pragma solidity ^0.5.0;

// lvl 1: equal split

contract AssociateProfitSplitter {

// @TODO: Create three payable addresses representing `employee\_one`, `employee\_two` and `employee\_three`.

address payable employee\_one;

address payable employee\_two;

address payable employee\_three;

constructor(address payable \_one, address payable \_two, address payable \_three) public {

employee\_one = \_one;

employee\_two = \_two;

employee\_three = \_three;

}

function balance() public view returns(uint) {

return address(this).balance;

}

function deposit() public payable {

// @TODO: Split `msg.value` into three

uint amount = msg.value/3;

// @TODO: Transfer the amount to each employee

employee\_one.transfer(amount);

employee\_two.transfer(amount);

employee\_three.transfer(amount);

// @TODO: take care of a potential remainder by sending back to HR (`msg.sender`)

msg.sender.transfer(msg.value-amount\*3);

}

function() external payable {

// @TODO: Enforce that the `deposit` function is called in the fallback function!

deposit();

}

}

pragma solidity ^0.5.0;

// lvl 2: tiered split

contract TieredProfitSplitter {

address payable employee\_one; // ceo

address payable employee\_two; // cto

address payable employee\_three; // bob

constructor(address payable \_one, address payable \_two, address payable \_three) public {

employee\_one = \_one;

employee\_two = \_two;

employee\_three = \_three;

}

// Should always return 0! Use this to test your `deposit` function's logic

function balance() public view returns(uint) {

return address(this).balance;

}

function deposit() public payable {

uint points = msg.value / 100; // Calculates rudimentary percentage by dividing msg.value into 100 units

uint total;

uint amount;

// @TODO: Calculate and transfer the distribution percentage

// Step 1: Set amount to equal `points` \* the number of percentage points for this employee

// Step 2: Add the `amount` to `total` to keep a running total

// Step 3: Transfer the `amount` to the employee

amount = points \* 60;

total += amount;

employee\_one.transfer(amount);

// @TODO: Repeat the previous steps for `employee\_two` and `employee\_three`

amount = points \* 25;

total += amount;

employee\_two.transfer(amount);

// @TODO: Repeat the previous steps for `employee\_two` and `employee\_three`

amount = points \* 15;

total += amount;

employee\_three.transfer(amount);

employee\_three.transfer(msg.value - total); // ceo gets the remaining wei

}

function() external payable {

deposit();

}

}

pragma solidity ^0.5.0;

// lvl 3: equity plan

contract DeferredEquityPlan {

address human\_resources;

address payable employee; // bob

bool active = true; // this employee is active at the start of the contract

// @TODO: Set the total shares and annual distribution

uint total\_shares = 1000;

uint annual\_distribution = 250;

uint start\_time = now; // permanently store the time this contract was initialized

// @TODO: Set the `unlock\_time` to be 365 days from now

uint public unlock\_time = now + 365;

uint public distributed\_shares = 0; // starts at 0

constructor(address payable \_employee) public {

human\_resources = msg.sender;

employee = \_employee;

}

function distribute() public {

require(msg.sender == human\_resources || msg.sender == employee, "You are not authorized to execute this contract.");

require(active == true, "Contract not active.");

// @TODO: Add "require" statements to enforce that:

// 1: `unlock\_time` is less than or equal to `now`

// 2: `distributed\_shares` is less than the `total\_shares`

require(unlock\_time <= now, "Error, time gone by!");

require(distributed\_shares <= total\_shares, "Error, Ouups something is wrong");

// @TODO: Add 365 days to the `unlock\_time`

unlock\_time += 365 days;

// @TODO: Calculate the shares distributed by using the function (now - start\_time) / 365 days \* the annual distribution

// Make sure to include the parenthesis around (now - start\_time) to get accurate results!

distributed\_shares = ((now - start\_time) / 365 days \* annual\_distribution);

// double check in case the employee does not cash out until after 5+ years

if (distributed\_shares > 1000) {

distributed\_shares = 1000;

}

}

// human\_resources and the employee can deactivate this contract at-will

function deactivate() public {

require(msg.sender == human\_resources || msg.sender == employee, "You are not authorized to deactivate this contract.");

active = false;

}

// Since we do not need to handle Ether in this contract, revert any Ether sent to the contract directly

function() external payable {

revert("Do not send Ether to this contract!");

}

}