TypeScript

Lab Manual

https://www.accelebrate.com
(877) 849-1850 ❖ info@accelebrate.com

Customized Technical Training

Accelebrate
ACCELERATED LEARNING, CELEBRATED RESULTS®

On-Site, Customized Private Training

Don't settle for a one-size-fits-all class! Let Accelebrate tailor a private class to your group's goals and experience. Classes can be delivered at your site or online (or a combination of both) worldwide. Visit our web site at **https://www.accelebrate.com** and contact us at **sales@accelebrate.com** for details.

Public Online Training

Need to train just 1-3 people? Attend one of our regularly scheduled, live, instructor-led public online classes. Class sizes are small (typically 3-6 attendees) and you receive just as much hands-on time and individual attention from your instructor as our private classes. For course dates, times, outlines, pricing, and registration, visit **https://www.accelebrate.com/public-training-schedule.**

Newsletter

Want to find out about our latest class offerings? Subscribe to our newsletter **https://www.accelebrate.com/newsletter**.

Blog

Get insights and tutorials from our instructors and staff! Visit our blog, **https://www.accelebrate.com/blog** and join the discussion threads and get feedback from our instructors!

Learning Resources

Get access to learning guides, tutorials, and past issues of our newsletter at the Accelebrate library, **https://www.accelebrate.com/library**.

Call us for a training quote!

877 849 1850

Accelebrate, Inc. was founded in 2002 with the goal of delivering **private training** that rapidly achieves participants' goals. Each year, our experienced instructors deliver hundreds of classes online and at client sites all over the US, Canada, and abroad. We pride ourselves on our **instructors' real-world experience** and ability to **adapt the training** to your team and their objectives. We offer a wide range of topics, including:

- Angular, React, and Vue
- JavaScript
- Data Science using R, Python, & Julia
- Tableau & Power BI
- .NET & VBA programming
- SharePoint & Office 365
- DevOps
- iOS & Android Development
- PostgreSQL, Oracle, and SQL Server
- Java, Spring, and Groovy
- Agile
- Web/Application Server Admin
- HTML5 & Mobile Web Development
- Ansible & Chef
- AWS & Azure
- Adobe & Articulate software
- Docker, Kubernetes, Ansible, & Git
- IT leadership & Project Management
- AND MORE *(see back)*

"I have participated in several Accelebrate training courses and I can say that the instructors have always proven very knowledgeable and the materials are always very thorough and usable well after the class is over."

— Rick, AT&T

Like us on Facebook + Follow us on Twitter + Watch us on YouTube

Visit our website for a complete list of courses!

Adobe & Articulate
Adobe Captivate
Adobe Creative Cloud
Adobe Presenter
Articulate Storyline / Studio
Camtasia
RoboHelp

AWS, Azure, & Cloud
AWS
Azure
Cloud Computing
Google Cloud
OpenStack

Big Data
Actian Matrix Architecture
Alteryx
Apache Spark
Greenplum Architecture
Kognitio Architecture
Teradata
Snowflake SQL

Data Science and RPA
Blue Prism
Django
Julia
Machine Learning
Python
R Programming
Tableau
UiPath

Database & Reporting
BusinessObjects
Crystal Reports
Excel Power Query
MongoDB
MySQL
NoSQL Databases
Oracle
Oracle APEX
Power BI
PivotTable and PowerPivot
PostgreSQL
SQL Server
Vertica Architecture & SQL

DevOps, CI/CD & Agile
Agile
Ansible
Chef
Docker
Git
Gradle Build System
Jenkins
Jira & Confluence
Kubernetes
Linux
Microservices
Red Hat
Software Design

Java
Apache Maven
Apache Tomcat
Groovy and Grails
Hibernate
Java & Web App Security
JavaFX
JBoss
Oracle WebLogic
Scala
Selenium & Cucumber
Spring Boot
Spring Framework

JS, HTML5, & Mobile
Angular
Apache Cordova
CSS
D3.js
HTML5
iOS/Swift Development
JavaScript
MEAN Stack
Mobile Web Development
Node.js & Express
React & Redux
Swift
Vue

Microsoft & .NET
.NET Core
ASP.NET
Azure DevOps
C#
Design Patterns
Entity Framework Core
F#
IIS
Microsoft Dynamics CRM
Microsoft Exchange Server
Microsoft Office 365
Microsoft Power Platform
Microsoft Project
Microsoft SQL Server
Microsoft System Center
Microsoft Windows Server
PowerPivot
PowerShell
Team Foundation Server
VBA
Visual C++/CLI
Web API

Other
Blockchain
C++
Go Programming
IT Leadership
ITIL
Project Management
Regular Expressions
Ruby on Rails
Rust
Salesforce
XML

Security
.NET Web App Security
C and C++ Secure Coding
C# & Web App Security
Linux Security Admin
Python Security
Secure Coding for Web Dev
Spring Security

SharePoint
Power Automate & Flow
SharePoint Administrator
SharePoint Developer
SharePoint End User
SharePoint Online
SharePoint Site Owner

SQL Server
Azure SQL Data Warehouse
Business Intelligence
Performance Tuning
SQL Server Administration
SQL Server Development
SSAS, SSIS, SSRS
Transact-SQL

Teleconferencing Tools
Adobe Connect
GoToMeeting
Microsoft Teams
WebEx
Zoom

Web/Application Server
Apache Tomcat
Apache httpd
IIS
JBoss
Nginx
Oracle WebLogic

Visit ***www.accelebrate.com/newsletter*** *to sign up and receive our* **newsletters** *with information about* **new courses**, **free webinars**, **tutorials**, *and* **blog articles**.

Call us for a training quote! 877 849 1850 (US/Canada) or +1 678 648 3133

TypeScript

Lab Manual

Create Project

1. Make a directory named `typescriptdemo`.
2. Open the directory `typescriptdemo` in your preferred editor.

Install TypeScript

1. Open a command-prompt or terminal in the `typescriptdemo` directory.

2. Run the following command to initialize the directory for npm packages (JavaScript libraries).

```
npm init -y
```

3. Install TypeScript by running the following command:

```
npm install typescript@4.3.5 --save-dev
```

Run TypeScript

1. Open a `command-prompt` or `terminal`.
2. Set the `current` directory to `typescriptdemo`.
3. Run the command:

```
npx tsc --init
```

- This creates a `tsconfig.json` file with the default commmand line options.
- Documentation for all TypeScript *compiler options are available here*.

About Compiler Strictness

1. Open `tsconfig.json` and notice the `strict` setting is `true`.

```
/* Strict Type-Checking Options */
"strict": true
...
```

2. Below are all the settings that are set by the `strict` flag.

--strict

> Enabling --strict enables --noImplicitAny, --noImplicitThis, --alwaysStrict, --strictBindCallApply, --strictNullChecks, --strictFunctionTypes and --strictPropertyInitialization.

> Strict null checks (--strictNullChecks) are a new concept to most developers. In strict null checking mode, the null and undefined values are not in the domain of every type and are only assignable to themselves and `any` (the one exception being that `undefined` is also assignable to `void`). Here is a some example code to demonstrate what this looks like although we will see many examples throughout the course.

```
errorMessage: string | undefined | null | "";
```

Run the Compiler

1. Create file `program.ts`
2. Add the following code:

```
function greeter(name: string) {
  console.log("Hi " + name);
}
greeter("Venkat");
```

> The colon after name is called a type annotation and ensures that any variable passed to name is of the type `string`. We will see what happens if we don't pass a string in the next section.

3. While at the command-prompt or terminal and in the `typescriptdemo` directory run the command:

```
npx tsc --watch
```

4. Open *another* (a new) command-prompt or terminal in the `typescriptdemo` directory; leaving the typescript compiler running in watch mode in the other terminal (`tsc --watch`).

   > In VS Code: Click the `+` or the split terminal icon

5. Run the command:

   ```
   node program.js
   ```

   > Verify that you DID NOT run the TypeScript file `node program.ts` instead of `node program.js`.

6. Result:

   ```
   Hi Venkat
   ```

Type Annotations

1. Code:

```
function greeter(name: string) {
  console.log("Hi " + name);
}
greeter(1);
```

2. Result (you may need to switch to another terminal to see this message):

```
program.ts(4,9): error TS2345: Argument of type '1' is not assignable to
parameter of type 'string'.
```

Output on Error

1. If you run the program while the error still exists you will still get output.

```
node program.js
```

2. Results: `Hi 1`

   This is because the default setting for the compiler option `--noEmitOnError` is false so output is emitted even if errors were reported.

   > For more information on why this is the default behavior see: https://github.com/Microsoft/TypeScript/issues/828

3. To fix this we would need to need to:

- Edit `tsconfig.json` and add the `noEmitOnError` setting.

```
{
"compileOnSave": false,
"compilerOptions": {
```

```
    "strict": true,
    "baseUrl": "./",
    "downlevelIteration": true,
    "importHelpers": true,
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "module": "es2020",
    "moduleResolution": "node",
    "experimentalDecorators": true,
    "target": "es2015",
    "typeRoots": ["node_modules/@types"],
    "lib": ["es2017", "dom"]
+ ,
+ "noEmitOnError": true
  },
  "angularCompilerOptions": {
    "strictInjectionParameters": true,
    "strictInputAccessModifiers": true,
    "strictTemplates": true
  }
  }
```

- Stop the compiler `Ctrl+C` (make sure you are in the `tsc --watch` terminal instance)
- Delete the `program.js` file
- Restart the compiler

```
npx tsc --watch
```

- Verify that no `program.js` was created (emitted).

Classes

Fields

1. Code:

```
class Person {
  first: string = "";
  last: string = "";
}

let person = new Person();
person.first = "Kanye";
person.last = "West";

console.log(person.first + " " + person.last);
```

2. Result:

```
Kanye West
```

> Class field declarations for JavaScript https://github.com/tc39/proposal-class-fields

Constructors

1. Code:

```
class Person {
  first: string;
  last: string;

  constructor(first: string, last: string) {
    this.first = first;
    this.last = last;
  }
}

let person = new Person("Kanye", "West");
console.log(person.first + " " + person.last);
```

2. Result:

```
Kanye West
```

Parameter Properties

1. Code:

```
class Person {

    constructor(public first: string,public last: string) {
    }
}

let person = new Person("Kanye", "West");
console.log(person.first + ' ' + person.last);
```

2. Result:

```
Sean Carter
```

Methods

1. Code:

```
class Person {
    constructor(public first: string, public last: string) {
    }

    getFullName() {
        return this.first + ' ' + this.last
    }
}

let person = new Person("Kanye", "West");
console.log(person.getFullName());
```

2. Result:

```
John Doe
```

Scope (var, let, const)

var

1. Code

```
var numbers = [1, 2, 3, 4];

for (var counter = 0; counter < numbers.length; counter++) {
  console.log(numbers[counter]);
}

console.log("at end: " + counter);
```

2. Result

```
1
2
3
4
at end: 4
```

let

1. Code

```
let numbers = [1, 2, 3, 4];

for (let counter = 0; counter < numbers.length; counter++) {
  console.log(numbers[counter]);
}

console.log("at end: " + counter);
```

2. Result

```
program.ts(7,26): error TS2304: Cannot find name 'counter'.
```

const

1. Code

```
const a = 1;
a = 2;
```

2. Result

```
error TS2540: Cannot assign to 'a' because it is a constant.
```

Arrow Functions

1. Code

Function

```
let numbers = [1, 2, 3, 4];
//verbose
numbers.forEach(function (n) {
  console.log(n);
});
```

1. Result

```
1
2
3
4
```

Arrow function

1. Code

```
let numbers = [1, 2, 3, 4];
numbers.forEach(n => console.log(n));
```

2. Result

```
1
2
3
4
```

Modules

First Module

1. Create file `my-module.ts`
2. Add the following code to `my-module.ts`

```
export function myFunction() {
  return "myFunction was run.";
}
```

3. Code in `program.ts`

- Show how editor can auto import module

```
import { myFunction } from "./my-module";
console.log(myFunction());
```

4. Result

```
myFunction was run.
```

Another Module

1. Code in `my-module.ts`

```
//my-module.ts
export function myFunction() {
  return "myFunction was run.";
}

function myPrivateFunction() {
  return "myPrivateFunction was run.";
}

let myObject = {
  name: "I can access myObject's name",
  myMethod: function () {
    return "myMethod on myObject is running.";
  }
};

export { myObject };

export const myPrimitive = 55;

export class MyClass {
  myClassMethod() {
    return "myClassMethod on myClass is running.";
  }
}
```

2. Code in `program.ts`

```
import { myFunction, myObject, myPrimitive, MyClass } from "./my-
module";

console.log(myFunction());

console.log(myObject.name);
console.log(myObject.myMethod());

console.log(myPrimitive);

let myClass = new MyClass();
console.log(myClass.myClassMethod());
```

3. Result

```
myFunction was run.
I can access myObject's name
myMethod on myObject is running.
55
myClassMethod on myClass is running.
```

- Show what happens if you try to import myPrivateFunction

Template Literals

1. Code

```
let verb = "ate";
let noun = "food";
let sentence = `I ${verb} ${noun}.
I enjoyed it.`;
console.log(sentence);
```

2. Result

```
I ate food.
I enjoyed it.
```

Default, Rest, Spread

Default

1. Code

```
function add(x: number, y: number = 2) {
  return x + y;
}

console.log(add(1, 1) === 2);
console.log(add(1) === 3);
```

2. Result

```
true
```

Rest

1. Code

```
function print( ...theArguments: any[]) {
  for (let argument of theArguments) {
    console.log(argument);
  }
}

print("a", "b", "c", "d");
```

2. Result

```
a
b
c
d
```

Spread

1. Code

```
let person = {
  first: "Thomas",
  last: "Edison",
  age: 5,
  twitter: "@tom"
};

let copyOfPerson = {  ...person, gender: "Male" };

console.log(copyOfPerson);
```

2. Result

```
{
  first: 'Thomas',
  last: 'Edison',
  age: 5,
  twitter: '@tom',
  gender: 'Male'
}
```

Optional Parameters

1. Code

```
function greeter(name?: string) {
  if (!name) return "Hi....You";
  return `Hi ${name}`;
}

console.log(greeter());
console.log(greeter("Josh"));
```

2. Result

```
Hi....You
Hi Josh
```

Destructuring

Objects

1. Code

```
let person = {
  first: "Thomas",
  last: "Edison",
  age: 5,
  twitter: "@tom"
};

let { first, last } = person;
console.log(first);
console.log(last);
```

2. Result

```
Thomas
Edison
```

Assignment is left to right with an object literal.

1. Code

```
let person = {
  first: "Thomas",
  last: "Edison",
  age: 5,
  twitter: "@tom"
};

let { first: firstName, last: lastName } = person;
console.log(firstName);
console.log(lastName);
```

2. Result

```
Thomas
Edison
```

Arrays

1. Code

```
let numbers = [1, 2, 3];

let [a, b, c] = numbers;
console.log(a);
console.log(b);
console.log(c);
```

2. Result

```
1
2
3
```

If you don't need an item just skip that item in the assignment.

1. Code

```
let numbers = [1, 2, 3];

let [, b, c] = numbers;
// console.log(a);
console.log(b);
console.log(c);
```

2. Result

```
2
3
```

Optional Parameters

1. Code

```
function buildName(first: string, last: string, middle?: string) {
  if (middle) {
    return `${first} ${middle} ${last}`;
  } else {
    return `${first} ${last}`;
  }
}

console.log(buildName("Craig", "McKeachie"));
console.log(buildName("Craig", "McKeachie", "D."));
```

2. Result

```
Craig McKeachie
Craig D. McKeachie
```

Resources

TypeScript

- TypeScript Deep Dive
- TypeScript Handbook
- TypeScript Compiler Options
- ECMAScript Compatibility Chart

npm

- Configuration
- Semantic Versioning