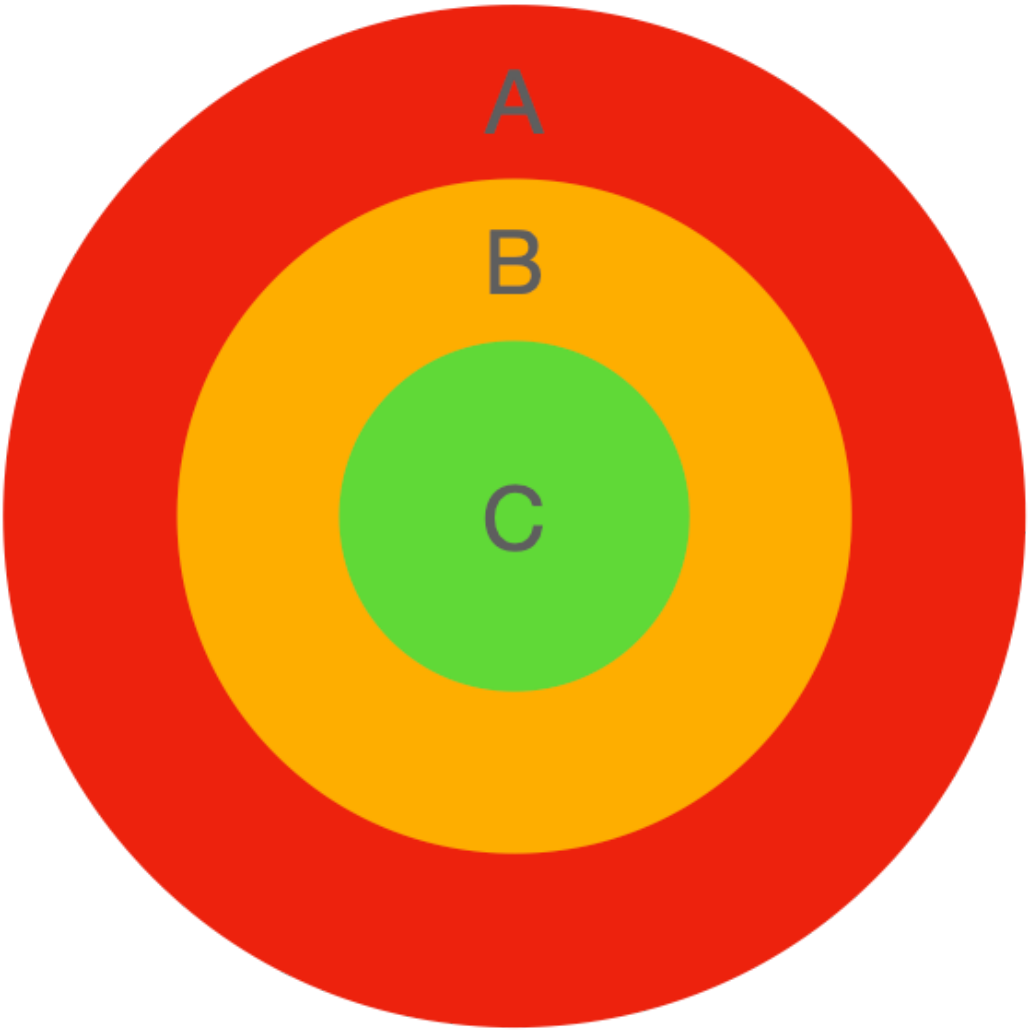
6장 고급 타입

- 강력한 타입 시스템

6.1.1 서브 타입, 슈퍼타입

- 서브 타입: 더 좁은범위의 타입
- 슈퍼 타입: 더 넓은범위의 타입



- any: 모든 타입의 슈퍼타입
- never: 모든 타입의 서브타입

```
type A = number | string

type B = string

A 는 B의 슈퍼타입
```

6.1.2 가변성

- 가변성

- 불변, 공변, 반변, 양변
- **공변: 사용할 타입의 서브타입만 이용 가능**

```
function a(value: strin) {}

a('abc');

function b(value: 'abc') {}

b('string') // Argument of type '"string"' is not assignable to parameter of type '"abc"'
```

- **반변: 파라미터로 함수를 전달할때 전달하는 함수의 인자 145p**

```
interface Animal {
  a: 1;
}

interface Bird extends Animal {
  b: 1;
}

interface Crew extends Bird {
  c: 1;
}

function a(f: (b: Bird) => (b: Bird) => Bird): void {}

const A: Bird = {
  a: 1,
  b: 1,
};

a((b: Bird) => (b: Bird) => {
  return A;
});
```

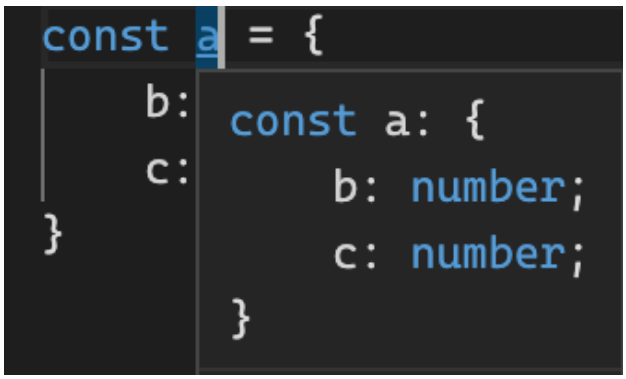
6.1.3 할당성

- 할당 가능하다 ⇒ 서브타입이다.

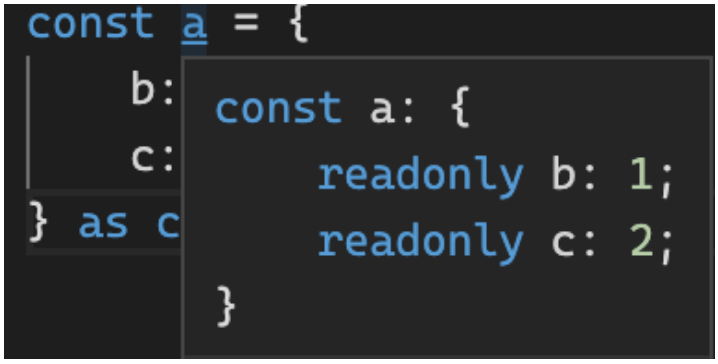
6.1.4 타입 넓히기

```
let a = 1 // a: number

a는 number로 추론된다. 타입스크립트의 기본 확장기능

const b = 1 // b: 1
```

- 변수들은 타입이 primitive type으로 넓혀진다.



- 객체의 경우, as const 어설션을 사용하면 property들을 readonly속성으로 사용할 수 있고, 타입이 좁혀진다.



6.1.5 정제

- typeof, instanceof, in 등의 연산자, 조건문에 의해 타입을 정제한다.

157p 설명은 어색하다.

```
type SquareOperatorParams = { type: 'square', payload: number}
type AddOperatorParams = { type: 'add', payload: [number, number]}

type OperatorParams = SquareOperatorParams | AddOperatorParams

function operate({ type, payload }: OperatorParams){
        if (type === 'square'){
                return payload ** 2 // payload: number
  }

        const [a, b] = payload // payload: [number, number]

        return a + b
}
```

6.2 종합성

- noimplicitReturns
- 반환타입 명시

https://github.com/gvergnaud/ts-pattern

6.3 고급 객체 타입

- key in(indexed access types)
- https://www.typescriptlang.org/docs/handbook/2/indexed-access-types.html

```
interface User {
  card: {
    expiredDt:string
    cardNumber: string
  }
}

type Card = User["card"]
```

User가 갖는 property인 card에 대한 타입을 새로 정의 하지 않고도 가져올 수 있음

- key of

```
interface User {
  card: ...
  age: ...
  name: ...
}

type A = keyof User // 'card' | 'age' | 'name'
```

객체의 키값들에 대한 유니온 타입

```
type WeekDay = 'Mon' | 'Tue' | 'Wed' | 'Thu' | 'Fri'
type Day = Weekday | 'Sat' | 'Sun'

const Map = Record<WeekDay, Day>

키, 밸류에 대한 타입을 지정, 누락된경우 타입 에러


type Record<K extends keyof any, T> = {
    [P in K]: T;
};
```

Mapped Types https://www.typescriptlang.org/docs/handbook/2/mapped-types.html

6.4 고급 함수 타입

- user-defined type guard 필요 예시

```
function a (value: unknown) {
    if(isString(value)){
        value // value: unknown
    }
}

function isString(value: unknown) {
```

```
    return typeof value === 'string'
}
```

user-defined type guard

```
function a (value: unknown) {
    if(isString(value)){
        value // value: string
    }
}

function isString(value: unknown): value is string {
    return typeof value === 'string'
}
```

6.5 조건부 타입

- extends 키워드 사용

```
type Exclude<T, U> = T extends U ? never : T;

type A = Exclude<1 | 2, 2 | 3> = (1 extends 2 | 3 ? never : 1) | (2 extends 2 | 3 ? never : 2) //

type Extract<T, U> = T extends U ? T : never;

type B = Exclude<1 | 2, 2 | 3> // 1
```

- infer 키워드

```
type Flatten<Type> = Type extends Array<infer Item> ? Item : Type;

type B = Flatten<number> // number
type C = Flatten<number[]> // number

type Awaited<Type> = Type extends Promise<infer Item> ? Item : never;

type B = Awaited<Promise<number>> // number
```

기타 유틸리티 타입들 https://www.typescriptlang.org/docs/handbook/utility-types.html

6.6.1 타입 어서션

- 슈퍼타입 또는 서브타입으로 타입을 단언

```
function a(value: string){
  b(value) // value: string // Argument of type 'string' is not assignable to parameter of type '"
  b(value as 'abc')

        b(1 as 'abc') // Conversion of type 'number' to type '"abc"' may be a mistake because neit
}

function b(value: 'abc'){
```

```
    value
}
```

6.8 프로토타입 안전하게 확장하기

- 런타임에 동적으로 프로토타입을 확장할 수 있지만, 안전하게 사용하기 위해 declare global로 타입 확장, 가장먼저 실행됨을 보장하자