months later, the TypeScript team announced that they had worked closely with the Google team, and the new version of the language (1.5) would have all the shiny new things AtScript had. And the Google team announced that AtScript was officially dropped, and that TypeScript was the new top-notch way to write Angular apps!

4.2. Enters TypeScript

I think this was a smart move for several reasons. For one, no one really wants to learn another language extension. And TypeScript was already there, with an active community and ecosystem. I never really used it before Angular, but I heard good things on it, from various people. TypeScript is a Microsoft project. But it's not the Microsoft you have in mind, from the Ballmer and Gates years. It's the Microsoft of the Nadella era, the one opening to its community, and, well, open-source. Google knows this, and it's far better for them to contribute to an existing project, rather than to have to bear the burden to maintain their own. And the TypeScript framework will gain a huge popularity boost: win-win, as your manager would say.

But the main reason to bet on TypeScript is the type system it offers. It's an optional type system that helps without getting in the way. In fact, after coding some time with it, I forgot about it: you can do Angular apps using TypeScript just for the parts where it really helps (more on that in a second) and simply ignore it everywhere else and write plain JS (ES6 in my case). But I do like what they have done, and we will have a look at what TypeScript offers in the next section. At the end, you'll have enough understanding to read any Angular code, and you'll be able to choose whether you want to use it or not (or just a little), in your apps.

You may be wondering: why use typed code in Angular apps? Let's take an example. Angular 1 and 2 have been built around a powerful concept named "dependency injection". You might already be familiar with it, as it is a common design pattern used in several frameworks for different languages and, as I said, already used in AngularJS 1.x.

4.3. A practical example with DI

To sum up what dependency injection is, think about a component of the app, let's say `RaceList`, needing to access the races list that the service `RaceService` can give. You would write `RaceList` like this:

```
class RaceList {
  constructor() {
    this.raceService = new RaceService();
    // let's say that list() returns a promise
    this.raceService.list()
    // we store the races returned into a member of `RaceList`
      .then(races => this.races = races);
    // arrow functions, FTW!
  }
}
```

But it has several flaws. One of them is the testability: it is now very hard to replace the `raceService`