1. What are the primitive types in TypeScript?

The primitive types in TypeScript are boolean, number, string, null, undefined, and symbol. These types are considered primitive because they are not objects and do not have any methods or properties.

1. Explain how the arrays work in TypeScript.

Arrays in TypeScript are similar to arrays in JavaScript. They can hold multiple values of the same or different data types. Arrays in TypeScript can be declared using the array notation, i.e., [] or the generic array notation, i.e., Array<elementType>. Arrays can be accessed using their index values, and their length can be determined using the length property.

1. What is any type, and when to use it?

The any type in TypeScript allows a variable to hold any type of value. It is used when the type of a value is not known or when the code needs to work with values of different types. However, using the any type too often can defeat the purpose of using TypeScript as it allows for less type checking and can result in runtime errors.

1. What is void, and when to use the void type?

The void type in TypeScript is used to indicate that a function does not return any value. It is commonly used when a function is called for its side effects, such as logging or modifying a data structure, rather than for its return value. Functions declared with a void return type must not return any value or else it will result in a compile-time error.

1. What is an unknown type, and when to use it in TypeScript?

The unknown type in TypeScript is similar to the any type in that it can hold values of any type. However, unlike any, the unknown type requires type checking before the value can be used. This is because the unknown type does not allow any operations to be performed on the value until its type is narrowed down using type guards. The unknown type is often used when the type of a value is not known, but it is necessary to perform type checking before using it.

1. What are the different keywords to declare variables in TypeScript?

In TypeScript, variables can be declared using the keywords var, let, and const. The var keyword declares a variable with function scope, while the let and const keywords declare variables with block scope. The let keyword declares a mutable variable that can be reassigned a new value, while the const keyword declares an immutable variable whose value cannot be reassigned.

1. Provide the syntax of a function with the type annotations.

The syntax of a function with type annotations in TypeScript is as follows:

function functionName(param1: type1, param2: type2): returnType {

// function body

}

Here, functionName is the name of the function, param1 and param2 are the parameters of the function with their respective types type1 and type2, and returnType is the type of the value returned by the function. The function body contains the code to be executed when the function is called.

1. How to create objects in TypeScript?

In TypeScript, objects can be created using object literal notation or by using the class keyword. Object literal notation involves defining an object using curly braces {} and specifying key-value pairs for the object properties. Here is an example:

let person = {

name: "John",

age: 30,

gender: "male"

}

1. How to specify optional properties in TypeScript?

In TypeScript, optional properties can be specified using the question mark (?) notation after the property name. Here is an example:

interface Person {

name: string;

age: number;

gender?: string;

}

let person: Person = {

name: "John",

age: 30

}

1. Explain the concept of null and its use in TypeScript.

In TypeScript, null is a special value that represents the intentional absence of any object value. It is often used to represent the absence of a value when a value is expected, but no valid value is available. Variables or properties that are potentially null are often defined with the union type of their original type and null (e.g., string | null). It is important to note that using null can introduce errors in code, and it is recommended to use undefined instead where possible.

1. What is undefined in TypeScript?

In TypeScript, undefined is a value that represents the absence of a value that has not been initialized or has been explicitly set to undefined. It is often used to indicate that a variable or property has not been assigned a value yet. Variables or properties that are potentially undefined are often defined with the union type of their original type and undefined (e.g., string | undefined).

1. Explain the purpose of the never type in TypeScript.

The never type in TypeScript represents the type of values that never occur. It is often used in functions that never return, such as functions that throw an error or enter an infinite loop. Functions that return the never type cannot have a reachable endpoint, and they cannot have a return statement or a function call. The never type is useful for enforcing type safety and eliminating potential errors in code.

1. Explain how enums work in TypeScript?

Enums in TypeScript are used to define a set of named constants. They provide a way to represent a group of related values as a single type, which makes the code more readable and easier to maintain. Enums are declared using the enum keyword, and their values can be accessed using dot notation. Here is an example:

enum Color {

Red,

Green,

Blue

}

let myColor = Color.Green;

console.log(myColor); // Outputs: 1

In this example, we have defined an enum called Color with three values. We then assign the value Color.Green to the variable myColor. When we log the value of myColor to the console, it outputs 1, which is the index of the Green value in the enum.

1. Explain the TypeScript class syntax.

In TypeScript, a class is a blueprint for creating objects that share the same properties and methods. The class syntax in TypeScript is similar to that in other object-oriented programming languages such as Java and C#. Here is an example of a class in TypeScript:

class Person {

private name: string;

private age: number;

constructor(name: string, age: number) {

this.name = name;

this.age = age;

}

greet() {

console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old`);

}

}

1. Explain the arrow function syntax in TypeScript.

In TypeScript, arrow functions provide a concise way to define a function. Arrow functions are also known as "fat arrow" functions due to the => syntax. Here is an example of an arrow function in TypeScript:

let multiply = (x: number, y: number): number => {

return x \* y;

}