New Object

This lesson goes over the creation of an object with the keyword new.

WE'LL COVER THE FOLLOWING

Instantiating a class into an object

TypeScript can create an object by instantiating a `class` using the keyword `new`. The scope of this lesson does not cover all object-oriented possibilities but in a nutshell, TypeScript lets you use the full set of object-oriented features, even if targeting ECMAScript before 2015 which does not contain Object or literal object.

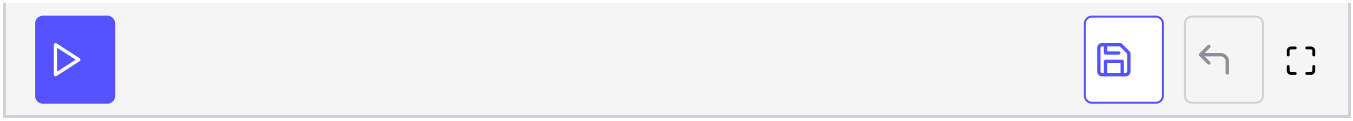
This means that if you want to use polymorphism, have multiple instances of a type, use a decorator, be able to use a pattern like the dependency of injection, or even just be able to mock specific functions of an object, the use of a class is mandatory. Creating an object with `new` creates an instance of an object which inherits all characteristics of the uppercase `Object`.

```
class MyClass{
    private value: string;
    constructor(val:string){
        this.value = val;
    }
}
const c = new MyClass("ABC");
```

The example above shows that to instantiate a class, a class must first exist. An instance means creating an object with `new`. Every instance is unique and

isolated. The following example creates two instances of `MyClass`. They are both objects. When comparing each of them, they return `false` because they are pointing to two different memory pointers. Changing the value of one object does not affect the second.

```
class MyClass{
    public value: string;
    constructor(val:string){
        this.value = val;
    }
}
const c1 = new MyClass("ABC");
const c2 = new MyClass("ABC");
console.log(c1===c2);
c1.value = "1";
console.log(c1.value);
console.log(c2.value);
```



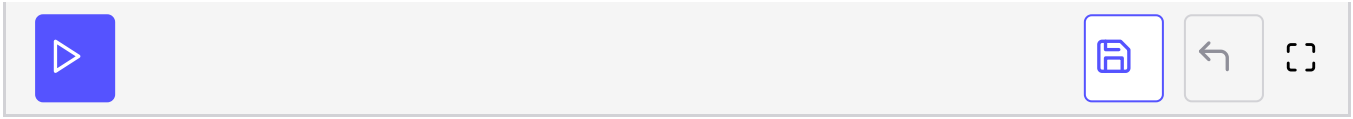
Object-oriented possibilities

Creating an object with the keyword `new` opens all the functionalities that are available in object-oriented programming. Without going into detail, an object with `new` can follow a dictated structure while being from different types of classes by implementing an interface.

In the following example, we see that the function at **line 13** takes an interface (the contract) of an object that has an `id` of type `number`. The interface is defined at **line 1**. Two different contracts are passed at **line 19** and **line 20**. The function does not distinguish the difference because it uses `MyContract` that both types share. Thus, **line 14** circumscribes the property that can be invoked to only `id`.

```
interface MyContract {
  id: number;
}

class ContractTypeA implements MyContract {
  constructor(public id: number){}
}

class ContractTypeB implements MyContract {
  constructor(public id: number){}
}

function showContractId(c: MyContract): void {
```

```
  console.log(c.id);
}

const c1 = new ContractTypeA(1);
const c2 = new ContractTypeB(2);
showContractId(c1);
showContractId(c2);
```



The value of object-oriented is that it brings the widest array of possibilities. It is possible to override functions, overload functions, abstract functions, having constructors, static functions, etc. All that with the help of `new`.