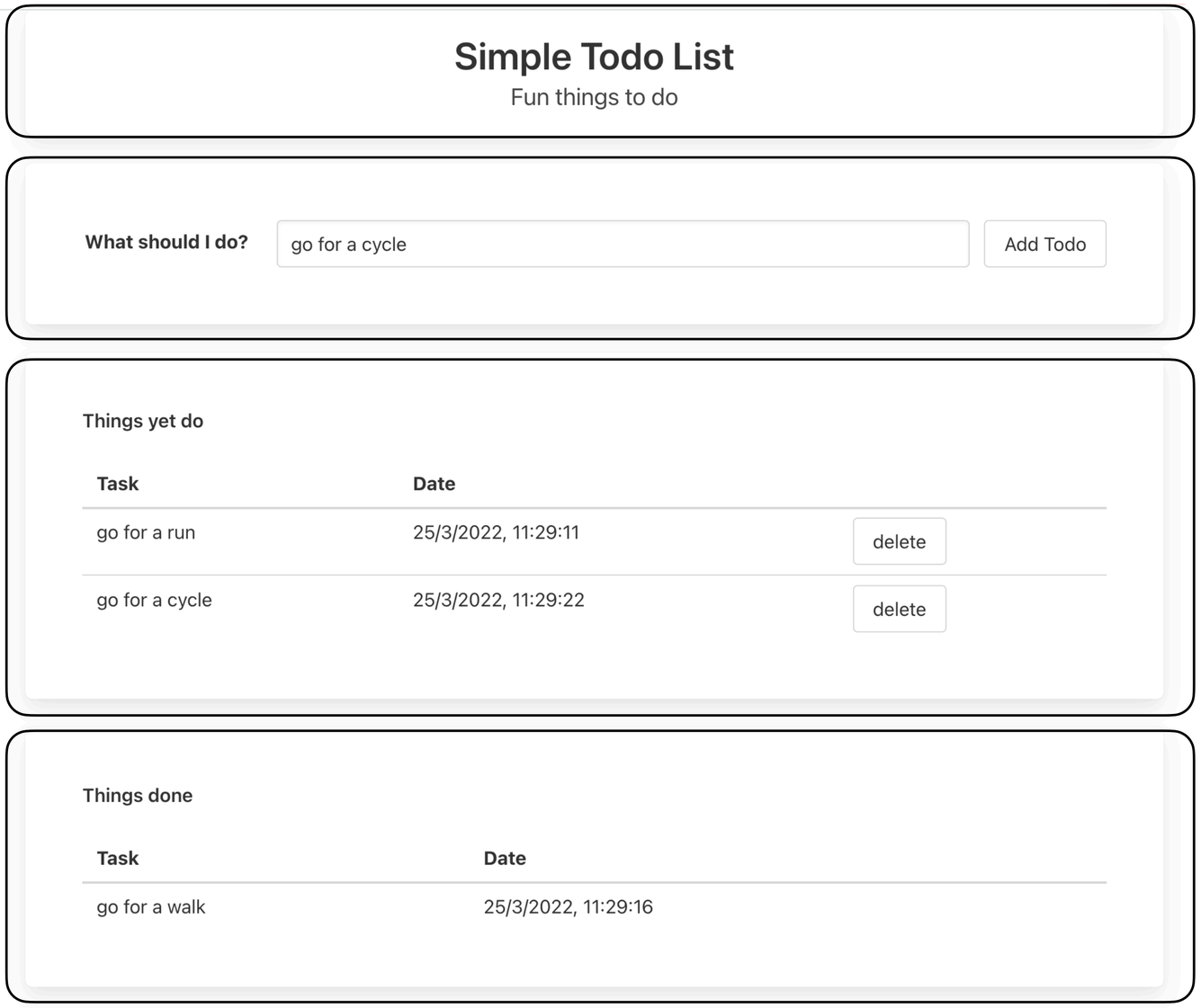
Todo DOM

Manipulating the DOM to implement TODO features

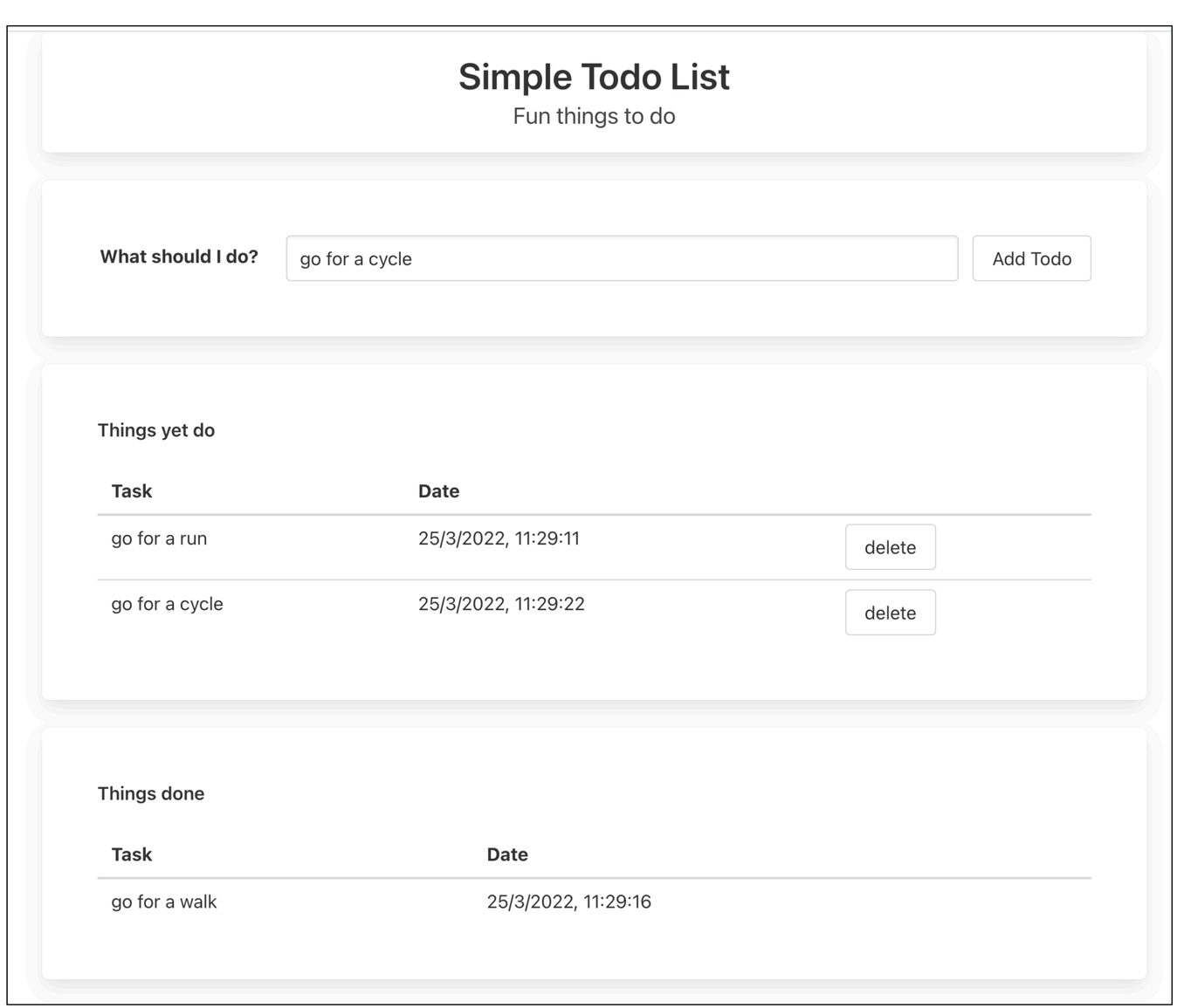
Todo UX




```
<div class="box has-text-centered">
  <div class="title"> Simple Todo List</div>
  <div class="subtitle">Fun things to do</div>
</div>
<div class="section box">
  <div class="field is-horizontal">
    <div class="field-label is-normal">
      <label class="label">What should I do?</label>
    </div>
    <div class="field-body">
      <div class="field">
        <p class="control">
          <input id="todo-id" class="input" type="text" placeholder
        </p>
      </div>
      <button onClick="addTodo()" class="button">Add Todo</button>
    </div>
  </div>
</div>
<div class="section box">
  <div class="title is-6">Things yet do</div>
  <table id="todo-table" class="table is-fullwidth">
    <thead>
      <th>Task</th>
      <th>Date</th>
      <th></th>
    </thead>
    <tbody>
      <tr></tr>
    </tbody>
  </table>
</div>
<div class="section box">
  <div class="title is-6">Things done</div>
  <table id="done-table" class="table is-fullwidth">
    <thead>
      <th>Task</th>
      <th>Date</th>
      <th></th>
    </thead>
    <tbody>
      <tr></tr>
    </tbody>
  </table>
</div>
```

Include Javascript

```
    </div>
    <div class="section box">
      <div class="title is-6">Things yet do</div>
      <table id="todo-table" class="table is-fullwidth">
        <thead>
          <th>Task</th>
          <th>Date</th>
          <th></th>
        </thead>
        <tbody>
          <tr></tr>
        </tbody>
      </table>
    </div>
    <div class="section box">
      <div class="title is-6">Things done</div>
      <table id="done-table" class="table is-fullwidth">
        <thead>
          <th>Task</th>
          <th>Date</th>
          <th></th>
        </thead>
        <tbody>
          <tr></tr>
        </tbody>
      </table>
    </div>
  </div>
  <script src="todo.js" type="text/javascript"></script>
</body>
</html>
```



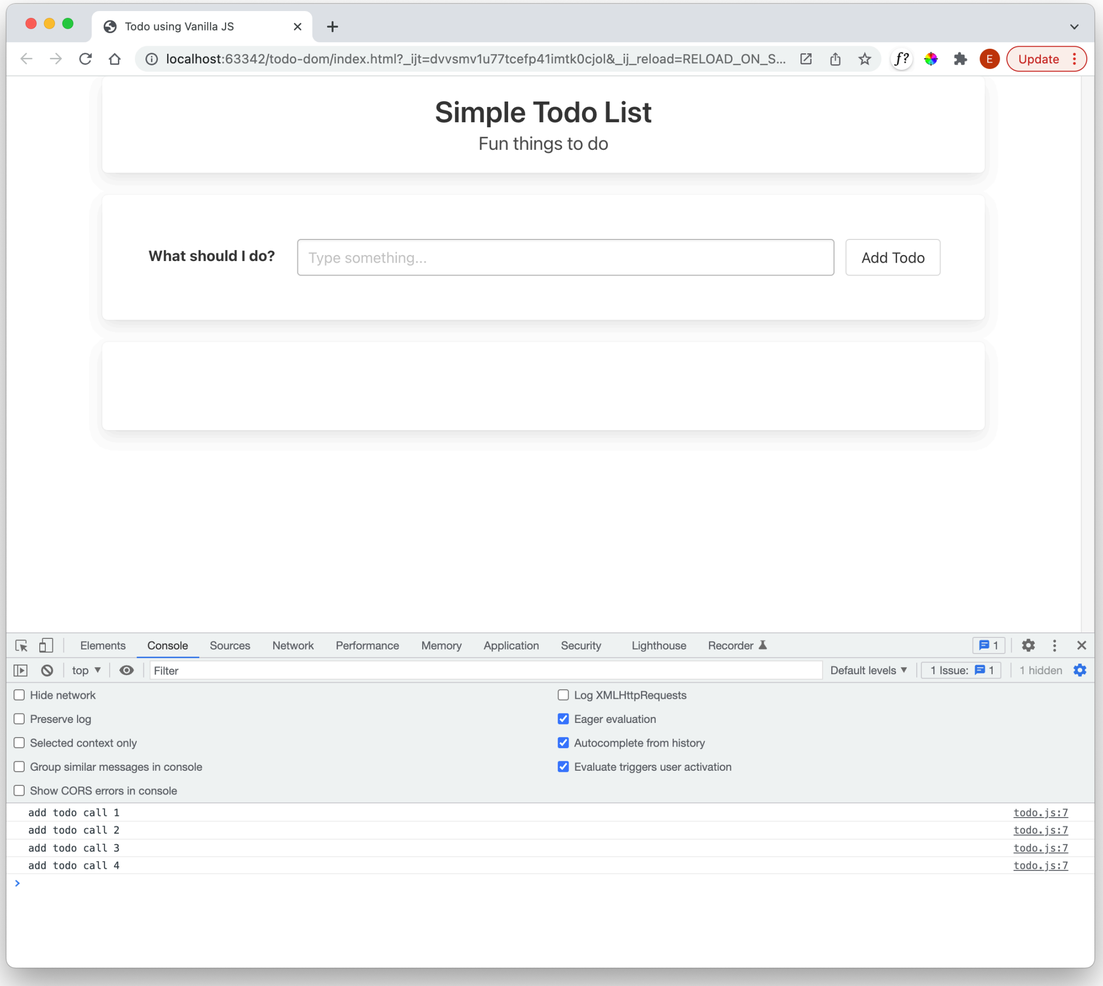
Incorporate Javascript into page

onClick

```
<div class="uk-width-1-2@m uk-card uk-card-default uk-padding">
  <fieldset class="uk-fieldset">
    <legend class="uk-legend">Enter todo item</legend>
    <div class="uk-margin">
      <input id="todo-id" class="uk-input" type="text" placeholder="Todo">
    </div>
  </fieldset>
  <button onClick="addTodo()" id="add-btn"
          class="uk-button uk-button-default">Add Todo</button>
</div>
```

```
let count = 0;

function addTodo() {
  count++;
  console.log(`add todo call ${count}`)
}
```



- onClick attribute establishes link from <button> to javascript function

DOM onevent handlers

https://developer.mozilla.org/en-US/docs/Web/Guide/Events/Event_handlers

Jump to section

Registering onevent handlers
HTML onevent attributes
Specifications
Browser compatibility

The Web platform provides several ways to be notified of DOM events. Two common approaches are `addEventListener()` and the specific `onevent` handlers. This page focuses on how the latter work.

Registering onevent handlers

The `onevent` handlers are properties on certain DOM elements to manage how that element reacts to events. Elements can be interactive (links, buttons, images, forms, and so forth) or non-interactive (such as the base `<body>` element). Events are actions like:

- Being clicked
- Detecting pressed keys
- Getting focus

The `onevent` handler is usually named with the event it reacts to, like `onclick`, `onkeypress`, `onfocus`, etc.

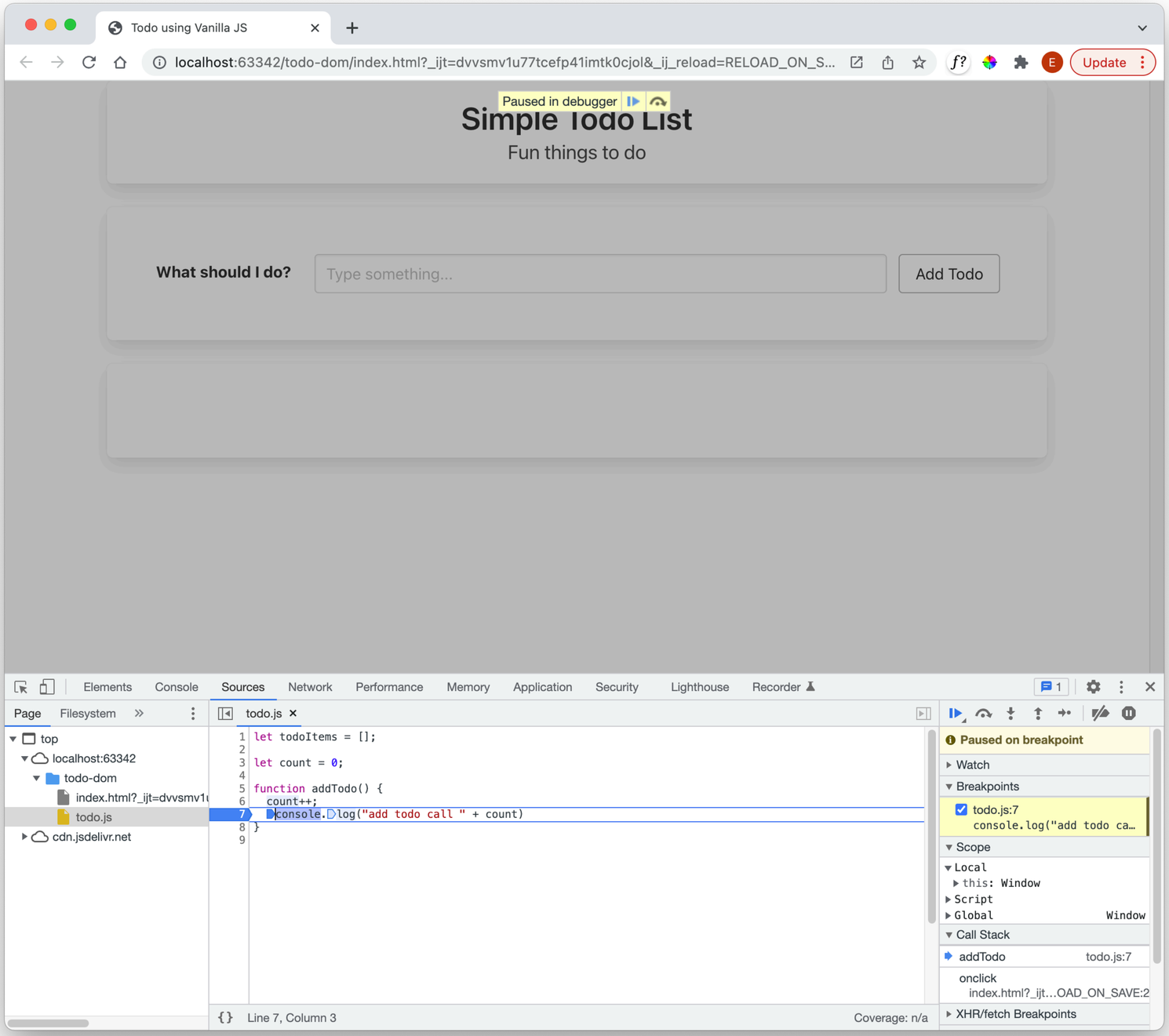
You can specify an `on<…>` event handler for a particular event (such as `click`) for a given object in different ways:

- Adding an HTML attribute named `on<eventtype>`:
  `<button onclick="handleClick()">`,
- Or by setting the corresponding property from JavaScript:
  `document.querySelector("button").onclick = function(event) { … }`.

Browser Debug

- Browsers support debugging of javascript programs

- Breakpoints, inspections, exceptions, etc…

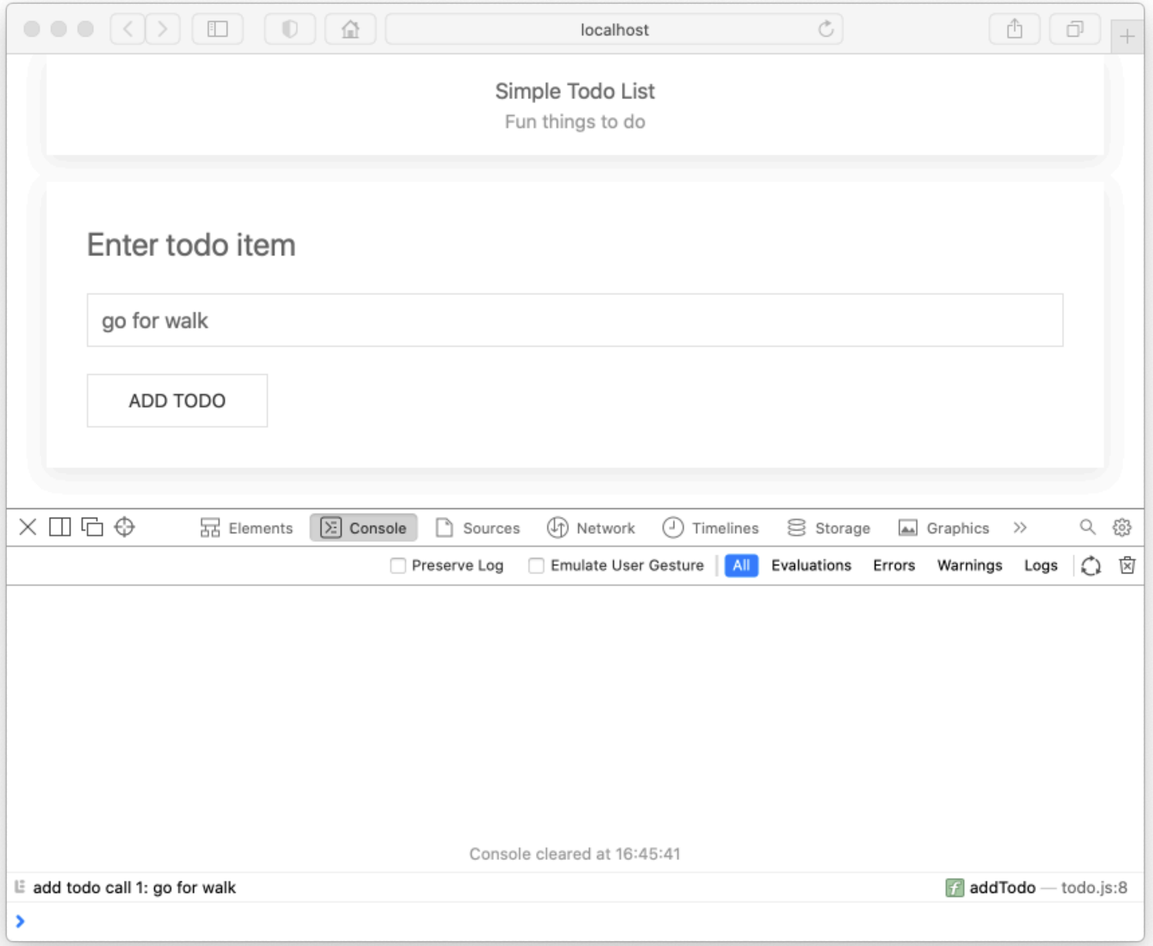


getElementById

```
<input id="todo-id" class="input" type="text" placeholder="Todo">
```

```
let count = 0;

function addTodo() {
  count++;
  const todoText = document.getElementById("todo-id").value;
  console.log(`add todo call ${count}: ${todoText}`)
}
```



- Retrieve the text entered into the field
- Log to console

https://developer.mozilla.org/en-US/docs/Web/API/Document/getElementById

Document.getElementById()

Jump to section

Syntax
Example
Usage notes
Example
Specifications
Browser compatibility
See also

Related Topics

Document Object Model

`Document`

▼ Constructor

`Document()`

The `Document` method **`getElementById()`** returns an `Element` object representing the element whose `id` property matches the specified string. Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.

If you need to get access to an element which doesn't have an ID, you can use `querySelector()` to find the element using any selector.

```
function addTodo() {
  count++;
  const todoText = document.getElementById("todo-id").value;
  console.log(`add todo call ${count}: ${todoText}`)
}
```

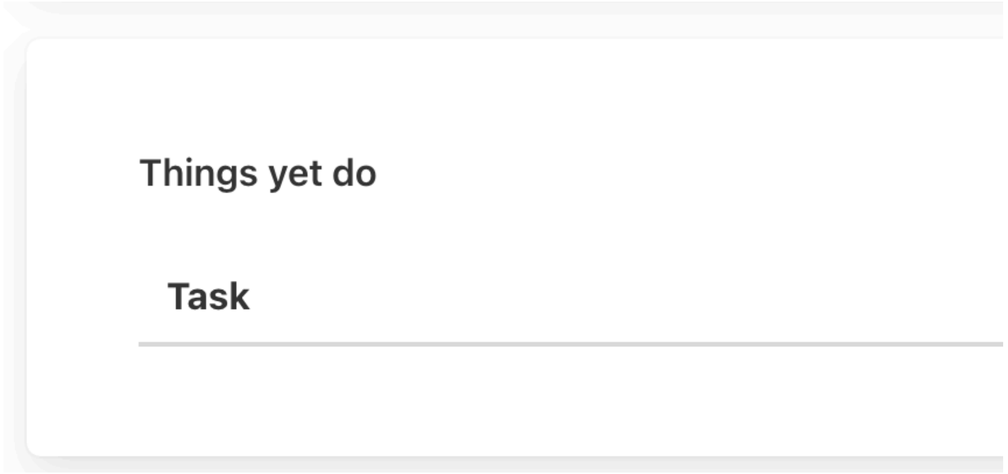
Syntax

```
var element = document.getElementById(id);
```

Parameters

`id`

The ID of the element to locate. The ID is case-sensitive string which is unique within the document; only one element may have any given ID.




```html
<div class="section box">
  <table id="todo-table" class="table is-fullwidth">
    <thead>
      <th>Task</th>
    </thead>
    <tbody>
      <tr></tr>
    </tbody>
  </table>
</div>
```

- Retrieve text from input field
- Create new row & insert text element

```
function addTodo() {
  count++;
  const todoText = document.getElementById("todo-id").value;
  console.log(`add todo call ${count}: ${todoText}`)

  const table = document.getElementById("todo-table");
  const row = table.insertRow(-1);
  const textCell = row.insertCell(0);
  textCell.innerText = todoText;
}
```

Retrieve Table DOM Object

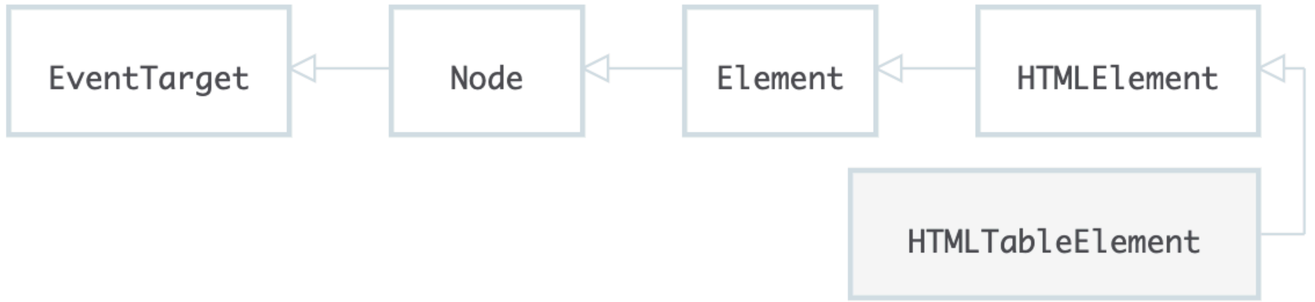
DOM Interaction

HTMLTableElement

Jump to section

Properties
Methods
Specifications
Browser compatibility
See also

The **`HTMLTableElement`** interface provides special properties and methods (beyond the regular `HTMLElement` object interface it also has available to it by inheritance) for manipulating the layout and presentation of tables in an HTML document.



https://developer.mozilla.org/en-US/docs/Web/API/HTMLTableElement

HTMLTable Methods

▼ Methods

createCaption()

createTFoot()

createTHead()

deleteCaption()

deleteRow()

deleteTFoot()

deleteTHead()

insertRow()

`HTMLTableElement.createTFoot()`

Returns an `HTMLElement` representing the first `<tfoot>` that is a child of the element. If none is found, a new one is created and inserted in the tree immediately before the first element that is neither a `<caption>`, a `<colgroup>`, nor a `<thead>`, or as the last child if there is no such element.

`HTMLTableElement.deleteTFoot()`

Removes the first `<tfoot>` that is a child of the element.

`HTMLTableElement.createCaption()`

Returns an `HTMLElement` representing the first `<caption>` that is a child of the element. If none is found, a new one is created and inserted in the tree as the first child of the `<table>` element.

`HTMLTableElement.deleteCaption()`

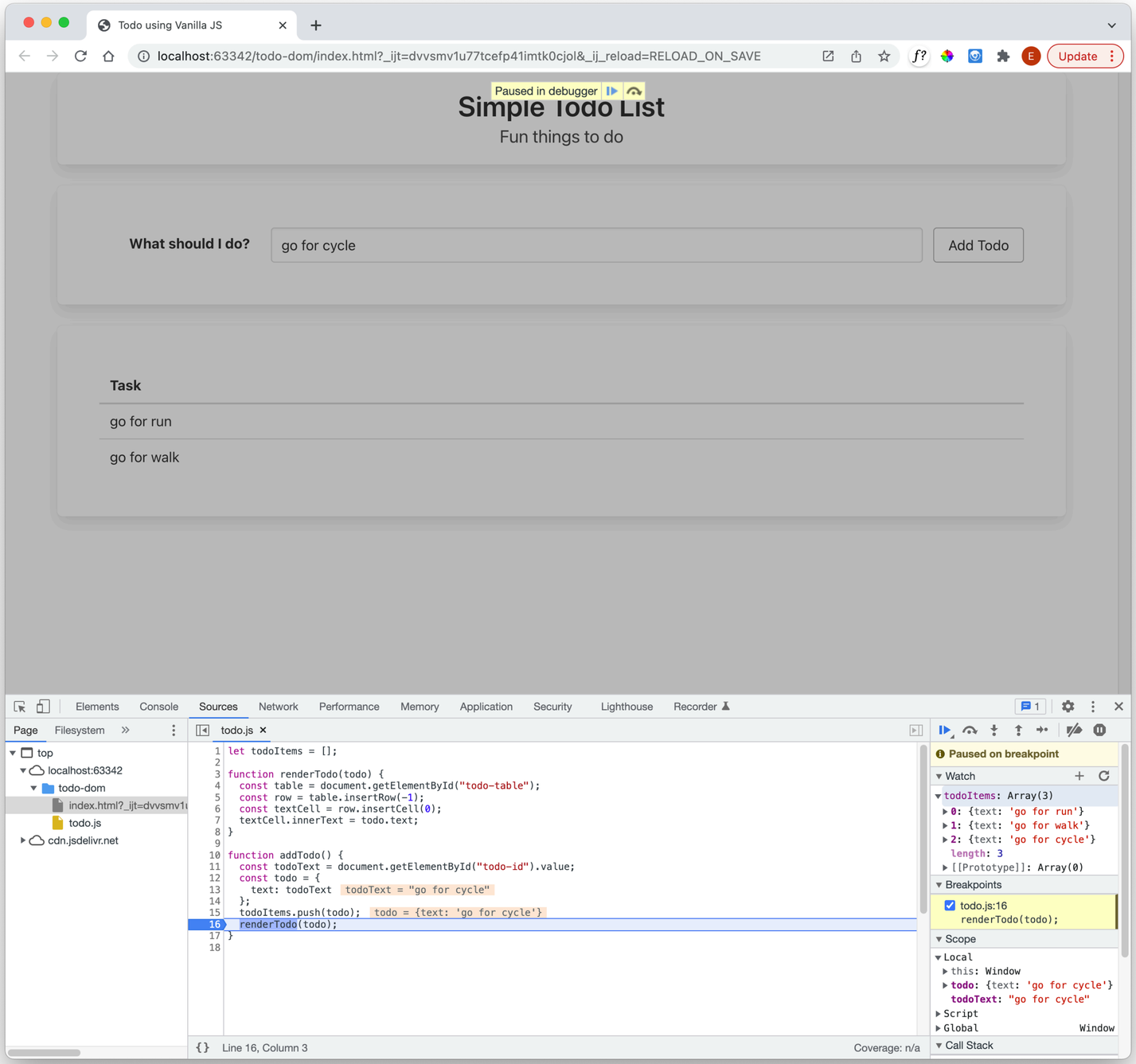
Removes the first `<caption>` that is a child of the element.

`HTMLTableElement.insertRow()`

Returns an `HTMLTableRowElement` representing a new row of the table. It inserts it in the rows collection immediately before the `<tr>` element at the given `index` position. If necessary a `<tbody>` is created. If the `index` is `-1`, the new row is appended to the collection. If the `index` is smaller than `-1` or greater than the number of rows in the collection, a `DOMException` with the value `IndexSizeError` is raised.

`HTMLTableElement.deleteRow()`

Removes the row corresponding to the `index` given in parameter. If the `index` value is `-1` the last row is removed; if it smaller than `-1` or greater than the amount of rows in the collection, a `DOMException` with the value `IndexSizeError` is raised.



Inspect Data Structure

- Developer Tools + Breakpoints
- Explore data structure in real time

Date Object

Date

▶ Jump to section

JavaScript `Date` objects represent a single moment in time in a platform-independent format. `Date` objects contain a `Number` that represents milliseconds since 1 January 1970 UTC.

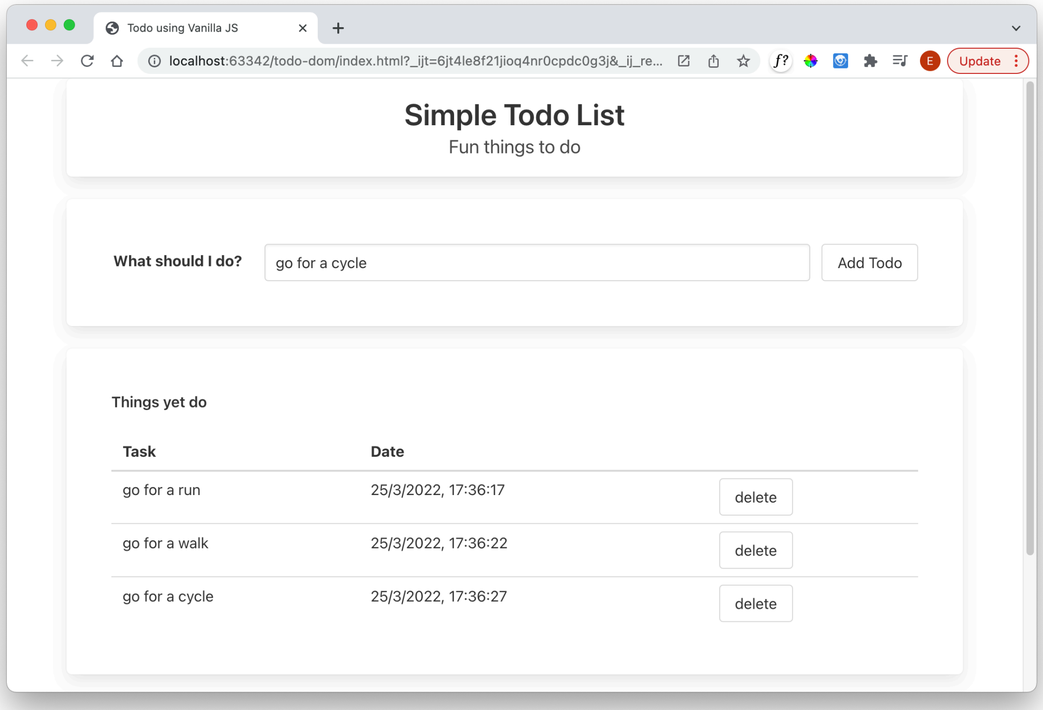
> TC39 is working on Temporal, a new Date/Time API.
> Read more about it on the Igalia blog and fill out the survey. It needs real-world feedback from web developers, but is not yet ready for production use!

Description

The ECMAScript epoch and timestamps

A JavaScript date is fundamentally specified as the number of milliseconds that have elapsed since midnight on January 1, 1970, UTC. This date and time are not the same as the **UNIX epoch** (the number of seconds that have elapsed since midnight on January 1, 1970, UTC), which is the predominant base value for computer-recorded date and time values.

Note: It's important to keep in mind that while the time value at the heart of a Date object is UTC, the basic methods to fetch the date and time or its components all work in the local (i.e. host system) time zone and offset.




```
let todoItems = [];

function renderTodo(todo) {
  const table = document.getElementById("todo-table");
  const row = table.insertRow(-1);
  const textCell = row.insertCell(0);
  textCell.innerText = todo.text;
  const dateCell = row.insertCell(1);
  dateCell.innerText = todo.date;
}

function addTodo() {
  const todoText = document.getElementById("todo-id").value;
  const todo = {
    text: todoText,
    date: new Date().toLocaleString("en-IE")
  };
  todoItems.push(todo);
  renderTodo(todo);
}
```

Extend Todo Object - Date

- Create todo object containing
  - *Text*
  - *Date*

Extend Todo Object - ID

```
function uuidv4() {
  return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function(c) {
    var r = Math.random() * 16 | 0, v = c == "x" ? r : (r & 0x3 | 0x8);
    return v.toString(16);
  });
}
```

UUIDs

Version 4 (random) [edit]

A version 4 UUID is randomly generated. As in other UUIDs, 4 bits are used to indicate version 4, and 2 or 3 bits to indicate the variant (10_2 or 110_2 for variants 1 and 2 respectively). Thus, for variant 1 (that is, most UUIDs) a random version-4 UUID will have 6 predetermined variant and version bits, leaving 122 bits for the randomly generated part, for a total of 2^{122}, or 5.3×10^{36} (5.3 undecillion) possible version-4 variant-1 UUIDs. There are half as many possible version-4 variant-2 UUIDs (legacy GUIDs) because there is one less random bit available, 3 bits being consumed for the variant.

Collisions [edit]

Collision occurs when the same UUID is generated more than once and assigned to different referents. In the case of standard version-1 and version-2 UUIDs using unique MAC addresses from network cards, collisions can occur only when an implementation varies from the standards, either inadvertently or intentionally.

In contrast to version-1 and version-2 UUID's generated using MAC addresses, with version-1 and -2 UUIDs which use randomly generated node ids, hash-based version-3 and version-5 UUIDs, and random version-4 UUIDs, collisions can occur even without implementation problems, albeit with a probability so small that it can normally be ignored. This probability can be computed precisely based on analysis of the birthday problem.[14]

For example, the number of random version-4 UUIDs which need to be generated in order to have a 50% probability of at least one collision is 2.71 quintillion, computed as follows:[15]

$$n \approx \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \times \ln(2) \times 2^{122}} \approx 2.71 \times 10^{18}.$$

This number is equivalent to generating 1 billion UUIDs per second for about 85 years. A file containing this many UUIDs, at 16 bytes per UUID, would be about 45 exabytes.

The smallest number of version-4 UUIDs which must be generated for the probability of finding a collision to be p is approximated by the formula

$$\sqrt{2 \times 2^{122} \times \ln \frac{1}{1-p}}.$$

Thus, the probability to find a duplicate within 103 trillion version-4 UUIDs is one in a billion.

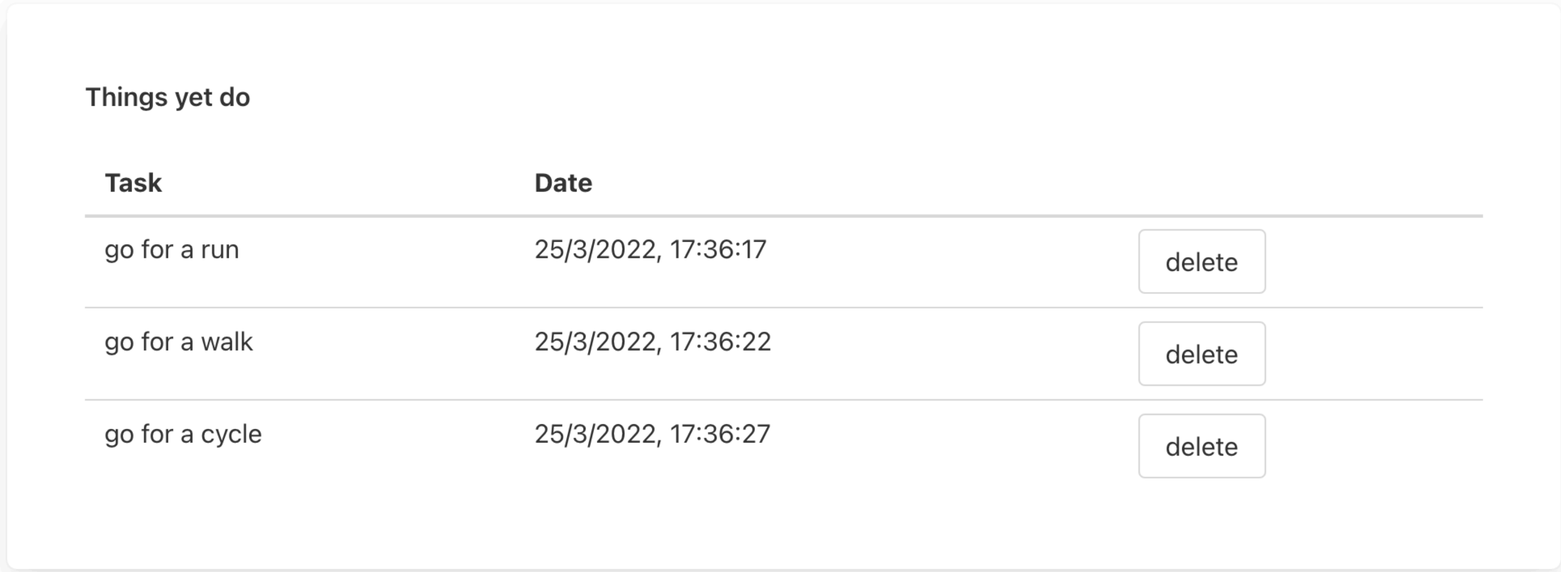
https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

ID for Delete

- Extend Todo Object to include ID

```
function addTodo() {
  const todoText = document.getElementById("todo-id").value;
  const todo = {
    text: todoText,
    date: new Date().toLocaleString("en-IE"),
    id: uuidv4()
  };
  todoItems.push(todo);
  renderTodo(todo);
}
```

- Try to include Delete button, and use ID to delete specific todo



Refactor

Data Structure

Generate ID

Render all Todos to DOM

Delete all Todos from DOM

```
let todoItems = [];

function uuidv4() {
  return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function(c) {
    var r = Math.random() * 16 | 0, v = c == "x" ? r : (r & 0x3 | 0x8);
    return v.toString(16);
  });
}

function renderAllTodos() {
  for (let i = 0; i < todoItems.length; i++) {
    renderTodo(todoItems[i]);
  }
}

function deleteAllTodos() {
  let table = document.getElementById("todo-table");
  for (let i = 0; i < todoItems.length; i++) {
    table.deleteRow(-1);
  }
}
```

innerText

- Append Delete Button to Todo row

```
function renderTodo(todo) {
  const table = document.getElementById("todo-table");
  const row = table.insertRow(-1);
  const textCell = row.insertCell(0);
  textCell.innerText = todo.text;
  const dateCell = row.insertCell(1);
  dateCell.innerText = todo.date;
  const deleteCell = row.insertCell(2);
  deleteCell.innerText = `<a onclick="deleteTodo('${todo.id}')" class="button">delete</a>`;
}
```

Task	Date	
go for a walk	25/3/2022, 10:55:56	delete

HTML Code appears - not button

innerHtml

- Append Delete Button to Todo row

```
function renderTodo(todo) {
  const table = document.getElementById("todo-table");
  const row = table.insertRow(-1);
  const textCell = row.insertCell(0);
  textCell.innerText = todo.text;
  const dateCell = row.insertCell(1);
  dateCell.innerText = todo.date;
  const deleteCell = row.insertCell(2);
  deleteCell.innerHtml = `<a onclick="deleteTodo('${todo.id}')" class="uk-button uk-button-default">delete</a>`;
}
```

Inner HTML

'A `DOMString` containing the HTML serialization of the element's descendants. Setting the value of `innerHTML` removes all of the element's descendants and replaces them with nodes constructed by parsing the HTML given in the string'

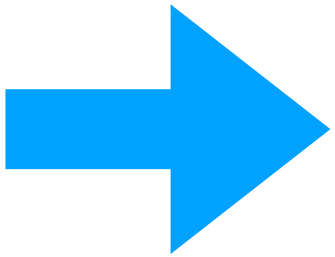
Task	Date	
go for run	25/3/2022, 10:57:23	delete

Render HTML as button object

deleteTodo function

- Append Delete Button to Todo row

```
function renderTodo(todo) {
  const table = document.getElementById("todo-table");
  const row = table.insertRow(-1);
  const textCell = row.insertCell(0);
  textCell.innerText = todo.text;
  const dateCell = row.insertCell(1);
  dateCell.innerText = todo.date;
  const deleteCell = row.insertCell(2);
  deleteCell.innerHtml = `<a onclick="deleteTodo('${todo.id}')" class="button">delete</a>`;
}
```



```
function deleteTodo(id) {
  deleteAllTodos();
  const found = todoItems.findIndex((todo) => todo.id == id);
  todoItems.splice(found, 1);
  renderAllTodos();
}
```

Todo DOM

Manipulating the DOM to implement TODO features