Section : Setting Up TS

app.ts

```
console.log("It works!");
```

Index. Html

```
<!doctype html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport"
        content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
   <meta http-equiv="X-UA-Compatible" content="ie=edge">
   <title>Learning TypeScript</title>
   <script src="app.js"></script>
</head>
<body>

</body>
</html>
```

Section : Types

app.js

```
// string
let myName: string = 'Max';
// myName = 28;

// number
let myAge: number = 27;
// myAge = 'Max';

// boolean
let hasHobbies: boolean = false;
// hasHobbies = 1;

// assign types
let myRealAge: number;
myRealAge = 27;
// myRealAge = '27';

// array
let hobbies: any[] = ["Cooking", "Sports"];
hobbies = [100];
// hobbies = 100;

// tuples
let address: [string, number] = ["Superstreet", 99];

// enum
enum Color {
   Gray, // 0
   Green = 100, // 100
   Blue = 2// 2
```

```
}
let myColor: Color = Color.Blue;
console.log(myColor);

// any
let car: any = "BMW";
console.log(car);
car = { brand: "BMW", series: 3};
console.log(car);

// functions
function returnMyName(): string {
   return myName;
}
console.log(returnMyName());

// void
function sayHello(): void {
   console.log("Hello!");
}

// argument types
function multiply(value1: number, value2: number): number {
   return value1 * value2;
}
// console.log(multiply(2, 'Max'));
console.log(multiply(10, 2));

// function types
let myMultiply: (a: number, b: number) => number;
// myMultiply = sayHello;
// myMultiply();
myMultiply = multiply;
console.log(myMultiply(5, 2));

// objects
```

```
let userData: { name: string, age: number } = {
   name: "Max",
   age: 27
};
// userData = {
//     a: "Hello",
//     b: 22
// };

// complex object
let complex: {data: number[], output: (all: boolean) => number[]} = {
   data: [100, 3.99, 10],

   output: function (all: boolean): number[] {
      return this.data;
   }
};
// complex = {};

// type alias

type Complex = {data: number[], output: (all: boolean) => number[]};

let complex2: Complex = {
   data: [100, 3.99, 10],

   output: function (all: boolean): number[] {
      return this.data;
   }
};

// union types
let myRealRealAge: number | string = 27;
myRealRealAge = "27";
// myRealRealAge = true;
```

```
// check types
let finalValue = 30;
if (typeof finalValue == "number") {
    console.log("Final value is a number");
}
```

Section : Classes

App.js

```
class Person {
    name: string;
    private type: string;
    protected age: number = 27;

    constructor(name: string, public username: string) {
        this.name = name;
    }

    printAge() {
        console.log(this.age);
        this.setType("Old Guy");
    }

    private setType(type: string) {
        this.type = type;
        console.log(this.type);
    }
}

const person = new Person("Max", "max");
console.log(person.name, person.username);
person.printAge();
// person.setType("Cool guy"); // Won't work with private method

// Inheritance
class Max extends Person {
    // name = "Max";

    constructor(username: string) {
```

```
        super("Max", username);
        this.age = 31;
    }
}
const max = new Max("max");
console.log(max);

// Getters & Setters
class Plant {
    private _species: string = "Default";

    get species() {
        return this._species;
    }

    set species(value: string) {
        if (value.length > 3) {
            this._species = value;
        } else {
            this._species = "Default";
        }
    }
}

let plant = new Plant();
console.log(plant.species);
plant.species = "AB";
console.log(plant.species);
plant.species = "Green Plant";
console.log(plant.species);

// Static Properties & Methods
class Helpers {
    static PI: number = 3.14;
    static calcCircumference(diameter: number): number {
        return this.PI * diameter;
```

```
    }
}
console.log(2 * Helpers.PI);
console.log(Helpers.calcCircumference(8));

// Abstract Classes
abstract class Project {
    projectName: string = "Default";
    budget: number = 1000;

    abstract changeName(name: string): void;

    calcBudget() {
        return this.budget * 2;
    }
}

class ITProject extends Project {
    changeName(name: string): void {
        this.projectName = name;
    }
}

let newProject = new ITProject();
console.log(newProject);
newProject.changeName("Super IT Project");
console.log(newProject);
```

Section : Interfaces

App.js

```
interface NamedPerson {
    firstName: string;
    age?: number;
    [propName: string]: any;
    greet(lastName: string): void;
}

function greet(person: NamedPerson) {
    console.log("Hello, " + person.firstName);
}

function changeName(person: NamedPerson) {
    person.firstName = "Anna";
}

const person: NamedPerson = {
    firstName: "Max",
    hobbies: ["Cooking", "Sports"],
    greet(lastName: string) {
        console.log("Hi, I am " + this.firstName + " " + lastNam
e);
    }
};

// greet({firstName: "Max", age: 27});
changeName(person);
greet(person);
person.greet("Anything");

class Person implements NamedPerson {
```

```
    firstName: string;
    lastName: string;

    greet(lastName: string) {
        console.log("Hi, I am " + this.firstName + " " + lastNam
e);
    };
}

const myPerson = new Person();
myPerson.firstName = "Maximilian";
myPerson.lastName = "Anything";
greet(myPerson);
myPerson.greet(myPerson.lastName);


// Function Types


interface DoubleValueFunc {
    (number1: number, number2: number): number;
}

let myDoubleFunction: DoubleValueFunc;
myDoubleFunction = function (value1: number, value2: number) {
    return (value1 + value2) * 2;
};

console.log(myDoubleFunction(10, 20));


// Interface Inheritance

interface AgedPerson extends NamedPerson {
    age: number;
}
```

```
const oldPerson: AgedPerson = {
    age: 27,
    firstName: "Max",
    greet(lastName: string) {
        console.log("Hello!");
    }
};

console.log(oldPerson);
```

Section : Generics

App.js

```
// Simple Generic
function echo(data: any) {
    return data;
}

console.log(echo("Max"));
console.log(echo(27));
console.log(echo({name: "Max", age: 27}));

// Better Generic
function betterEcho<T>(data: T) {
    return data;
}
console.log(betterEcho("Max").length);
console.log(betterEcho<number>(27));
console.log(betterEcho({name: "Max", age: 27}));

// Built-in Generics
const testResults: Array<number> = [1.94, 2.33];
testResults.push(-2.99);
console.log(testResults);

// Arrays
function printAll<T>(args: T[]) {
    args.forEach((element) => console.log(element));
}
printAll<string>(["Apple", "Banana"]);

// Generic Types
const echo2: <T>(data: T) => T = betterEcho;
```

```
console.log(echo2<string>("Something"));

// Generic Class
class SimpleMath<T extends number | string, U extends number |
 string> {
    baseValue: T;
    multiplyValue: U;
    calculate(): number {
        return +this.baseValue * +this.multiplyValue;
    }
}

const simpleMath = new SimpleMath<string, number>();
simpleMath.baseValue = "10";
simpleMath.multiplyValue = 20;
console.log(simpleMath.calculate());
```