**Solidity for Loop**

**Introduction:**

In this tutorial, we’ll set up the `withdraw` function to enable the retrieval of accumulated funds. We’ll then reset all the mappings to zero using a method known as a for loop.

**For Loops:**

A for loop is a widely-used concept in many programming languages that enables the execution of a block of code a repeated amount of times.

For example, consider this list:

//[1, 2, 3, 4] elements

// 0 1 2 3 indexes

The elements of the list are the numbers 1 through 4, with indices ranging from 0 to 3: the number 1 is at index 0, the number 2 is at index 1, and so on. To access all the elements in this list, we can use their indexes.

A typical `for` loop structure in programming languages involves:

1. Initializing at some starting index
2. Iterating until an end index
3. Incrementing by certain steps

For instance, if we start the loop at index 0, end at index 10, and increment by 1 each time, we’ll get:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

However, if we start at index 3, end at index 12, and increment by 2 each time, we get:

3, 5, 7, 9, 11

**Using for Loops in the FundMe Contract:**

To implement this concept in the `FundMe` contract:

Uint256 funderIndex;

For (funderIndex = 0; funderIndex < funders.length; funderIndex++) {

Address funder = funders[funderIndex];

addressToAmountFunded[funder] = 0; }

The loop begins at index 0 and goes through all the elements in the `funders` array until it reaches the final element. With each iteration, it performs the following actions:

1. Accesses the `funder` address at the current index
2. Resets the corresponding funding amount in the `addressToAmountFunded` mapping to zero, clearing the `funder`’s record.

**Note:** The addressToAmountFunded map connects addresses with the respective amounts they funded.

**Shortcuts:**

Additionally, we have used two shorthands in our code:

1. `funderIndex++`: shorthand for `funderIndex = funderIndex + 1`.

2. `+=`: adds a value to an existing one. `x = x + y` is equivalent to `x += y`.

To illustrate the code snippet, we start from `funderIndex` 0. When entering the loop, we get the address of the funder at the 1st position in the `funders` array and set its amount to zero. After that, we repeat the loop, incrementing the `funderIndex` by 1 and checking whether it is still less than the total number of funders. We then get the address of the funder at the next position and so on.

**Conclusion:**

In this lesson, we learned how to implement the `withdraw` function to manage accumulated funds and reset mappings using a `for loop`. This process ensures efficient handling of funders’ records.