! Let's delve deeper into variables in TypeScript:

8. **Type Narrowing**:
   TypeScript uses control flow analysis to narrow down the type of a variable within a specific code block based on conditions.

   ```typescript
   let value: string | number;
   if (typeof value === "string") {
       // Within this block, value is treated as a string
       console.log(value.toUpperCase());
   } else {
       // Within this block, value is treated as a number
       console.log(value.toFixed(2));
   }
   ```

   Type narrowing helps TypeScript understand more specific types within conditional branches, improving type safety.

9. **Type Aliases**:
   TypeScript allows you to create aliases for types, which can be especially useful for complex types or types that are reused frequently.

   ```typescript
   type MyString = string;
   type Point = {
       x: number;
       y: number;
   };

   let str: MyString = "Hello, type alias!";
   let point: Point = { x: 10, y: 20 };
   ```

   Type aliases improve code readability and maintainability by providing descriptive names for types.

10. **Object Destructuring**:
    TypeScript supports object destructuring, allowing you to extract values from objects and bind them to variables.

    ```typescript
    let user = { name: "Alice", age: 30 };
    ```

```
let { name, age } = user;
console.log(name); // "Alice"
console.log(age);  // 30
```

Destructuring simplifies code by providing a concise way to extract values from objects.

11. **Array Destructuring**:
Similar to object destructuring, TypeScript supports array destructuring for extracting values from arrays.

```typescript
let numbers = [1, 2, 3];
let [first, second, third] = numbers;
console.log(first);  // 1
console.log(second); // 2
```

Array destructuring is handy for working with arrays and simplifies code by providing a more readable syntax.

12. **Rest Parameters and Spread Syntax**:
TypeScript supports rest parameters and spread syntax, allowing functions to accept an indefinite number of arguments or spread array elements into function calls or array literals.

```typescript
function sum(...numbers: number[]): number {
    return numbers.reduce((acc, curr) => acc + curr, 0);
}

let nums = [1, 2, 3];
console.log(sum(...nums)); // 6
```

Rest parameters and spread syntax provide flexibility and concise syntax for working with functions and arrays.

Variables in TypeScript offer a wide range of features and capabilities, empowering developers to write robust and maintainable code with strong typing support. Whether it's through type annotations, type inference, or advanced features like destructuring and type narrowing, TypeScript provides tools to enhance productivity and code quality.