Open-Source Software Practice

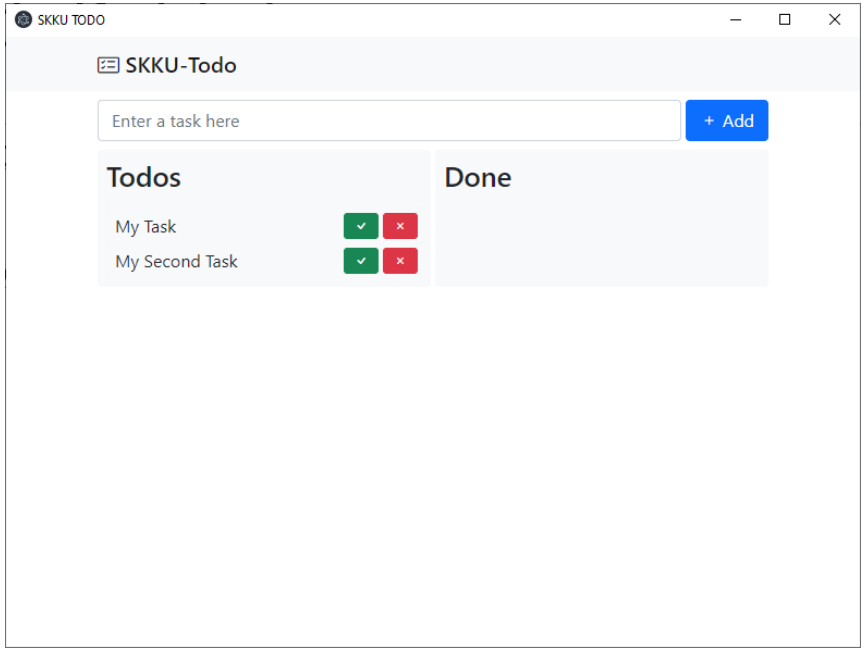
10. Desktop App

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

SKKU-Todo

- Let’s improve SKKU-Todo.

- ~~Add a task~~
- **Remove a task**
- **Save and restore**
- Mark as done



SKKU-Todo-2

- GitHub repository: https://github.com/e-/skku-todo-2
- Web demo: https://e-.github.io/skku-todo-2/

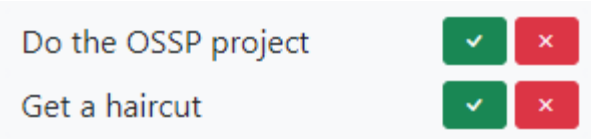
- Starter template: https://github.com/e-/skku-todo-2/blob/main/skeleton.html

- **Make sure that no Korean characters are included in the path to your HTML file.**
  - Electron build can fail.

```html
<!DOCTYPE html>
<html>

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta3/dist/css/bootstrap.min.css" rel="stylesheet"
        integrity="sha384-
eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6" crossor
igin="anonymous">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.4.1/font/bootstrap-icons.css">

    <title>SKKU TODO</title>
    <style>
        .container {
            width: 640px;
        }
    </style>
</head>

<body>
    <nav class="navbar navbar-light bg-light">
        <div class="container">
            <span class="navbar-brand mb-0 h1"><i class="bi bi-card-
checklist"></i> SKKU-Todo</span>
        </div>
    </nav>
    <div class="container">
        <div class="d-flex align-items-center mb-2 mt-2">
            <input type="text" class="form-control" id="task-
input" placeholder="Enter a task here">
            <button type="button" id="add" class="btn btn-primary ms-
1 text-nowrap"><i class="bi bi-plus"></i>
                Add</button>
        </div>
        <div class="d-flex">
            <div class="flex-grow-1 bg-light rounded-2 p-2 me-1 w-50">
                <h3>Todos</h3>
                <div id="todo-list">
                </div>
            </div>
            <div class="flex-grow-1 bg-light rounded-2 p-2 w-50">
                <h3>Done</h3>
                <div id="done-list">
                </div>
            </div>
        </div>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-
beta3/dist/js/bootstrap.bundle.min.js"
        integrity="sha384-
JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
        crossorigin="anonymous"></script>
    <script>
        button.addEventListener("click", () => {
            // 1. Read the text in #task-input.
            let input = document.querySelector("#task-input");
            let text = input.value;

            if (!text.length) return;

            // 2. Create a new Task object.
            // 3. Append the new Task object to tasks
            // 4. Create a new task item and attach it to #todo-list.
            // 5. Clear #task-input.
            input.value = "";
        });
    </script>
</body>
</html>
```

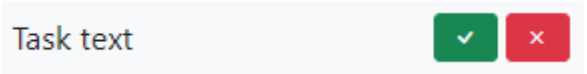
HTML

- We need two buttons for each task item.



- Let’s code HTML first.

```
<div id="todo-list">
    <div class="task bg-light p-1 rounded-2 ps-2 d-flex align-items-center">
        <span class="me-auto">Task text</span>
        <button class="btn btn-sm btn-success me-1">
            <i class="bi bi-check"></i>
        </button>
        <button class="btn btn-sm btn-danger">
            <i class="bi bi-x"></i>
        </button>
    </div>
</div>
```



Adding a Task

- We will manage a task as an object with the following two properties:
- *text*: string, the task name
- *type*: *number*, the task type (1 for todo and 2 for done).

```
const Type = {
    Todo: 1,
    Done: 2,
};
```

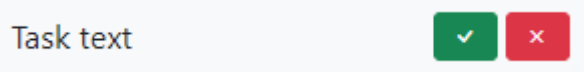
```
// 1. Read the text in #task-input.
let input = document.querySelector("#task-input");
let text = input.value;

if (!text.length) return;

// 2. Create a new Task object.
let task = {
    text: text,
    type: Type.Todo
};
```

Adding a Task

```
let button = document.querySelector("#add");
button.addEventListener("click", () => {
    // 1. Read the text in #task-input.
    let input = document.querySelector("#task-input");
    let text = input.value;

    if (!text.length) return;

    // 2. Create a new Task object.
    let task = {
        text: text,
        type: Type.Todo
    };

    // 3. Create a new task item and attach it to #todo-list.
    addToList(task);

    // 4. Clear #task-input.
    input.value = "";
});
```

Adding a Task

- Function *addToList(task)* accepts a Task object *task* and creates a list item.



- If *task.type* is *Type.Todo* append the item to *#todo-list.*
- Otherwise, append it to *#done-list*.

Adding a Task

- In *addToList*, we need to generate the HTML code on the right programmatically.

- Our weapons:
  - *document.createElement(name)*
  - *element.appendChild(child)*
  - *element.className = “abc”*
  - *element.innerHTML = “”*

```html
<div class="task bg-light p-1 rounded-2 ps-2 d-flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```javascript
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```html
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```javascript
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```html
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

You can use *div.classList.add("task", ...)* as in the previous class, but the className attribute is more convenient for this case.

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Don’t forget to append an element to the parent. Or the element will exist only in memory and thus be invisible.

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

You can use *document.createElement("i")* to create <i>, but using the *innerHTML* property is more convenient for this case.

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

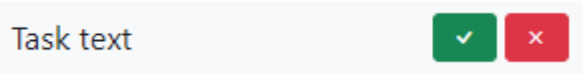
Why did I use single-quotes ‘’ for this case?

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Append the “Done” button only for a task in the todo list.



Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Adding a Task

```
function addToList(task) {
    let div = document.createElement("div");
    div.className = "task bg-light p-1 rounded-2 ps-2 d-flex align-
items-center";

    let span = document.createElement("span");
    span.className = "me-auto";
    span.textContent = task.text;
    div.appendChild(span);

    if (task.type === Type.Todo) {
        let buttonDone = document.createElement("button");
        buttonDone.className = "btn btn-sm btn-success me-1";
        buttonDone.innerHTML = '<i class="bi bi-check"></i>';
        div.appendChild(buttonDone);
    }

    let buttonRemove = document.createElement("button");
    buttonRemove.className = "btn btn-sm btn-danger";
    buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
    div.appendChild(buttonRemove);

    let list = document.querySelector(task.type === Type.Todo ? "#to
do-list" : "#done-list");
    list.appendChild(div);
}
```

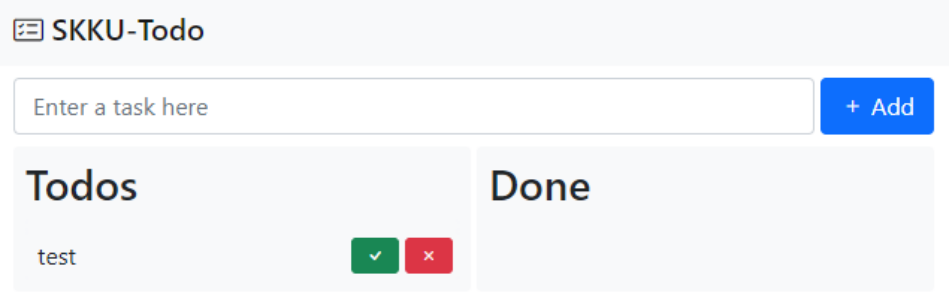
```
<div class="task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center">
    <span class="me-auto">Task text</span>
    <button class="btn btn-sm btn-success me-1">
        <i class="bi bi-check"></i>
    </button>
    <button class="btn btn-sm btn-danger">
        <i class="bi bi-x"></i>
    </button>
</div>
```

Finally, we attach to one of the lists depending on the type of the Task object.
a ? b : c evaluates to *b* if *a* is true, otherwise *c*.

Removing a Task

- We can add tasks, but the remove button does not work.

- If the user clicks on a remove button,
- remove the element from the document.



Removing a Task

- *element.remove()* removes *element* from the HTML document.

```
let buttonRemove = document.createElement("button");
buttonRemove.className = "btn btn-sm btn-danger";
buttonRemove.innerHTML = '<i class="bi bi-x"></i>';
div.appendChild(buttonRemove);

buttonRemove.addEventListener("click", () => {
    div.remove();
});

let list = document.querySelector(task.type === Type.Todo ?
 "#todo-list" : "#done-list");
list.appendChild(div);
```

Removing a Task

- We define an event handler (an arrow function) and register it for click events.
- Wait.. the variable *div* is ***NOT*** defined in the function!

```
let div = document.createElement("div");
div.className = "task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center";

// ...

buttonRemove.addEventListener("click", () => {
    div.remove();
});

// ...
```

Removing a Task

- Inner functions have access to the variables of outer functions.
- Although *div* is not defined in the function, it could be accessed in the scope where the function is defined.
- *div* is included in the environment of the function (or in the function's ***closure***).

```
let div = document.createElement("div");
div.className = "task bg-light p-1 rounded-2 ps-2 d-
flex align-items-center";

// ...

buttonRemove.addEventListener("click", () => {
    div.remove();
});

// ...
```

Saving and Loading the State

- Your tasks are ***volatile***. They are gone each time you refresh the page.

- Let's save the tasks. But, how?
  - The browser cannot write to a local file.

- You can use *localStorage*. Data items added to *localStorage* persist even if you exit the browser.

Saving and Loading the State

- *localStorage.setItem(key, value)* associates a string *value* to *key*.
- *localStorage.getItem(key)* returns the value associated with *key*.

```
> localStorage.setItem("test", "1")
<· undefined
> localStorage.getItem("test")
<· "1"
```

Refresh!

```
> localStorage.getItem("test")
<· "1"
```

1. Defining the State

- When implementing saving/loading, you should consider the following questions:

- What is the “state” of a program?
- How do user interactions change the state?
- How to save the state
- How to load the state and initialize interfaces from it.

1. Defining the State

- The state of SKKU-Todo can be defined as the array of tasks that the user added.

- We maintain this state using a global array, *tasks*.

```
let tasks = [];
```

2. Changing the State

- (Addition) When the user adds a new task, we push the corresponding Task object to *tasks*.
- (Deletion) When the user removes a task, we filter out the corresponding Task object from *tasks*.

```
// 2. Create a new Task object.
let task = {
    text: text,
    type: Type.Todo
};


// 3. Append the new Task object to tasks
tasks.push(task);
```

```
// in function addToList(task)

buttonRemove.addEventListener("click", () => {
    div.remove();
    tasks = tasks.filter(t => t !== task);
});
```

3. Saving the State

- Save the *tasks* array in the local storage.
- Since the local storage can only store string values, we need to convert an array of objects to a string.
- *JSON.stringify(obj)* does this.

```
function saveTasks() {
    localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

```
> tasks
<· ▼(2) [{…}, {…}]
    ▶0: {text: "test", type: 1}
    ▶1: {text: "test2", type: 1}
     length: 2
    ▶__proto__: Array(0)
> JSON.stringify(tasks)
<· "[{"text":"test","type":1},{"text":"test2","type":1}]"
```

3. Saving the State

- Call *saveTasks()* each time the user changes the state.

```
function saveTasks() {
    localStorage.setItem("tasks", JSON.stringify(tasks));
}
```

```
// 2. Create a new Task object.
let task = {
    text: text,
    type: Type.Todo
};


// 3. Append the new Task object to tasks
tasks.push(task);
saveTasks();
```

```
// in function addToList(task)

buttonRemove.addEventListener("click", () => {
    div.remove();
    tasks = tasks.filter(t => t !== task);
    saveTasks();
});
```

4. Loading the State

- When the app is loaded,
- Read the state string from the local storage
- Set *tasks*
- Populate list items

- When the app is loaded -> event handling
  - This event is fired by the browser not by the user.

```
window.addEventListener("load", () => {
    loadTasks();
});
```

4. Loading the State

- *JSON.parse(string)* converts *string* to an object.

```
> lastTasks = localStorage.getItem("tasks")
<· "[{"text":"test","type":1},{"text":"test2","type":1}]"
> tasks = JSON.parse(lastTasks)
<· ▼(2) [{…}, {…}]
    ▶0: {text: "test", type: 1}
    ▶1: {text: "test2", type: 1}
     length: 2
    ▶__proto__: Array(0)
```

```
function loadTasks() {
    let lastTasks = localStorage.getItem("tasks");
    if (!lastTasks) return;

    tasks = JSON.parse(lastTasks);
    tasks.forEach(t => {
        addToList(t);
    });
}
```

- Read the state string from the local storage
- Set *tasks*
- Populate list items

4. Loading the State

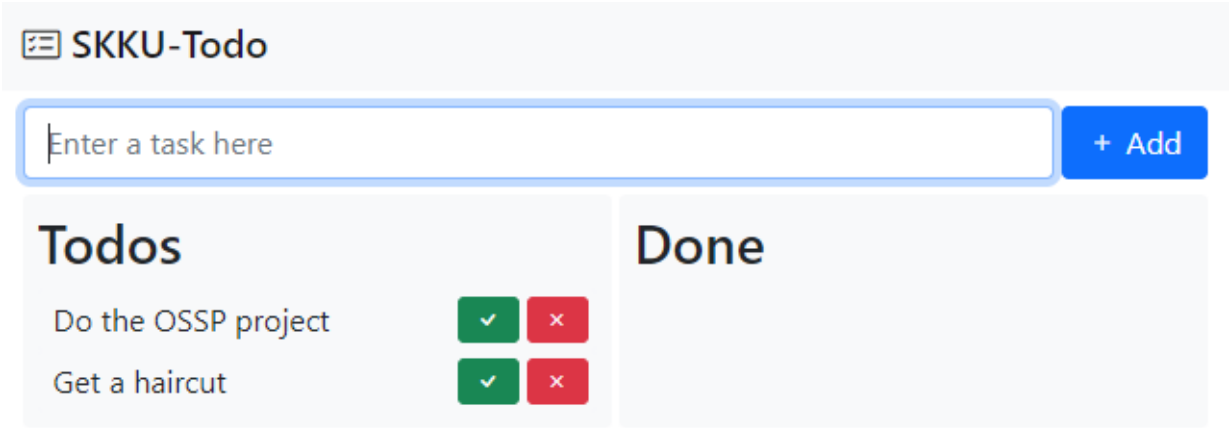
- Example 1 passes an arrow function that takes one argument *t* and calls *addToList(t)*.
- But *addToList* itself is the very function that takes *t* and adds *t* to the list.

```
tasks.forEach(t => {
    addToList(t);
})
```

```
tasks.forEach(addToList);
```

The Final Version

- Add a task
- Remove a task
- Save and restore
- Mark as done (you will do this as an assignment)



Let’s Publish

- Let’s publish SKKU-Todo as a desktop app.

- **Electron** is a framework that builds cross-platform desktop apps using Web technologies.
  - Slogan: If you can build a website, you can build a desktop app.

- https://www.electronjs.org/
- https://www.electronjs.org/docs/tutorial/quick-start

Let’s Publish

- You need at least these four files:

```
my-electron-app/
├── package.json
├── main.js
├── preload.js
└── index.html
```

- **package.json**: Create one using `npm init`
- **main.js**: Let’s use the default file from the official documentation.
  - https://www.electronjs.org/docs/tutorial/quick-start
  - https://github.com/e-/skku-todo-2/blob/main/main.js
- **preload.js**: Create an empty JS file
- **index.html**: The HTML file we coded today

Let’s Publish

Main.js

```
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow() {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  win.loadFile('index.html')
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow()
    }
  })
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})
```

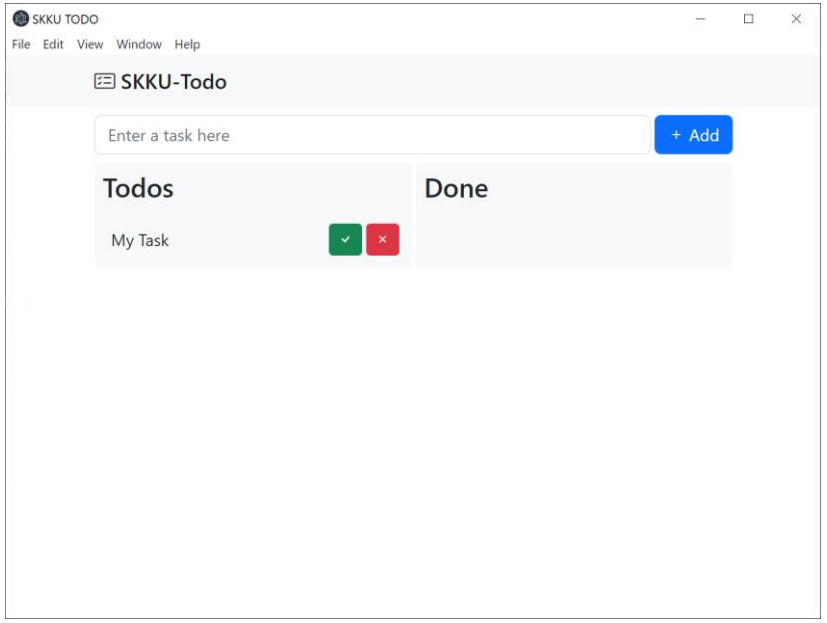
Let’s Publish

- `npm install --save-dev @electron-forge/cli`
- `npx electron-forge import`
- package.json will be converted.

```
D:\skku-todo-2>npx electron-forge import
✔ Checking your system
✔ Initializing Git Repository
✔ Writing modified package.json file
✔ Installing dependencies
✔ Writing modified package.json file
✔ Fixing .gitignore


We have ATTEMPTED to convert your app to be in a format that electron-forge understands.

Thanks for using Electron Forge!!!
```

Let’s Publish

- `npm start`
- SKKU-Todo became a desktop app!
- Check if all features work as expected.



Remove the Menu Bar

Main.js

```
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow() {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  win.setMenuBarVisibility(false);
  win.loadFile('index.html')
}

app.whenReady().then(() => {
  createWindow()

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow()
    }
  })
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})
```

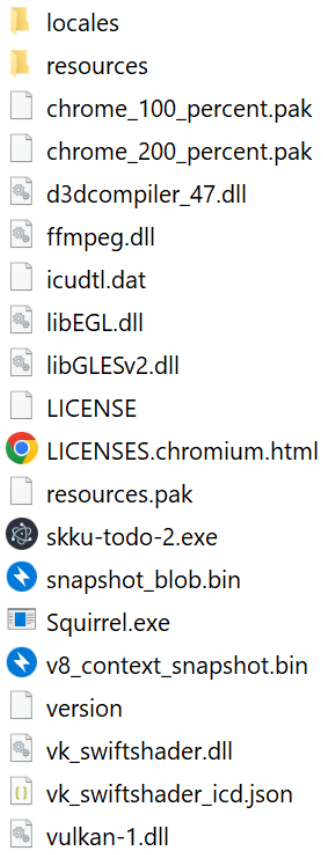
Let’s Publish

- Let’s make a package for distribution.
- `npm run make`
  - It will take a while.
  - Make sure Hangul is not included in the path to the project.

```
D:\skku-todo-2>npm run make

> skku-todo-2@1.0.0 make
> electron-forge make

✔ Checking your system
✔ Resolving Forge Config
We need to package your application before we can make it
✔ Preparing to Package Application for arch: x64
✔ Preparing native dependencies
✔ Packaging Application
Making for the following targets: squirrel
✔ Making for target: squirrel - On platform: win32 - For arch: x64
```

Let’s Publish

- Find the package in the “out” directory.
- Zip and ship all files to distribute your app!



What’s Next?

- We learned HTML, CSS, and JS.
- We learned Node.js, Bootstrap, and unit testing.
- We built and published a command-line interface (*skku-menu*), a Web app (*skku-todo*), and a Desktop app (*skku-todo-2*).

What’s Next?

- To level up your dev skills further, learn

- **JS components of Bootstrap**, Twitter’s toolkit to enrich your user interface
- **React**, Facebook’s JS library to build reactive apps
- **TypeScript**, Microsoft’s Typed JavaScript
- **Flutter**, Google’s UI toolkit for building cross platform apps.

- Mastering a few of these will make you survive.