

TypeScript Cheat Sheet

Basic Types

- **Primitives:** `string`, `number`, `boolean`, `null`, `undefined`, `symbol`, `bigint`
- **Arrays:** `number[]` or `Array<number>`
- **Tuples:** Fixed-length typed arrays


```
let user: [number, string] = [1, "Alice"];
let optionalTuple: [string, number?] = ["Bob"];
let restTuple: [string, ...number[]] = ["Scores", 90, 85];
```
- **Enums:**

```
enum Direction { Up = 1, Down, Left, Right }
const enum FastStatus { Ready, Done } // Inlined at compile time
```
- **Special Types:**
 - **any:** Disables type checking (allows all operations)
 - **unknown:** Type-safe counter of **any** (requires narrowing)
 - **never:** Represents values that never occur (exhaustiveness)
 - **void:** Return type for functions returning no value

Interface vs Type Alias

Interface (Extendable & Object Models)

```
interface User {
  readonly id: number;
  name: string;
  age?: number; // Optional property
}

// Inheritance & Declaration Merging
interface Admin extends User {
  role: string;
}
```

Type Alias (Unions, Primitives, Tuples)

```
type ID = string | number;
type Point = { x: number; y: number };
type Callback = (data: string) => void;
type Status = "idle" | "loading" | "success" | "error";
```

Union & Intersection

- **Union (|):** Value can be one of several types


```
function printId(id: string | number) { ... }
```
- **Intersection (&):** Combines multiple types into one


```
type Person = { name: string };
type Employee = Person & { employeeId: number };
```

Type Narrowing

Refine broad types to specific types using run-time checks.

- **typeof check:** `typeof x === "string"`
- **instanceof check:** `err instanceof CustomError`
- **in operator:** `"role" in account`
- **Discriminated Unions:**

```
type Shape =
  | { kind: "circle"; radius: number }
  | { kind: "square"; size: number };

function getArea(s: Shape): number {
  switch (s.kind) {
    case "circle": return Math.PI * s.radius ** 2;
    case "square": return s.size * s.size;
  }
}
```
- **Custom Type Guards:**

```
function isString(val: unknown): val is string {
  return typeof val === "string";
}
```

Generics

Parameterized types for reusable code.

```
function identity<T>(arg: T): T {
  return arg;
}
```

```
// Generic Interface with Defaults
interface ApiResponse<T = any> {
  code: number;
```

```
data: T;
}

// Generic Constraints
interface Lengthwise { length: number; }
function logLength<T extends Lengthwise>(item: T): number {
  return item.length;
}
```

Utility Types

Built-in type transformation helpers.

- **Partial<T>:** Makes all properties in **T** optional
- **Required<T>:** Makes all properties in **T** required
- **ReadOnly<T>:** Makes all properties in **T** read-only
- **Record<K, T>:** Constructs an object type with keys **K** and values **T**
- **Pick<T, K>:** Extracts properties **K** from **T**
- **Omit<T, K>:** Removes properties **K** from **T**
- **Exclude<T, U>:** Excludes types assignable to **U** from union **T**
- **Extract<T, U>:** Extracts types assignable to **U** from union **T**
- **NonNullable<T>:** Removes `null` and `undefined` from **T**
- **ReturnType<T>:** Extracts return type of function type **T**
- **Parameters<T>:** Extracts parameters of function type **T** as tuple
- **Awaited<T>:** Unwraps `Promise<T>` value type

Advanced Types & Operators

- **keyof operator:** Produces union of known object keys


```
type UserKeys = keyof User; // "id" | "name" | "age"
```
- **Indexed Access Types:**

```
type UserName = User["name"]; // string
```
- **Conditional Types:**

```
type IsString<T> = T extends string ? true : false;
```
- **Template Literal Types:**

```
type Event = `on${"Click" | "Hover"}`;
// "onClick" | "onHover"

• satisfies Operator: Validates expression type without widening
const palette = {
  red: [255, 0, 0],
  green: "#00ff00",
} satisfies Record<string, string | number[]>;
```

Classes & Access Modifiers

```
class Animal {
  public name: string;
  protected age: number;
  #internalId: string; // ECMAScript Private field

  // Parameter property shorthand
  constructor(public readonly breed: string, age: number) {
    this.name = breed;
    this.age = age;
    this.#internalId = Math.random().toString();
  }
}
```

Key tsconfig.json Settings

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "strict": true,
    "noImplicitAny": true,
    "strictNullChecks": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  }
}
```