**Creating custom errors**

**Implementing the entranceFee**

Great! Let's move on with writing the contract.

Previously we defined the i\_entranceFee variable. This is the amount the user has to send to enter the raffle. How do we check this?

function enterRaffle() external payable {

require(msg.value >= i\_entranceFee, "Not enough ETH sent");

}

First, we changed the visibility from public to external. external is more gas efficient, and we won't call the enterRaffle function internally.

We used a require statement to ensure that the msg.value is higher than i\_entranceFee. If that is false, we will yield an error message "Not enough ETH sent".

**Note: The require statement is used to enforce certain conditions at runtime. If the condition specified in the require statement evaluates to false, the transaction is reverted, and any changes made to the state within that transaction are undone. This is useful for ensuring that certain prerequisites or validations are met before executing further logic in a smart contract.**

In Solidity 0.8.4 a new and more gas-efficient way has been introduced.

**Custom errors**

[Custom errors](https://docs.soliditylang.org/en/v0.8.25/contracts.html#errors-and-the-revert-statement) provide a way to define and use specific error types that can be used to revert transactions with more efficient and gas-saving mechanisms compared to the require statements with string messages. If you want to find out more about how custom errors decrease both deploy and runtime gas click [here](https://soliditylang.org/blog/2021/04/21/custom-errors/).

I know we just wrote this using the require statement, we did that because require is used in a lot of projects, that you might get inspiration from or build on top of and so on. But from now on we will perform checks using the if statement combined with custom errors.

We will refactor enterRaffle, but before that let's define our custom error. Be mindful of the layout we talked about in the previous lesson

error Raffle\_NotEnoughEthSent();

function enterRaffle() external payable {

// require(msg.value >= i\_entranceFee, "Not enough ETH sent!");

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

}

You will see that we named the custom error using the Raffle\_\_ prefix. This is a very good practice that will save you a ton of time when you need to debug a protocol with 20 smart contracts. You will run your tests and then ask yourself Ok, it failed with this error ... but where does this come from?. Because you thought ahead and used prefixes in naming your error you won't have that problem! Awesome!

**Note:**

**In Solidity, like in many other programming languages, you can write if statements in a single line for brevity, especially when they are simple and only execute a single statement. This is purely a stylistic choice and does not affect the functionality or performance of the code.**

There is no difference between this:

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

and this:

if(msg.value < i\_entranceFee) {

revert Raffle\_\_NotEnoughEthSent();

}

Amazing work! Let's learn about events!

**From Soidity 0.8.26**

The example below shows custom error usage with the revert statement in function transferWithRevertError, as well as the newer approach with require in function transferWithRequireError.

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.26;

/// Insufficient balance for transfer. Needed `required` but only

/// `available` available.

/// @param available balance available.

/// @param required requested amount to transfer.

error InsufficientBalance(uint256 available, uint256 required);

// This will only compile via IR

contract TestToken {

mapping(address => uint) balance;

function transferWithRevertError(address to, uint256 amount) public {

if (amount > balance[msg.sender])

revert InsufficientBalance({

available: balance[msg.sender],

required: amount

});

balance[msg.sender] -= amount;

balance[to] += amount;

}

function transferWithRequireError(address to, uint256 amount) public {

require(amount <= balance[msg.sender], InsufficientBalance(balance[msg.sender], amount));

balance[msg.sender] -= amount;

balance[to] += amount;

}

// ...

}