TypeScript Cheat Sheet

1. Introduction to TypeScript

What is TypeScript?

- **TypeScript** is a superset of JavaScript that adds optional static typing and modern ECMAScript features.

- It is compiled to plain JavaScript, making it compatible with all browsers and environments.

Key Features

- **Static Typing**: Catch errors early and improve code quality.

- **ES6+ Features**: Use modern JavaScript features.

- **Tooling**: Enhanced developer tools with IntelliSense and refactoring.

---

2. Basic Syntax and Types

Variables and Constants

```
let variableName: type = value;
const constantName: type = value;
```

Basic Types

- **Number**: `let num: number = 10;`

- **String**: `let str: string = "Hello";`

- **Boolean**: `let bool: boolean = true;`

- **Array**: `let arr: number[] = [1, 2, 3];`

- **Tuple**: `let tuple: [string, number] = ["age", 25];`

- **Any**: `let dynamic: any = "or any other type";`

- **Void**: `function logMessage(): void { console.log("Message"); }`

- **Null and Undefined**: `let n: null = null; let u: undefined = undefined;`

- **Never**: `function error(message: string): never { throw new Error(message); }`

Type Assertions

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

---

3. Interfaces and Types

Interfaces

```
interface Person {
    name: string;
    age?: number; // Optional property
    readonly id: number; // Read-only property
    greet(phrase: string): void;
}
```

Type Aliases

```
type Name = string;
type Age = number;
type Person = { name: Name, age: Age };
```

Extending Interfaces

```
interface Employee extends Person {
    employeeId: number;
}
```

Union Types

```
type ID = number | string;
```

Intersection Types

```
type EmployeePerson = Person & Employee;
```

---

4. Classes and Objects

Classes

```
class Animal {
    name: string;
    constructor(name: string) {
        this.name = name;
    }
    move(distance: number) {
        console.log(`${this.name} moved ${distance}m.`);
    }
}
```

Inheritance

```
class Dog extends Animal {
    bark() {
        console.log("Woof! Woof!");
    }
}
```

Access Modifiers

- **Public**: `public name: string;` (default)
- **Private**: `private age: number;`
- **Protected**: `protected id: number;`

Readonly and Static

```
class MyClass {
    readonly version: string = "1.0";
    static instanceCount: number = 0;
}
```

---

5. Modules and Namespaces

Modules

```
// math.ts
export function add(a: number, b: number): number {
    return a + b;
}
```

```
// main.ts
import { add } from './math';
console.log(add(2, 3)); // 5
```

Namespaces

```
namespace MyNamespace {
    export interface MyInterface { }
    export class MyClass { }
}
```

---

6. Generics

Generic Functions

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

Generic Classes

```
class GenericClass<T> {
    value: T;
    constructor(value: T) {
        this.value = value;
    }
}
```

Generic Constraints

```
function loggingIdentity<T extends Lengthwise>(arg: T): T {
    console.log(arg.length);
    return arg;
}
```

---

7. Advanced Types

Type Guards

```
function isString(value: any): value is string {
    return typeof value === "string";
}
```

Discriminated Unions

```
type Shape = Square | Rectangle;
interface Square { kind: "square"; size: number; }
interface Rectangle { kind: "rectangle"; width: number; height: number;
}
```

Mapped Types

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

Conditional Types

```
type TypeName<T> =
    T extends string ? "string" :
    T extends number ? "number" :
    T extends boolean ? "boolean" :
    "object";
```

---

8. Decorators

Class Decorators

```
function sealed(constructor: Function) {
    Object.seal(constructor);
    Object.seal(constructor.prototype);
}

@sealed
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
```

```
        return "Hello, " + this.greeting;
    }
}
```

Method Decorators

```
function enumerable(value: boolean) {
    return function (target: any, propertyKey: string, descriptor:
PropertyDescriptor) {
        descriptor.enumerable = value;
    };
}

class MyClass {
    @enumerable(false)
    method() { }
}
```

---

9. TypeScript Compiler (tsc)

Basic Usage

```
tsc file.ts
```

Configuration File (tsconfig.json)

```
{
    "compilerOptions": {
        "target": "ES6",
        "module": "commonjs",
        "strict": true,
        "outDir": "./dist"
    },
    "include": ["src/**/*"],
    "exclude": ["node_modules"]
}
```

Watch Mode

```
tsc --watch
```

---

10. Tips and Tricks

Type Inference

- TypeScript can infer types, so you don't always need to specify them explicitly.

Strict Mode

- Enable `strict: true` in `tsconfig.json` for better type safety.

Using `any` Wisely

- Avoid using `any` unless absolutely necessary. Prefer specific types.

Debugging with Source Maps

- Generate source maps to debug TypeScript code directly in the browser.

Using `unknown`

- Use `unknown` instead of `any` when you don't know the type but want to enforce type checking.

This cheat sheet provides a comprehensive overview of TypeScript, covering its essential features, syntax, and best practices. Use it as a quick reference to enhance your TypeScript development skills.

By Ahmed Baheeg Khorshid

ver 1.0