String Literal and Overload Function

In this lesson, we approach the string literal with overload function

WE'LL COVER THE FOLLOWING

Parameter with a String Literal

A function can use a *string literal* in its parameter to know which return type is possible. For example, it's possible to have a function with a string parameter and have many overloads where a specific string value will cause execution of one function or another.

In the example below, the method accepts only the strings `"batman"` and `"superman"` even if the signature shows string. See **line 12** and **line 13**. TypeScript unites both overloads to create a *string literal* of all the overloads.

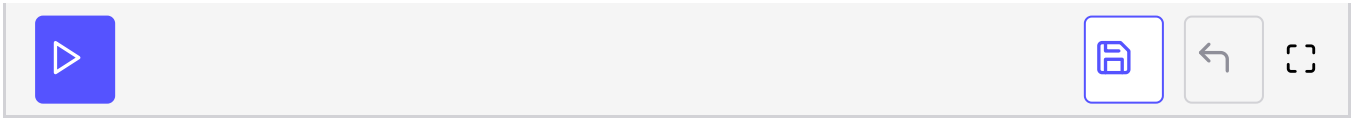
For example, if you are using an IDE that supports TypeScript, the IDE will show the return type `Batman` if the parameter is `"batman"`. **Line 14** combines all the overload.

```
interface SuperHero {
    attackName: string;
}
interface Batman extends SuperHero {
    jumpLength: number;
}
interface SuperMan extends SuperHero {
    flyingSpeed: number;
}

function createSuperHero(name: "batman"): Batman;
function createSuperHero(name: "superman"): SuperMan;
function createSuperHero(name: string): Batman | SuperMan | SuperHero {
    if (name === "batman") {
```

```
        return {
            attackName: "Kick",
            jumpLength: 12,

        };
    } else if (name === "superman") {
        return {
            attackName: "Punch",
            flyingSpeed: 120,
        };
    }
    return {
        attackName: "Run",
    };
}
const hero1 = createSuperHero("batman");
console.log(`Batman can jump ${hero1.jumpLength} feet`);
const hero2 = createSuperHero("superman");
console.log(`Superman can fly ${hero2.flyingSpeed} miles per hour`);
```



Advantage of String Literal

The main advantage is that inside a scope where the *string literal* is compared adjust the parameters to the definition of the *string literal*. At **line 15** the `name` compares the string `batman`. When the value is `batman`, all the closure, from **line 16** to **line 19** acts like if the function signature was only **line 12** returning the type `Batman`.