**Smart contracts events**

**Tracking participants**

Ok, our user paid the entrance fee, but how do we track his registration? We can't simply take the money and run! We need a storage structure that keeps track of all registered users from where to pick the winner.

Take a moment and decide what would be the best from the following:

1. Mapping
2. Array
3. A bunch of address variables and limit the number of participants

. . .

Congratulations if you chose the **Array** option! To be more specific, a dynamic array that grows in size with each new participant. Mappings can't be looped through, and a bunch of address variables is not feasible.

Add the array below the i\_entranceFee declaration: address payable[] private s\_players;

We've made it address payable because one of the participants registered in this array will be paid the ETH prize, hence the need for the payable attribute.

Back in the enterRaffle function, we need to add the address that paid into the s\_players array:

function enterRaffle() external payable {

if(msg.value < i\_entranceFee) revert Raffle\_\_NotEnoughEthSent();

s\_players.push(payable(msg.sender));

}

The .push method is used to append an element to an array, increasing its length by 1.

s\_players.push(payable(msg.sender)); performs a modification of the state by adding the payable address msg.sender in the array. It is customary to emit an **event** every time we perform a state modification.

**Events**

Events are a way for smart contracts to communicate with the outside world, primarily with the front-end applications that interact with these contracts. Events are logs that the Ethereum Virtual Machine (EVM) stores in a special data structure known as the blockchain log. These logs can be efficiently accessed and filtered by external applications, such as dApps (decentralized applications) or off-chain services. The logs can also be accessed from the blockchain nodes. Each emitted event is tied up to the smart contract that emitted it.

Please click [here](https://docs.soliditylang.org/en/v0.8.25/contracts.html#events) to find out more about events.

How can we use events?

Imagine we have a more complex function that changes an important parameter, let's say we are recording the exchange rate of BTC/USDC. We change it by calling the function changeER(). After we perform the call and the exchange rate is changed we need to make sure this also gets picked up by our front-end. We make the front-end listen for the BTCUSDCupdated event. An example of that event could be this:

event BTCUSDCupdated(

uint256 indexed oldER,

uint256 indexed newER,

uint256 timestamp,

address sender

)

You will see that some of the emitted parameters are indexed and some are not. Indexed parameters, also called topics, are much more easy to search than non-indexed ones.

For an event to be logged we need to emit it.

Let's come back to our Raffle contract where we'll also learn how to emit them.

First, we define the event (be mindful of where the events should go in terms of our defined layout)

event EnteredRaffle(address indexed player);

Then, we emit it in enterRaffle:

function enterRaffle() external payable {

if(msg.value < i\_entranceFee) revert Raffle\_\_NotEnoughEthSent();

s\_players.push(payable(msg.sender));

emit EnteredRaffle(msg.sender);

}

Great! I know there is a possibility you don't quite understand the importance/usage of this event, but don't worry, we'll get back to it in the testing section.

But before that, let's discuss randomness.