
# TypeScript in a JavaScript world

*Or: How I learned to Stop Worrying and Love the Compiler*

# What is TypeScript?

**TypeScript is a** typed **superset of JavaScript that compiles to plain JavaScript.**

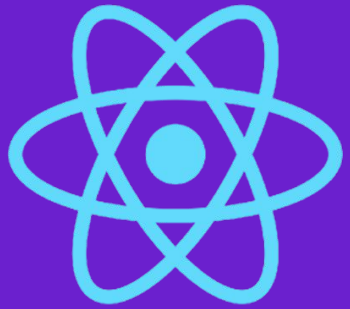
**TypeScript is JavaScript that** scales**.**

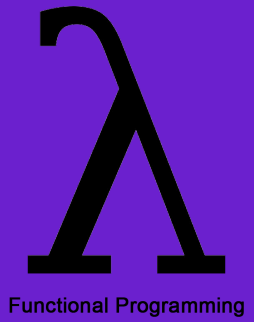
# Scale?

- TypeScript enables your tools to work better
  - Editors can use the compiler and associated infrastructure to show helpful errors and warnings
  - ⌘ + Space to see function arguments, or even JSX-style Props
- Reduce cognitive load, focus on the code you're writing now
  - Curried Dependency Injection is awesome for testing in JS, but now our editors lost any auto-completing since we are no longer importing anything
  - Look in another module to see what gets passed in, switch back to code
- Other developers are contributing code
  - Let the Interface do the talking, no more code sleuthing!

# TypeScript non-goals

- **Be Java / C#**
  - **TypeScript is a** superset **of JavaScript, all valid JavaScript will compile**
  - **Escape hatches from the type system are available**
- **Protect against run-time type errors**
  - **The compiler will protect against a lot of mistakes, but at the end of the day we are still dealing with JavaScript**
  - **Type system is not academically rigorous, it tries to strike a balance between correctness and productivity**
- **Force you to write types for everything**
  - Powerful **type inference**
- **“Fix” JavaScript**
  - **TS is JS: Warts and all**
  - **Provide modern features through transpiling**




```
const businessSearch = async (lat, lng, radius) => {
  try {
    const response = await businessSearchApi(lat, lng, radius);
    const jsonResponse = await response.json();

    const businesses = transformResponse(jsonResponse);
    return businesses;
  } catch (err) {
    return [];
  }
}
```

# TypeScript Cons

- Kind of weird type system
  - There will be a learning curve
- Porting a project can take some time
  - You may just learn how janky your code is :P
- Relying on typing stubs
  - Either community provided or bundled with libraries
  - No common “style” to these typing stubs
  - Potentially can be out of sync

Your code might work now...

But your tooling isn’t doing you any favors

**Interface**

```
interface Bar {
  x: number
  y?: number
}
```

```
class Foo implements Bar {
  x: number
  y?: number
  constructor() {
    this.x = 100
  }
}
class Baz extends Foo ...
```

## Union Types

const foo: Bar | Baz

**foo is either a bar or a baz**

## Intersection Types

const foo: Bar & Baz

**foo has all properties of bar AND baz**

## Type Alias

```
type MyInt = number | null
```

# Difference between Type and Interface?

???

General guidance is to use Interfaces most of the time

# Unit Type

type Filter = ‘all’ | ‘incomplete’

‘all’ and ‘incomplete’ are **TYPES**, not values, in this usage.
They are types with only one possible value (‘all and ‘incomplete’)

**Other unit types are** null **and** undefined

# Types Are Defined By Their Shape

```
interface Foo {
  x: string
  y?: number
}
const bar = (x: Foo) => x.x

// typeof bar == (x: Foo) => string  Type Inference!!!1!

bar({ x: ‘baz’ })
```

**We define an object literal, not a Foo.**
**TypeScript checks for “assignability”, does the Shape of the object conform to the required interface**
**(it does)**

# Show the code!!!