# The TypeScript Handbook

## Literal Types

A literal is a more concrete sub-type of a collective type. What this means is that "Hello World" is a string, but a string is not "Hello World" inside the type system.

There are three sets of literal types available in TypeScript today: strings, numbers, and booleans; by using literal types you can allow an exact value which a string, number, or boolean must have.

## Literal Narrowing

When you declare a variable via var or let, you are telling the compiler that there is the chance that this variable will change its contents. In contrast, using const to declare a variable will inform TypeScript that this object will never change.

// We're making a guarantee that this variable

// helloWorld will never change, by using const.

// So, TypeScript sets the type to be "Hello World", not string

const helloWorld = "Hello World";

// On the other hand, a let can change, and so the compiler declares it a string

let hiWorld = "Hi World";[Try](https://www.typescriptlang.org/play/#code/PTAEHUFMHICdNAWwIYGsCWA7A5qZpsBXZWZTAF0gXIAtlzRb0BnUANxPWQCMAbSAFAhQNSL14B7cBNi8AJqADu6caEyQ2kWKADGdHJAA0obgE9QhZllw6JmZuQB0AoWADKE4wBVTAB0huOrDovgzMkOSstNR+1BImCABEABJikhAy8onGmBJh5ME4Arb2DKLiUpkKALygKWnx0rJyiQDcLsIA8piMoqB5otr6csb4-Aw6ZLr62EZ4mArM8dG6Eoi+KlqgcpA6vCSQrOgM+A6F2ALjIuhN8qC1KegZzW1AA)

The process of going from an infinite number of potential cases (there are an infinite number of possible string values) to a smaller, finite number of potential case (in helloWorld’s case: 1) is called narrowing.

## String Literal Types

In practice string literal types combine nicely with union types, type guards, and type aliases. You can use these features together to get enum-like behavior with strings.

type Easing = "ease-in" | "ease-out" | "ease-in-out";

class UIElement {

animate(dx: number, dy: number, easing: Easing) {

if (easing === "ease-in") {

// ...

} else if (easing === "ease-out") {

} else if (easing === "ease-in-out") {

} else {

// It's possible that someone could reach this

// by ignoring your types though.

}

}

}

let button = new UIElement();

button.animate(0, 0, "ease-in");

button.animate(0, 0, "uneasy");

Argument of type '"uneasy"' is not assignable to parameter of type 'Easing'.Argument of type '"uneasy"' is not assignable to parameter of type 'Easing'.[Try](https://www.typescriptlang.org/play/#code/PTAEAEFMCdoe2gZwFygEwGYAsBWAUAC4CeADpKAKICGiAlgHYDmoAvKAESQ2QC0D7oAD4cuiXnACuBAcM7c+9HpOkBuPHgDGAGxqJQAVQCSFLZAC2kegVABvPKFBV6tM1QKQAFABMAHqnoSZgBGMAA0oF5E-oEh0OGiDIyo1HRMAJS29g6gtABmoB4JTKwsbHJiCuwZdtnZIKAAdE1ZDgC+oJBaYjn5hTSJJWWi4lJVmbXtnd15BUXMpUPyDEqj1S2gk13kNbWg9YYEAOR6JHCIdEGmoAQAFm6giHAWcPTkGpJaXqDQXBo31zdaIh1g56kEiDlGPQEAMiJJoNdSJA9LdJIwbg11q0stjsXhTNYglICC9WKBXgB3AzGUwWKweNJqIkEEn0BpOFxuTwABnCvJES3oVSZxJe7OcrncHn5-PYEleNCIwqAA)

You can pass any of the three allowed strings, but any other string will give the error

Argument of type '"uneasy"' is not assignable to parameter of type '"ease-in" | "ease-out" | "ease-in-out"'

String literal types can be used in the same way to distinguish overloads:

function createElement(tagName: "img"): HTMLImageElement;

function createElement(tagName: "input"): HTMLInputElement;

// ... more overloads ...

function createElement(tagName: string): Element {

// ... code goes here ...

}

## Numeric Literal Types

TypeScript also has numeric literal types, which act the same as the string literals above.

function rollDice(): 1 | 2 | 3 | 4 | 5 | 6 {

return (Math.floor(Math.random() \* 6) + 1) as 1 | 2 | 3 | 4 | 5 | 6;

}

const result = rollDice();[Try](https://www.typescriptlang.org/play/#code/GYVwdgxgLglg9mABAJzgGzQERhApgCgEoAuRARkQB9EAmKxAZnoBZ6BWegNkQG8AoRClxQQyJPgCyAQygALAHTA0cOMkkyFyKWAAmcALZFEAKkSdCiANTkLUgM7l6dak2qtqHapwDcfAL58fBAIdlBCdiBoYQC8KOhYOASE3kA)

A common case for their use is for describing config values:

interface MapConfig {

lng: number;

lat: number;

tileSize: 8 | 16 | 32;

}

setupMap({ lng: -73.935242, lat: 40.73061, tileSize: 16 });[Try](https://www.typescriptlang.org/play/#code/PQKhAIGECcFMEMAusDO57gLbwA7gMawB2ycAJuouADYD2+w1S4IwAUGbPk3OAGYBXIvkQBLWkXApYiATgCyuABT4JfUQHMAXOEU5IazQEodAN1qiyAbjbBg4ALRP8AxE4dtRJWND7xCurgGROoa4ADebOA0RNrgRAKYAEY+NtFMiDoJyalR4GLUsADKogBesDoAHOAAPuAAjABsteAAzABMNgC+bGzSsgrK4TFxDgDsrQB0AJytAKztACztADQ0SDqLAAyTE1uN9WsFxWUVDc1dRlZAA)

## Boolean Literal Types

TypeScript also has boolean literal types. You might use these to constrain object values whose properties are interrelated.

interface ValidationSuccess {

isValid: true;

reason: null;

}

interface ValidationFailure {

isValid: false;

reason: string;

}

type ValidationResult = ValidationSuccess | ValidationFailure;