TypeScript

Introduction & Overview

New Horizons®

Prerequisites

- JavaScript !!!
- HTML
- CSS

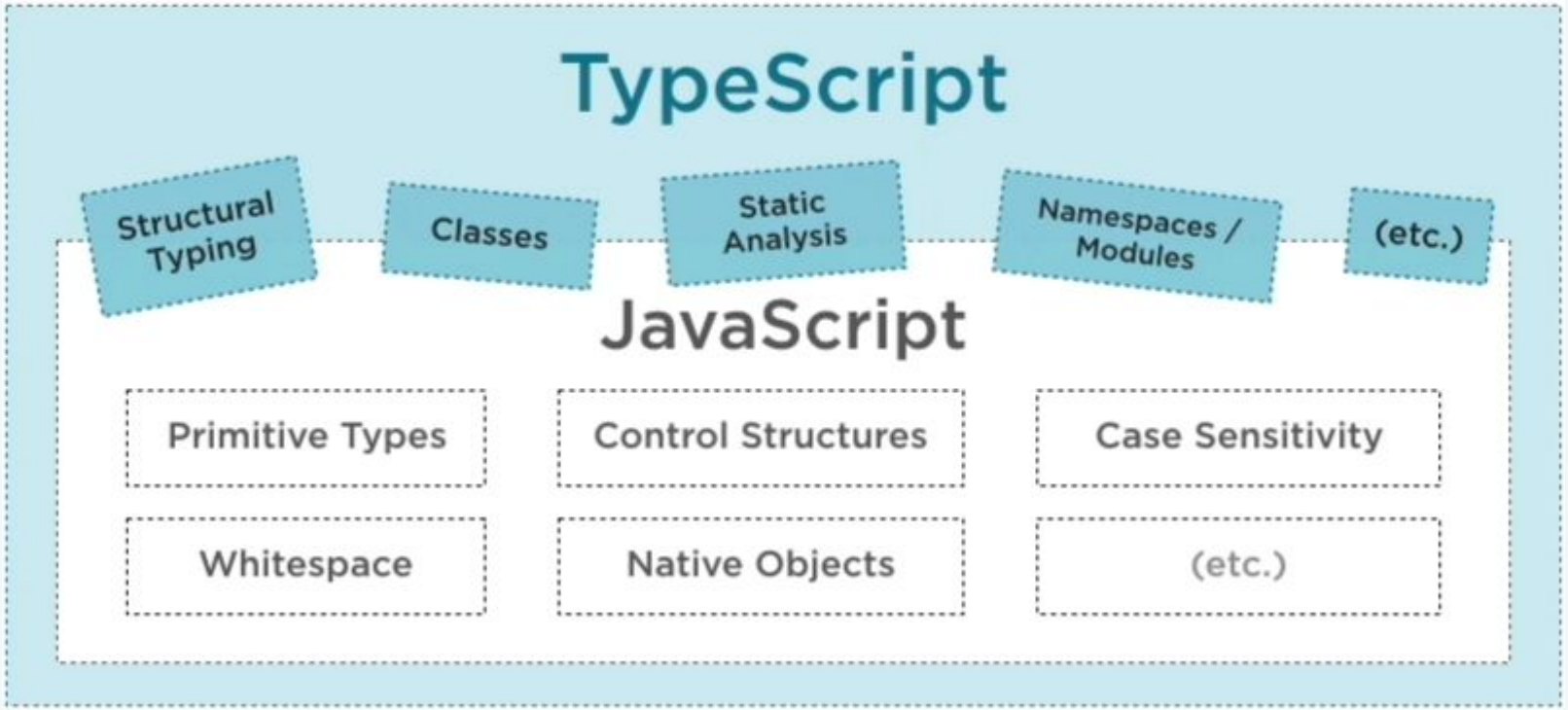
The only reason TypeScript exists is to improve the JavaScript experience!

Prerequisites

TypeScript is a **superset** of JavaScript.

Prerequisites

It is not a standalone language, it’s build on top.



Why TypeScript?

- JavaScript is EVERYWHERE
- It’s built in the browsers and every application and website uses it
- However, developing complex applications with JavaScript is… challenging

Why TypeScript?

- JavaScript is amazing at what it does on small scale
- Big applications require more complex architecture and tools to develop the code. So once we encounter a bigger project we are going to start asking questions and we are not going to like the answers.

Why TypeScript?

- How do we define specific type for a variable? We can’t.
- How do we organize the code into classes or modules or namespace? We don’t.
- When dealing with multiple files can we affect JS code in different files via something like an interface or header file? No.
- Do we at least have very good tooling and debugging support so we can get real time feedback as we are writing code? ...

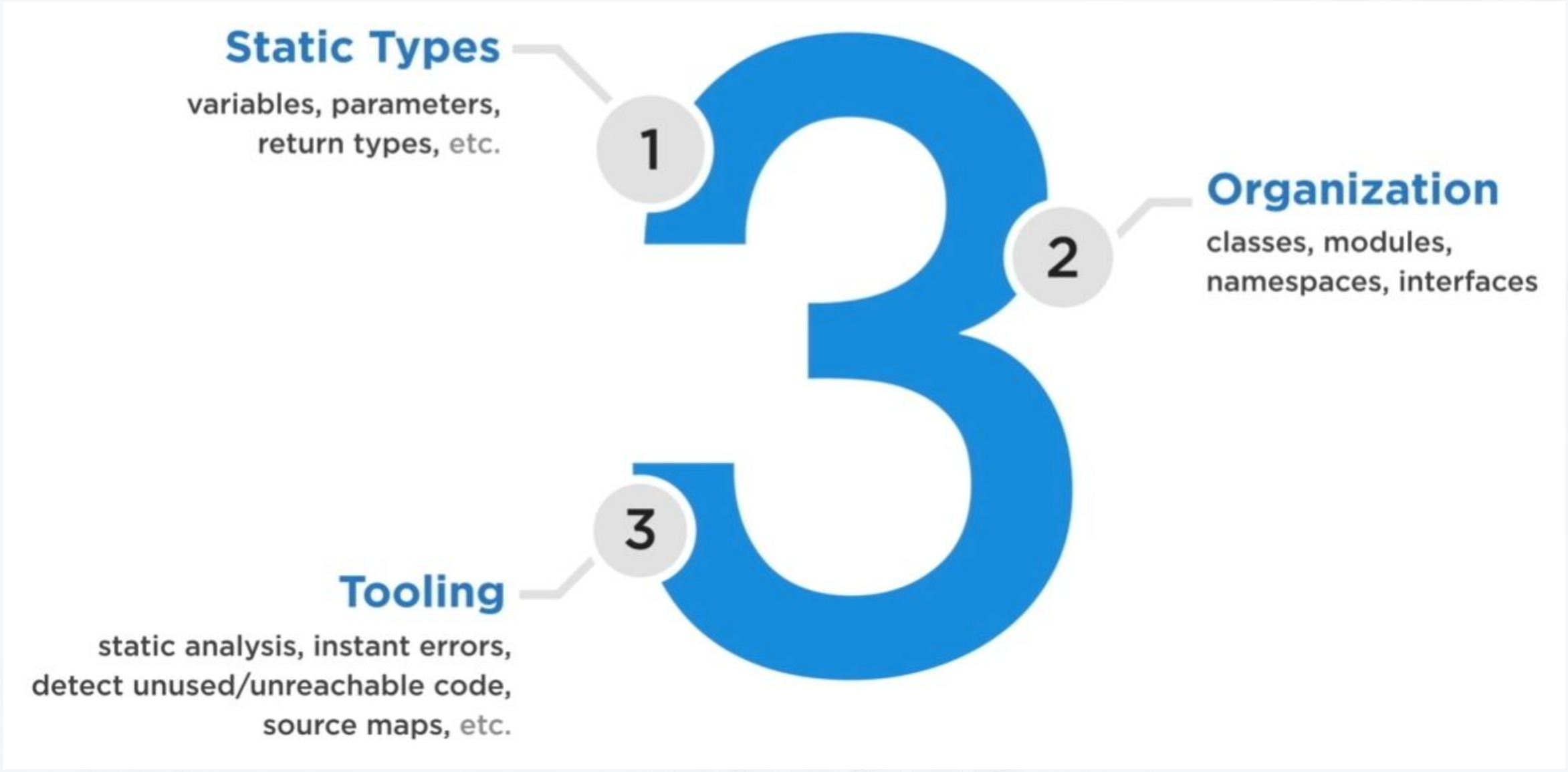
Why TypeScript?

- There are many solutions and changes to the JavaScript code
- TypeScript compiles directly into JavaScript
- It’s not an interpreted language itself so we don’t need a TypeScipt engine or runtime to work with it

Why TypeScript?

- Using TypeScript does not change anything in the way we run the application
- Any type of client / server software, any libraries or frameworks, any operating system, any browser - it’s all going to be the same because it’s still JavaScript
- It’s all about making the development easier with some extra features and rules

TypeScript Benefits



TypeScript Benefits

- Variable in JS do not have a strict type declaration
- In big projects that simply leads to unpredictable behavior because JS does not evaluate anything before run time

```
// Variables in JavaScript

var myVariable = "Here's a basic string";

// sometime later

myVariable = 42;

// and later still...

myVariable = [true,true,false,false,true];

// and yet later

myVariable = function() {
    console.log("Here's a simple log message");
}
myVariable();

// perhaps later still...

myVariable = null;
```

TypeScript Benefits

- Classes were introduced to ECMAScript 2015 and you can use them in both JS and TS with all structural benefits that they bring - inheritance, constructors, etc.
- TS also brings to the table namespaces and modules as you don’t want everything to be globally available
- It also brings Intefaces

SIDEBAR NOTE

Many of the features included in TypeScript were and are in a long awaited JavaScript wish lists. They are slowly getting integrated in newer ECMAScript releases but TS is much quicker into grasping the new concepts and making them available to developers.

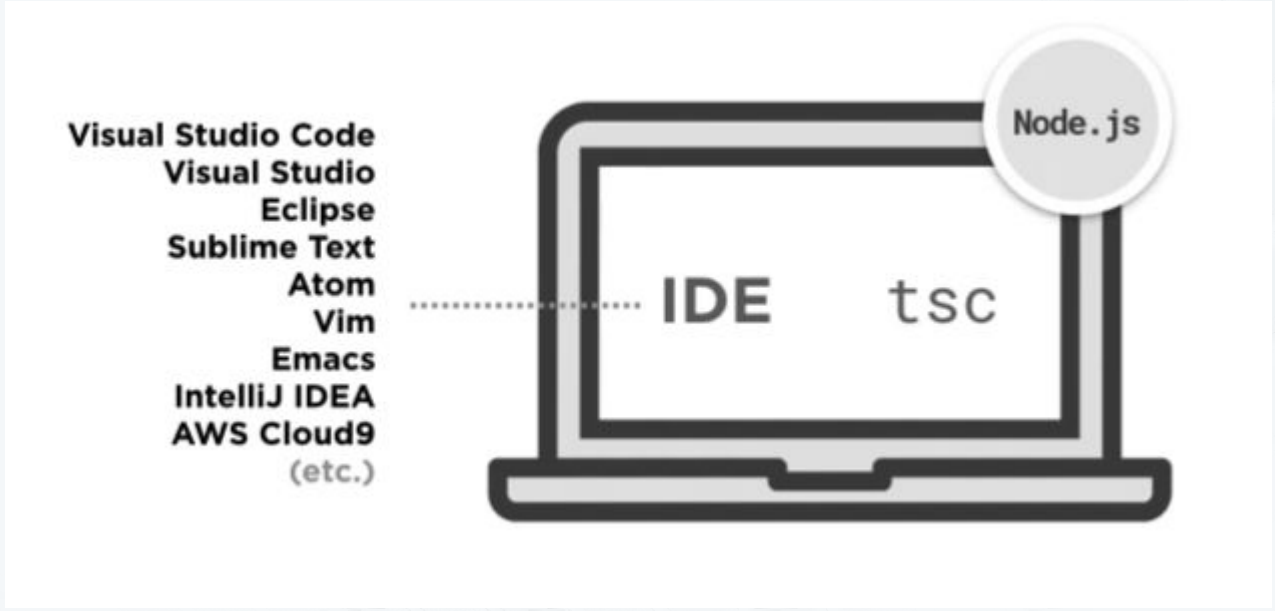
TypeScript Benefits

- Basic autocompletion and color coding
- Static type analysis and “instant” errors
- Detect unused and unreachable code
- Source mapping - allows you to debug the JS in the browser which then links back to the TS that generated that specific code

TypeScript Benefits

The Development Environment

- Apart from IDE (VS Code) you will also need a TypeScript compiler (called tsc) which will generate the JavaScript files
- Node.js has one embedded



The Development Environment

- Visual Studio Code:

  https://code.visualstudio.com/download

- Node.js:

  https://nodejs.org/en/download

  (it comes with npm - Node Package Manager)

The Development Environment

- npm is a great tool to access code packages, add them to your projects, update them and never worry about versioning
- Once you have installed Node.js, you can run the “npm” command straight in your cmd or terminal
- Installing TypeScript then becomes a breeze:

  npm install -g typescript

  NOTE: -g flag makes the installation global to your machine

The Development Environment

- You can confirm that you have TS by:

  tsc - v

- To see a list of compiler options type:

  tsc -h

Writing TypeScript

- Small pieces of code would not benefit from TS much so if you want a “Hello World” example - use JavaScript
- One of TS main goals is to provide a structuring mechanism for **larger pieces** of code
- Exclusion: you can benefit from writing small pieces of TS code while learning it

Writing TypeScript Demo

- Create basic HTML file with a h1 title
- Create a TypeScript file (.ts) with a class including a constructor and a method that returns a string
- Instantiate the class and call the method
- Change the text on the page replacing the h1 content
- Generate a JS file using tsc and include it in the project

Writing TypeScript Questions

- Can we change the generated JavaScript file?

  - Yes, but there's no point - change the TS instead

- Do you have to manually execute tsc after every edit?

  - No, you can setup automatic watches that will regenerate the JS files upon changes to the TS files

- Can we add tools for minifying, package bundling, etc.?

  - You do not need to change your existing pipeline as whatever tools work with your existing JS code will continue to do so with the new files generated by TypeScript

Writing TypeScript Questions

- What if we want to use different JS version?

  - tsc --target ES2015 basic.ts

- What if we want to output the JS files in different folder?

  - tsc --ourDir js basic.ts

NOTE: Run a test with both these options included and note that the newly generated JS file now contains the new ECMAScript syntax for classes instead of the old JavaScript prototype syntax

Writing TypeScript Questions

- Do we have to type all of this configuration options every time?

  - No, this process can be automated by using a tsconfig.json file which overwrites the defaults if not present (such as compiling to ECMAScript version 3 and generating file with the same name in the same folder)

Writing TypeScript Questions

- You can create sample file by using: tsc -- init
- Note inside the target for the ES version and uncomment the outDir (may be change to “./js”) and the noUnusedLocals, also feel free to explore the rest
- From now on you can just type tsc in the command line and it will generate the JS files using the configuration
- It will look for all your .ts files in all of your directories and compile them all into JavaScript files

TypeScript Syntax

Let's create a JavaScript file .js with the following perfectly valid code inside.

After you are done, rename the file to .ts instead and see what happens.

```
let firstName = "Alice";
let age = 72;
let activeMember = true;
firstName = 5;
```

TypeScript Syntax

- If you hover over the variable declarations, you are going to see the TypeScript annotation which TS derives from the code to figure out the type of each variable

```
let activeMember: boolean
```

- Declare a new one: let someVar;

  Note the type here is any which is a valid type in TS if we don’t know what will be the type of the value

TypeScript Syntax

- You can explicitly declare a variable to hold values of a certain type using that same annotation:
  `let firstName: string;`
- Note that if you are initializing with a value, you can skip the annotation and let it be inferred automatically
- However, if you are not providing a value, you'd like to use the annotation to provide the specific type and avoid the default of `any`

TypeScript Syntax

- We can use the annotations when we declare functions

```
function simpleFunction(name: string, isActive: boolean): number {
    // code goes here...
    return 0;
}
```

- This is a function that takes two input parameters, first one must be a string, second one must be a boolean and the return type must be a number

- You can use : void if the function doesn't return a value

TypeScript Syntax

- We already saw example of a class defined in TS
- Inheritance is supported:

  class SpecialCustomer extends Customer { ... }
- Data encapsulation modifiers are supported:

  public, private, protected
- There is also support for getters and setters, readonly modifiers, abstract classes and more

TypeScript Syntax

- We often need to reuse code and JavaScript finally added easier support for including modules in ECMAScript2015
- Before that everything was handled manually or via third party libraries such as CommonJS
- You can specify which type of syntax and annotation you'd like to use in TS in your configuration file by changing the module value in tsconfig.json

TypeScript Syntax

- That eventually allows you to specify that a certain piece of code will be used elsewhere in the application
- In order to achieve that you need to export it first:

```
export class Customer { ... }
```

TypeScript Syntax

- You can then import that elsewhere in your application and reuse it (syntax may differ based on method used):

```
import { Customer } from “./customer”; //customer.js

let myCustomer = new Customer(“Bob”);
...
```

More TypeScript

- Visit the official website:

  https://www.typescriptlang.org/

- Documentation, examples, tutorials, integration with other technologies and a playground

- Another good tutorial site to learn from:

  https://www.typescripttutorial.net/

Any Questions?

Thank You
For Your Time!