Lowercase vs UpperCase Object

This lesson will teach you how to differentiate object and Object.

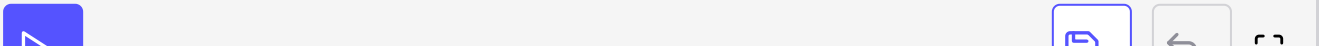
WE'LL COVER THE FOLLOWING

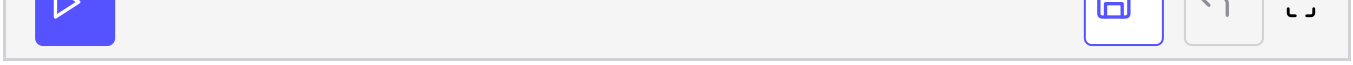
Lowercase `object`

With TypeScript 2.2, the lowercase `object` type was born. The lowercase `object` type contains all types that are *not primitive*. It's easy to confuse this with the uppercase `Object`, which every kind of object has in common. Here is a list of the types that are *not* an object:

1. undefined
2. null
3. number
4. string
5. Boolean
6. symbol

```
let o: object;
o = 1; // Primitive = does not work
o = { a: "123" }; // Anonymous object = work
interface MySchema {
    val: string;
}
let interfaceObject: MySchema = { val: "Test" };
o = interfaceObject; // Typed object = work
o = null; // Does not work
o = undefined; // Does not work
let x = new Array();
o = x; // "new" object = work
```





Several examples that transpile and does not transpile

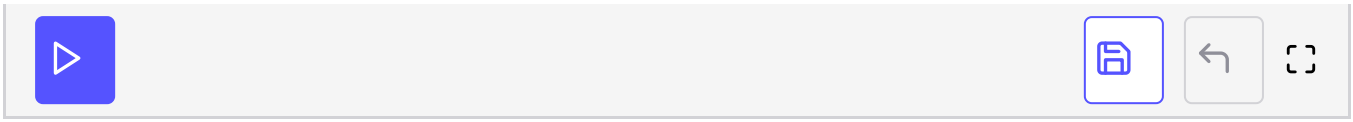
> 📜**Note:** The code above does not transpile ✕

The code above defines a single variable `o` of type `object`. Many different assignations occur from a `number` at **line 2** that does not compile to a curly object that compile. Follow along with the code and check the comments to see many examples of type that transpile successfully or not.

Uppercase `Object`

The uppercase `Object` is the one from JavaScript that gives basic functions like `toString`, `hasOwnProperty`, etc. These functions are also available naturally via the prototype chain if you are using curly braces to create an empty object. Direct use of `Object` is rare. However, when you instantiate a class, it summons a new uppercase `Object`.

```
let x = { y: 1 };
let obj: Object = x;
console.log(obj.toString());
```

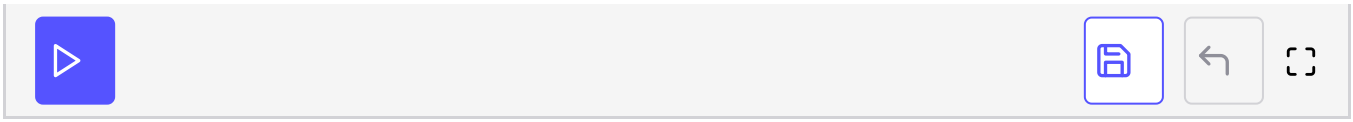


Deciding between the lowercase `object` and uppercase `Object` can be confusing. The rule of thumb is to use the lowercase object when a non-primitive is required. The use of the uppercase `Object` is to access one of the members from the ECMAScript Object, like:

1. `hasOwnProperty`
2. `toString`
3. `isPrototypeOf`
4. `propertyIsEnumerable`
5. `toLocaleString`
6. `watch`

7. `unwatch`
8. `toSource`
9. `valueOf`

```
print(1);
print({ id: 1 });
print({ id: 2, y: 1 });
print({ id: 3, y: 2 });
function print(o: Object) {
    if (o.hasOwnProperty("y")) {
        console.log(o);
    } else if (typeof o === "number"){
        console.log("Number is " + o);
    }
}
```



The use of `Object` is preferred to `any` or `unknown` because you still have access to a subset of functions. As seen in the previous example, it is possible to use `hasOwnProperty` which would not have been possible without casting with `any`. With `Object`, your IDE will provide the Object's properties as well.