javascript 안티 패턴

작업 경험에 따른 지양하는 코드에 대한 이야기.

👎 배열을 순환할 때 무조건 for 문 사용

```
const dropdownBtns = document.querySelectorAll('.dropdown_btn');

for(let d = 0; d < dropdownBtns.length; d++) {
  // 중략...
  dropdownBtns[d].addEventListener('click', () => {
  // 중략...
  });
};
```

초반 작업 시 배열을 순환할 때 무조건 for 문으로 사용하였다. (후에 forEach()로 변경) 따라서 for 문, forEach()에 대한 구분이 필요해 여러 포스팅을 읽어보니 forEach()도 무조건적인 것은 아니였고 내용도 난이도가 있었다.

👉 결론적으로는 코드에 따라 구분해서 사용해야 하며 **배열의 순환 시 for of의 사용**을 추천(Object X)하여 앞으로 자주 써볼 예정이다. (링크 참고)

💡 **for vs forEach()**
https://m.blog.naver.com/rlaalsdn456456/221818157118

for vs forEach() vs for in vs for of
https://miiingo.tistory.com/347
https://bbaktaeho-95.tistory.com/32
https://thecodebarbarian.com/for-vs-for-each-vs-for-in-vs-for-of-in-javascript

👎 스타일 변경을 inline으로 진행

```
// 중략...
inputArea[i].addEventListener('focusin', (event) => {
  const target = event.currentTarget;
  target.style.borderBottomColor = '#42588a';
});
```

```
<!-- 브라우저 -->
<div class="input_area" style="border-bottom-color: rgb(190, 201, 212);">
  <input type="text" id="inputStyle0101" placeholder="Placeholder">
</div>
```

공통 input text의 이벤트에 따른 스타일 제어를 inline style로 넣었었다. 변경해야하는 스타일도 한 두개이고 평소 class로 제어했기 때문에 안 써본 걸로 써보자 하는 단순한 생각이었다. 😑

그 결과

- 가독성이 떨어짐
- 수정이 번거로움
- html를 브라우저에서 복사하는 경우가 대부분이라 script로 제어되는 스타일까지 복사하여 자잘한 오류를 발생시켰음
- 개발 막바지 단계에서는 스타일 수정이라도 js를 수정한다는 것에 대한 부담이 있음

👉 모든 경우는 아니지만 inline으로 제어하는 것은 불필요한 경우가 많음으로 **스타일 요소에 변화는 css로 접근할 수 있는 class로 조정**하는 것을 지향한다.

👎 함수문법 복합사용과 let을 var처럼 사용

```
// 함수표현식
const checkedAllAction = () => {
  // 중략 ..
};

// 함수선언문
function agreeAllAciton () {
  // 중략 ..
};
```

```
let checkedAll = document.querySelector('input[data-role="checked-all"]');
checkedAll.addEventListener('click', checkedAllAction);

const checkedAllAction = () => {
  if(checkedAll.checked) {
    checkedAllList.forEach((checkbox) => {
      checkbox.checked = true;
    });
  }
  // 중략...
};
```

작업 스타일에 따라 함수문과 변수 쓰임이 복합적이여서 통일성이 떨어졌으며 함수는 문법에 따라 호이스팅(끌어올림)과 관련이 있기 때문에 **함수 문법 통일과 변수를 의미에 맞게 사용하는 것이 중요**하다고 생각한다.

- 함수선언문: 선언 위치에 관련 없이 내부적으로 호이스팅이 됨
- 함수표현식: 실행 후 함수를 선언하며 호출되지 않을 수 있으며 호이스팅에 대한 비용이 발생함

👉 호이스팅을 고려한 변수/ 함수표현식 선언

```
const checkedAll = document.querySelector('input[data-role="checked-all"]');
const checkedAllAction = () => {
  // 중략 ..
};

checkedAll.addEventListener('click', checkedAllAction);
```

- 변수, 함수표현식은 실행 전 선언하여 호이스팅 발생과 코드 미실행 등의 오류 최소화한다.
- 재할당 여부를 판단해서 const, let를 구분하여 사용한다.

👎 비객체화 코드

```
const btnValueReset = inputArea[i].querySelector('.delete_icon');

if(btnValueReset !== null) {
```

```
    const activeBtnReset = () => {
      // 중략...
    };

    inputAreaTextField.addEventListener('input', activeBtnReset);
};
```

script을 요소에 대한 캐시 여부 확인 ⇒ 함수 코드 ⇒ EventListener에 함수 호출의 형태로 작업하였는데 이 점이 코드 퀄리티를 낮추는데 큰 원인인 것 같다.
객체타입으로 리팩토링을 했어야 했는데 구현하는데 급급해서 정리를 하지 못하다 보니 같이 작업하신 분들도 상위 구조의 형태로 작업을 할 수 밖에 없었다. 이 점을 보안할 예정이다.

👉 객체화 코드

```
const commonInput = {
  textFoucsAction () => {
    // 코드..
  },
  textResetBtnAction () => {
    // 코드..
  },
  // 중략..
};

const btnValueReset = document.querySelector('.delete_icon');
if(btnValueReset !== null) {
    inputAreaTextField.addEventListener('input', commonInput.textResetBtnAction());
}
```