CHECKLIST MVC WITH WEB STORAGE

1. Set up. We will work on `checklist.js` referred to in `checklist.html`:

- `"use strict";`: This line is a directive that tells the JavaScript engine to enforce stricter parsing rules, which can help catch errors early on. Future-proofs code.
- Variable Declarations:
  - `var todoList`: This variable stores a reference to the HTML element with the class `todo-list`. This element will likely be an unordered list (`<ul>`) where todo items are displayed.
  - `var todoForm`: This variable stores a reference to the HTML element with the class `add-todo`. This element is likely a form that allows users to add new todo items.
  - `var removeList`: This variable stores a reference to the HTML element with the class `remove-list`. This element might be a button that allows users to clear the entire todo list.

```
"use strict";

var todoList = document.querySelector('.todo-list');
var todoForm = document.querySelector('.add-todo');
var removeList = document.querySelector('.remove-list');
```

2. Initializing todo items:

- `var items = ...`: This line declares a variable named `items` and assigns it an array.
  - The `JSON.parse(localStorage.getItem('todoList'))` part tries to retrieve data from browser's Local Storage with the key `'todoList'`. If data exists and is valid JSON, it gets parsed into an array and assigned to `items`.
  - The `||` (OR) operator is a short-circuit operator. If the localStorage data retrieval fails (is null or invalid), the right side of the OR is used.
  - The right side of the OR is a default array containing two example todo items with title and done status.

```
//short circuit "or" tries to get from Web Storage, then JSON
var items = JSON.parse(localStorage.getItem('todoList')) || [
  {
    title: 'Learn JavaScript',
    done: false
  },
  {
    title: 'TypeScript',
    done: false
  }

];
```

3. Add a todo item:

- `function addTodo(e) { ... }`: This defines a function named `addTodo` that takes an event object (`e`) as a parameter.
    - `e.preventDefault();`: This line prevents the default form submission behavior, which would normally cause a page reload.
    - `var title = this.querySelector('[name=item]').value;`: This line retrieves the value entered in the form field with the name `'item'`. This field likely corresponds to an input box where users enter the todo text.
    - `var todo = { ... }`: Creates a new JavaScript object with properties `title` (from the form field) and `done` set to `false` (not completed).
    - `items.push(todo);`: This line adds the newly created todo object to the `items` array.
    - `saveTodos();`: This calls the `saveTodos` function (explained later) to persist the updated list in Local Storage.
    - `this.reset();`: This resets the form, clearing the input field for the next todo entry.

```
function addTodo(e) {
  //e.preventDefault();
  var title = this.querySelector('[name=item]').value;
  var todo = {
    title: title,
    done: false
  };
  items.push(todo);
  saveTodos();
  this.reset();
  e.preventDefault();
}
```

4. Create the todo list:

- `function createList(list = [], listTarget) { ... }`: This defines a function named `createList` that takes two arguments:
    - `list` (optional): An array of todo items to be displayed. Defaults to an empty array.

    - `listTarget`: The HTML element where the list will be displayed (presumably the `todoList` variable).
- The function iterates through the `list` array using `map`. For each item:
    - It creates an HTML list item (`<li>`) element.
    - It creates a checkbox (`<input type="checkbox">`) with a unique ID (`todo` + index) and a `data-index` attribute set to the current item's index in the array.
    - The checkbox is checked if the item's `done` property is true.
    - It creates a label (`<label>`) element for the checkbox, displaying the item's title.
    - It creates a span (`<span>`) element with the class `remove` and `data-index` attribute set to the current item's index. This span likely displays an "X" button to remove the todo item.
- Finally, it joins all the created HTML elements into a single string and sets the `innerHTML` property of the `listTarget` (the todo list) to display the list.

```
function createList() {
  var list = arguments.length > 0 && arguments[0] !==
undefined ? arguments[0] : [];
  var listTarget = arguments[1];

  listTarget.innerHTML = list.map(function (item, i) {
    return '<li><input type="checkbox" id="todo' + i + '" data-
index="' + i + '"' + (item.done ? 'checked' : '') + ' /><label
for="todo' + i + '">' + item.title + '<span class="remove" data-
index="' + i + '">X</span></li>';
  }).join('');
}
```

5. Mark a todo item as done/undone:

- **`function toggleDone(e) { ... }`**: This defines a function named `toggleDone` that takes an event object (`e`) as a parameter.
    - **`var el = e.target;`**: This line gets a reference to the element that triggered the click event.

```
function toggleDone(e) {
  //if(!e.target.matches('input')) return;
  var el = e.target;
  //dataset gets all data- attributes
  var index = el.dataset.index;
  items[index].done = !items[index].done;
  saveTodos();
}
```

6. Remove a single todo item:

- `function removeSingle(e) { ... }`: This defines a function named `removeSingle` that takes an event object (`e`) as a parameter.
    - `if (e.target.className != "remove") { return; }`: This line checks if the clicked element has the class name `"remove"`. This likely corresponds to the "X" button on a todo item. If not, the function exits without doing anything.
    - `var el = e.target;`: This line gets a reference to the element that triggered the click event (the "X" button).
    - `var index = el.dataset.index;`: This line retrieves the `data-index` attribute from the clicked element. This attribute stores the index of the corresponding todo item in the `items` array.
    - `items.splice(index, 1);`: This line removes the todo item at the specified `index` from the `items` array using the `splice` method.
    - `saveTodos();`: This calls the `saveTodos` function (explained later) to persist the updated list in Local Storage.

```
function removeSingle(e) {
  if (e.target.className != "remove") {
  return;
saveTodos();
  }

  var el = e.target;
  var index = el.dataset.index;
  items.splice(index, 1);
  saveTodos();
}
```

7. Saving the todo list:

- `function saveTodos() { ... }`: This defines a function named `saveTodos`.
    - `localStorage.setItem('todoList', JSON.stringify(items));`: This line converts the `items` array (containing todo objects) into JSON format using `JSON.stringify` and then stores it in Local Storage with the key `'todoList'`. This allows the todo list to persist even after the browser window is closed.
    - `createList(items, todoList);`: This calls the `createList` function (explained earlier) to regenerate and update the displayed todo list based on the updated `items` array.

```
function saveTodos() {
  localStorage.setItem('todoList', JSON.stringify(items));
  createList(items, todoList);
}
```

8. Removing all todo items:

- `function removeData() { ... }`: This defines a function named `removeData`.
    - `items = [];`: This line sets the `items` array to an empty array, effectively removing all todo items from memory.
    - `localStorage.removeItem('todoList');`: This line removes the stored todo list data from Local Storage using `localStorage.removeItem`.
    - `createList(items, todoList);`: This calls the `createList` function (explained earlier) to regenerate and display an empty todo list.

```
function removeData() {
  items = [];
  localStorage.removeItem('todoList');
  createList(items, todoList);
}
```

9. Event listeners and initialization:

- These lines set up event listeners for user interactions:
    - `todoList.addEventListener('click', toggleDone);`: This listens for `click` events on the `todoList` element (the entire todo list container). When a click occurs, it calls the `toggleDone` function to handle marking todo items as done/undone.
    - `todoList.addEventListener('click', removeSingle);`: This listens for `click` events on the `todoList` element as well. When a click occurs, it calls the `removeSingle` function to handle removing individual todo items.
    - `todoForm.addEventListener('submit', addTodo);`: This listens for `submit` events on the `todoForm` element (the form for adding new items). When the form is submitted, it calls the `addTodo` function to handle adding a new todo item to the list.
    - `removeList.addEventListener('click', removeData);`: This listens for `click` events on the `removeList` element (the button for clearing the entire list). When clicked, it calls the `removeData` function to remove all todo items from the list.

- Finally, `createList(items, todoList);` is called one last time to ensure the initial todo list is displayed based on the data retrieved from Local Storage or the default items.

```
todoList.addEventListener('click', toggleDone);
todoList.addEventListener('click', removeSingle);
todoForm.addEventListener('submit', addTodo);
removeList.addEventListener('click', removeData);

// Init list
createList(items, todoList);
```