Tricky & Interesting TypeScript Questions for Senior Angular Developers

A curated list of advanced and scenario-based TypeScript questions to assess senior Angular developers.

■ Type Inference & Narrowing Example

```
function process(value: string | number) {
  if (typeof value === 'string') {
    return value.toUpperCase();
  }
  return value.toFixed(2);
}
const result = process(Math.random() > 0.5 ? 5 : 'five');
```

■ Discuss how TypeScript infers return types and how changing return type annotations affects inference.

■ Generics with keyof Constraint

```
function getProperty<T, K extends keyof T>(obj: T, key: K): T[K] {
  return obj[key];
}
```

■ Explain why we use `K extends keyof T` and how it improves type safety.

■ Cache Service Scenario

```
class CacheService<T> {
  private store = new Map<string, T>();
  setItem(key: string, value: T): void {
    this.store.set(key, value);
  }
  getItem(key: string): T | undefined {
    return this.store.get(key);
  }
}
```

■ How can this class be extended to handle multiple types dynamically for different cache keys?

■ Mapped Type Example

```
type Mutable<T> = {
  -readonly [P in keyof T]: T[P];
};
```

■ Explain what removing `-readonly` does and how mapped types are used.

■ Custom Decorator Example

```
function LogMethod(target: any, propertyKey: string, descriptor: PropertyDescriptor) {
  const original = descriptor.value;
  descriptor.value = function (...args: any[]) {
    console.log(`Called ${propertyKey} with`, args);
    return original.apply(this, args);
  };
}
```

```
class UserService {
  @LogMethod
  getUser(id: number) {
    return { id, name: 'John' };
  }
}
```

■ How would you make this decorator type-safe for async methods?

■ Structural Typing Example

```
type Point = { x: number; y: number };
type Coordinate = { x: number; y: number };

const move = (p: Point) => console.log(p.x + p.y);
const coord: Coordinate = { x: 5, y: 10 };

move(coord);
```

■ Why does this work under TypeScript's structural typing model?

■ Conditional Type Example

```
type Response<T> = T extends Array<infer U> ? U : T;
```

■ How does the `infer` keyword work, and what will `Response` evaluate to?

■ Dynamic Form Scenario

```
form = this.fb.group({
  name: [''],
  age: [0],
});
```

■ How could you infer control and value types from a `FormGroup` using mapped types?

■ HttpClient Wrapper Example

```
get<T>(url: string): Observable<T> {
  return this.http.get<T>(url);
}
```

■ How can you combine runtime validation (zod/io-ts) with TypeScript generics for safer Angular API calls?

■ Edge Case - unknown vs any

■ When should you use `unknown` over `any`, and what's the difference?