Exercise 1: Basic TypeScript Setup

Objective: Set up a TypeScript environment and compile a basic TypeScript file.

Instructions:

1. Install TypeScript globally using npm.
2. Create a new file named `hello.ts`.
3. Write a TypeScript program that prints "Hello, TypeScript!" to the console.
4. Compile the TypeScript file to JavaScript.

Solution:

```sh
npm install -g typescript
```

Create `hello.ts`:

```typescript
console.log("Hello, TypeScript!");
```

Compile:

```sh
tsc hello.ts
```

Run the compiled JavaScript:

```sh
node hello.js
```

Exercise 2: Basic Data Types

Objective: Use different basic data types in TypeScript.

Instructions:

1. Create a new file named `dataTypes.ts`.
2. Declare variables of types `string`, `number`, `boolean`, `null`, and `undefined`.
3. Print each variable to the console.

Solution:

```typescript
let myString: string = "Hello";
let myNumber: number = 42;
let myBoolean: boolean = true;
let myNull: null = null;
let myUndefined: undefined = undefined;

console.log(myString, myNumber, myBoolean, myNull, myUndefined);
```

Exercise 3: Arrays

Objective: Create and manipulate arrays in TypeScript.

Instructions:

1. Create a new file named `arrays.ts`.
2. Declare an array of numbers.

3. Add and remove elements from the array.
4. Print the array before and after modification.

Solution:
```typescript
let numbers: number[] = [1, 2, 3, 4, 5];
console.log("Before:", numbers);

numbers.push(6);
numbers.pop();
console.log("After:", numbers);
```

Exercise 4: Tuples

Objective: Use tuples in TypeScript.

Instructions:
1. Create a new file named `tuples.ts`.
2. Declare a tuple that holds a string and a number.
3. Access and print the elements of the tuple.

Solution:

```typescript
let myTuple: [string, number] = ["TypeScript", 2023];
console.log("Tuple:", myTuple[0], myTuple[1]);
```

Exercise 5: Enums

Objective: Define and use enums in TypeScript.

Instructions:

1. Create a new file named `enums.ts`.
2. Define an enum representing the days of the week.
3. Write a function that takes an enum value and prints the corresponding day.

Solution:

```typescript
enum DaysOfWeek {
```

```
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}

function printDay(day: DaysOfWeek): void {
    console.log(DaysOfWeek[day]);
}

printDay(DaysOfWeek.Monday);
```

Exercise 6: Any and Void

Objective: Use `any` and `void` types in TypeScript.

Instructions:

1. Create a new file named `anyAndVoid.ts`.

2. Declare a variable of type `any` and assign it different types of values.

3. Write a function with a `void` return type that prints a message to the console.

Solution:

```typescript
let anything: any = 42;
anything = "Now I'm a string";
console.log(anything);

function logMessage(): void {
    console.log("This function returns void");
}

logMessage();
```

Exercise 7: Null and Undefined

Objective: Use `null` and `undefined` in TypeScript.

Instructions:

1. Create a new file named `nullUndefined.ts`.
2. Declare variables of type `null` and `undefined`.
3. Write a function that checks if a variable is `null` or `undefined`.

Solution:

```typescript
let nothing: null = null;
let notAssigned: undefined = undefined;

function checkNullUndefined(value: any): void {
    if (value === null) {
        console.log("Value is null");
    } else if (value === undefined) {
        console.log("Value is undefined");
    } else {
```

```
        console.log("Value is neither null nor undefined");
    }
}

checkNullUndefined(nothing);
checkNullUndefined(notAssigned);
```

Exercise 8: Type Inference

Objective: Understand type inference in TypeScript.

Instructions:

1. Create a new file named `typeInference.ts`.
2. Declare variables without specifying their types and assign values to them.
3. Print the types of these variables.

Solution:

```typescript
let inferredString = "TypeScript";
let inferredNumber = 2023;
let inferredBoolean = true;

console.log(typeof inferredString, typeof inferredNumber, typeof inferredBoolean);
```

Exercise 9: Type Casting

Objective: Perform type casting in TypeScript.

Instructions:

1. Create a new file named `typeCasting.ts`.
2. Declare a variable of type `any` and assign it a string value.
3. Cast the variable to a `string` and print its length.

Solution:

```typescript
```

```
let someValue: any = "Hello, TypeScript";
let strLength: number = (<string>someValue).length;
console.log("String length:", strLength);
```

Exercise 10: Let and Var

Objective: Understand the difference between `let` and `var`.

Instructions:

1. Create a new file named `letVar.ts`.
2. Declare a variable using `var` inside a function and another using `let` inside a block.
3. Print the variables inside and outside their scopes.

Solution:

```typescript
function testVarLet() {
    if (true) {
        var varVariable = "I'm a var";
```

```
    let letVariable = "I'm a let";
    console.log("Inside block:", varVariable, letVariable);
  }
  console.log("Outside block:", varVariable);
  // console.log("Outside block:", letVariable); // This will cause an error
}

testVarLet();
```

Exercise 11: Const Declaration

Objective: Use `const` in TypeScript.

Instructions:

1. Create a new file named `constDeclaration.ts`.
2. Declare a constant object and modify its properties.
3. Attempt to reassign the constant object.

Solution:

```typescript
const person = { name: "Alice", age: 25 };
person.age = 26;
console.log("Modified object:", person);

// person = { name: "Bob", age: 30 }; // This will cause an error
```

Exercise 12: Classes and Methods

Objective: Define and use classes and methods in TypeScript.

Instructions:

1. Create a new file named `classes.ts`.
2. Define a class with properties and methods.
3. Create an instance of the class and call its methods.

Solution:

```typescript
class Person {
    name: string;
    age: number;

    constructor(name: string, age: number) {
        this.name = name;
        this.age = age;
    }

    greet() {
        console.log(`Hello, my name is ${this.name}`);
    }
}

let alice = new Person("Alice", 25);
alice.greet();
```

Exercise 13: Inheritance

Objective: Implement class inheritance in TypeScript.

Instructions:

1. Create a new file named `inheritance.ts`.
2. Define a base class and a derived class.
3. Create an instance of the derived class and call its methods.

Solution:

```typescript
class Animal {
    name: string;

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

    move(distance: number) {
```

```
        console.log(`${this.name} moved ${distance} meters.`);
    }
}

class Dog extends Animal {
    bark() {
        console.log("Woof! Woof!");
    }
}

let dog = new Dog("Buddy");
dog.bark();
dog.move(10);
```

Exercise 14: Abstract Classes

Objective: Use abstract classes in TypeScript.

Instructions:

1. Create a new file named `abstractClasses.ts`.
2. Define an abstract class and a concrete subclass.
3. Create an instance of the subclass and call its methods.

Solution:

```typescript
abstract class Shape {
    abstract area(): number;
}

class Circle extends Shape {
    radius: number;

    constructor(radius: number) {
        super();
        this.radius = radius;
    }

    area(): number {
        return Math.PI * this.radius * this.radius;
```

```
    }
}

let circle = new Circle(5);
console.log("Area of circle:", circle.area());
```

Exercise 15: Interfaces

Objective: Define and implement interfaces in TypeScript.

Instructions:
1. Create a new file named `interfaces.ts`.
2. Define an interface with properties and methods.
3. Implement the interface in a class.

Solution:
```typescript
interface ICar {
    make: string;
```

```
    model: string;
    drive(): void;
}

class Car implements ICar {
    make: string;
    model: string;

    constructor(make: string, model: string) {
        this.make = make;
        this.model = model;
    }

    drive() {
        console.log(`Driving a ${this.make}
${this.model}`);
    }
}

let car = new Car("Toyota", "Corolla");
car.drive();
```

```

### Exercise 16: Type Assertion

**Objective:** Perform type assertion in TypeScript.

**Instructions:**

1. Create a new file

 named `typeAssertion.ts`.

2. Declare a variable of type `any` and assign it a number value.

3. Use type assertion to treat the variable as a `number` and perform arithmetic operations.

**Solution:**

```typescript
let someValue: any = 42;

let numValue: number = someValue as number;

console.log("Double the value:", numValue * 2);
```
```

Exercise 17: Union Types

Objective: Use union types in TypeScript.

Instructions:

1. Create a new file named `unionTypes.ts`.
2. Declare a variable that can hold a `string` or a `number`.
3. Write a function that accepts the variable and prints its type and value.

Solution:

```typescript
let value: string | number;
value = "Hello";
console.log("Value:", value);

value = 42;
console.log("Value:", value);
```

```
function printValue(val: string | number) {
    if (typeof val === "string") {
        console.log("String:", val);
    } else {
        console.log("Number:", val);
    }
}

printValue(value);
```

Exercise 18: Generics

Objective: Use generics in TypeScript.

Instructions:

1. Create a new file named `generics.ts`.
2. Write a generic function that returns the passed argument.
3. Call the function with different types of arguments.

Solution:

```typescript
function identity<T>(arg: T): T {
    return arg;
}

console.log(identity<string>("Hello"));
console.log(identity<number>(42));
console.log(identity<boolean>(true));
```

Exercise 19: Readonly Properties

Objective: Use readonly properties in TypeScript.

Instructions:

1. Create a new file named `readonlyProperties.ts`.
2. Define a class with a readonly property.
3. Attempt to modify the readonly property after initialization.

Solution:

```typescript
class Book {
    readonly title: string;

    constructor(title: string) {
        this.title = title;
    }
}

let myBook = new Book("TypeScript Guide");
console.log("Book title:", myBook.title);

// myBook.title = "New Title"; // This will cause an error
```

Exercise 20: Optional Properties in Interfaces

Objective: Use optional properties in TypeScript interfaces.

Instructions:

1. Create a new file named `optionalProperties.ts`.
2. Define an interface with optional properties.
3. Implement the interface in a class and create an object with some properties missing.

Solution:

```typescript
interface User {
    name: string;
    age?: number;
}

class Person implements User {
    name: string;
    age?: number;

    constructor(name: string, age?: number) {
```

```
        this.name = name;
        if (age) {
            this.age = age;
        }
    }

    printInfo() {
        console.log(`Name: ${this.name}, Age: ${this.age}`);
    }
}

let user1 = new Person("Alice");
let user2 = new Person("Bob", 30);

user1.printInfo();
user2.printInfo();
```