## 1. Primitive Types (Kiểu nguyên thủy):

Các kiểu nguyên thủy trong TypeScript là những kiểu dữ liệu cơ bản, không thể thay đổi giá trị của chúng. Một số kiểu nguyên thủy phổ biến là:

- `number`: Đại diện cho các giá trị số.
- `string`: Đại diện cho chuỗi ký tự.
- `boolean`: Đại diện cho giá trị `true` hoặc `false`.
- `null`: Đại diện cho một giá trị null (rỗng).
- `undefined`: Đại diện cho một biến chưa được khởi tạo giá trị.
- `symbol`: Đại diện cho các giá trị độc nhất trong hệ thống.
- `bigint`: Đại diện cho các số nguyên lớn.

## 2. Object Types (Kiểu đối tượng):

Kiểu đối tượng là những cấu trúc phức tạp có thể chứa các thuộc tính và phương thức. Trong TypeScript, kiểu đối tượng có thể là:

- **Object**: Là kiểu cơ bản đại diện cho bất kỳ đối tượng nào.
- **Mảng (Array)**: Là kiểu đối tượng có thể chứa các phần tử cùng loại.
- **Hàm (Function)**: Là kiểu đối tượng đại diện cho một hàm.
- **Class**: Là kiểu đối tượng do người dùng định nghĩa với các thuộc tính và phương thức.

## 3. Union Types (Kiểu hợp):

Kiểu hợp cho phép một giá trị có thể là một trong các kiểu được chỉ định. Dùng dấu `|` để định nghĩa kiểu hợp.

```
let value: string | number;
value = "Hello"; // hợp lệ
value = 123;     // hợp lệ
```

## 4. Intersection Types (Kiểu giao):

Kiểu giao cho phép một giá trị có thể mang các thuộc tính của nhiều kiểu khác nhau. Dùng dấu `&` để định nghĩa kiểu giao.

```
type Person = { name: string };
type Age = { age: number };
type PersonWithAge = Person & Age;
let person: PersonWithAge = { name: "Alice", age: 30 };
```

## 5. Interface vs Type Alias:

- **Interface**: Thường dùng để định nghĩa cấu trúc của đối tượng hoặc lớp. `Interface` có thể được kế thừa và mở rộng.

- **Type Alias**: Dùng để tạo tên mới cho các kiểu dữ liệu. `Type` có thể đại diện cho bất kỳ kiểu nào, bao gồm kiểu hợp (union), kiểu giao (intersection), hoặc tuple.
- Sự khác biệt chính là `interface` có thể mở rộng, còn `type alias` không thể.

```
interface Person {
    name: string;
}
type Car = { make: string; model: string };
```

## 6. Generics (Kiểu tổng quát):

**Generics** là một tính năng mạnh mẽ của TypeScript cho phép tạo ra các cấu trúc dữ liệu hoặc hàm có thể làm việc với nhiều kiểu khác nhau mà không làm mất đi tính an toàn về kiểu.

```
function identity<T>(arg: T): T {
    return arg;
}
let result = identity(42); // result có kiểu number
```

## 7. Decorators (Trang trí):

**Decorators** là một tính năng trong TypeScript (và ESNext) cho phép gắn các hành vi bổ sung vào các lớp, phương thức, thuộc tính, hoặc tham số. Nó giúp bạn làm việc với các lớp một cách linh hoạt hơn.

- Ví dụ, có thể dùng decorator để thêm logic vào lớp:

```
function log(target: any, key: string) {
    console.log(`${key} has been called.`);
}

class MyClass {
    @log
    myMethod() {}
}
```

## 8. Optional & Readonly Properties (Thuộc tính tùy chọn và chỉ đọc):

- **Optional Properties**: Các thuộc tính có thể không được khai báo trong đối tượng. Dùng dấu `?` để định nghĩa thuộc tính tùy chọn.
- `interface Person {`
- `    name: string;`
- `    age?: number; // Thuộc tính age là tùy chọn`
- `}`
- **Readonly Properties**: Các thuộc tính không thể thay đổi giá trị sau khi được khởi tạo. Dùng từ khóa `readonly` để định nghĩa thuộc tính chỉ đọc.
- `interface Person {`
- `    readonly name: string;`
- `}`

## 9. Enum & Tuples:

- **Enum**: Là kiểu dữ liệu đặc biệt trong TypeScript giúp định nghĩa một tập hợp các giá trị hằng số có tên.
- ```
  enum Direction {
  ```
- ```
      Up = "UP",
  ```
- ```
      Down = "DOWN",
  ```
- ```
      Left = "LEFT",
  ```
- ```
      Right = "RIGHT"
  ```
- ```
  }
  ```
- **Tuples**: Là kiểu dữ liệu giống mảng nhưng có độ dài cố định và mỗi phần tử có thể có kiểu khác nhau.
- ```
  let tuple: [string, number] = ["hello", 42];
  ```
  ```