Generic with Construction Functions

This lesson explains how to apply generic to a construction function.

WE'LL COVER THE FOLLOWING

- Restricting the signature of a constructor

Restricting the signature of a constructor

In some scenarios, someone may need to instantiate a generic object from a function.

```
const obj = funct(TypeIWantToInstantiate, parameter1, parameterx);
```

The goal is to have a *type constraint* on the construction of an object type.

It is possible to use the *constructed signature* in a **construction function** to dynamically create a new instance of a specific type that requires an **exact** constructor signature.

> A construction function is a function that returns a *new* object.

> A constructor signature is the type and number of the parameters.

For example, a function taking `T` as a generic argument can require having its argument of type `{new(): T; }` and returns `T`. An interface with a member of `new` indicates that the class must have a constructor with the defined parameter. In the following code, **line 2** constrains any type passed to have a constructor with a single parameter of type `string`.

```
interface IMyInterfaceWithConstructor<T> {
```

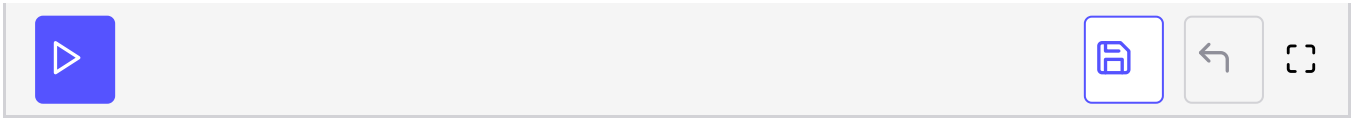
```
    new(param: string): T; // Force to have a constructor with the signature of 1 parameter t
}

function createInstance<T>(ctor: IMyInterfaceWithConstructor<T>, param1: string): T { // Crea
    return new ctor(param1);
}

class C1 {
    constructor(name: string) { // We can create from createInstance function because 1 param
        console.log("Initializing class C1 with string: " + name);
    }
}

class C2 {
    constructor(name: string) { // We can create from createInstance function because 1 param
        console.log("Initializing class C2 with string: " + name);
    }
}

const inst1 = createInstance(C1, "Instance 1");
const inst2 = createInstance(C2, "Instance 2");
```



Line 5 is the function that creates an instance of a class. The constructor function has the specification of having two parameters. The first one is the constraint. It specifies a class with a constructor defined by the interface previously defined to have a single parameter constructor.

Classes `C1` and `C2` are both respect that constraint and both have a constructor that takes a single parameter of type `string`. **Line 21** and **line 22** call the function that instantiates respectively `C1` and `C2` as it is the type passed in the first parameter. The second parameter could be anything, including nothing. In this example, it is just passing the parameter into the creating function that passes it down to the actual class' constructor.