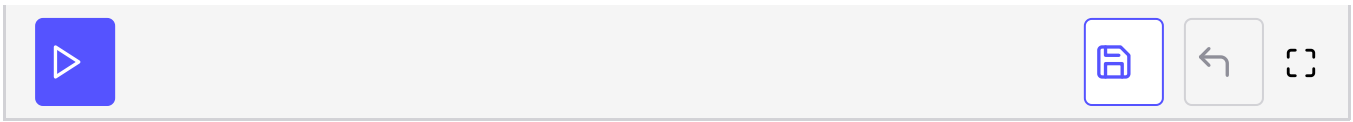
Intersecting with Types, Interfaces, and Generics

In this lesson, we will look at how to intersect types, interfaces and generics.

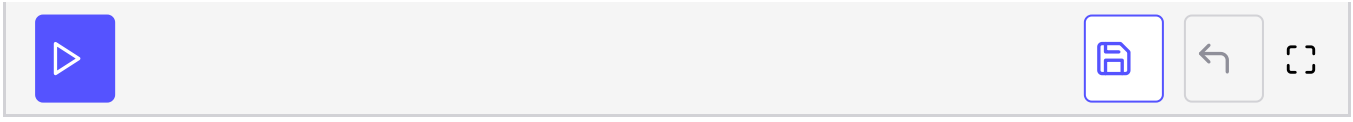
TypeScript can manipulate types by combining any of them in one of several ways. The first way is to specify a type to be an intersection type, using an ampersand `&`. In the end, the type will have the sum of all members.

```
type T1 = { x: string };
type T2 = { y: number };
type T3 = { z: boolean };
type IntersectType1 = T1 & T2 & T3;
type IntersectType2 = T1 & T2;
type IntersectType3 = T3 & T1;
const inter1: T1 = { x: "x1", y: 2 }; // Won't compile: y does not exist in T1
const inter2: T1 & T2 = { x: "x1", y: 2 }; // Compile
const inter3: IntersectType2 = { x: "x1", y: 2 }; // Compile
```



Interfaces work well with intersecting as well.

```
interface I1 {
    x: string;
}
interface I2 {
    y: number;
}
interface I3 {
    z: boolean;
}
type IntersectWithInterface = I1 & I2 & I3;
const with3Interfaces: IntersectWithInterface = { x: "1", y: 1, z: true };
```



Or it can be used for extending other interfaces. However, intersect shines with a dynamic composition of types. The inheritance of interfaces would require creating additional interfaces to inherit the existing interface and

eventually pollute the codebase, depending on what must be combined.

```
    interface CombineI1 extends I1, I2, I3 {}
    interface CombineI2 extends I1, I2 {}
    interface CombineI3 extends I3, I1 {}
```

Setting the result of intersecting to a variable is more reusable and pragmatic than inheritance. However, it is also convenient to use an intersect on-the-fly for a parameter or a return type.

```
function intersectOnTheFly(t: I1 & I2){ }
```

Intersect and generic types can be combined to give flexibility. In the following example, the function is generic and combines the two generic types in the return type.

```
function intersectGeneric<TT1, TT2>(t1: TT1, t2: TT2): TT1 & TT2 {
    const returnValue = <TT1 & TT2>{};
    for (let index in t1) {
        (returnValue as TT1)[index] = t1[index];
    }
    for (let index in t2) {
        (returnValue as TT2)[index] = t2[index];
    }
    return returnValue;
}
```