# 아이템13. 타입과 인터페이스 차이점 알기

# 아이템13 타입과 인터페이스 차이점 알기

명명된 타입 (named type) 정의하는 방법

- 타입 사용

```
type TState = {
    name: string;
    capital: string;
}
```

- 인터페이스 사용

```
interface IState {
    name: string;
    capital: string;
}
```

# 아이템13 타입과 인터페이스 차이점 알기

IState와 TState를 추가속성과 함께 할당할 경우

```
type TState = {
    name: string;
    capital: string;
}
const wyoming: TState = {
    name: 'Wyoming',
    capital: 'Cheyenne',
    population: 500_000
};
```

# 아이템13 타입과 인터페이스 차이점 알기 - 비슷한점

인덱스 시그니처

```ts
type TDict = {[key: string]: string };
interface IDict {
    [key: string] : string;
}
```

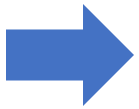
함수 타입

```ts
type TFunction = (x: number) => string;
interface IFunction {
    (x: number): string;
}
```

# 아이템13 타입과 인터페이스 차이점 알기 - 비슷한점

함수 타입

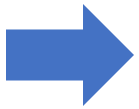
추가 속성



```ts
type TFnWithProperties = {
    (x: number): number;
    prop: string;
}
interface IFnWithPropertise {
    (x: number): number;
    prop: string;
}
```

# 아이템13 타입과 인터페이스 차이점 알기 - 비슷한점

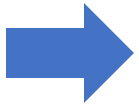
제너릭 가능



```ts
type TPair<T> = {
    first: T;
    seond: T;
}
interface IPair<T> {
    first: T;
    seond: T;
}
```

# 아이템13 타입과 인터페이스 차이점 알기 - 비슷한점

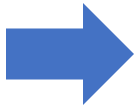
확장



```ts
interface IState{
    name: string;
    capital: string;
}

type TState = {
    name: string;
    capital: string;
}

interface IStateWithPop extends TState {
    population: number;

}
type TStateWithPop = IState & {population: number; };
```

# 아이템13 타입과 인터페이스 차이점 알기 - 비슷한점

클래스 구현



```
class StateT implements TState{
    name: string = '';
    capital: string= '';
}

class StateI implements IState {
    name: string = '';
    capital: string = '';
}
```

# 아이템13 타입과 인터페이스 차이점 알기 - 다른점

## 유니온 타입 확장

type AorB = 'a' | 'b' ;

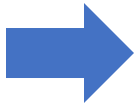
```
type Input = {};
type Output = {};
interface VariableMap {
    [name: string]: Input | Output;
}
```

type NamedVariable = (Input | Output) & {name: string};

# 아이템13 타입과 인터페이스 차이점 알기 - 다른점

## 튜플 | 배열 타입

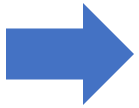
```
type Pair = [number, number];
type StringList = string[];
type NamedNums = [string, ... number[]];
```

튜플  TYPE

# 아이템13 타입과 인터페이스 차이점 알기 - 다른점

## 인터페이스 '보강'

```ts
interface IState {
    name: string;
    capital: string;
}
interface IState{
    population: number;
}
const wyoming: IState = {
    name: 'Wyoming',
    capital: 'Cheyenne',
    population: 500_000
};
```



선언 병합

# 아이템13 타입과 인터페이스 차이점 알기

- 타입과 인터페이스 비슷한점 / 차이점 알기
- 한 타입을 type 과 interface 두가지 다 사용하여 작성하는 방법 터득하기
- 프로젝트에서 어떤 문법을 사용할지 결정할 때 일관된 스타일을 확립하기
- 보강기법 필요한지 고려하기

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

## DRY 원칙

```
interface Person{
    firstName: string;
    lastName: string;
}

interface PersonWithBirthDate{
    firstNmae: string;
    lastName: string;
    birth: Date;
}
```

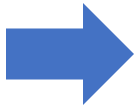
# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

```ts
interface Person {
    firstName: string;
    lastName: string;
}

interface PersonWithBirthDate extends Person{
    birth: Date;
}
```

```ts
interface Person {
    firstName: string;
    lastName: string;
}

type PersonWithBirthDate = Person & {birth: Date};
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

```ts
interface State{
    userId: string;
    pageTitel: string;
    recentFiles: string[];
    pageContents: string;
}

interface TopNavState {
    userId: string;
    pageTitle: string;
    recentFiles: string[];
}
```



```ts
type TopNavState = {
    userId: State['userId'];
    pageTitle: State['pageTitle'];
    recentFiles: State['recentFiles'];
}
```

```ts
type TopNavState = {
    [k in 'userId' | 'pageTitle' | 'recentFiles']: State[k]
};
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

```
type Pick<T,K> = {[k in K]: T[k] };

type TopNavState = Pick<State, 'userId' | 'pageTitle' | 'recenFiles'>;
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

```
interface SaveAction {
    type: 'save';
}
interface LoadAction {
    type: 'load';
}

type Action =  SaveAction | LoadAction ;
type ActionType = 'save' | 'load';


type ActionType = Action['type'];
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

```ts
interface Options {
    width: number;
    height: number;
    color: string;
    lable: string;
}

interface OptionsUpdate {
    width?: number;
    height?: number;
    color?: string;
    lable?: string;
}
class UIWidget{
    constructor(init: Options){}
    update(options: OptionsUpdate){}
}
```

```ts
type OptionsUpdate = {[k in keyof Options]? : Options[k]};
```

```ts
type OptionsKeys = keyof Options;
```

```
"width" | "height" | "color" | "label"
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

```
const INIT_OPTIONS ={
    width: 640,
    height: 480,
    color: '#00FF00',
    label: 'VGA',
};
interface Options {
    width: number,
    height:number,
    color: 'string',
    label: 'string',
}

type Options1 = typeof INIT_OPTIONS;
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

## 제너릭 타입 extends

```ts
interface Name{
    first:string,
    last:string,
}
type DancingDuo<T extends Name> = [T,T];

const couple1: DancingDuo<Name> = [
    {first: 'Fred', last:'Astaire'},
    {first: 'Ginger', last:'Rogers'},
];

const couple2: DancingDuo<{first: string}> = [
    {first: 'Sonny'},
    {first: 'Cher'}
];
```

```ts
interface Name{
    first:string,
    last:string,
}
type DancingDuo<T extends Name> = [T,T];

const couple1: DancingDuo = [
    {first: 'Fred', last:'Astaire'},
    {first: 'Ginger', last:'Rogers'},
];

const couple2: DancingDuo = [
    {first: 'Sonny'},
    {first: 'Cher'}
];
```

# 아이템14. 타입 연산과 제너릭 사용으로 반복 줄이기

## 제너릭 타입 extends

```
type Pick1<T, K> = {
    [k in K]: T[k]
};

type Pick2<T, K extends keyof T> = {
    [k in K]: T[k]
};
```

'string | number | symbol'

# 아이템15. 동적 데이터에 인덱스 시그니처 사용하기

# 아이템15. 동적 데이터에 인덱스 시그니처 사용하기

```ts
const rocket1 = {
    name: 'Falcon 9 ',
    variant: 'Block 5',
    thrust: '7,607 kN',
};
```

```ts
type Rocket = {[property: string]: string};
const rocket2 = {
    name: 'Falcon 9 ',
    variant: 'Block 5',
    thrust: '7,607 kN',
};
```

## 인덱스 시그니처

- 키의 이름
- 키의 타입
- 값의 타입

# 아이템15. 동적 데이터에 인덱스 시그니처 사용하기

```ts
interface Rocket {
    name: string;
    variant: string;
    thrust_kN: number;

}

const falconHeavy: Rocket = {
    name: 'Falcon Heavy',
    variant: 'v1',
    thrust_kN: 15_200
};
```

# 아이템15. 동적 데이터에 인덱스 시그니처 사용하기

연관 배열 associative array → Map 타입

광범위한 String 타입 → 1.Record

2.매핑된 타입