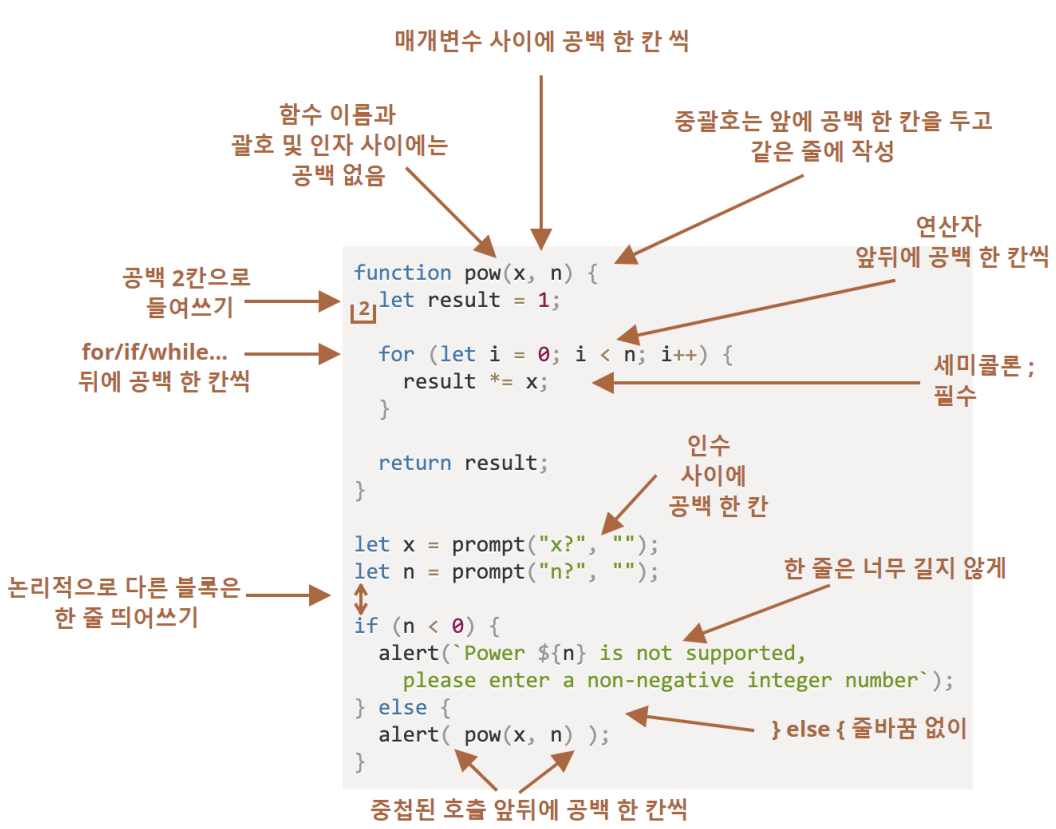
03. 코드 품질

코드 스타일

- 프로그래머스로 알고리즘 풀 때, 어떤 코드가 가독성이 좋을지 생각해보자.
- 모범 사례



- 함수 위치
    - 헬퍼 함수를 사용하는 코드 *위*에서 헬퍼 함수를 모아 선언하기

```
// 함수 선언
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}

// 헬퍼 함수를 사용하는 코드
let elem = createElement();
setHandler(elem);
walkAround();
```

    - 코드를 먼저, 함수는 그 다음에 선언하기

```
// 헬퍼 함수를 사용하는 코드
let elem = createElement();
setHandler(elem);
walkAround();

// --- 헬퍼 함수 ---
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}
```

- 사실 난 함수를 먼저 선언하고 코드를 작성하는 방식을 사용해왔다.
  - 애초에 JS말고 다른 언어에서는 이게 당연한 방식이다.
- 하지만 JS 튜토리얼에서는 후자를 권장하고 있다.
  - 이름만 보고도 함수의 역할을 유추할 수 있게 작명했다면, 함수를 읽을 필요가 없다는 게 이유
  - 함수형 프로그래밍의 관점에서 봤을 때에도 후자가 더 적절할 수 있겠다.

주석

- 좋은 코드엔 설명이 담긴 주석이 많지 않아야 한다.
  - 주석 역시도 관리 대상이기에, 리소스가 투입된다는 걸 명심해야한다.
  - 모익 때, 모든 컴포넌트/함수에 주석 작성하다가 오히려 관리에 실패한걸 명심해라..

함수 분리하기

- 아래의 함수를 분리해 *자기 설명적인(self-descriptive)* 코드를 만들어보자
- 함수 이름 자체가 주석 역할을 해 가독성을 높이는 코드를 *자기 설명적인* 코드라고 한다.

```
//before
function showPrimes(n) {
  nextPrime:
  for (let i = 2; i < n; i++) {

    // i가 소수인지를 확인함
    for (let j = 2; j < i; j++) {
      if (i % j == 0) continue nextPrime;
    }

    alert(i);
  }
}
```

```
//after
function showPrimes(n) {
  for (let i = 2; i < n; i++) {
    if (!isPrime(i)) continue;

    alert(i);
  }
```

```
}

function isPrime(n) {
  for (let i = 2; i < n; i++) {
    if (n % i == 0) return false;
  }
  return true;
}
```

- 함수는 주석이 없어도 그 존재 자체가 무슨 역할을 하는지 설명할 수 있어야 한다.
  - 즉, 읽기 쉽도록 적절히 분리하고 이름을 알맞게 지어줘야한다.

좋은 주석

- '설명이 담긴' 주석은 대체적으로 좋지 않지만, 좋은 주석도 있다.
  - 아키텍처를 설명하는 주석
    - 고차원 수준 컴포넌트 개요, 컴포넌트 간 상호작용에 대한 설명, 상황에 따른 제어 흐름 등에 대한 주석은 조감도 역할을 해준다.
    - 아예 아키텍처를 직접 그려보는 것도 좋다.
  - 함수 용례와 매개변수를 설명하는 주석 by JSDoc
    - 나름 이해하기 쉽게 코드를 작성했다고 생각했지만, 코드 리뷰하는 사람도 생각해주자.

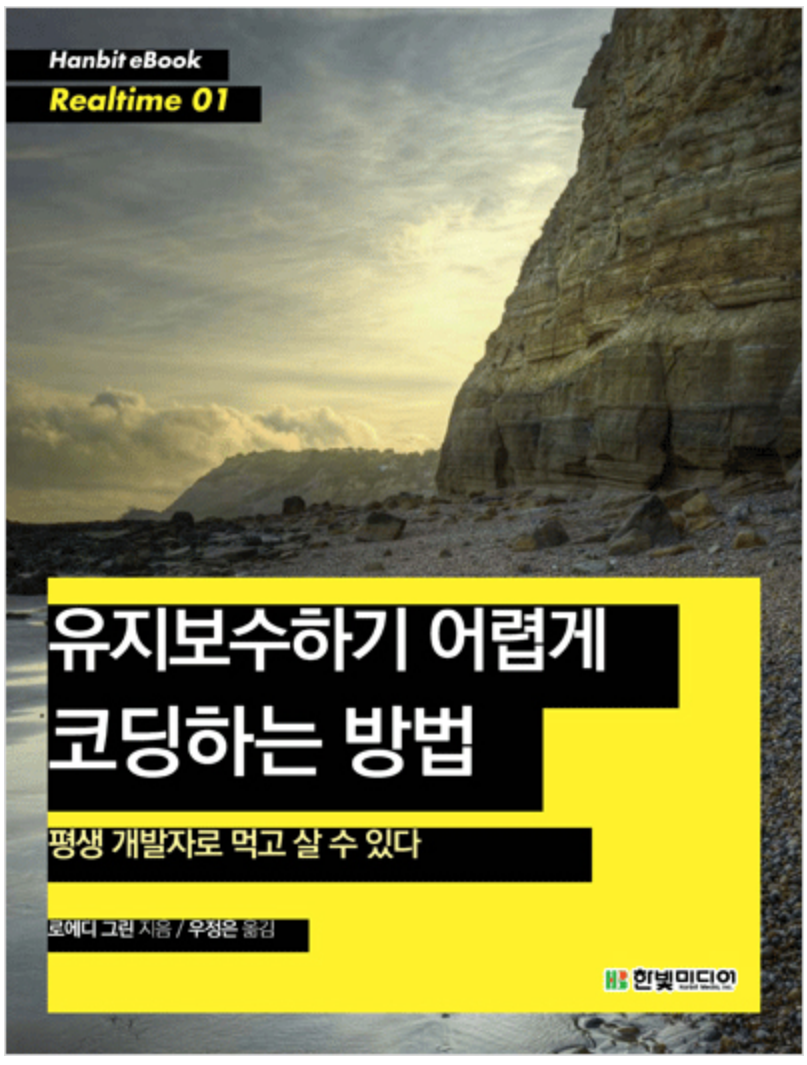
```
const RelationshipStatusCalculate = (
  relationshipStatus: RelationshipStatus,
  locked: boolean,
) => {
  switch (relationshipStatus) {
    case 'self':
      return 'self';
    case 'following':
      return 'none';
    case 'none':
      return locked ? 'pending' : 'following';
    case 'pending':
      return 'none';
    default:
      return relationshipStatus;
  }
};
```

  - 왜 이런 방법으로 문제를 해결했는지를 설명하는 주석
    - 이게 없다면?
    1. 시간이 지난 후 코드를 봤을 때, 다른 방법이 떠오른다
    2. "그때는 내가 멍청했구나. 하지만 지금은 더 똑똑해졌지"라고 생각하며, 이전보단 '더 명확하고 올바른' 방법으로 코드를 개선한다.
    3. n시간이 지난 뒤 '더 명확'하다고 생각했던 방법은 이미 시도해봤지만 문제가 있던 방법이라는 것을 깨닫는다. 뒤늦게 작성한 코드를 되돌렸지만, 시간은 돌아오지 않는다.
  - 미묘한 기능이 있고, 이 기능이 어디에 쓰이는지를 설명하는 주석

닌자코드

> *생각 없이 배우기만 하면 얻는 것이 없고,*
>
> *생각만 하고 배우지 않으면 오류나 독단에 빠질 위험이 있다.*
>
> "공자"

- 어떻게 하면 악질적으로 코드를 쓸 수 있는가



- 코드를 최대한 짧게 써라
  - 가독성은 필요없다

```
// 음 맛있다
i = i ? i < 0 ? Math.max(0, len + i) : i : 0;
```

- 글자 하나만 써서 변수를 작명해라
  - for문에서의 `i`만 빼고 ㅋ
- 최대한 모호한 약어를 써라
  - `btn`같이 쉬운거 말고 `browser` → `brsr`는 어떨까?
- 포괄적인 명사를 써라
  - 변수명으로 `data`, `value`, `str`, `num`를 쓰는 것을 추천한다.
  - 이미 있다면 `data1`, `value3`같이 숫자를 붙이는 것도 추천한다.
- 철자가 유사한 단어를 써라
  - `date`와 `data`를 적절하게 섞어 쓰는 건 어떨까?
- 이음동의어를 적극적으로 사용해라
  - 무언가를 보여주는 함수를 만들 때, `display`, `show`, `render`, `paint` 등을 번갈아가며 쓰는 건 어떨까? 문학적으로 다양한 코드를 작성할 수 있다.
    - 근데 이거 진짜 헷갈리더라, 다음 프로젝트에서는 무조건 통일하고 들어가야 할 것 같다.
- 변수 이름 재사용하기
  - 함수를 구현 중이라면 내부 변수를 선언하지 않고, 매개변수에서 넘어온 값만 사용하자
- 의미없는 언더스코어 사용하기
- 의미없는 형용사 사용하기
- 외부 변수 덮어쓰기
- 사이드 이펙트가 있는 코드 작성하기
  - `is..`, `check..`, `find...`등의 접두사가 붙은 함수가 뭔가를 바꿀 수 있는 기능을 넣어주면 다들 좋아할 것

- `check...` 함수의 return 값이 `boolean` 이 아닌 것은 어떨까?

- 함수에 다양한 기능 넣기
  - `validateEmail(email)` 함수에 유효한 이메일 주소를 확인하는 기능 외에, 에러 메세지 출력이 메일 주소 재입력 기능을 추가해보자

테스트 자동화와 Mocha

- 이 부분은 한 단락으로 하고 넘기기 아쉬워서 따로 작성

https://cksxkr5193.tistory.com/67

폴리필

- Js는 계속해서 진화하는 언어
  - 새로운 제안이 계속해서 등록, 분석되고 가치가 있다면 명세서(ECMA-262)에 추가됨
  - 한국 사람 중에서도 최근 카이스트 류석영 교수의 제안이 등록

NEWS
사이트 설명을 간략히 기록 해주세요.
https://kaist.ac.kr/news/html/news/?mode=V&mng_no=36610&skey=prof&sval=류석영&list_s_date=&list_e_date=&GotoPage=1

바벨(Bable)

- 최신 JavaScript(ES2015+)로 작성한 코드는 구 버전 브라우저의 엔진에서는 동작하지 않음
- 이 때, 사용하는 것이 트랜스파일러(Transpiler)인 바벨

Babel · Babel
The compiler for next generation JavaScript
https://babeljs.io/

- 바벨의 주요 역할은 두 가지다.
  1. 트랜스파일러
  2. 폴리필
     - 구현이 누락된 새로운 기능을 메꿔주는 역할
     - 예시
       - 오래된 브라우저에는 객체나 메서드에 대한 구현이 없기에, 아래의 코드는 실패함

```
[1, 2, 3].at(-1);

Promise.resolve(1);

new Set(1, 2, 3);
```

       - 이런 문제를 해결하기 위해서는 없는 구현을 채워줘야함
       - 아래와 같이 브라우저의 구현 포함 여부를 체크하고, 없으면 값을 채워줌

```
Array.prototype.at = Array.prototype.at ?? /* Array.prototype.at에 대한 자체 구현
```

똑똑하게 브라우저 Polyfill 관리하기
현대적인 JavaScript를 쓰면서도 넓은 범위의 기기를 지원하기 위한 Polyfill을 어떻게 똑똑하게 설정할 수 있는지 소개합니다.
https://toss.tech/article/smart-polyfills