**Arrays And Structs**

**Introduction:**

Up to this point, the `SimpleStorage` contract allows for storing, updating, and viewing a single favorite number. In this lesson, we’ll enhance the code to store multiple numbers, enabling more than one person to store their values. We’ll learn how to create a list of favorite numbers using arrays, and we’ll explore the `struct` keyword for creating new types in Solidity.

**Arrays and struct:**

First we need to replace the `uint256 favoriteNumber` with a list of `uint256` numbers:

uint256[] list\_of\_favorite\_numbers;

The brackets indicate that we have a list of `uint256`, an array of numbers. If we want to initialize this array we can do so by specifying its content:

uint256[] list\_of\_favorite\_numbers = [0, 78, 90];

**Note:** Arrays are zero-indexed: the first element is at position zero (has index 0), the second element is at position one (has index 1), and so on.

The Issue with this data structure is that we cannot link the owner with its favorite value. One solution is to establish a \*\*new type\*\* using the `struct` keyword, named `Person`, which is made of two \_attributes\_: a favorite number and a name.

The Issue with this data structure is that we cannot link the owner with its favorite value. One solution is to establish a “new type” using the `struct` keyword, named `Person`, which is made of two attributes: a favorite number and a name.

struct Person {

Uint256 my\_favorite\_number;

String name;

}

**warning:** Rename the variables `favorite\_number` to avoid name clashes

From this struct, we can instantiate a variable `my\_friend` that has the type `Person`, with a favorite number of seven and the name ‘Pat’. We can retrieve these details using the getter function that was generated by the `public` keyword.

Person public my\_friend = Person(7, ‘Pat’);

/\* equals to

Person public my\_friend = Person({

Favorite\_number:7,

Name:’Pat’});

\*/

**Array of struct:**

Creating individual variables that represent several people might become a tedious task, due to the repetitive steps of the process. Instead of manually instantiating a variable for each person, we can combine the two concepts we just learned about: arrays and structs.

Person[] public list\_of\_people; // this is a dynamic array

Person[3] public another\_list\_of\_three\_people; // this is a static array

When using a dynamic array, we can add as many `Person` objects as we like, as the size of the array it’s not static but can grow and shrink. We can access each `Person` object in our array by its index.

To add people to this list, we can create a function that populates the array:

function add\_person(string memory \_name, uint256 \_favorite\_number) public {

List\_of\_people.push(Person(\_favorite\_number, \_name));

}

`add\_person` is a function that takes two variables as input – the name and favourite number of the person. It creates first a new `Person` object and then it pushes it to our `list\_of\_people` array.

**Conclusion:**

With these features, our Solidity contract can now store multiple favorite numbers, each associated with a specific person. The `add\_person` function creates a new `Person` struct and adds it to the `list\_of\_people` state variable. We can then view each person’s name and favorite number by accessing the `Person` object through the array index.