What Is TypeScript?

TypeScript is an open source language and is a superset of JavaScript

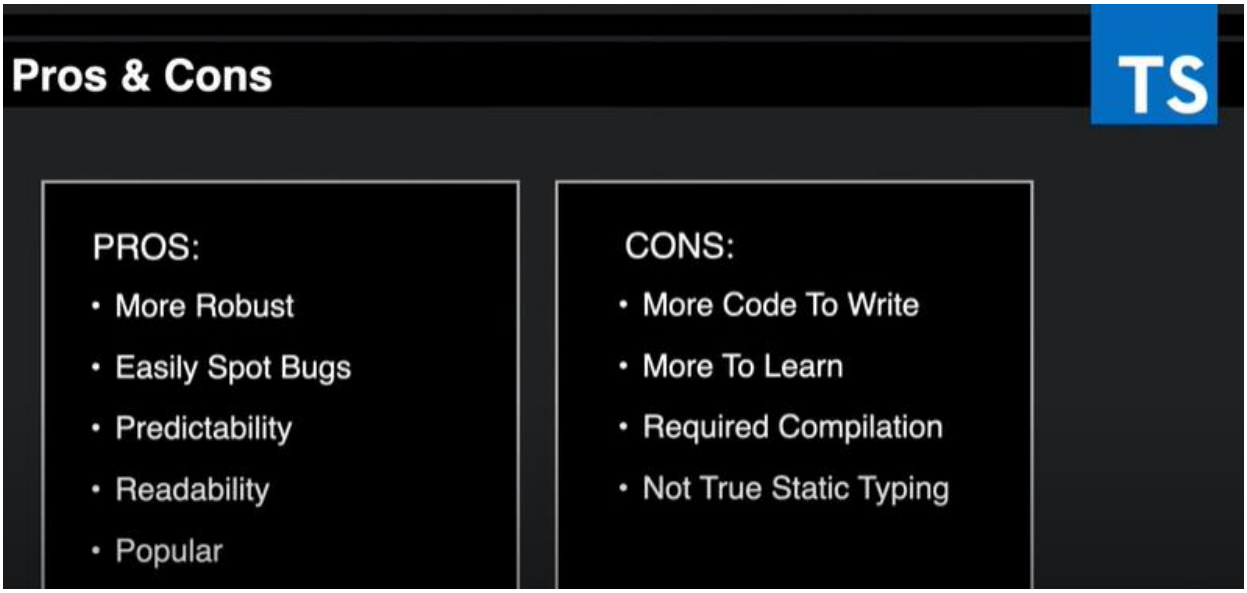
- Offers additional features to JavaScript including static types
- Using types is completely **optional**
- Compiles down to regular JS
- Can be used for front-end JS as well as backend with Node.js
- Includes most features from ES6, ES7 (classes, arrow functions, etc)
- Types from 3rd party libraries can be added with **type definitions**

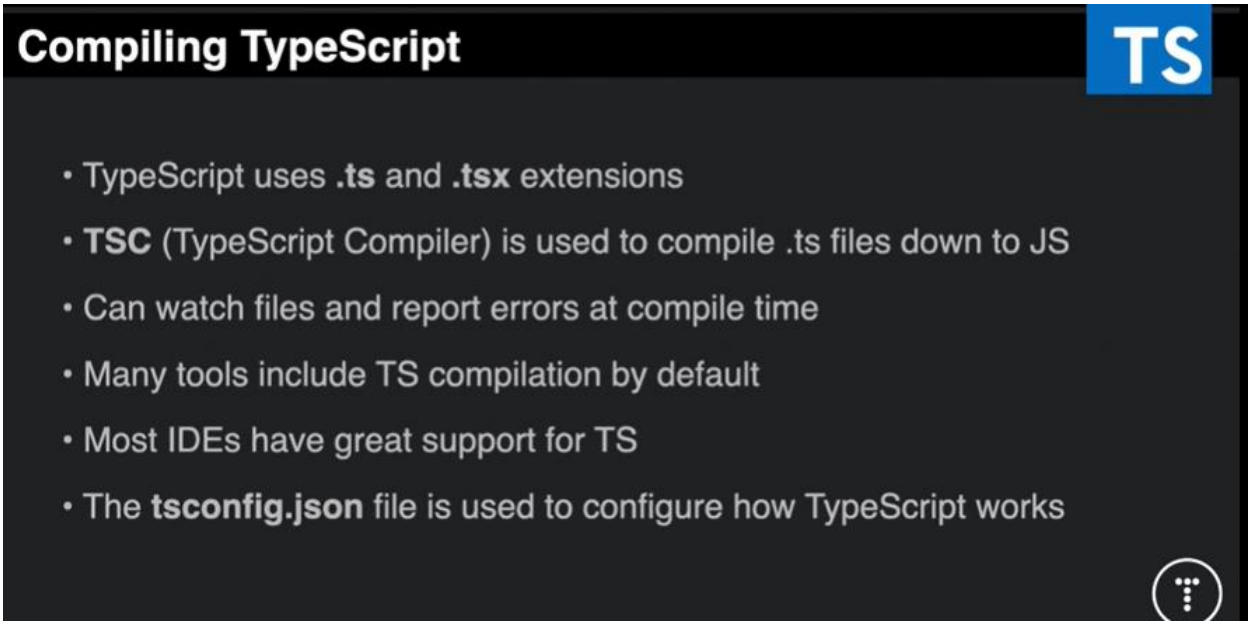
Dynamic vs Static Typing

In **dynamically typed languages**, the types are associated with run-time values and not named explicitly in your code

In **statically typed languages**, you explicitly assign types to variables, function parameters, return values, etc

Static Examples:	**Java, C, C++, Rust, Go**
Dynamic Examples:	**JavaScript, Python, Ruby, PHP**





Installation globally

```
npm i -g typescript
```

```
tsc —init
```

Used to create tsconfig file; some useful flags

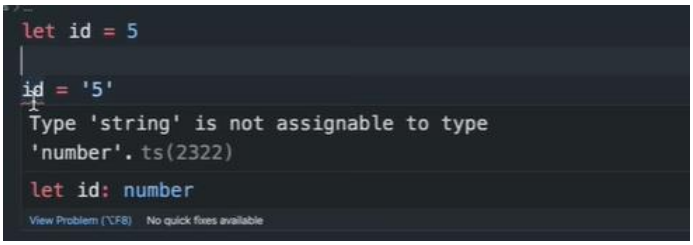
```
“target

“rootDir

“outDir
```

```
tsc --watch index
```

Type inference: even though type is not mentioned, error will be thrown




```
// Basic Types
let id: number = 5
let company: string = 'Traversy Media'
let isPublished: boolean = true
let x: any = 'Hello'

let ids: number[] = [1, 2, 3, 4, 5]
let arr: any[] = [1, true, 'Hello']

// Tuple
let person: [number, string, boolean] = [1, 'Brad', true]
// Tuple Array
let employee: [number, string][]
```

```
// Union
let pid: string | number
pid = '22'
```

```ts
// Enum
enum Direction1 {
  Up = 1,
  Down,
  Left,
  Right,
}

enum Direction2 {
  Up = 'Up',
  Down = 'Down',
  Left = 'Left',
  Right = 'Right',
}
```

```ts
// Objects
type User = {
  id: number,
  name: string
}

const user: User = {
  id: 1,
  name: 'John'
}
```

```ts
// Type Assertion
let cid: any = 1
let customerId = <number>cid

customerId = true
```

```ts
// Functions
function addNum(x: number, y: number): number {
  return x + y
}

function log(message: string | number): void {
  console.log(message)
}
```

```ts
// Interfaces
interface UserInterface {
  id: number
  name: string
}

const user1: UserInterface = {
  id: 1,
  name: 'John',
}
```

```ts
// Interfaces
interface UserInterface {
  readonly id: number
  name: string
  age?: number
}

const user1: UserInterface = {
  id:  Cannot assign to 'id' because it is a read-only
  name property. ts(2540)
}
      (property) UserInterface.id: any
View Problem (⌥F8)  No quick fixes available
user1.id = 5
```

```ts
interface MathFunc {
  (x: number, y: number): number
}

const add: MathFunc = (x: number, y: string): number => x + y
```

```ts
type Point = number | string
const p1: Point = 1
```

```
class Person {
  id: number
  name: string

  constructor(id: number, name: string) {
    this.id = id
    this.name = name
  }
}

const brad = new Person()
const mike = new Person()
```

```
interface PersonInterface {
  id: number
  name: string
  register(): string
}

// Classes
class Person implements PersonInterface {
  id: number
  name: string

  constructor(id: number, name: string) {
    this.id = id
```

```
class Employee extends Person {
  position: string

  constructor(id: number, name: string, position: string) {
    super(id, name)
    this.position = position
  }
}
```


// Generics
function getArray<T>(items: T[]): T[] {
return new Array().concat(items)
}
let numArray = getArray<number>([1, 2, 3, 4])
let strArray = getArray<string>(['brad', 'John', 'Jill'])
strArray.push(1)