https://jsmastery.pro

8 TypeScript Tips You Need to Know

Explore Union & Intersection Types

Leverage union types for flexibility and intersection types for combining types

```
type Status = 'success' | 'error';
type User = { id: number } & { name: string };
```

Benefit from Conditional Types

Use conditional types to create dynamic type branching based on conditions.

```
type Check<T> = T extends string ? 'string' : 'non-string';
let result: Check<number>; // Type of result: 'non-string'
```

Leverage Type Assertion Carefully

Be cautious with type assertions; use them sparingly and validate assumptions.

```
const userInput: any = getUserInput();
// Use type assertion when certain about the type
const formattedInput = (userInput as string).toUpperCase();
```

Explore 'readonly' Modifier

Use 'readonly' to prevent accidental modifications to objects or arrays.

```
const numbers: readonly number[] = [1, 2, 3];
numbers.push(4);
// Error: Property 'push' does not exist on type 'readonly number[]'.
```

Avoid 'any' When Possible

Minimize the use of 'any' to maintain strong typing and catch potential errors.

```
// Avoid using 'any' when the type is known
function parseJSON(jsonString: string): any {
  return JSON.parse(jsonString);
}
```

Explore Mapped Types

Use mapped types for creating new types based on existing ones.

```
type OptionalProps<T> = { [P in keyof T]?: T[P] };
type OptionalUser = OptionalProps<{ name: string; age: number }>;
```

Leverage Optional and Default Parameters

Make good use of optional and default parameters for more flexible function calls

```
function greet(name: string, greeting: string = 'Hello') {
  console.log(`${greeting}, ${name}!`);
}
```

Take Advantage of Tuples

Use tuples for fixed-length arrays with specific types at each position.

```
let coordinate: [number, number] = [10, 20];
```