**The modulo operation**

**Understanding Modulo**

We ended the previous lesson when we defined the following function:

function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords) internal override {}

As we've said before, this function is going to be called by the VRF service. Here we will be given 1 random word (1 because of the NUM\_WORDS we defined in the previous lesson). This isn't a word as in a string of letters like pizza, this is a big and random uint256. Being a number we can use it to do math.

What we need is to use the modulo operator denoted as % in Solidity.

The modulo operation (often abbreviated as "mod") is a mathematical operation that finds the remainder when one integer is divided by another. In other words, given two numbers, a and b, the modulo operation a % b returns the remainder of the a / b division.

The modulo operation can be calculated using this equation:

a % b = a - floor(a / b) \* b

Examples:

5 % 2 = 1 // Because 5 is 2 \* 2 + 1

11 % 3 = 2 // Because 11 is 3 \* 3 + 2

159 % 50 = 9 // Because 153 is 50 \* 3 + 9

1000 % 10 = 0 // Because 1000 is 100 \* 10 + 0

We are going to use this function to pick a random winner.

Let's say we have 10 players (s\_players.length = 10).

Now let's say Chainlink VRF sends back the number 123454321 (I know, super random).

Given that the % 10 operation can yield a value between [0:9] we can use the result of the randomNumber % 10 as the s\_players index corresponding to the winner.

Using the actual numbers:

123454321 % 10 = 1

This means that the player with index 1 (s\_players[1]) is the winner of our raffle! The random number will always be different and sufficiently large. Using s\_players.length will ensure that we always include all the players who paid a ticket. Perfect!

**Picking the winner**

Enough theory, let's implement it in code!

function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords) internal override {

uint256 indexOfWinner = randomWords[0] % s\_players.length;

address payable winner = s\_players[indexOfWinner];

}

Now let's record this last winner in state and send them their prize.

function fulfillRandomWords(uint256 requestId, uint256[] memory randomWords) internal override {

uint256 indexOfWinner = randomWords[0] % s\_players.length;

address payable winner = s\_players[indexOfWinner];

s\_recentWinner = winner;

(bool success,) = winner.call{value:address(this).balance}("");

if (!success) {

revert Raffle\_\_TransferFailed();

}

}

Let's define the Raffle\_\_TransferFailed() custom error and the s\_recentWinner variable in the state variables section.

error Raffle\_\_NotEnoughEthSent();

error Raffle\_\_TransferFailed();

// Raffle related variables

uint256 private immutable i\_entranceFee;

uint256 private immutable i\_interval;

uint256 private s\_lastTimeStamp;

address payable[] private s\_players;

address payable private s\_recentWinner;

Amazing! Let's keep going!