Mastering Functions in TypeScript

1. Problem Statement

The city library needs a **Report Generator** module to automate routine tasks:

- Display member details (ID, name, optional email).
- Calculate total fines for overdue books (variable number per member).
- Compute discounted membership fees with default rates.
- Register daily visitors via callback.
- Support different report formats via function overloading.
  Manual scripts are tangled and error-prone. You need reusable, well-typed functions to organize this logic.

---

2. Learning Objectives

By the end of this tutorial, you will be able to:

- Declare and call typed functions.
- Use optional, rest, and default parameters.
- Write anonymous and arrow functions.
- Implement recursion and function overloading.
- Define function types and use type aliases.
- Pass functions as arguments (higher-order functions).

---

3. Concept Introduction with Analogy

Analogy: The Library Service Desks

Imagine the library's front lobby organized into specialized desks, each demonstrating a key TypeScript function concept through its real-world role:

1. **Information Desk (Function Declaration)**
    - This desk greets you by name, looks up your membership details, and tells you which section to visit.
    - Just as the Information Desk has a clear menu of services ("show account," "renew books," "check due dates"), a declared function names its purpose and defines exactly what inputs it accepts and what output it provides.
2. **Fine Collection Desk (Parameterized + Rest Parameters)**
    - Patrons bring in one or many overdue slips all at once. The clerk takes a variable number of slips and tallies them in one go.
    - Like that desk, a function with rest parameters accepts a fixed initial piece of information (your member ID) plus any number of additional values (the fines), processing them all in the same streamlined step.

3. **Membership Counter (Default Parameters)**
   - New members pay a standard fee, but returning members with a coupon get a special discount. If no coupon is shown, the clerk defaults to the regular rate.
   - Similarly, a function can have parameters that default to a predefined value when no explicit argument is provided, ensuring consistent behavior without extra steps.
4. **Visitor Kiosk (Callback Functions)**
   - A touchscreen asks your name then calls a custom greeting routine-perhaps displaying your photo, printing a welcome card, or sending an SMS. The kiosk simply invokes whatever greeting function you've configured.
   - This mirrors higher-order functions: you supply the behavior (a callback) and the base function handles user input, then hands off control to your custom routine.
5. **Special Services Desk (Function Overloads)**
   - Patrons can request the library catalog in different formats: a printed handout, a PDF, or a USB drive with structured data. The same desk accepts the request and delivers the correct format.
   - Function overloads work the same way: a single name offers multiple "signatures" so callers can choose text, JSON, or other formats, with one underlying implementation adapting to the request.

---

4. Technical Deep Dive: Functions in TypeScript

- **Declaration vs. Expression**: Named (`function foo(){}`) vs. anonymous (`const foo = function(){}`)
- **Parameters & Return Types**: `function fn(a: number): string { … }`
- **Optional Parameters**: Marked `?`, must be last
- **Rest Parameters**: `...nums: number[]`, gather variable args
- **Default Parameters**: `rate: number = 0.1`
- **Arrow Functions**: Concise syntax `(a, b) => a + b`
- **Recursion**: Functions calling themselves with a base case
- **Function Overloads**: Multiple signatures, single implementation
- **Function Types & Aliases**: `(x: string) => void`; `type Handler = (msg: string) => void`
- **Higher-Order Functions**: Functions accepting other functions as parameters

---

5. Step-by-Step Code Walkthrough

```
// 1. Simple Declaration & Optional Parameter
function displayMember(id: number, name: string, email?: string): void {
  console.log(`ID: ${id}, Name: ${name}`);
  if (email) console.log(`Email: ${email}`);
}

// 2. Rest Parameters for Fines Tally
function calculateFines(...fines: number[]): number {
  let total = 0;
  for (let fine of fines) total += fine;
  return total;
}

// 3. Default Parameter for Discount
function membershipFee(price: number, discountRate: number = 0.1): number {
  return price - price * discountRate;
}

// 4. Anonymous Function & Callback
function greetVisitor(visitor: string, formatter: (name: string) => void): void {
  formatter(visitor);
}
const vipGreet = (name: string) => console.log(`Welcome VIP ${name}!`);

// 5. Recursion: Factorial (for demonstration)
function factorial(n: number): number {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
}

// 6. Function Overloads for Report Generation
function generateReport(data: object[]): string;
function generateReport(data: object[], format: "json"): string;
function generateReport(data: any[], format?: string): string {
  if (format === "json") {
    return JSON.stringify(data, null, 2);
  }
  return data.map(item => item.toString()).join("\n");
}

// 7. Function Type & Alias
type VisitorFormatter = (name: string) => void;
let consoleGreet: VisitorFormatter = (n) => console.log(`Hello, ${n}!`);
```

---

6. Interactive Challenge / Mini-Project

Your Turn!

1. Call `displayMember` for two members: one with email, one without.
2. Use `calculateFines` to sum fines: 5, 10, 2.5.
3. Compute a membership fee for $100 with default discount, then with 20%.
4. Greet visitors “Alice” and “Bob” using both `vipGreet` and `consoleGreet`.
5. Compute `factorial(5)`.
6. Generate a text report and a JSON report for an array of sample objects (e.g., `{ title: "1984" }`).

---

7. Common Pitfalls & Best Practices

- **Optional parameters** must come last.
- **Rest parameter** can only appear once at the end.
- **Default vs. Optional**: Don’t mix both on the same parameter.
- **Recursive functions** need a clear base case to avoid infinite loops.

- **Provide `break` in overloads**: ensure `switch` or `if` chains cover all types.
- **Explicit function types** improve readability for callbacks.

8. Quick Recap & Key Takeaways

- Functions organize code into reusable tasks.
- TypeScript adds safety with parameter and return type annotations.
- Optional, rest, and default parameters handle flexible argument patterns.
- Anonymous, arrow, and constructor functions offer varied declaration styles.
- Recursion and overloads provide advanced capabilities.
- Function types and aliases clarify expected function shapes.