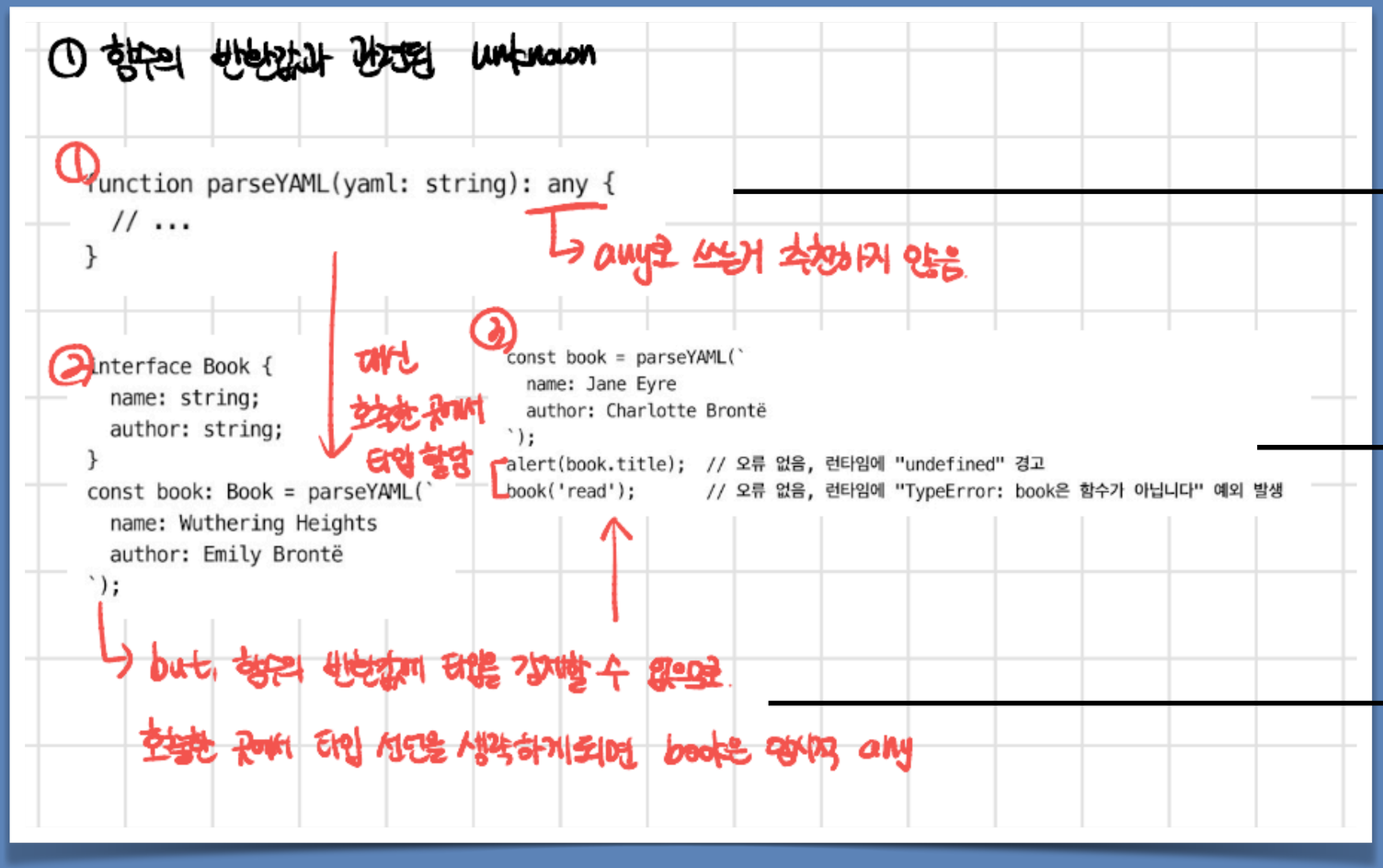
ITEM 42. 모르는 타입의 값에는 ANY 대신 UNKNOWN 사용하기

EDITED '23. 03. 04. (SAT) 신현호(@SWARVY)

1. 함수의 반환값과 관련된 UNKNOWN



반환형 any는 추천하지 않아요

parseYML이 any타입으로 반환되었기에
book은 암시적 any타입이에요

parseYML의 타입이 any이기에
호출한 곳에서 타입을 할당했어요

대신 unknown으로 사용하는것 추천 (더 안전함)

```
function safeParseYAML(yaml: string): unknown {
  return parseYAML(yaml);
}
const book = safeParseYAML(`
  name: The Tenant of Wildfell Hall
  author: Anne Brontë
`);
alert(book.title);
   // ~~~~ 개체가 'unknown' 형식입니다.
book("read");
// ~~~~~~~~~~~~ 개체가 'unknown' 형식입니다.
```

any대신 unknown을 사용하세요

함수 반환형을 any로 하는 것 보다 unknown으로 하는것이 더 안전해요

- any가 강력하면서도 위험한 이유

① 어떠한 타입이든 any타입에 할당 가능하다.

② any 타입은 어떠한 타입으로도 할당 가능하다.

한 집합은 다른 모든 집합의 부분 집합이면서 동시에 상위 집합이 될 수 있다.

→ 이 점에서 any는 타입 시스템과 상충되는 면을 가지고 있다.

이러한 점이 any의 강력함의 원천이면서 동시에 문제를 일으키는 원인이 된다.

타입 체커는 집합 개념이 때문에 any를 사용하면 타입 체커가 무용지물이 된다는 것을 인지해야 한다.

any는 강력한 기능을 가지기에 위험해요

어떠한 타입이던 할당 가능하고,
어떠한 타입으로도 할당 가능하기에
많은 문제들을 일으켜요

unknown은 any 대신 쓸 수 있는 타입 시스템의 부합하는 타입이다.

unknown은 ① 어떠한 타입이든 할당이 가능하다 는 만족하지만

② 어떠한 타입으로든 할당이 가능하다 는 만족하지않는다.

(unknown은 unknown 혹은 any에만 할당가능)

unknown은 any 대신 사용 가능한 타입이에요

근데 unknown인 채로 사용하려하면 오류 생겨요

항상 unknown 인 채로 사용하려하면 오류 발생.

(unknown인 값 출력. 연산도 마찬가지)

```
const book = safeParseYAML(`
  name: Villette
  author: Charlotte Brontë
`) as Book;
alert(book.title);
        // ~~~~~ 'Book' 형식에 'title' 속성이 없습니다.
book('read');
// ~~~~~~~~~~ 이 식은 호출할 수 없습니다.
```

반환된 타입인 unknown 그대로 사용 불가능하기때문에 타입 단언문 필요.

never라는 타입도 존재해요

any의 대용으로 쓰이는 친구는 아니고
unknown의 성질과 정반대라는걸
알아두시면 좋을 것 같아요

반대로 never는 unknown과 정반대이다.

① 어떠한 타입이던 할당이 가능하다 는 명확하지 않음.
(어떠한 값도 never에 할당할 수 없음)

② 어떠한 타입으로도 할당 가능하다 는 명확.


2. 변수와 관련된 UNKNOWN

② 변수 선언과 관련된 unknown.

어떠한 값이 있지만 그 타입을 모르는 경우에 unknown 사용

+ 혹은 타입을 예상할 수 없는 경우.

```
interface Feature {
  id?: string | number;
  geometry: Geometry;
  properties: unknown;
}
```

타입을 모르거나,
타입을 예상할 수 없으면
변수 타입을 unknown으로 사용하세요

물론 이것도 그냥은 못쓰니까
타입 단언문 등으로 변경해서 사용하세요

```
const book = safeParseYAML(`
  name: Villette
  author: Charlotte Brontë
`) as Book;
alert(book.title);
        // ~~~~ 'Book' 형식에 'title' 속성이 없습니다.
book('read');
// ~~~~~~~~~~ 이 식은 호출할 수 없습니다.
```

타입 단언문이 unknown에서 원하는 타입으로 변환하는 유일한 방법은 아니다.

instanceof 쓰세요.

```
function processValue(val: unknown) {
  if (val instanceof Date) {
    val  // 타입이 Date
  }
}
```

타입 단언문 말고도 타입 변환 가능해요
Instanceof 쓰면 바껴요

또한, 사용자 정의 타입 가드로 unknown에서 원하는 타입으로 변환할 수 있다.

```
function isBook(val: unknown): val is Book {
  return (
    typeof(val) === 'object' && val !== null &&
    'name' in val && 'author' in val
  );
}
function processValue(val: unknown) {
  if (isBook(val)) {
    val;  // 타입이 Book
  }
}
```

물론 직접 만드시는 방법도 있어요

unknown 타입의 범위를 좁히기 위해서는 상당히 많은 노력을 해야한다.

어떤 방법을 쓰시던간에
unknown타입이면 그냥 그대로는 못써요
취향껏 바꿔주세요

제네릭도 가능은 한데 그냥 앞에 나온 방법 쓰세요

```
function safeParseYAML<T>(yaml: string): T {
  return parseYAML(yaml);
}
```

↳ 쓰지마세요. unknown을 반환하고 사용자가 원하는대로 지정 안전하게 강제하게 더 좋다.


3. 단언문과 관련된 UNKNOWN

③ 단언문과 관련된 unknown.

```
declare const foo: Foo;
let barAny = foo as any as Bar;
let barUnk = foo as unknown as Bar;
```

기능적으로 동일하지만 단언문을 분리하는 리팩터링을 한다면

unknown이 더 안전.

① any → 분리하는 순간 영향력이 전염병처럼 번짐

② unknown → 분리하는 순간 오류.

리팩터링하면서 단언문을 분리할 수도 있는데
any를 분리하면 영향력이 전염병처럼 번져요

unknown은 분리하는 순간 오류가 발생하니 더 안전해요

unknown처럼 쓰는 object와 {}

- {} 타입은 null과 undefined를 제외한 모든 값을 포함
- object 타입은 모든 비원형 (non-primitive)로 이루어짐.

  여기에는 true, 12, "foo"와 같은 값들은 포함 X, 객체/배열 O

unknown이랑 비슷한 타입도 있어요

{}만 가끔 사용하는데
Null하고 undefined가 불가능한 경우에
Undefined 대신 사용하시면 돼요