Typescript

- type alias
  - type Nome-do-tipo = tipos.
- union
  - tipo | tipo | tipo.
- intersection
  - tipo & tipo.
- KeyOf ⇒ possui as possibilidades das keys de um tipo.
- Mapped types ⇒ [key: string] : boolean | horse;
  - Mostra que um objeto que possui esse tipo tem as chaves do tipo string e os valores do tipo boolean | horse, evitando repetição.
  - Podemos também usar o keyOf para pegar os tipos das keys de um outro tipo.

```
type OptionsFlags<Type> = {
  [Property in keyof Type]: boolean;
};
```

- Classes
  - Tipos de propriedades
    - Private: não pode ser alterado fora da classe criadora.
    - Protected: pode ser alterado apenas pela classe criadora e pelas classes herdeiras.
    - Public: todo mundo faz o que quer.
    - Readonly: só pode ser lido por qualquer um.
  - Getters e Setters
    - São usados como propriedade.

        - get method-name() {} ⇒ a.method-name
        - set method-name2() {} ⇒ a.method-name2 = 'o'
    - Interfaces
        - São projetos de classes, mostrando como será a estrutura do objeto.
        - pode extender outras interfaces.
        - pode ser usada como tipo.
        - pode ser implementada por uma classe.
- Type alias x Interfaces
    - tipos podem receber números primitivos.
    - tipos podem declarar tuplas.
    - Pode declarar só uma vez o tipo por escopo.
        - Melhor para passar props.
    - Pode declarar várias vezes uma interface, ela será adicionada as anteriores.
        - Quando criar uma lib, use interfaces por serem extensíveis.
- Type utilities
    - Partial<Type> ⇒ Constrói um tipo com todas as propriedades de Type como opcionais.
    - Required<Type> ⇒ Constrói um tipo com todas as propriedades de Type como necessárias.
    - Readonly<Type> ⇒ Constrói um tipo com todas as propriedades de Type como apenas para leitura.
    - Record<Keys, Type> ⇒ Mapeia as propriedades de um tipo para um outro tipo.
    - Pick<Type, Keys> ⇒ Pega algumas propriedades (keys) de um tipo (Type).
    - Omit<Type, Keys> ⇒ Pega todas as propriedades de um tipo (Type) menos as (keys).
    - Exclude<Type, ExcludeUnion> ⇒ cria um tipo (Type) sem as propriedades de (ExcludeUnion).

- Decorator
  - Vigia e e executa ações.
  - São funções que recebe parâmetros específicos.
  - basta colocar @nome-do-decorator
  - Usamos o padrão de factory
  - Para classes
    - recebe um target que é o construtor da classe.
  - Para propriedades
    - recebe o target.
    - recebe a key que é o nome da propriedade.
    - Podemos fazer getters e setters e utilizar Object.defineProperty(target, key, { get: getter, set: setter}) para fazer validações e outros.
  - Para métodos
    - recebe target, key e o property descriptor(descrição do método).