```
function startRace(race: Race): Race {
  race.status = RaceStatus.Started;
  return race;
}
```

If the function returns nothing, you can show it using `void`:

```
function startRace(race: Race): void {
  race.status = RaceStatus.Started;
}
```

5.4. Interfaces

That's a good first step. But as I said earlier, JavaScript is great for its dynamic nature. A function will work if it receives an object with the correct property:

```
function addPointsToScore(player, points) {
  player.score += points;
}
```

This function can be applied to any object with a `score` property. How do you translate this in TypeScript? It's easy: you define an interface, like the "shape" of the object.

```
function addPointsToScore(player: { score: number; }, points: number): void {
  player.score += points;
}
```

It means that the parameter must have a property called `score` of the type `number`. You can name these interfaces, of course:

```
interface HasScore {
  score: number;
}
    function addPointsToScore(player: HasScore, points: number): void {
      player.score += points;
    }
```

5.5. Optional arguments

Another treat of JavaScript is that arguments are optional. You can omit them, and they will become `undefined`. But if you define a function with typed parameter in TypeScript, the compiler will shout at you if you forget them: