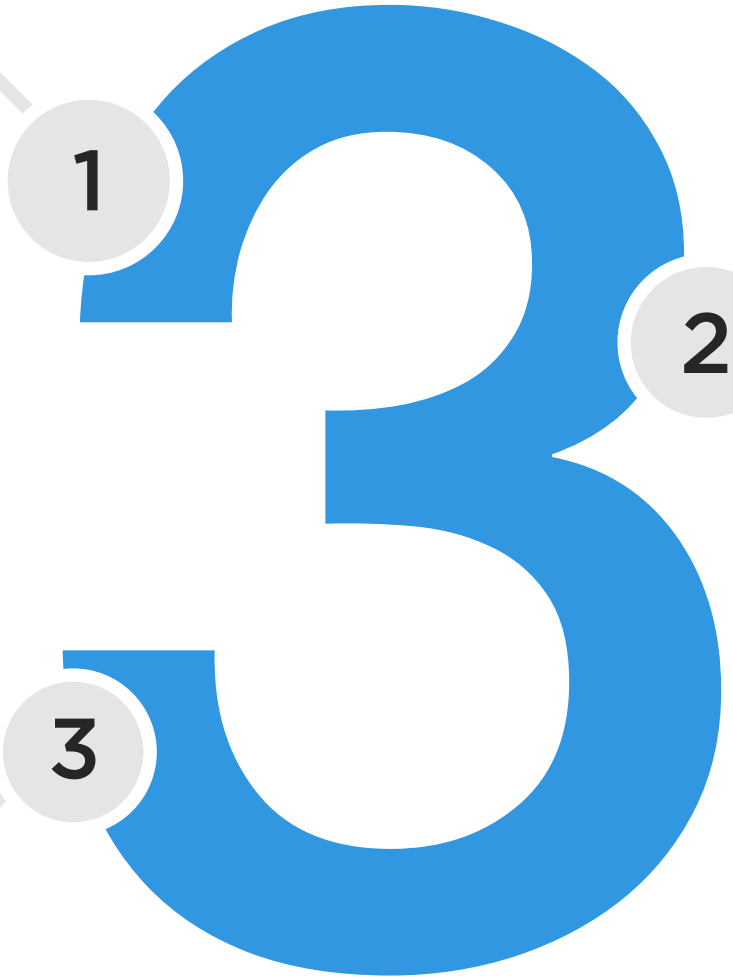
Writing TypeScript Applications

Simon Allardice
STAFF AUTHOR, PLURALSIGHT

@allardice www.pluralsight.com

Benefits of TypeScript



1

Static Types

variables, parameters,
return types, etc.

2

Organization

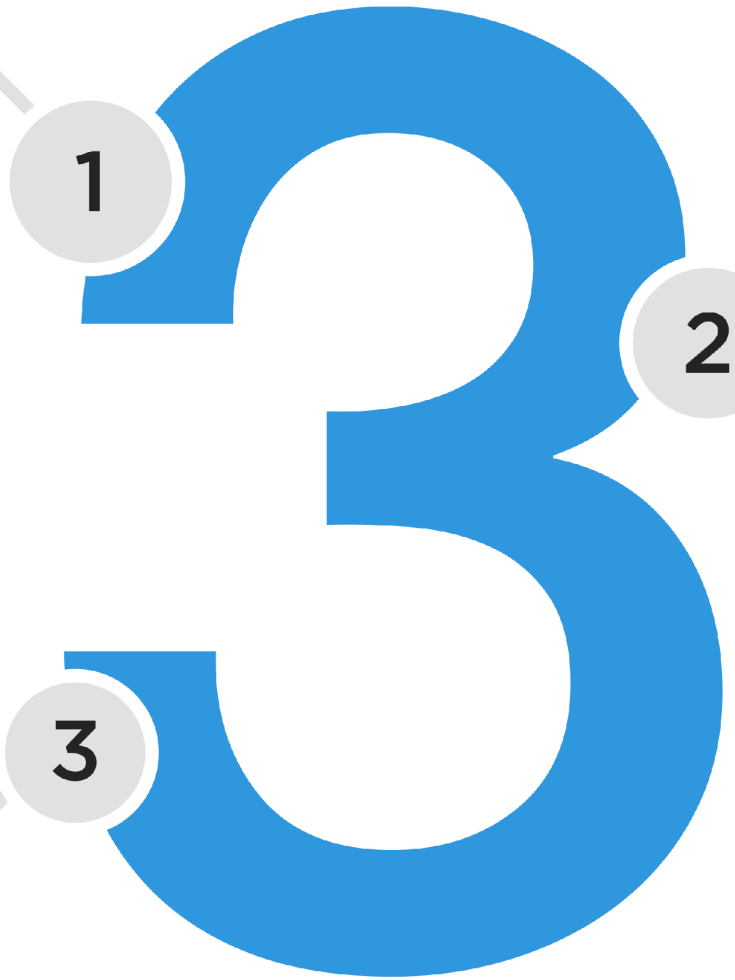
classes, modules,
namespaces, interfaces

3

Tooling

static analysis, instant errors,
detect unused/unreachable code,
source maps, etc.

Benefits of TypeScript



1

Static Types

variables, parameters,
return types, etc.

2

Organization

classes, modules,
namespaces, interfaces

3

Tooling

static analysis, instant errors,
detect unused/unreachable code,
source maps, etc.

Static Types

variables, parameters,
return types, etc.

1

Declaring Variables

variables.js

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

firstName = 1;  // OK in JavaScript
```

variables.ts

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

firstName = 1;  // NOT OK in TypeScript
```

```
Type '123' is not assignable to type 'string'. ts(2322)
```

Type Inference

```
let firstName = "Alice";      // inferred as a string
let age = 72;                 // inferred as a number
let activeMember = true;      // inferred as a boolean

firstName = 123;
```

```
Type '123' is not assignable to type 'string'. ts(2322)
```

Type Information in Functions

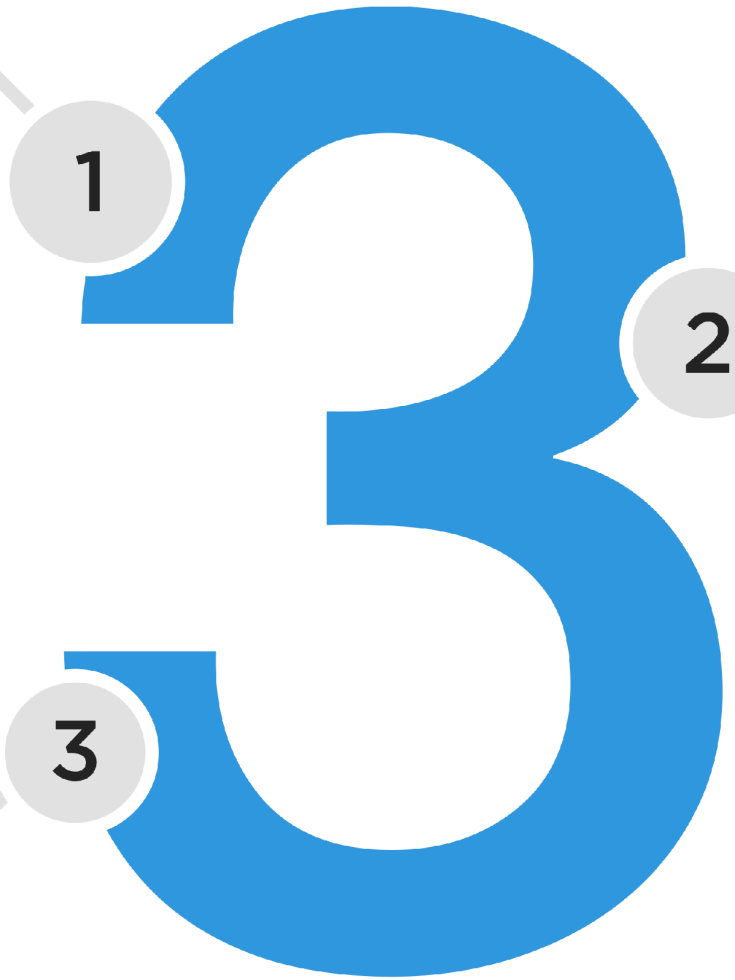
```
// TypeScript style function
function simpleFunction(name: string, isActive: boolean): void {
    // code here...
    return 0; // need to return a number!
}
```

Static Types

variables, parameters,
return types, etc.

1

Benefits of TypeScript



1

Static Types

variables, parameters,
return types, etc.

2

Organization

classes, modules,
namespaces, interfaces

3

Tooling

static analysis, instant errors,
detect unused/unreachable code,
source maps, etc.

2

Organization

classes, modules, namespaces, interfaces

Working with Modules in TypeScript

CommonJS

From Wikipedia, the free encyclopedia

For usage in Wikipedia, see Special:Mypage/common.js, Wikipedia:Common.js and common.css, and MediaWiki:Common.js.

CommonJS was a project with the goal to establish conventions on module ecosystem for JavaScript outside of the web browser. The primary reason of its creation was a major lack of commonly accepted form of JavaScript scripts module units which could be reusable in environments different from that provided by a conventional web browser e.g. web server or native desktop applications which run JavaScript scripts.

CommonJS maintains specifications (including drafts) and a list of implementations on a MediaWiki site. All editing activities have ceased since November 2014, marking the effective end of its life.[1]

Contents [hide]

requirejs.org

CommonJS - Wikipedia | RequireJS | import - JavaScript | MDN

A JAVASCRIPT
MODULE LOADER


```
/* ---

RequireJS is a JavaScript file and module loader.
It is optimized for in-browser use, but it can be
used in other JavaScript environments, like Rhino
and Node. Using a modular script loader like
RequireJS will improve the speed and quality of
your code.

IE 6+ .......... compatible ✔
Firefox 2+ ..... compatible ✔
Safari 3.2+ .... compatible ✔
Chrome 3+ ...... compatible ✔
Opera 10+ ...... compatible ✔

Get started then check out the API.

--- */
```

Latest Release: 2.3.6

import

On this Page

The static **`import`** statement is used to import bindings which are exported by another module. Imported modules are in `strict mode` whether you declare them as such or not. The `import` statement cannot be used in embedded scripts unless such script has a `type="module"`.

There is also a function-like dynamic **`import()`**, which does not require scripts of `type="module"`.

Backward compatibility can be ensured using attribute `nomodule` on the script tag.

Dynamic import is useful in situations where you wish to load a module conditionally, or on-demand. The static form is preferable for loading initial dependencies, and can benefit more readily from static analysis tools and tree shaking.

Related Topics

JavaScript

Tutorials:

- Complete beginners

tsconfig.json — TypeScript Demo

tsconfig.json ●

tsconfig.json > { } compilerOptions

```
1   {
2     "compilerOptions": {
3       /* Basic Options */
4       // "incremental": true,                   /* Enable incremental compilation */
5       "target": "es2015",                          /* Specify ECMAScript target version: 'ES3'
        (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019' or 'ESNEXT'. */
6       "module": "es2015",  /* Specify module code generation: 'none', 'commonjs', 'amd', 'system',
        'umd', 'es2015', or 'ESNext'. */
7       // "lib": [],                             /* Specify library files to be included in the
        compilation. */
8       // "allowJs": true,                       /* Allow javascript files to be compiled. */
9       // "checkJs": true,                       /* Report errors in .js files. */
10      // "jsx": "preserve",                     /* Specify JSX code generation: 'preserve',
        'react-native', or 'react'. */
11      // "declaration": true,                   /* Generates corresponding '.d.ts' file. */
12      // "declarationMap": true,                /* Generates a sourcemap for each corresponding
        '.d.ts' file. */
13      // "sourceMap": true,                     /* Generates corresponding '.map' file. */
14      // "outFile": "./",                       /* Concatenate and emit output to single file. */
15      "outDir": "./js",                        /* Redirect output structure to the directory. */
16      // "rootDir": "./",                       /* Specify the root directory of input files.
        Use to control the output directory structure with --outDir. */
17      // "composite": true,                     /* Enable project compilation */
```

0 ⚠ 0 Ln 16, Col 38 Spaces: 2 UTF-8 LF JSON with Comments Go Live

About
Governance
Community
Working Groups
Releases
Resources
Trademark
Privacy Policy

About Node.js®

Edit on GitHub

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
```

angular.io

About | Node.js | Angular | React – A JavaScript library for building user interfaces

ANGULAR

FEATURES DOCS RESOURCES EVENTS BLOG

Search

One framework.
Mobile & desktop.

GET STARTED

DEVELOP ACROSS ALL PLATFORMS

Learn one way to build applications with Angular and reuse

reactjs.org

About | Node.js | Angular | React – A JavaScript library for building user interfaces

React Docs Tutorial Blog Commun Search v16.12.0 Languages GitHub

React

A JavaScript library for building user interfaces

Get Started Take the Tutorial

Declarative

React makes it painless to create interactive

Component-Based

Build encapsulated components that

Learn Once, Write Anywhere

We don't make assumptions about the rest

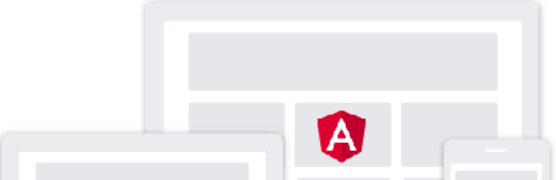
angular.io

About | Node.js | Angular | React – A JavaScript library for building user interfaces

ANGULAR FEATURES DOCS RESOURCES EVENTS BLOG Search

One framework.
Mobile & desktop.

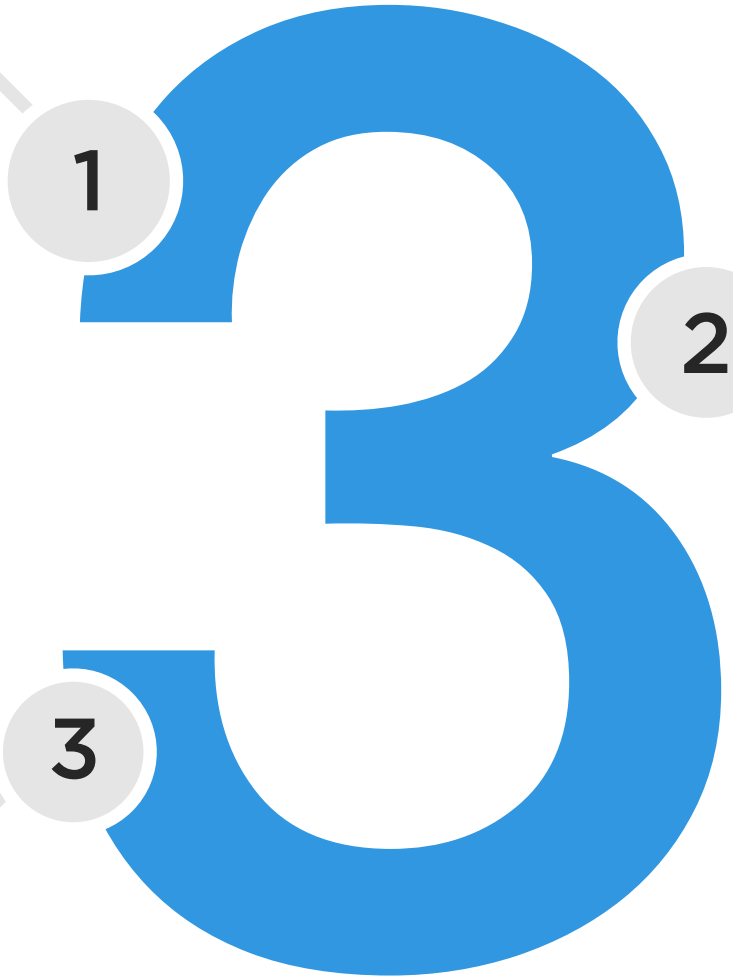
Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS. Wikipedia



DEVELOP ACROSS ALL PLATFORMS

Learn one way to build applications with Angular and reuse

Benefits of TypeScript



1

Static Types

variables, parameters,
return types, etc.

2

Organization

classes, modules,
namespaces, interfaces

3

Tooling

static analysis, instant errors,
detect unused/unreachable code,
source maps, etc.