword() returns false and an error message is displayed by removing the hidden class from the error message.
- If the password meets all 3 criteria, isStrongPassword() returns true and the hidden class is added to the error message to hide the error message.

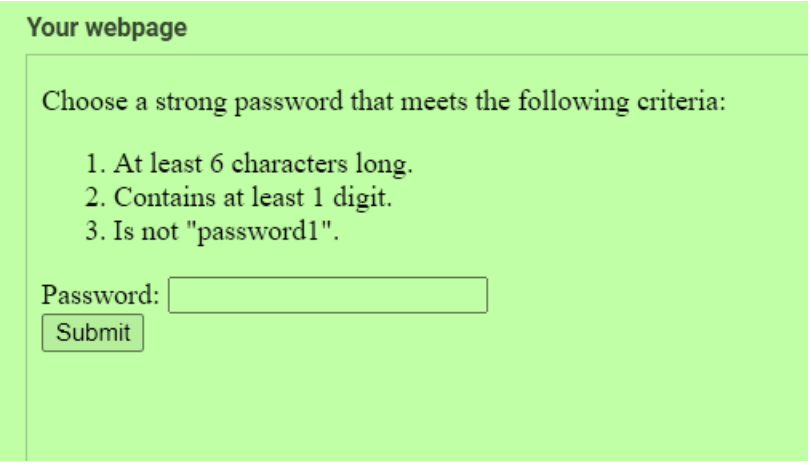
Enter some passwords that cause the error message to be visible and then hidden. Ex: Enter "abc" and press Submit to see the error message, then "abcdef1" to hide the error message.

Modify the submitBtnClick() function to do the following:

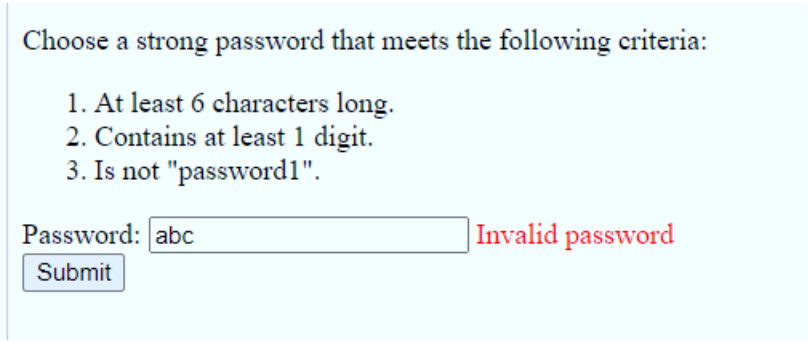
1. If isStrongPassword() returns true, then remove the error-textbox class from the password text box.
2. If isStrongPassword() returns false, then add the error-textbox class to the password text box.

After making the modifications, verify the password text box is highlighted in red only when entering an invalid password.

For an extra challenge, add the error class to the criteria that is violated when an invalid password is entered. Ex: If the password is not long enough, add the error class to the first <li> so the item becomes red.



Sample Run:



4. Download DOMLab9.html **10 points**
   Complete the JavaScript checkForm() function so that checkForm() sets the input style.backgroundColor to LightGreen for each field that passes the validation check and sets the input field's style.backgroundColor to Orange if the validation fails.

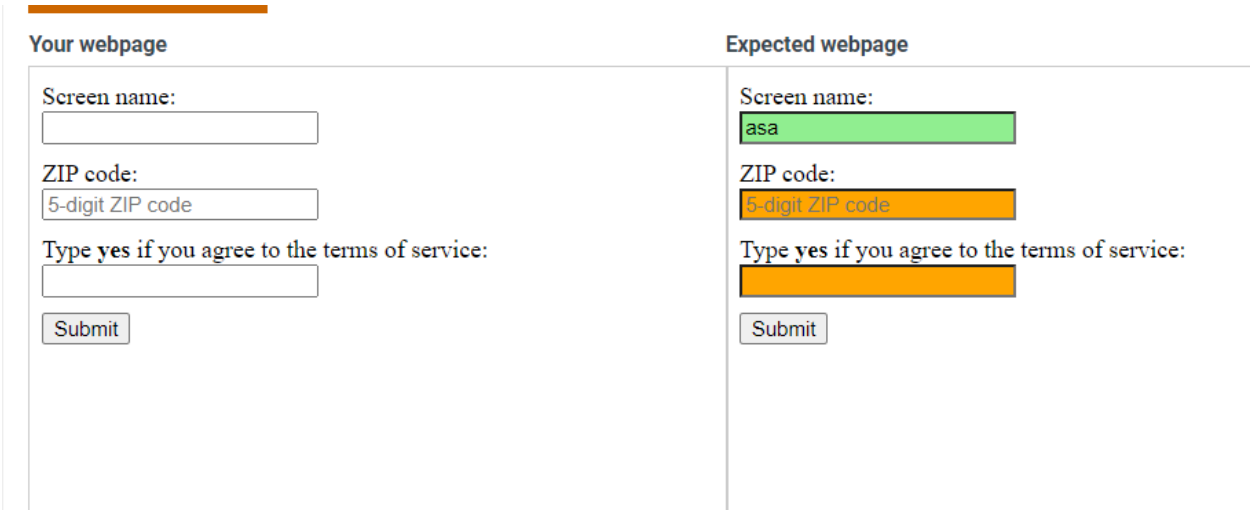
Validation rules:

The screen name field must not be empty.

The ZIP code field must be of length 5.

The TOS field must contain "yes".

Note that some browsers will override the light green color with another color if the user chooses an autofill option instead of typing a value.



Validating each field as data is entered

Alternatively, form data can be validated as the user enters data in the form by:

For each field that should be validated:

a) Register an input event handler for the field.
b) Create a global variable to track whether the field is currently valid. In most cases, this global variable should be initialized to false since the form typically starts with the field as invalid.
c) Modify the global variable as appropriate within the field's event handler.
d) Register a submit event handler for the form that verifies the global variables for each field are true.
e) If one or more of the global variables are false, call the preventDefault() method on the submit event to prevent the form from submitting to the server.

The example below uses a regular expression to verify the user enters five digits for the ZIP code. Regular expressions are discussed in more detail elsewhere. The form does not submit unless the ZIP is valid.

```html
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="UTF-8">
      <title>Terms of Service</title>
      <script src="validate.js" defer></script>
   </head>
   <body>
      <form id="tosForm">
         <label for="zip">ZIP:</label>
         <input type="text" id="zip">
         <input type="submit">
      </form>
   </body>
</html>
```

```javascript
// validate.js

let zipCodeValid = false;
let zipCodeWidget = document.querySelector("#zip");
zipCodeWidget.addEventListener("input", checkZipCode);

function checkZipCode() {
   let regex = /^\d\d\d\d\d$/;
   let zip = zipCodeWidget.value.trim();
   zipCodeValid = zip.match(regex);
}

let tosForm = document.querySelector("#tosForm");
tosForm.addEventListener("submit", checkForm);

function checkForm(event) {
   if (!zipCodeValid) {
      event.preventDefault();
   }
}
```

5. You will write three JavaScript functions to generate a bar graph of letter grades from a distribution of scores. Download the ZIP file below containing HTML and JavaScript files. Add all your code to index.js. **10 points**

Implement parseScores() (1 point)

a) Implement the parseScores() function to take a space-separated string of scores as an argument and return an array of score strings. Each score is a number in the range [0, 100].

Ex: parseScores("45 78 98 83 86 99 90 59") should return ["45", "78", "98", "83", "86", "99", "90", "59"].

Hint: The string method split() can create the array with one line of code.

b) Implement buildDistributionArray() (2 points)

Implement the buildDistributionArray() function to take an array of scores, built by parseScores(), as an argument. The function should return a grade distribution array of length 5.

The function should loop through the scores array and tally up the number of A, B, C, D, and F scores using the standard scoring system (90 and above = A, 80-89 = B, 70-79 = C, 60-69 = D, 59 and below = F). The grade totals should be stored in a distribution array where the number of As is the first number, number of Bs is the second number, etc.

Ex: buildDistributionArray(["45", "78", "98", "83", "86", "99", "90", "59"]) should return [3, 2, 1, 0, 2].

buildDistributionArray() should return [0, 0, 0, 0, 0] when the scoresArray argument is an empty array.

c) Implement setTableContent() (7 points)

Implement the setTableContent() function to take a space-separated string of scores as an argument. setTableContent() should call parseScores() and buildDistributionArray() and produce a grade distribution graph by setting the table row's inner HTML.

The table's first row (id="firstRow") should use a for each bar. Each bar gains 10 pixels in height per grade occurrence. Apply the classes from the embedded stylesheet so that each bar is a different color. The CSS vertical-align property is set for <td> elements so that the bars are aligned at the bottom of the containing cells.

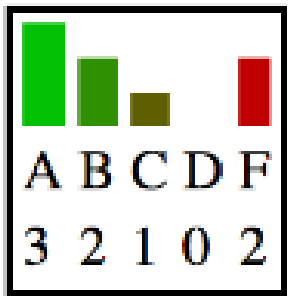
Below is a sample of what might be generated for the table's first row.

```
<tr id="firstRow">
   <td><div style="height:30px" class="bar0"></div></td>
   <td><div style="height:20px" class="bar1"></div></td>
   <td><div style="height:10px" class="bar2"></div></td>
   <td><div style="height:0px"  class="bar3"></div></td>
   <td><div style="height:20px" class="bar4"></div></td>
</tr>
```

The table's second row contains letter grade labels.

The third row (id="thirdRow") should contain the number of occurrences of each grade.

Ex: setTableContent("45 78 98 83 86 99 90 59") should produce the following table:

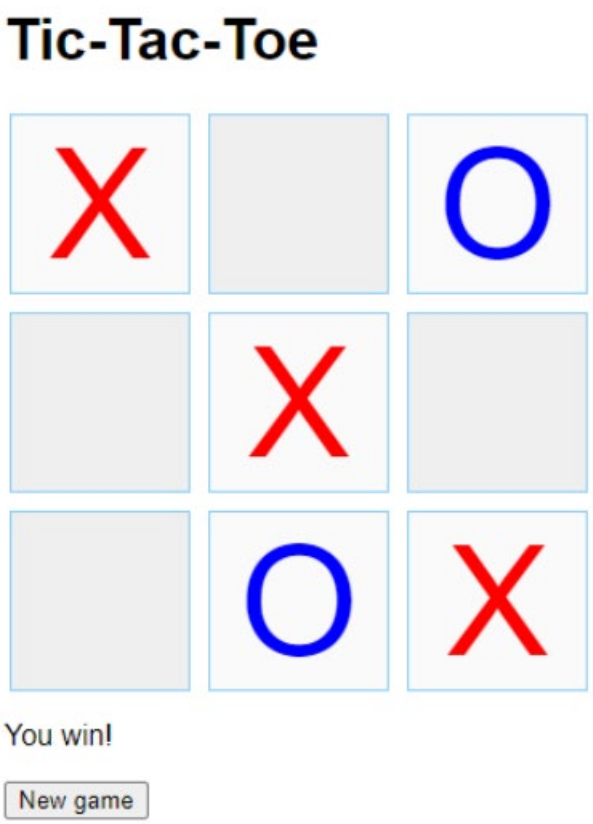


Testing your solution

The given index.js file calls setTableContent() with several different scores. Replace the scores string with other scores to verify the webpage produces a table with the correct bar graph and distributions.

6. write JavaScript to implement a Tic-Tac-Toe game, as shown below. The human is X, and the computer is O. A "New game" button starts a new game. **10 points**

Tic-Tac-Toe web page showing a 3x3 game board with an X win



Download the ZIP file below containing HTML, CSS, and JavaScript files. The index.html file contains all needed page elements for a game of Tic-Tac-Toe:

- A div with ID gameBoard and 9 buttons forms the game board. CSS in tictactoe.css converts the div and buttons into a 3x3 grid.
- A paragraph with ID turnInfo, initially containing text "TURN INFO", indicates the turn is the player's or computer's.
- A "New game" button with ID newGameButton allows the player to clear the board and start a new game.

Investigate the stylesheet

The tictactoe.css file declares .x and .o rules to set the X and O button colors. Other CSS rules style the grid and buttons.

Investigate the JavaScript

The tictactoe.js script has six declarations:

- playerTurn: Boolean variable that is true when the turn belongs to the player and false when the turn belongs to the computer.

- computerMoveTimeout: ID of an active timeout for the computer's move, or 0 if no such timeout exists.
- gameStatus: Object that contains four possible game statuses. The checkForWinner() function returns the appropriate game status.
- domLoaded(): Function that is called when the DOM loads to start the game. Events for the "New game" button click and game board button clicks are registered. Then newGame() is called to start the game. The domLoaded() function is implemented for you and requires no alteration.
- getGameBoardButtons(): Function that returns an array of the 9 <button> elements from the game board. The first 3 elements are the top row, the next 3 the middle row, and the last 3 are the bottom row. The getGameBoard() function is implemented for you and requires no alteration.
- checkForWinner(): Function that returns a gameStatus value indicating if the human has won, if the computer has won, if a draw occurs, or if more moves are available.

Implement newGame() (2 points)

Implement the newGame() function to do the following:

1. Use clearTimeout() to clear the computer's move timeout and then set computerMoveTimeout back to 0.
2. Loop through all game board buttons and set the inner HTML of each to an empty string. Also remove the class name and disabled attribute. The disabled attribute prevents the user from clicking the button, but all the buttons should be clickable when starting a new game.
3. Allow the player to take a turn by setting playerTurn to true.
4. Set the text of the turn information paragraph to "Your turn".

Implement boardButtonClicked() (2 points)

Implement the boardButtonClicked() function to do the following:

- If playerTurn is true:
    1. Set the button's inner HTML to "X".
    2. Add the "x" class to the button.
    3. Set the button's disabled attribute to true so the button cannot be clicked again.
    4. Call switchTurn() so the computer can take a turn.

Implement switchTurn() (3 points)

Implement the switchTurn() function to do the following:

- Call checkForWinner() to determine the game's status.
- If more moves are left, do the following:

1. If switching from the player's turn to the computer's turn, use setTimeout() to call makeComputerMove() after 1 second (1000 milliseconds). Assign the return value of setTimeout() to computerMoveTimeout. The timeout simulates the computer "thinking", and prevents the nearly-instant response to each player move that would occur from a direct call to makeComputerMove().
2. Toggle playerTurn's value from false to true or from true to false.
3. Set the turn information paragraph's text content to "Your turn" if playerTurn is true, or "Computer's turn" if playerTurn is false.

- In the case of a winner or a draw game, do the following:

    1. Set playerTurn to false to prevent the user from being able to place an X after the game is over.
    2. If the human has won, display the text "You win!" in the turn info paragraph.
    3. If the computer has won, display the text "Computer wins!" in the turn info paragraph.
    4. If the game is a draw, display the text "Draw game" in the turn info paragraph.

Implement makeComputerMove() (3 points)

Implement the makeComputerMove() function to do the following:

1. Choose a random, available button, and set the button's inner HTML to "O".
2. Add the "o" class to the button.
3. Set the button's disabled attribute to true.
4. Call switchTurn() at the end of the function to switch back to the player's turn.

Submissions :

Submit the html ,css and .js files in one folder. Zip the folder and submit.

Total Point : 60