LESSON SC 3 – **Conditional Logic and Loops**

University of West Attica

Department of Electrical and Electronics Engineering

Ioannis Christidis

Christoforos Kachris

Support by Ethereum Foundation ESP

What will we accomplish!

This is the second lesson for our project *Instant Car Renting*.

In this lesson we will learn about *conditional logic and loops in solidity.*

We will continue on our `ICR.sol`.

Conditional Logic: `if, else if, else`

The *`if` statement allows you to execute a block of code if a specific condition evaluates to true*. Solidity also supports *else if and else for additional conditions and default behavior.*

```
if (condition) {
    // Executes if condition is true
} else if (anotherCondition) {
    // Executes if anotherCondition is true
} else {
    // Executes if no condition is true
}
```

For our ICR SC, we will create a function to change a car's `occupied` to `true` or `false` depending on its previous state.

`If car is occupied => make it unoccupied`

`If car is unoccupied => make it occupied`

Conditional Logic: `if, else if, else` (2)

```
    function changeCarOccupie(uint256 _carId) public {
        Car storage car = cars[_carId]; // find the car on the mapping
        if(car.occupied) { // if it is occupied
            car.occupied = false; // make it unoccupied
        } else { // else
            car.occupied = true; // make it occupied
        }
    }
```

The function is called `changeCarOccupie` and *it takes as input the ID of the car we want to change its state.* Since we want to change the state, we will make a transaction when calling it.

To permanently make a change to an existing entry on our mapping we will need to use the `storage` keyword. `storage` is a keyword that we use when we want to make a permanent change in general. *Any state variable we define in a SC is stored in `storage`.*

So we find the `Car` with `carId` and we name it `car`. Now, we can access its properties by calling `car.property`. In our example we can use it the `occupied bool`:

```
car.occupied
```

Conditional Logic: `if, else if, else` (3)

Previously, we mentioned that *if we try to get the value by using a key that does not have a value yet, we will still get a result, but it will be the default value of that data type*.

In this situation this is dangerous since *if we add as a parameter an ID of a car that has not yet been rented it will change its* `occupied` *property to* `true` *since a* `bool` *default value is* `false`*.*

To avoid that, we will add another `if` statement that *checks if the ID we provide as input is less than the* `nextCarId` *meaning that the* `car` *was already registered*. If it is, then, we will change its state.

```
    function changeCarOccupie(uint256 _carId) public {
        if(_carId < nextCarId){ // check if the car is registered
            Car storage car = cars[_carId];
            if(car.occupied) {
                car.occupied = false;
            } else {
                car.occupied = true;
            }
        }
    }
```

Loops

Solidity supports three types of loops:

`for` Loop: Repeats a block of code a specific number of times.

```
for (initialization; condition; increment/decrement) {
    // Code to execute
}
```

`while` Loop: Repeats a block of code as long as a condition is true.

```
while (condition) {
    // Code to execute
}
```

`do...while` Loop: Executes the block at least once and then checks the condition.

```
do {
    // Code to execute
} while (condition);
```

Important Considerations for Loops in Solidity

Gas Costs:

Each iteration of a loop consumes gas. For long-running loops, this can exceed the ***block gas limit****, causing the transaction to fail.

Avoid unbounded loops (e.g., while (true) or for loops with unknown termination).

Avoid Complex Logic:

Complex conditions or nested loops can increase gas usage significantly.

Use Alternatives When Possible:

If possible, *use mappings or external tools to perform complex operations off-chain.*

For these reasons we will not use any loops in our code.

***The block gas limit** is the maximum amount of computational work, expressed in gas units, that can be included in a single block by the network.

Outro

Finished the conditional logic and loops.

We also learned about storage!

Next, we will go over the concept of `enum`.