Cheat Sheet for comprehensive TypeScript

Basic Syntax and Types

Variables and Constants

- **`let`**: Mutable variable declaration.

```
let count: number = 10;
```

- **`const`**: Immutable variable declaration.

```
const PI: number = 3.14;
```

Basic Types

- **`number`**: For numeric values.

```
let age: number = 25;
```

- **`string`**: For text values.

```
let name: string = "Alice";
```

- **`boolean`**: For true/false values.

```
let isStudent: boolean = true;
```

- **`any`**: For dynamic types.

```
let dynamicValue: any = "Hello";
```

- **`void`**: For functions that do not return a value.

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

Arrays and Tuples

Arrays

- **Declaration**:

```
let numbers: number[] = [1, 2, 3];
let names: Array<string> = ["Alice", "Bob"];
```

- **Methods**:

- `push()`, `pop()`, `shift()`, `unshift()`, `splice()`, `slice()`, `map()`, `filter()`, `reduce()`

Tuples

- **Declaration**:

```
let person: [string, number] = ["Alice", 25];
```

- **Accessing Elements**:

```
let name = person[0]; // "Alice"
let age = person[1]; // 25
```

Functions

Function Declaration

- **Basic**:

```
function add(a: number, b: number): number {
  return a + b;
}
```

- **Optional Parameters**:

```
function greet(name: string, age?: number): string {
  return `Hello, ${name}!`;
}
```

- **Default Parameters**:

```
function greet(name: string, age: number = 25): string {
  return `Hello, ${name}! You are ${age} years old.`;
}
```

- **Rest Parameters**:

```
function sum(...numbers: number[]): number {
  return numbers.reduce((acc, num) => acc + num, 0);
}
```

Arrow Functions

- **Basic**:

```
const multiply = (a: number, b: number): number => a * b;
```

- **Implicit Return**:

```
const square = (x: number): number => x * x;
```

Interfaces and Types

Interfaces

- **Declaration**:

```
interface Person {
  name: string;
  age: number;
  greet(): string;
}
```

- **Implementation**:

```
const alice: Person = {
  name: "Alice",
  age: 25,
  greet() {
    return `Hello, I'm ${this.name}!`;
  }
};
```

Type Aliases

- **Declaration**:

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

- **Usage**:

```
const origin: Point = { x: 0, y: 0 };
```

Classes and Inheritance

Classes

- **Declaration**:

```
class Animal {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
  speak(): void {
    console.log(`${this.name} makes a noise.`);
  }
}
```

- **Instantiation**:

```
const dog = new Animal("Dog");
dog.speak(); // "Dog makes a noise."
```

Inheritance

- **Declaration**:

```
class Dog extends Animal {
  constructor(name: string) {
    super(name);
  }
  speak(): void {
    console.log(`${this.name} barks.`);
```

```
  }
}
```

- **Usage**:

```
const myDog = new Dog("Rex");
myDog.speak(); // "Rex barks."
```

Generics

Basic Usage

- **Declaration**:

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

- **Usage**:

```
let output = identity<string>("Hello");
```

Generic Classes

- **Declaration**:

```
class Box<T> {
  private value: T;
  constructor(value: T) {
    this.value = value;
  }
  getValue(): T {
    return this.value;
  }
}
```

- **Usage**:

```
const box = new Box<number>(123);
console.log(box.getValue()); // 123
```

Modules and Namespaces

Modules

- **Export**:

```
export const PI = 3.14;
export function area(radius: number): number {
  return PI * radius * radius;
}
```

- **Import**:

```
import { PI, area } from './math';
console.log(area(5)); // 78.5
```

Namespaces

- **Declaration**:

```
namespace Geometry {
  export interface Point {
    x: number;
    y: number;
  }
  export function distance(p1: Point, p2: Point): number {
    return Math.sqrt((p1.x - p2.x) ** 2 + (p1.y - p2.y) ** 2);
  }
}
```

- **Usage**:

```
const p1: Geometry.Point = { x: 0, y: 0 };
const p2: Geometry.Point = { x: 3, y: 4 };
console.log(Geometry.distance(p1, p2)); // 5
```

Advanced Types

Union Types

- **Declaration**:

```
let value: string | number;
value = "Hello";
value = 123;
```

Intersection Types

- **Declaration**:

```
type A = { a: number };
type B = { b: string };
type AB = A & B;
```

- **Usage**:

```
let ab: AB = { a: 1, b: "Hello" };
```

Type Guards

- **`typeof`**:

```
function printValue(value: string | number) {
  if (typeof value === "string") {
    console.log("String: " + value);
  } else {
    console.log("Number: " + value);
  }
}
```

- **`instanceof`**:

```
class Dog extends Animal {}
function printAnimal(animal: Animal) {
  if (animal instanceof Dog) {
    console.log("Dog: " + animal.name);
  } else {
    console.log("Animal: " + animal.name);
  }
}
```

Decorators

Class Decorators

- **Declaration**:

```
function logClass(target: Function) {
  console.log(`Class: ${target.name}`);
}
```

- **Usage**:

```
@logClass
class MyClass {}
```

Method Decorators

- **Declaration**:

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

- **Usage**:

```
class MyClass {
  @logMethod
  greet() {
    console.log("Hello!");
  }
}
```

Tips and Tricks

Type Inference

- **Automatic Type Inference**:

```
let count = 10; // TypeScript infers `count` as `number`
```

Type Assertions

- **Usage**:

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

Null and Undefined

- **Strict Null Checks**:

```
let value: string | null = null;
if (value !== null) {
  console.log(value.length);
}
```

Type Compatibility

- **Structural Typing**:

```
interface Point2D {
  x: number;
  y: number;
}
let point: Point2D = { x: 1, y: 2 };
let point3D = { x: 1, y: 2, z: 3 };
point = point3D; // Valid due to structural typing
```

Configuration

`tsconfig.json`

- **Basic Configuration**:

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

Compiler Options

- **`target`**: Specifies ECMAScript target version.

- **`module`**: Specifies module system.
- **`strict`**: Enables all strict type-checking options.
- **`outDir`**: Specifies output directory for compiled files.
- **`rootDir`**: Specifies root directory of input files.

Debugging and Tooling

Debugging with VSCode

- **Launch Configuration**:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "program": "${workspaceFolder}/src/index.ts",
      "outFiles": ["${workspaceFolder}/dist/**/*.js"]
    }
  ]
}
```

ESLint Integration

- **Installation**:

```
npm install eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin --save-dev
```

- **Configuration**:

```
{
  "parser": "@typescript-eslint/parser",
  "plugins": ["@typescript-eslint"],
  "extends": ["eslint:recommended", "plugin:@typescript-eslint/recommended"]
}
```

Best Practices

- **Use `const` by default**: Prefer `const` over `let` for immutability.

- **Leverage Type Inference**: Let TypeScript infer types when possible.

- **Strict Mode**: Always enable `strict` mode in `tsconfig.json`.

- **Use Interfaces for Object Shapes**: Prefer interfaces over type aliases for object shapes.

- **Avoid `any`**: Minimize the use of `any` to maintain type safety.

- **Consistent Naming Conventions**: Follow consistent naming conventions for variables, functions, and classes.

Conclusion

This cheat sheet provides a comprehensive overview of TypeScript's essential features, syntax, and best practices. By mastering these concepts, you can write robust, maintainable, and type-safe code in TypeScript.

By Ahmed Baheeg Khorshid

ver 1.0