TypeScript Cheat Sheet

22 January 2024 by Huzaif Sayyed

TypeScript has gained significant popularity in the world of web development due to its ability to bring static typing to JavaScript. It enhances code quality, provides better tooling, and ultimately leads to more robust applications. However, mastering TypeScript can be a journey in itself, and having a handy cheatsheet can make that journey smoother. In this blog post, we'll provide a TypeScript cheatsheet that you can refer to for quick guidance and reminders.

Links

- Typescript Official Wesite
- Javascript Cheatsheet
- Download TypeScript Cheat Sheet
- ★ Want More Cheatsheet

Hello World! Program in TypeScript

Create a file `app.ts`

```
let message: string = 'Hello, World!';
console.log(message);
```

Run `tsc app.ts` and this will create `app.js`

Basic Types

```
let isDone: boolean = false;
let age: number = 25;
let name: string = "John";
let fruits: string[] = ["apple", "banana", "orange"];
let tuple: [string, number] = ["example", 42];
let anyType: any = "anything";
```

Functions

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

const greet = (name: string): string => `Hello, ${name}!`;
```

Interfaces

```
interface Person {
  name: string;
  age: number;
}

const user: Person = {
  name: "Alice",
  age: 30,
};
```

Classes

```
class Animal {
  constructor(public name: string) {}

  makeSound(): void {
    console.log("Some generic sound");
  }
}

class Dog extends Animal {
  makeSound(): void {
    console.log("Woof! Woof!");
  }
}

const myDog = new Dog("Buddy");
myDog.makeSound();
```

Enums

```
enum Color {
  Red,
  Green,
  Blue,
}

let myColor: Color = Color.Green;
console.log(myColor); // Outputs: 1
```

Enums with Values

```
enum Size {
  Small = "S",
  Medium = "M",
  Large = "L",
}

let mySize: Size = Size.Medium;
console.log(mySize); // Outputs: M
```

Generics

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

const result = identity<string>("Hello, TypeScript!");
```

Type Assertions

```
let someValue: any = "this is a string";
let strLength: number = (someValue as string).length;
```

Type Guards

```
function isNumber(value: any): value is number {
  return typeof value === "number";
}

if (isNumber(myVar)) {
  // myVar is now recognized as a number
}
```

Modules

```
// Exporting
export const myVar: string = "Hello from module!";

// Importing
import { myVar } from "./myModule";
```

Declaration Merging

```
interface User {
  name: string;
  age: number;
}

interface User {
  email: string;
}

const newUser: User = {
  name: "John",
  age: 25,
  email: "john@example.com",
};
```

Decorators

```
function log(target: any, key: string, descriptor: PropertyDescriptor): void {
  const originalMethod = descriptor.value;

  descriptor.value = function (...args: any[]): any {
    console.log(`Calling ${key} with arguments: ${args}`);
    const result = originalMethod.apply(this, args);
    console.log(`${key} returned: ${result}`);
    return result;
  };
}

class Example {
  @log
  add(x: number, y: number): number {
    return x + y;
  }
}

const example = new Example();
example.add(2, 3); // Console logs with method details
```

Union and Intersection Types

Union Types

```
let value: string | number;
value = "10";
value = 10;
```

Intersection Types

```
type Car = { brand: string; model: string };
type ElectricCar = { batteryLife: number };

const electricCar: Car & ElectricCar = {
  brand: "Tesla",
  model: "Model S",
  batteryLife: 300,
};
```

Literal Types

```
let status: "active" | "inactive";
status = "active";

function handleStatus(status: "active" | "inactive"): void {
  // Do something based on status
}

handleStatus("inactive");
```

Type Aliases

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

function calculateDistance(point1: Point, point2: Point): number {
  return Math.sqrt((point2.x - point1.x) ** 2 + (point2.y - point1.y) ** 2);
}
```

Mapped Types

```
type Flags = {
  option1: boolean;
  option2: boolean;
};

type ReadOnlyFlags = {
  readonly [K in keyof Flags]: boolean;
};

const readOnlyFlags: ReadOnlyFlags = {
  option1: true,
  option2: false,
};
```

Async/Await

```
function fetchData(): Promise<string> {
  return new Promise(resolve => {
    setTimeout(() => resolve("Data fetched!"), 2000);
  });
}

async function fetchDataAsync(): Promise<void> {
  const result: string = await fetchData();
  console.log(result);
}

fetchDataAsync();
```

Conditional Types

```
type Check<T> = T extends string ? boolean : number;

const isString: Check<"hello"> = true;
const isNumber: Check<42> = 42;
```

Nullish Coalescing and Optional Chaining

Nullish Coalescing

```
const defaultValue = "Default Value";
const userInput = null;

const result = userInput ?? defaultValue;
console.log(result); // Outputs: Default Value
```

Optional Chaining

```
type Person = {
  name: string;
  address?: {
    city?: string;
  };
};

const person: Person = { name: "John" };
const cityName = person?.address?.city;
console.log(cityName); // Outputs: undefined
```

String Literal Template Types

```
type Greeting<T extends string> = `Hello, ${T}!`;

const helloJohn: Greeting<"John"> = "Hello, John!";
```

Using “unknown” Type

```
let userInput: unknown;

// Type assertion is needed
let userName: string = userInput as string;

// Type checking function
function isString(input: unknown): input is string {
  return typeof input === "string";
}

if (isString(userInput)) {
  userName = userInput;
}
```

Type Narrowing with “never” Type

```
function throwError(message: string): never {
  throw new Error(message);
}

let result: string;

try {
  result = fetchData();
} catch (error) {
  throwError("Failed to fetch data");
}
```

Readonly Arrays and Tuples

```
let numbers: readonly number[] = [1, 2, 3];
let tuple: readonly [string, number] = ["example", 42];

// This will result in a compile-time error
numbers[0] = 4;
tuple[0] = "new example";
```

Destructuring with Types

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

function printCoordinates({ x, y }: Point): void {
  console.log(`Coordinates: x=${x}, y=${y}`);
}

const point: Point = { x: 10, y: 20 };
printCoordinates(point);
```

Mixins

```
class Timestamped {
  timestamp = Date.now();
}

class Activatable {
  isActive = false;

  activate() {
    this.isActive = true;
  }

  deactivate() {
    this.isActive = false;
  }
}

interface User extends Timestamped, Activatable {
  name: string;
}

const user: User = {
  name: "Alice",
  timestamp: 1642949983235,
  isActive: true,
  activate: () => {/*...*/},
  deactivate: () => {/*...*/}
};
```

Congratulations on reaching the end of this Typescript Programming Language Cheatsheet! This resource is designed to make your coding experience easy and efficient. Feel free to bookmark this page or download the PDF for future reference. Happy coding with TypeScript!

Cheatsheet
JavaScript Programming Language, Programming Language Cheatsheet, TypeScript Programming Language

‹ PowerShell Cheat Sheet

Type cheatsheet, code
Search

Related Posts

- PowerShell Cheat Sheet
- Bash Cheat Sheet
- Java Cheat Sheet
- Zig Cheat Sheet
- Shell Scripting Cheat Sheet
- JavaScript Cheat Sheet
- Python Cheat Sheet

More Cheatsheet Tutorials ⟫