Functions in TypeScript are essential for organizing code into reusable blocks, and TypeScript provides robust support for defining functions with various features such as type annotations, optional parameters, default parameters, and arrow functions. Here's an introduction to functions in TypeScript:

1. **Function Declaration**:
   In TypeScript, you can declare functions using the `function` keyword followed by the function name and parameters enclosed in parentheses, followed by the function body enclosed in curly braces.

   ```typescript
   function greet(name: string): void {
       console.log(`Hello, ${name}!`);
   }
   ```

   This defines a function `greet` that takes a parameter `name` of type `string` and logs a greeting message to the console.

2. **Function Parameters**:
   You can define parameters for functions with specified types.

   ```typescript
   function add(a: number, b: number): number {
       return a + b;
   }
   ```

   In this example, `add` takes two parameters `a` and `b`, both of type `number`, and returns their sum, also of type `number`.

3. **Return Types**:
   TypeScript allows you to specify the return type of a function using a type annotation after the parameter list and a colon `:`.

   ```typescript
   function multiply(a: number, b: number): number {
       return a * b;
   }
   ```

   The return type `number` indicates that this function returns a numerical value.

4. **Optional Parameters**:

   You can make function parameters optional by appending a question mark `?` to their names in the parameter list.

```typescript
function sayHello(name?: string): void {
    if (name) {
        console.log(`Hello, ${name}!`);
    } else {
        console.log("Hello, World!");
    }
}
```

   In this example, `name` is an optional parameter. If provided, it will greet the specified name; otherwise, it will greet "World".

5. **Default Parameters**:
   TypeScript allows you to specify default values for parameters using the assignment operator `=`.

```typescript
function greetWithPrefix(name: string, prefix: string = "Hello"): void {
    console.log(`${prefix}, ${name}!`);
}
```

   If `prefix` is not provided when calling the function `greetWithPrefix`, it defaults to `"Hello"`.

6. **Arrow Functions**:
   Arrow functions provide a concise syntax for defining functions, especially for anonymous functions and callbacks.

```typescript
const square = (x: number): number => x * x;
```

   This defines an arrow function `square` that takes a parameter `x` of type `number` and returns its square.

Functions in TypeScript are powerful constructs for building modular and reusable code, and TypeScript's static typing features ensure type safety and clarity throughout the development process. Whether it's defining simple functions, handling parameters and return types, or utilizing advanced features like optional parameters and arrow functions, TypeScript provides a flexible and expressive syntax for writing functions.