Let's delve deeper into functions in TypeScript:

7. **Function Overloading**:
   TypeScript allows you to define multiple function signatures for a single function, known as function overloading. This enables the function to accept different combinations of parameters and return types.

   ```typescript
   function combine(a: string, b: string): string;
   function combine(a: number, b: number): number;
   function combine(a: any, b: any): any {
       return a + b;
   }
   ```

   Here, `combine` is overloaded to accept either two strings or two numbers and return a string or number accordingly.

8. **Rest Parameters**:
   TypeScript supports rest parameters, which allow a function to accept an indefinite number of arguments as an array.

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

   console.log(sum(1, 2, 3, 4, 5)); // Output: 15
   ```

   The rest parameter `...numbers: number[]` collects all arguments into an array called `numbers`.

9. **Function Types**:
   In TypeScript, functions are first-class citizens, meaning they can be assigned to variables, passed as arguments, and returned from other functions. You can define function types using the arrow function syntax or the `Function` type.

   ```typescript
   type Operation = (x: number, y: number) => number;

   const add: Operation = (a, b) => a + b;
   const subtract: Operation = (a, b) => a - b;
   ```

Here, `Operation` is a type representing a function that takes two numbers and returns a number. `add` and `subtract` are variables of this type.

10. **Recursive Functions**:
    TypeScript supports recursive functions, which call themselves within their own definition.

    ```typescript
    function factorial(n: number): number {
        if (n === 0 || n === 1) {
            return 1;
        } else {
            return n * factorial(n - 1);
        }
    }

    console.log(factorial(5)); // Output: 120
    ```

    This `factorial` function calculates the factorial of a non-negative integer `n` using recursion.

11. **Higher-Order Functions**:
    TypeScript allows you to define higher-order functions, which are functions that take other functions as arguments or return functions as results.

    ```typescript
    function applyOperation(x: number, y: number, operation: (a: number, b: number) => number): number {
        return operation(x, y);
    }

    console.log(applyOperation(5, 3, add)); // Output: 8
    ```

    `applyOperation` is a higher-order function that takes two numbers and a function representing an operation to apply to those numbers.

Functions in TypeScript are versatile constructs that enable you to create modular, reusable, and expressive code. Whether it's through function overloading, rest parameters, function types, recursion, or higher-order functions, TypeScript provides powerful tools for defining and working with functions in a statically-typed environment.