Type vs Interface
- Interfaces can only describe object shapes.
- Interfaces can be extended by declaring it multiple times.
- In performance critical types interface comparison checks can be faster.

Think of Types Like Variables
Much like how you can create variables with the same name in different scopes, a type has similar semantics.

Build with Utility Types
TypeScript includes a lot of global types which will help you do common tasks in the type system. Check the site for them.

Object Literal Syntax
```typescript
type JSONResponse = {
  version: number;  
  /** In bytes */
  payloadSize: number; 
  outOfStock?: boolean; 
  update: (retryTimes: number) => void; 
  [key: number]: JSONResponse; 
  new (s: string): JSONResponse; 
}
```
Note: Field for attached docs
Optional
Function Type
Type is callable
Accepts any index
Newable
Readonly property

Loop through each field in the type generic parameter "Type"

Sets type as a function with original type as param

Conditional Types
Acts as "if statements" inside the type system. Created via generics, and then commonly used to reduce the number of options in a type union.

```typescript
type HasFourLegs<Animal> = Animal extends { legs: 4 } ? Animal : never;
type Animals = Bird | Dog | Ant | Wolf;
type FourLegs = HasFourLegs<Animals>  
// Dog | Wolf
```

Template Union Types
A template string can be used to combine and manipulate text inside the type system.

```typescript
type SupportedLangs = "en" | "pt" | "zh";
type FooterLocaleIDs = "header" | "footer";
type AllLocaleIDs = 
  `{${SupportedLangs}.${FooterLocaleIDs}.id}`;
  // "en_header_id" | "en_footer_id" 
  | "pt_header_id" | "pt_footer_id" 
  | "zh_header_id" | "zh_footer_id"
```