**TypeScript Basic Questions:**   
  
TypeScript allows developers to add to JavaScript.

JavaScript is a typed language.  
  
The TypeScript compiler can be configured with which file?

Tsconfig.json

**TypeScript Simple Types:**

1. There are two main ways TypeScript assigns a type:  
1.Explicit  
2.Implicit  
  
2. Create a "firstName" variable, string type using Implicit type:  
let firstName = "Dylan";

3. Create a "firstName" variable, string type using Explicit type:  
let firstName : String = "Dylan";

**TypeScript Special Types:**  
1. Create an empty "myVar" variable, and disable type checking:  
let myVar: any;

2. Create an empty "myVar" variable, and specify it should be an unknown type:  
let myVar: unknown;

**TypeScript Arrays:**

Prevent the array from being changed:  
const names: readonly string[] = ["Dylan"];

**TypeScript Tuples:**  
1. The order of value types does not matter for Tuples:  
False  
  
2. Define ourTuple as string and boolean, in that order:  
let ourTuple: [string, boolean];

**TypeScript Object Types:**

1. Add the correct types for the object below:

const car: { type: string, model: string, year:number} = {

type: "Toyota",

model: "Corolla",

year: 2009 };

2. Specify that the second property, called model, should be optional:  
const car: { type: string, model?: string } = {

type: "Toyota"

}

**TypeScript Enums:**

1. Create an enum called myEnum, with 2 constants (myFirstConst, mySecondConst) with default values:  
enum myEnum{

myFirstConst,

mySecondConst

};

1. Create an enum called myEnum, with 2 constants (myFirstConst, mySecondConst) with the string values "first" and "second":  
     
   enum myEnum{

myFirstConst= "first",

mySecondConst= "second" };

**TypeScript Aliases & Interfaces:**

1. Create a Type Alias for a string, called carType:  
type carType = string  
  
2. Create an interface called myInterface, with the property myProp as a string:  
interface myInterface {

myProp:string

}

1. Extend the myInterface interface from last exercise, and add a property myExtProp as a number:  
   interface myExtInterface extends myInterface {

myExtProp : number

}

**TypeScript Union Type:**

Specify that the paramater "myVar" for the function can be either string or number:  
function myFunc(myVar:string |number) {

console.log(myVar)

}

**TypeScript Functions:**

1. Create a function that returns the string "Learning is Fun!", with the return type explicitly defined:  
function myFunc(): string {

return "Learning is Fun!";

}

2. Create a function that specifically does not return a value:  
function myFunc(): void {

console.log("Learning is Fun!");

}  
  
3. Create a function with 2 parameters (myVar1 and myVar2), that are both strings:  
function myFunc(myVar1:string, myVar2:string ) {

return(myVar1 + myVar2);

}

1. Create a function with 2 parameters (myVar1 and myVar2,in that order), that are both strings

Specify that myVar2 should be optional:

function myFunc(myVar1: string, myVar2?: string) {

return(myVar1 + (myVar2 || ""));}

**TypeScript Casting:**

Cast the "unknown" variable myVar as a string, using the as keyword:

let myVar: unknown = "Hello world!";

console.log((myVar as string).length);  
  
Cast the "unknown" variable myVar as a string, using < >:  
let myVar: unknown = "Hello world!";

console.log((<string>myVar).length);

**TypeScript Classes:**

Specify that Person.name can only be accessed within the class, but that the method Person.getName() can be accessed anywhere:  
class Person {

private name: string;

public constructor(name: string) {

this.name = name;

}

public getName(): string {

return this.name;

}

}

**TypeScript Basic Generics:**

Complete the Generic:  
function createPair<typeX,typeY> (x: typeX, y: typeY): [typeX, typeY] {

return [x, y];

}

console.log(createPair<string, number>('Meaning', 42));

**TypeScript Utility Types:**

1. Declare an object kindPerson from the Person interface, where all the properties are optional:  
   interface Person {

age: number;

firstName: string;

lastName: string;

}

let kindPerson:Partial<Person> = {};

1. Declare an object kindPerson from the Person interface, where all the properties are required.  
   interface Person {

age: number;

firstName: string;

lastName?: string;

}

let kindPerson: Required<Person> = {

age: 1800,

firstName: "Santa",

lastName: "Claus"

};

3. Complete the sentence:  
Record<string, number> is equivalent to { [key: string]: number }