TypeScript 4.4 Cheat Sheet

Install	`npm install TypeScript`
Run	`npx tsc`
Run with a specific config	`npx tsc --project configs/my_tsconfig.json`
Triple slash directives	
Reference built-in types	`/// <reference lib="es2016.array.include" />`
Reference other types	`/// <reference path="../my_types" />` `/// <reference types="jquery" />`
AMD	`/// <amd-module name="Name" />` `/// <amd-dependency path="app/foo" name="foo" />`
Compiler comments	
Don’t check this file	`// @ts-nocheck`
Check this file (JS)	`// @ts-check`
Ignore the next line	`// @ts-ignore`
Expect an error on the next line	`// @ts-expect-error`

TypeScript 4.4 Cheat Sheet

Operators (TypeScript-specific and draft JavaScript)

`??` (nullish coalescing)

```
function getValue(val?:
number): number | 'nil' {
  // Will return 'nil' if
`val` is falsey (including 0)
  // return val || 'nil';

  // Will only return 'nil'
if `val` is null or undefined
  return val ?? 'nil';
}
```

`?.` (optional chaining)

```
function countCaps(value?:
string) {
  // The `value` expression
be undefined if `value` is
null or
  // undefined, or if the
`match` call doesn't find
anything.
  return value?.match(/[A-
Z]/g)?.length ?? 0;
}
```

`!` (null assertion)

```
let value: string |
undefined;

// ... Code that we're sure
will initialize `value` ...

// Assert that `value` is
defined
console.log(`value is
${value!.length} characters
long`);
```

`&&=`

```
let a;
let b = 1;

// assign a value only if
current value is truthy
```

TypeScript 4.4 Cheat Sheet

	`a &&= 'default'; // a is still undefined`
	`b &&=  5; // b is now 5`

`||=`	`let a;`
	`let b = 1;`
	
	`// assign a value only if current value is falsy`
	
	`a ||= 'default'; // a is 'default' now`
	`b ||=  5; // b is still 1`

`??=`	`let a;`
	`let b = 0;`
	
	`// assign a value only if current value is null or undefined`
	
	`a ??= 'default'; // a is now 'default'`
	`b ??=  5; // b is still 0`

Basic types	
Untyped	`any`
A string	`string`
A number	`number`
A true / false value	`boolean`
A non-primitive value	`object`

TypeScript 4.4 Cheat Sheet

Uninitialized value	`undefined`
Explicitly empty value	`null`
Null or undefined (usually only used for function returns)	`void`
A value that can never occur	`never`
A value with an unknown type	`unknown`
Object types	
Object	`{ requiredStringVal: string; optionalNum?: number; readonly readOnlyBool: bool; }`

Object with arbitrary string properties (like a hashmap or dictionary)	`{ [key: string]: Type; }`
	`{ [key: number]: Type; }`
	`{ [key: symbol]: Type; }`
	`{ [key: `data-${string}`]: Type; }`

Literal types	
String	`let direction: 'left' | 'right';`
Numeric	`let roll: 1 | 2 | 3 | 4 | 5 | 6;`

TypeScript 4.4 Cheat Sheet

Arrays and tuples	

Array of strings	`string[]`
	or
	`Array<string>`

Array of functions that return strings	`(() => string)[]`
	or
	`{ (): string; }[]`
	or
	`Array<() => string>`

Basic tuples	`let myTuple: [ string, number, boolean? ];` `myTuple = [ 'test', 42 ];`

Variadic tuples	`type Numbers = [number, number];`
	`type Strings = [string, string];`
	
	`type NumbersAndStrings = [...Numbers, ...Strings];`
	`// [number, number, string, string]`
	
	`type NumberAndRest = [number, ...string[]];`
	`// [number, varying number of string]`
	
	`type RestAndBoolean = [...any[], boolean];`
	`// [varying number of any, boolean]`

Named tuples	`type Vector2D = [x: number, y: number];`

TypeScript 4.4 Cheat Sheet

	`function createVector2d(...args: Vector2D) {}` `// function createVector2d(x: number, y: number): void`
Functions	

Function type	`(arg1: Type, argN: Type) => Type;`
	or
	`{ (arg1: Type, argN: Type): Type; }`

Constructor	`new () => ConstructedType;`
	or
	`{ new (): ConstructedType; }`

Function type with optional param	`(arg1: Type, optional?: Type) => ReturnType`
Function type with rest param	`(arg1: Type, ...allOtherArgs: Type[]) => ReturnType`
Function type with static property	`{ (): Type; staticProp: Type; }`
Default argument	`function fn(arg1 = 'default'): ReturnType {}`

Arrow function	`(arg1: Type): ReturnType => { ...; return value; }`
	or
	`(arg1: Type): ReturnType => value;`

TypeScript 4.4 Cheat Sheet

`this` typing	`function fn(this: Foo, arg1: string) {}`

Overloads	`function conv(a: string): number;`
	`function conv(a: number): string;`
	`function conv(a: string | number): string | number {`
	`  ...`
	`}`

Union and intersection types	
Union	`let myUnionVariable: number | string;`
Intersection	`let myIntersectionType: Foo & Bar;`
Named types	

Interface	`interface Child extends Parent, SomeClass {`
	`  property: Type;`
	`  optionalProp?: Type;`
	`  optionalMethod?(arg1: Type): ReturnType;`
	`}`

Class	`class Child extends Parent`
	
	`implements Child, OtherChild {`
	`  property: Type;`
	`  defaultProperty = 'default value';`
	`  private _privateProperty: Type;`

TypeScript 4.4 Cheat Sheet

	`    private readonly _privateReadonlyProperty: Type;`
	`    static staticProperty: Type;`
	
	`    static {`
	`        try {`
	
	`Child.staticProperty = calcStaticProp();`
	`        } catch {`
	
	`Child.staticProperty = defaultValue;`
	`        }`
	`    }`
	
	`    constructor(arg1: Type) {`
	`        super(arg1);`
	`    }`
	
	`    private _privateMethod(): Type {}`
	
	`    methodProperty: (arg1: Type) => ReturnType;`
	`    overloadedMethod(arg1: Type): ReturnType;`
	`    overloadedMethod(arg1: OtherType): ReturnType;`
	`    overloadedMethod(arg1: CommonT): CommonReturnT {}`
	`    static staticMethod(): ReturnType {}`
	`    subclassedMethod(arg1: Type): ReturnType {`
	
	`super.subclassedMethod(arg1);`
	`    }`
	`}`

Enum	`enum Options {`
	`    FIRST,`
	`    EXPLICIT = 1,`
	`    BOOLEAN = Options.FIRST | Options.EXPLICIT,`
	`    COMPUTED = getValue()`
	`}`

TypeScript 4.4 Cheat Sheet

	<pre>enum Colors {
	    Red = "#FF0000",
	    Green = "#00FF00",
	    Blue = "#0000FF"
	}</pre>

Type alias	<pre> type Name = string;
	
	    type Direction = 'left' | 'right';
	
	    type ElementCreator = (type: string) => Element;
	
	    type Point = { x: number, y: number };
	
	    type Point3D = Point & { z: number };
	
	    type PointProp = keyof Point; // 'x' | 'y'
	
	    const point: Point = { x: 1, y: 2 };
	
	    type PtValProp = keyof typeof point; // 'x' | 'y'</pre>

Generics	
Function using type parameters	<pre><T>(items: T[], callback: (item: T) => T): T[]</pre>

Interface with multiple types	<pre>interface Pair<T1, T2> {
	    first: T1;
	    second: T2;
	}</pre>

Constrained type parameter	<pre><T extends ConstrainedType>(): T</pre>

TypeScript 4.4 Cheat Sheet

Default type parameter	`<T = DefaultType>(): T`
Constrained and default type parameter	`<T extends ConstrainedType = DefaultType>(): T`

Generic tuples	`type Arr = readonly any[];`
	
	`function concat<U extends Arr, V extends Arr>(a: U, b: V): [...U, ...V] { return [...a, ...b] }`
	
	`const strictResult = concat([1, 2] as const, ['3', '4'] as const);`
	`const relaxedResult = concat([1, 2], ['3', '4']);`
	
	`// strictResult is of type [1, 2, '3', '4']`
	`// relaxedResult is of type (string | number)[]`

Index, mapped, and conditional types	

Index type query (`keyof`)	`type Point = { x: number, y: number };`
	`let pointProp: keyof Point = 'x';`
	
	`function getProp<T, K extends keyof T>(`
	`  val: T,`
	`  propName: K`
	
	`): T[K] { ... }`

Mapped types	`type Stringify<T> = { [P in keyof T]: string; }` `type Partial<T> = { [P in keyof T]?: T[P]; }`

TypeScript 4.4 Cheat Sheet

Conditional types	`type Swapper = <T extends number | string>`
	`(value: T) => T extends number ? string : number;`
	is equivalent to
	`(value: number) => string`
	if T is number, or
	`(value: string) => number`
	if T is string

Conditional mapped types	`interface Person {`
	`firstName: string;`
	`lastName: string;`
	`age: number;`
	`}`
	
	`type StringProps<T> = {`
	`[K in keyof T]: T[K]`
	`extends string ? K : never;`
	`};`
	
	`type PersonStrings =`
	`StringProps<Person>;`
	
	`// PersonStrings is`
	`"firstName" | "lastName"`

Utility types	

Partial	`Partial<{ x: number; y:`
	`number; z: number; }>`
	is equivalent to
	`{ x?: number; y?: number; z?:`
	`number; }`

Readonly	`Readonly<{ x: number; y:` `number; z: number; }>`

TypeScript 4.4 Cheat Sheet

	is equivalent to `{ readonly x: number; readonly y: number; readonly z: number; }`

Pick	`Pick<{ x: number; y: number; z: number; }, 'x' | 'y'>`
	is equivalent to
	`{ x: number; y: number; }`

Record	`Record<'x' | 'y' | 'z', number>`
	is equivalent to
	`{ x: number; y: number; z: number; }`

Exclude	`type Excluded = Exclude<string | number, string>;`
	is equivalent to
	`number`

Extract	`type Extracted = Extract<string | number, string>;`
	is equivalent to
	`string`

NonNullable	`type NonNull = NonNullable<string | number | void>;`

TypeScript 4.4 Cheat Sheet

	is equivalent to `string | number`

ReturnType	`type ReturnValue = ReturnType<() => string>;`
	is equivalent to
	`string`

InstanceType	`class Renderer() {}`
	`type Instance = InstanceType<typeof Renderer>;`
	is equivalent to
	`Renderer`

Type guards	

Type predicates	`function isThing(val: unknown): val is Thing {`
	`  // return true if val is a Thing`
	`}`
	
	`if (isThing(value)) {`
	`  // value is of type Thing`
	`}`

`typeof`	`declare value: string | number | boolean;`
	`const isBoolean = typeof value === “boolean”;`
	
	`if (typeof value === "number") {`
	`  // value is of type Number`
	`} else if (isBoolean) {`

TypeScript 4.4 Cheat Sheet

	`// value is of type Boolean`
	`} else {`
	`// value is a string`
	`}`

`instanceof`	`declare value: Date | Error | MyClass;`
	
	`const isMyClass = value instanceof MyClass;`
	
	`if (value instanceof Date) {`
	`// value is a Date`
	`} else if (isMyClass) {`
	`// value is an instance of MyClass`
	`} else {`
	`// value is an Error`
	`}`

`in`	`interface Dog { woof(): void; }`
	`interface Cat { meow(): void; }`
	
	`function speak(pet: Dog | Cat) {`
	`if ('woof' in pet) {`
	`pet.woof()`
	`} else {`
	`pet.meow()`
	`}`
	`}`

Assertions	

Type	`let val = someValue as string;`
	
	or
	
	`let val = <string>someValue;`

TypeScript 4.4 Cheat Sheet

Const (immutable value)	`let point = { x: 20, y: 30 } as const;`
	or
	`let point = <const>{ x: 20, y: 30 };`

Ambient declarations	
Global	`declare const $: JQueryStatic;`

Module	`declare module "foo" {`
	`  export class Bar { ... }`
	`}`

Wildcard module	`declare module "text!*" {`
	`  const value: string;`
	`  export default value;`
	`}`