Aliases with Generic Types and Recursivity

In this lesson we will see how to use an alias with generic types and recursivity.

WE'LL COVER THE FOLLOWING

- Generic types and aliases
- Recursive type with aliases
- Type intersect

Generic types and aliases

A generic type can also leverage an alias. It is possible to assign a name to a generic structure, similarly as we can do with an interface. Using the less-than `<` and, greater-than `>` signs as with an interface, a generic type can be specified to a different type.

At **line 1** and **line 7**, both interfaces have a generic type named `TOption` to define a type within the interface.

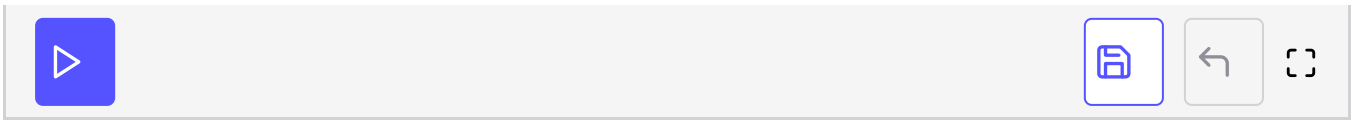
```
interface Car<TOption> {
  name: string;
  options: TOption
}


type Car2<TOption> = {
  name: string;
  options: TOption
};

let c1: Car<string> = { name: "Vroom", options: "fast" };
let c2: Car2<string> = { name: "Pouttt", options: "slow" };

// Interchangeable because it is the same structure:
c1 = c2
```

Recursive type with aliases

Aliases are a great way to self-reference a type. For example, if you have a `Group` that can have nested groups inside the main group, then you can use an alias that refers to itself.

The example has at **line 3** a reference to the type itself.

```
type Group = {
  name: string;
  child?: Group; // Refer to itself
};

let g: Group = {
  name: "G1",
  child: {
    name: "Generation1",
    child: {
      name: "Generation2"
    }
  }
};
```



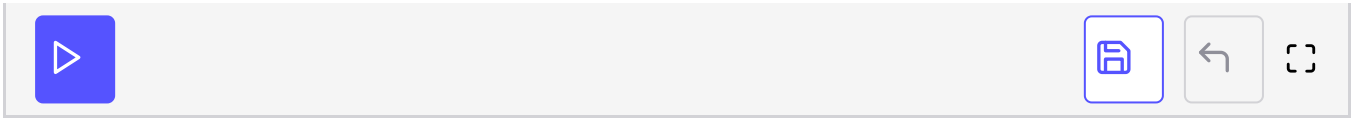
Type intersect

Any type can be intersected with another type, creating a junction of many types. The merging of several types is similar to an interface that extends many interfaces.

The following code has at line 14 an intersection with `Child<Person>` and to `Person`.

```
type Group = {
  name: string;
}

type Person = {
  firstName: string;
  lastName: string;
}

type Child<T> = {
  child: T;
}

type PersonGroupWithChild = Group & Person & Child<Person>;
```

```
let p: PersonGroupWithChild = {
  name: "GroupName",
  firstName: "Fist",
  lastName: "Last",
  child: {
    firstName: "ChildFirst",
    lastName: "ChildLast"
  }
};

console.log(p);
```



Generic type and aliases work well together and are often required for a type that must include a nested member that is of the same type.