Problem 01. index.html 파일의 Line 8~10에서는 별도로 작성된 스크립트 파일들을 로드하고 있습니다. 하지만 여러분은 스크립트 파일이 제대로 작동하지 않고, 다음과 같은 오류가 발생하는 것을 발견했습니다. (7점)

TypeError: Cannot read properties of null (reading 'addEventListener')

(1) 문제의 원인은 무엇인가요? (3점)

**script.js 파일이 html 파일의 요소들이 로드되기 이전에 실행되기 때문에 addEventListener 메서드가 호출 될 때 null이 반환되어 호출할 수 없게 된다.**

(2) 해결 가능한 방법을 2가지 이상 작성하세요. (4점)

**1) script.js 파일이 html 파일의 요소들이 로드된 이후에 실행되도록 문장의 위치를 body 태그 밑으로 내린다.**

**2) 같은 방식으로 로드될 수 있도록 script 태그에 defer 속성을 추가한다.**

Problem 02. index.html 파일의 Line 14~25에서는 게시글 검색 시 검색 방식을 선택하기 위한 HTML element들의 구조를 표현하고 있습니다. 하지만 여러분은 검색 방식 선택 기능이 의도했던 것과 다르게 동작한다는 사실을 발견했습니다. (7점)

(1) 검색 방식 선택이 어떻게 원래 의도와 다르게 동작하나요? (3점)

**querySelector를 사용하여 체크된 것을 찾는 부분에서 input:checked를 정확히 선택하지 못하고 있다. 여러 개가 동시에 선택될 수 있는 상태이다.**

(2) 의도했던 것처럼 기능이 동작하도록 하기 위해서 수정해야 하는 부분을 설명해 주세요. (4점)

**input 태그가 type=”radio” 속성값을 가지도록 하여 하나의 옵션만 선택할 수 있게 한다.**

Problem 03. script.js 파일의 Line 3에서는 DOM에서 제공하는 메서드를 이용해 HTML 요소를 조회하고 있습니다. 하지만 여러분은 게시글을 검색할 때 이전 검색 결과가 삭제되지 않고 계속해서 누적된다는 사실을 발견했습니다. (8점)

(1) 문제의 원인은 무엇인가요? (3점)

**this.element가 하나의 요소를 가리키기 때문에 clear 메서드가 게시글을 제대로 제거하지 않는다.**

(2) 해결 가능한 방법을 작성하세요. (2점)

**document.querySelector(“#post-container”)를 이용하여 요소를 선택하면 모든 자식 요소들을 한 번에 제거할 수 있다.**

(3) 현재 방식과 (2)에서 변경한 방식 간의 또 다른 차이점에 대해 작성하세요. (3점)

**변경한 방식에서는 한 번의 DOM 조작을 통해 모든 요소를 제거하기 때문에 대규모로 처리할 때 효율적이다.**

Problem 04. interface.js 파일의 Line 11~13에서는 게시글의 ID로 게시글을 검색하는 기능을 제공하기 위해 사용되는 메서드를 정의하고 있습니다. 여러분은 다른 검색 방식은 의도한 대로 정확히 동작하지만, ID 값으로 검색할 때는 값을 정확히 입력했음에도 검색 결과가 나타나지 않는다는 문제를 발견했습니다. (7점)

(1) 문제의 원인은 무엇인가요? (3점)

**입력 값으로는 문자열 타입의 값을 받지만 저장되어 있는 값은 숫자 타입이기 때문에 ===을 사용하면 같다고 처리해야 할 것이 다르게 처리될 것이다.**

(2) 해결 가능한 방법을 2가지 이상 작성하세요. (4점)

**1) postId = parseInt(postId); 라는 문장을 추가해 숫자 타입으로서 비교할 수 있게 한다.**

**2) postId = Number(postId); 라는 더욱 엄격히 변환하는 문장을 추가할 수 있다.**

Problem 05. 여러분은 Problem 04에서 발생한 문제를 해결하면서, 게시글의 ID로 게시글을 검색하는 기능에서 사용자가 잘못된 입력을 했을 때 오류 메시지를 발생시켜야 하겠다는 생각을 하게 되었습니다. interface.js 파일의 Line 11~13을 수정해서 이 기능을 구현해 봅시다. (단, Problem 04에서 발생한 문제는 해결되었다고 가정합니다.) (6점)

(1) 먼저, 사용자의 잘못된 입력을 확인하는 조건을 작성하고, (3점)

**if (isNAN(postId))**

(2) 조건에 해당할 때, BOM에서 제공하는 메서드를 이용해 브라우저에 알림을 보내세요. (3점)

**{ alert("게시글의 Id는 숫자 타입으로 입력되어야 합니다."); }**

Problem 06. interface.js 파일의 Line 14~16에서는 단일 사용자를 사용자 ID로 조회하는 메서드를 정의하고

있습니다. 여러분은 아직 이 메서드를 작성하지 않았습니다. 이제, 다음 사실을 이용해 메서드를 구현해 봅시다. (6점)

JavaScript에서 제공하는 Standard built-in object 중 하나인 Array Object는 다양한 instance 메서드를 제공합니다. 주요 메서드 목록: length, concat, filter, find, findIndex, forEach, includes, join, map, pop, push, reverse, slice, sort

(1) Array Object의 instance 메서드를 사용해서 메서드를 구현해 보세요. (3점)

```
getUserById: function (userId) {
  return data.users.find(user => user.id === userId);
}
```

(2) Array Object의 instance 메서드를 사용하지 않고 메서드를 구현해 보세요. (3점)

```
getUserById: function (userId) {
  for (let i = 0; i < data.users.length; i++) {
    if (data.users[i].id === userId) {
      return data.users[i];
    }
  }
}
```

Problem 07. 제작된 웹 어플리케이션은 다섯 개 파일 (index.html, style.css, script.js, interface.js, data.js) 로 구성되어 있습니다. 여러분은 이 웹 어플리케이션이 동작하는 원리에 대해 완벽히 알고자 합니다. (8점)

(1) 각 파일이 웹 어플리케이션에서 어떤 역할을 하는지 설명하고, (5점)

**index.html: 사용자에게 보이는 요소들을 구조화하고 css, js 파일들을 로드하여 동작할 수 있도록 한다.**

**style.css: 레이아웃, 디자인, 색상 등을 설정하여 보이는 화면을 꾸민다.**

**script.js: 동적인 동작을 담당한다. 사용자와의 상호작용에 반응하여 DOM 요소 조작, 데이터 처리 등을 통해 필요한 기능을 구현한다.**

**interface.js: script.js가 데이터 파일을 이용할 수 있도록 해 준다.**

**data.js: 사용자, 게시글 등의 정보를 저장한다.**

(2) 사용자가 이 웹 어플리케이션에 접속해 정보를 확인하기까지의 과정을 간단히 소개해 보세요. (3점)

**웹 브라우저에서 주소를 통해 웹 페이지에 접속한다.**

**웹 브라우저는 서버에서 index.html 파일을 받아온다.**

**index.html이 로드되고 style.css 파일에 의해 디자인이 구체화된다.**

**script.js가 로드되고 사용자와의 상호작용에 필요한 동작이 실행된다.**

**interface.js 파일에 의해 사용자의 동작에 따라 data.js에 저장되어있던 필요한 데이터를 이용하게 된다.**

**사용자가 검색 방식을 선택하고 검색어를 입력하여 게시물을 찾는다.**

**script.js가 interface.js를 호출하여 data.js의 데이터를 검색하고 그 결과를 html요소로 렌더링한다.**

Problem 08. Figure 1은 브라우저의 너비가 720px일 때 index.html 페이지를 조회한 사진입니다. 여러분은 브라우저의 너비가 1080px일 때의 상태도 능숙하게 추론할 수 있습니다. (10점)

(1) 웹 페이지를 반응형으로 디자인하기 위한 방법을 간단히 소개해 보세요. (2점)

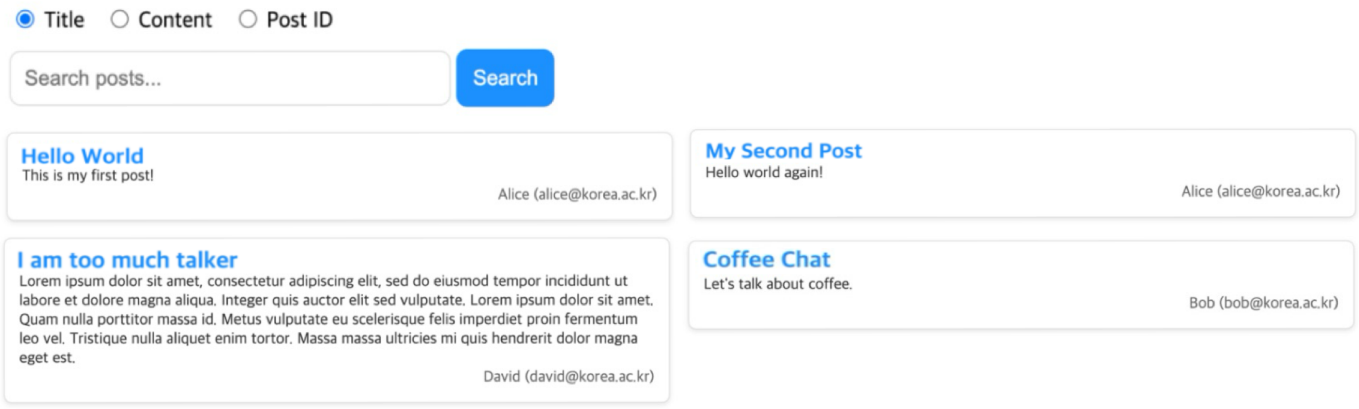
**미디어 쿼리를 이용할 수 있다.**

**@media (max-width: 720px) {**

**}**

**이와 같은 문장을 여러 개 사용하여 특정 너비 범위에 따른 동작을 구분하여 실행할 수 있다.**

(2) 브라우저의 너비가 1080px일 때 index.html 페이지를 조회한 결과를 그려 보세요. (8점)




**아래 반복**

Problem 09. 사용자의 편의를 위해 Text Input에서 Enter 키를 누를 때 자동으로 검색이 이루어지도록 하는 기능을 구현해 봅시다. 아래는 사용자가 브라우저에서 Enter 키를 누를 때 발생하는 이벤트입니다. (8점)

KeyboardEvent {isTrusted: true, key: 'Enter', code: 'Enter', keyCode: 13, location: 0, ...}

(1) 적절한 HTML 요소, 적절한 이벤트를 선택해서 Event Listener 추가하고, (4점)

**document.querySelector("#Search-input").addEventListener("keypress",**

(2) 이벤트 객체를 받아 누른 키가 Enter인지 확인해서 검색을 실행하면 됩니다. (4점)

```
function(event) {
        if ( event.key === "Enter" ) {
                const value = document.querySelector( "#search-input" ).value;
                const option = document.querySelector( ".search-option input:checked ").value;
                board.search(value,option);
        }
    }
);
```

Problem 10. 이번에는 각 게시글의 사용자 정보 부분에 이메일을 보내는 하이퍼링크를 추가하려고 합니다. 이메일을 전송하기 위한 리소스 주소는 mailto:<address>와 같이 표현합니다. 하지만 여러분이 script.js의 Line 45에 다음과 같은 코드를 추가해도 하이퍼링크가 동작하지 않습니다. 하이퍼링크가 제대로 동작하도록 해 봅시다. (8점)

userSpan.setAttribute("href", "mailto:" + user.email);

(1) 위 코드가 동작하지 않는 원인이 무엇인가요? (2점)

**userSpan이 span 태그이며 span 태그를 하이퍼링크를 지원하지 않기 때문에 href 속성으로 링크를 작동하게 할 수 없다.**

(2) 위 코드 대신 script.js의 Line 45에 어떤 의미의 코드를 추가해야 하이퍼링크가 제대로 동작할까요? (2점)

**span 태그 대신 a 태그를 이용할 수 있게 해야 한다.**

(3) 실제로 코드를 작성해 봅시다. 단, innerHTML 메서드는 사용하지 않아야 합니다. (4점)

```
function addEmailLink(user, containerId) {
        const userLink = document.createElement('a');
        userLink.setAttribute('href', 'mailto:' + user.email);
        userLink.textContent = user.name;
        const container = document.getElementById(containerId);
        container.appendChild(userLink);
}
```

Problem 11. Problem 10에서 하이퍼링크를 추가하니, 브라우저 엔진에 의해 강제로 삽입된 다음 CSS가 적용된 것을 확인할 수 있었습니다. 사용자가 보는 화면이 원래와 동일해질 수 있도록 CSS 코드를 추가로 작성해 봅시다. (5점)

{ color: -webkit-link; cursor: pointer; text-decoration: underline; }

(1) 적절한 선택자를 사용해서, (2점)

**#user-info-container a 선택자를 이용하여 a 태그에 스타일을 적용하도록 함.**

(2) 적절한 CSS 속성들을 적용해 보세요. (3점)

```
#user-info-container a {
        color: blue;
        cursor: pointer;
        text-decoration: underline;
}
```

Problem 12. 버튼 위에 커서를 올렸을 때, 커서 모양이나 버튼의 색이 변경되는 디자인은 사용자 경험을 개선합니다. 커서 모양이 변경되는 디자인은 이미 적용되어 있습니다. 이제 버튼의 색이 변경되는 디자인을 적용해 봅시다. (5점)

(1) 적절한 가상 클래스 선택자를 사용해서, (2점)

**hover 가상 클래스 선택자를 사용.**

(2) 여러분이 선호하는 CSS 속성을 5가지 이상 적용해 보세요. (3점)

```
button: hover {
        background-color: red;
        color: white;
        border-color: red;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        transform: scale(1.05);
}
```

Problem 13. 게시글 본문과 같은 긴 텍스트의 경우에는, 내용 전체를 보여주지 않고 넘치는 부분을 줄임표로 대체하기 위해 다음과 같은 속성을 적용하기도 합니다. 브라우저의 크기가 800px 미만일 때만 이 방식을 적용해 봅시다. (4점)

{ overflow: hidden; white-space: nowrap; text-overflow: ellipsis; }

```
@media (max-width: 800px) {
        .post-content {
                overflow: hidden;
                white-space: nowrap;
                text-overflow: ellipsis;
        }
}
```

Problem 14. script.js 파일의 Line 4~7에서는 게시판을 초기화하기 위한 메서드가 정의되어 있습니다. 이 메서드를 익명 함수로 다음과 같이 작성하였을 때, 발생할 수 있는 문제점을 설명해 보세요. (4점)

init: () => { this.posts = interface.getPosts(); this.render(); },

**this 가 가리키는 곳이 함수 자신이 아닌 상위의 this를 유지하기 때문에 의도치 않은 동작이 실행되는 문제가 발생할 수 있다.**

Problem 15. script.js 파일의 Line 8~25에서는 검색 결과에 따라 게시판을 갱신하기 위한 메서드가 정의되어 있습니다. 여러분은 ES5 표준에 맞게 작성된 이 메서드에서 var 키워드를 제거해 리팩터링 하려고 합니다. (7점)

(1) 단순히 var 키워드를 let 키워드로 변경하면 오류가 발생하게 됩니다. 이유를 설명해 보세요. (3점)

**var 키워드는 변수 선언이 최상단에서 이루어지도록 하는 기능을 가지지만 let 키워드는 그렇지 않다. 따라서 변수를 선언하기 전에 접근하려는 경우 오류가 발생할 수 있다.**

(2) 이 메서드를 var 키워드를 사용하지 않고 다시 작성해 보세요. (4점)

```
search: function (value, option) {
        this.clear();
        let posts;
        switch (option) {
                case "title":
                        posts = interface.getPostsByTitle(value);
                        break;
                case "content":
                        posts = interface.getPostsByContent(value);
                        break;
                case "id":
                        posts = interface.getPostsById(value);
                        break;
        }
        this.posts = posts;
        this.render();
},
clear: function () {
        while (this.elements.length > 0) {
                this.elements[0].remove();
        }
},
```