퀀터스테크놀로지스-과제유형 1

Description
1. 숫자 `1` 혹은 `0`을 임의로 return 하는 `get_1_or_0` 함수를 구현하세요.

>
```
const get_1_or_0 = () => Math.round(Math.random())
```

(ramdom.js)

1번에는 구현 방법에 따로 제한이 없었으므로 Math.round를 통해 간단하게 난수를 생성했습니다.

2. 1.에서 작성한 `get_1_or_0`을 이용하여 숫자 `n`을 인자로 받아 `0~n` 사이의 임의의 정수를 반환하는 `get_random` 함수를 구현하세요.

```
  const get_random = (n) => {
  const typeValidity = typeof n === "number"
  const rangeValidity = n > 0 && n % 1 === 0

  if (typeValidity && rangeValidity) {
    const randomNum =
      parseInt(require('crypto').randomBytes(n.toString().length).toString('hex'), 16)
    const rangedRandom =
      (randomNum*n).toString().substring(0,((randomNum*n).toString().length)-(randomNum.toString().length))*1

    for (i = rangedRandom; i <= n; i++) {
      if (get_1_or_0() || i == n) {
        return i;
        break;
      }
    }
  }
 };

module.exports = get_random

for (j=0; j<100; j++) {
  console.log(get_random(100))
}
```

(ramdom.js)

1) 데이터 타입을 체크하여 숫자만 거릅니다.

2) 정수만 통과하도록 1로 나누어 떨어지지 않을 경우는 배제했습니다.
   (입력가능한 숫자 범위가 명확히 지정되지는 않았으나, 반환 범위가 0~n 이므로, 0과 음수도 배제했습니다.)

3) crypto의 randomBytes 메소드로 랜덤한 숫자를 추출한 뒤, 범위에 맞게 조정하였습니다.
   반복문으로 랜덤하게 숫자를 추가해 랜덤적인 요소를 추가했습니다.

3. 2.에서 작성한 `get_random` 함수에 대한 테스트 함수를 **최대한 넓은 범위**를 cover할 수 있도록 작성하고, 테스트 내역에 대한 보고서를 자유 형식으로 제출하세요.

> (test.js)

1) 우선 작은숫자(1)부터 큰 숫자(9007199254740991)까지, 결과값의 타입이 number인지 확인했습니다.

2) 0과 음수, 소수점을 포함한 숫자의 결과값이 undefined로 나오는지 확인했습니다.
   (이 부분에 대해 따로 지정해주신 것이 없어 undefined로 두었습니다)

3) 문자열로 된 숫자, 문자열, Boolean, object 등 다른 타입 투입시 undefinedr가 나오는지 확인했습니다.

> (ramdom.js)

1) 1부터 100까지 테스트시, 결과값이 고르게 나오는지 확인했습니다.

```
PASS ./test.js
  √ numbers/true (2 ms)
  √ numbers/false (1 ms)
  √ otherTypes/false (1 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.459 s, estimated 1 s
Ran all test suites.
```

퀀터스테크놀로지스-FE 과제유형 1

Description
`assginment-fe1.js` 파일은 다음과 같은 기능을 하는 컴포넌트를 구현한 파일입니다.

1. 컴포넌트가 마운트 된 후, 임의의 숫자를 받은 후 그것을 표현
2. 현재 화면의 수직 스크롤 값 `(window.scrollY)`을 표현

위 컴포넌트의 구현에서 이슈가 되는 부분을 **가능한 많이** 찾고 그것을 고쳐서 제출하세요.

> (Comp.js)

```
import React, { useState, useEffect } from "react";

// imitation of a request to the server. just get the number asynchronously
export const generateRandomNumber = () => Promise.resolve(Math.random());

const Comp = () => {
  const [num, setNum] = useState(null);
  const [scroll, setScroll] = useState(null);

  // 서버에서 숫자를 받아오는 함수 : async-await가 정상적으로 작동하도록 고쳤습니다.
  const getNum = async () => {
    setNum(await generateRandomNumber());
  };

  useEffect(() => {
    // 숫자 받아오는 함수를 useEffect 외부로 이동하였습니다.
    // async-await 관련 수정만 한다면 useEffect 안에서 작성하여도 관계없겠으나,
    // 서버에서 숫자를 받아오는 함수는 useEffect 혹 외부에서도 사용할 일이 있을 듯 하여 이동시켰습니다.

    getNum();

    // 스크롤 이벤트가 지나치게 많이 발생하는 것을 방지하기 위해 interval을 설정했습니다.
    const timer = setInterval(() => {
      window.addEventListener("scroll", () => setScroll(window.scrollY));
    }, 500);

    return () => {
      clearInterval(timer);
      window.removeEventListener("scroll", () => setScroll(window.scrollY))
    };

    // 의존성 배열이 누락되어 있어 우선 비워 둔 상태로 추가했습니다.
  }, []);


  return (
    <div>
    {/* 스크롤 이벤트가 정상 작동하는지 확인하기 위해 임의로 추가한 코드입니다. */}
      <div style={{position:'fixed', width:'100%', textAlign:"center"}}>
        <div> Number: {num} </div>
        <div> Scroll: {scroll} </div>
      </div>

    {/* 스크롤 이벤트가 정상 작동하는지 확인하기 위해 임의로 추가한 코드입니다. */}
      {Array.from({length:40}, (item, idx) => idx).map(i => <p key={i}>.</p>) }

    </div>
  );
};

// 컴포넌트를 export 하는 부분 추가하였습니다.
export default Comp;
```