TypeScript

Lab Manual

Create Project

```
mkdir typescriptdemo
cd typescriptdemo
code . //opens Visual Studio Code
```

Install TypeScript & RxJS

In a command-prompt or terminal

```
npm init -y
npm install typescript@2.7.2 --save-dev
npm install rxjs@6.2.0 --save
```

Run TypeScript

1. Open `package.json`
2. Add the following `npm script` to run the TypeScript compiler (`tsc`).

```
"scripts": {
    "tsc": "tsc"
  }
```

You can replace the existing `test` script.

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

```
npm run-script tsc -- --init
```

- This creates a `tsconfig.json` file with the default commmand line options.
- Documentation for all TypeScript compiler options is available here.
- The double-hypen `--` forwards command-line arguments to the underlying program (`tsc` in this case).

6. Open `tsconfig.json` and change the `strict` setting from `true` to `false`

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

7. `npm run tsc -- -w`
8. Create file `program.ts`
9. Code:

```
function greeter(name) {
  console.log('Hi ' + name);
}
greeter('Craig');
```

10. Open another command-**prompt** or terminal in the `typescriptdemo` directory.

- In VS Code: View> Integrated Terminal

11. Run the command: `node program.js`
12. Result:

```
Hi Craig
```

Type Annotations

1. Code:

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

2. Result:

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

Classes

Properties

1. Code:

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

let person = new Person();
person.first = 'Craig';
person.last = 'McKeachie';

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

2. Result:

```
Craig McKeachie
```

Constructors

1. Code:

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

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

let person = new Person('Craig', 'McKeachie');
console.log(person.first + ' ' + person.last);
```

2. Result:

```
Craig McKeachie
```

Auto Properties

1. Code:

```
class Person {

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

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

2. Result:

```
Craig McKeachie
```

Methods

1. Code:

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

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

let person = new Person("John", "Doe");
console.log(person.getFullName());
```

1. 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 or a read-only
property.
```

Arrow Functions

1. Code

Function

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

2. 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, y = 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

```
function add(x, y, z) {
  return x + y + z;
}

// Pass each elem of array as argument
console.log(add( ...[1, 2, 3]));
```

2. Result

```
program.ts(6,13): error TS2556: Expected 3 arguments, but got 0 or more.
```

Change the code to make parameters optional using ?

```
function add(x?, y?, z?) {
  return x + y + z;
}
```

```
6
```

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 opposite than 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
```

Object.assign()

1. Code

```
let o1 = { a: 1, b: 1, c: 1 };
let o2 = { b: 2, c: 2 };
let o3 = { c: 3 };

let obj = Object.assign({}, o1, o2, o3);
console.log(obj);
```

2. Result Initially you will receive the compiler error:

```
Property 'assign' does not exist on type 'ObjectConstructor'.
```

3. Open `tsconfig.json` and uncomment the `lib` setting and add the following values.

```
"lib": [
  "es2015",
  "dom"
]
```

4. The compiler error will go away but may require closing the editor and then opening it again.

5. Run the program.

```
program node.js
```

6. Result.

```
{ a: 1, b: 2, c: 3 }
```