Optional and Default Parameters in TypeScript

1. Problem Statement

You are building a **Flexible Greeting System** for a social app. The system must:

- Greet users by name.
- Optionally include their age if provided.
- Use a default age when none is given.
- Handle missing or undefined parameters gracefully.

Previously, missing data caused crashes or incorrect messages. You need functions that safely handle both optional and default parameters.

---

2. Learning Objectives

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

- Define functions with optional parameters.
- Implement default parameters and understand their behavior.
- Distinguish between optional and default parameters.
- Safely handle `undefined` values inside functions.
- Apply proper parameter ordering and best practices.

---

3. Concept Introduction with Analogy

Analogy: The Personalized Invitation Card Maker

Imagine you run a service that prints invitation cards:

- **Name** is mandatory on every card.
- **Age** is optional-if the guest's age isn't provided, you leave it off or use a standard placeholder.
- Sometimes the age is unknown (explicitly `undefined`) and you must handle that case too.

Your card printer (function) must accept these variations without error:

- If age is supplied, print it.
- If age is omitted, leave it blank.
- If age is missing entirely, treat it as `undefined` and apply placeholder logic.
- For a default case, automatically use a standard age if none is provided.

This mirrors TypeScript's optional (`?`) and default (`=`) parameters: they let your functions adapt to missing or omitted arguments safely and predictably.

---

4. Technical Deep Dive

Optional Parameters

- Syntax: `paramName?: type`
- May be omitted; value is `undefined` when not passed.
- Must follow all required parameters in signature.
- Inside function, use type guards to check for `undefined`.

Default Parameters

- Syntax: `paramName: type = defaultValue`
- Automatically optional; if omitted or explicitly `undefined`, uses `defaultValue`.
- Cannot combine `?` and `=` on the same parameter.
- Must follow required parameters.

Key Differences

- **Optional**: no built-in value; must check `undefined`.
- **Default**: has a built-in fallback value; no need for manual check.

Parameter Order

1. Required parameters
2. Optional parameters
3. Default parameters

5. Step-by-Step Code Walkthrough

Example 1: Optional Parameter

```
function greet(name: string, age?: number): void {
  if (typeof age === "number") {
    console.log(`Hello ${name}, you are ${age} years old.`);
  } else {
    console.log(`Hello ${name}`);
  }
}

greet("Alice");       // Hello Alice
greet("Bob", 30);     // Hello Bob, you are 30 years old.
```

Example 2: Default Parameter

```ts
function greet(name: string, age: number = 25): void {
  console.log(`Hello ${name}, you are ${age} years old.`);
}

greet("Charlie");     // Hello Charlie, you are 25 years old.
greet("Diana", 40);   // Hello Diana, you are 40 years old.
```

Example 3: Incorrect Optional Ordering (Error)

```ts
// Error: Optional parameters must follow required ones
function add(x?: number, y: number): number {
  return (x || 0) + y;
}
```

Example 4: Optional + Default Conflict (Error)

```ts
// Error: Cannot combine ? and = on same parameter
function add(x: number, y?: number = 10): number {
  return x + y;
}
```

Example 5: Default as Optional

```ts
function add(x: number, y: number = 10): number {
  return x + y;
}

console.log(add(5));    // 15
console.log(add(5, 20)); // 25
```

6. Interactive Challenge / Mini-Project

Your Turn!

1. **`describePerson`**
   - Required: `name: string`
   - Optional: `age?: number`
   - Print `"Name: <name>, Age: <age>"` or `"Name: <name>, Age: Unknown"`.
2. **`calculatePrice`**
   - Required: `basePrice: number`
   - Default: `discount: number = 0.1`
   - Return price after discount.
3. Test calls:

```
describePerson("Eve");
describePerson("Frank", 28);
console.log(calculatePrice(100));      // 90
console.log(calculatePrice(100, 0.2)); // 80
```

---

7. Common Pitfalls & Best Practices

- Place optional and default parameters **after** all required ones.
- **Do not** combine `?` and `=` on the same parameter.
- Use **type guards** to handle `undefined` optional values.
- Prefer default parameters for simple fallbacks to reduce branching.
- Document parameter behavior (optional vs. default) in comments or docs.

---

8. Quick Recap & Key Takeaways

- **Optional parameters** (`?`) may be omitted; value is `undefined` .
- **Default parameters** (`=`) supply a fallback value when omitted.
- Parameter order: **required → optional → default**.
- Type guards and clear checks ensure runtime safety.
- Default parameters serve as implicit optionals, simplifying code.
