Advanced Types

Summary

- Using a type alias we can create a new name (alias) for a type. We often use type aliases to create custom types.
- With union types, we can allow a variable to take one of many types (eg number | string).
- With intersection types, we can combine multiple types into one (eg Draggable & Resizable).
- Using optional chaining (?.) we can simplify our code and remove the need for null checks.
- Using the Nullish Coalescing Operator we can fallback to a default value when dealing with null/undefined objects.
- Sometimes we know more about the type of a variable than the TypeScript compiler. In those situations, we can use the **as** keyword to specify a different type than the one inferred by the compiler. This is called type assertion.
- The **unknown** type is the type-safe version of **any**. Similar to **any**, it can represent any value but we cannot perform any operations on an **unknown** type without first narrowing to a more specific type.
- The **never** type represents values that never occur. We often use them to annotate functions that never return or always throw an error.

Cheat Sheet

Type alias

```
type Employee = {
   id: number;
   name: string;
   retire: (date: Date) => void
```

Union types

```
let weight: number | string = 1;
weight = '1kg';
```

Intersection types

```
type UIWidget = Draggable & Droppable;
```

Literal types

```
type Quantity = 50 | 100;
```

Nullable types

```
let name: string | null = null;
```

Optional chaining (?.)

```
customer?.birthdate?.getFullYear();
customers?.[0];
log?.('message');
```

Nullish coalescing operator

```
someValue ?? 30
```

Type assertion

```
obj as Person
```

The unknown type

```
function render(document: unknown) {
  // We have to narrow down to a specific
  // type before we can perform any operations
  // on an unknown type.
  if (typeof document === 'string') {

  }
}
```

The never type

```
function processEvents(): never {
  // This function never returns because
  // it has an infinite loop.
  while (true) {}
}
```

Compiler Options

Option	Description
`strictNullChecks`	When enabled, null and undefined will not be acceptable values for variables unless you explicitly declare them as nullable. So, you'll get an error if you set a variable to null or undefined.
`allowUnreachableCode`	When set the false, reports error about unreachable code.