Mistake: treating a value type as a reference type

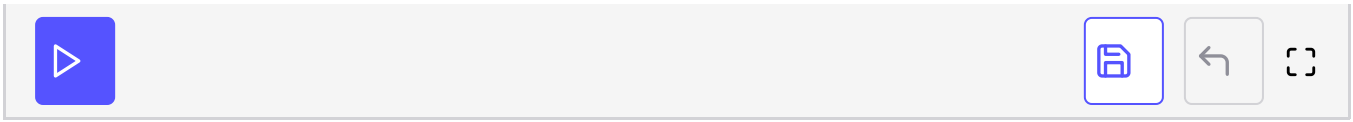
This lesson will show two erroneous attempts of assuming a value type as a reference type.

Suppose that the `increaseAge` method was erroneously implemented like this:

```
var people = [
    { name: 'John Hill', age: 22 },
    { name: 'Jack Chill', age: 27 }
];

function increaseAge( age ) {
    age += 1;
}

console.log( 'Before:', people[0].age );
increaseAge( people[0].age );
console.log( 'After:', people[0].age );
```



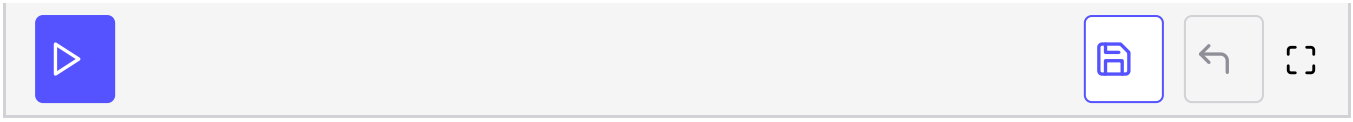
The age stays `22` even after executing the `increaseAge` function. Primitive types are passed by value. Changing the copied value has no effects on the original value. Always encapsulate the fields you would like to change with an array or an object.

Another example is an erroneous attempt to replace a person with another one. Suppose that the `replacePerson` function accepts 3 arguments: the person to be replaced, and the name and age of the new person.

```
var people = [
    { name: 'John Hill', age: 22 },
    { name: 'Jack Chill', age: 27 }
];

function replacePerson( person, name, age ) {
    person = { name: name, age: age };
}

replacePerson( people[1], 'Jack Newtown', 35 );
console.log( people );
```



This function does not do anything to the `people` object as the `person` argument was assigned a new value. The `people` data structure stays intact. The `person` handle inside the function references a completely new object after the assignment. Once this function terminates, the new object is thrown away.