Sure, TypeScript enums provide a way to define a set of named constants. This can be particularly useful when you have a fixed set of values that a variable can take.

Here's how you define an enum in TypeScript:

```typescript
enum Direction {
  Up,
  Down,
  Left,
  Right,
}
```

In this example, `Direction` is the name of the enum, and it contains four members: `Up`, `Down`, `Left`, and `Right`. By default, enums in TypeScript are zero-based, which means `Up` is assigned the value `0`, `Down` is `1`, `Left` is `2`, and `Right` is `3`. However, you can explicitly set the values if needed:

```typescript
enum Direction {
  Up = 1,
  Down,
  Left,
  Right,
}
```

In this case, `Up` is `1`, `Down` is `2`, `Left` is `3`, and `Right` is `4`.

You can also mix string and numeric members:

```typescript
enum Direction {
  Up = "UP",
  Down = "DOWN",
  Left = "LEFT",
  Right = "RIGHT",
}
```

Enums can be used as types:

```typescript
```

```typescript
let playerDirection: Direction;
playerDirection = Direction.Up;
```

Enums in TypeScript are bidirectional. This means you can convert from the numeric value to the enum's name and vice versa:

```typescript
let directionString: string = Direction[0]; // Returns "Up"
let directionNumber: number = Direction.Up; // Returns 0
```

Using enums can help make your code more readable and maintainable, especially when dealing with a fixed set of related values.