Crowdfunding Service in Solidity

Triet Lieu

Cryptocurrency & Blockchain Course

Spring 2024

Table of Contents

2. Introduction

3. Background

4. Design Choices

6. Conclusion

7. GoFundMe Demonstration

11. Kickstarter Demonstration

13. Code

1 Introduction

Crowdfunding is the mean of raising funding for a project or for a person by people the crowdfunder doesn't know. It is done via the Internet. The two most popular crowding platforms are GoFundMe and Kickstarter. GoFundMe and Kickstarter differ in the "reward" mechanism.

GoFundMe doesn't have a formal reward mechanism. A funder is promised nothing in return for contributing, except for an expression of gratitude. Thus, GoFundMe is initially intended for soliciting donations for a person and is the simpler service to build first. After the donation functionality is designed, Kickstarter's tiers functionality can be added. Thus, there is a contract aspect to Kickstarter in that a donor purchases a reward for a donor and the crowdfunder is expected to deliver. Thus, a smart contract is perfect for implementing this relation.

2 Features

A. Tier

In contrast to GoFundMe, Kickstarter allows a creator to offer rewards through the tier mechanism. Funders can choose the tiers of funding and each tier requires contributing a minimum amount. There is a default tier with a donation of $1 that offers no reward. In return, each tier promises different rewards, with the higher tiers offering greater rewards. A crowdfunder on Kickstarter can limit the number of slots available for each tier.

B. All Or Nothing

Both GoFundMe and Kickstarter require a crowdfunder to specify their target goal of contributions. However, Kickstarter is all-or-nothing, while GoFundMe is not. Since Kickstarter is intended to fund projects, if a crowdfunder cannot achieve their goal of funding, they are assumed to abandon the project. Thus, if the goal is not reached on Kickstarter, the funds are given back. In my service, the contract constructor takes 3 arguments `(goal, durationInMinutes, _isKickstarter).` If a crowdfunder passes "true" as the final argument, they create a Kickstarter project, while "false" creates a GoFundMe project

For a GoFundMe campaign, if the goal is not reached, the crowdfunder can still withdraw the funds that were donated. This behavior is implemented in my service.

3 Design Choices

1. Tracking donations by using `mapping(address => uint256)` rather than via 2 distinct arrays of address[] and uint[]. Each approach has its disadvantage.

   To use a hashtable mapping from donor to donation as I did, if I know the address of a donor, it is a simple O(1) search to get the donation. This lets me efficiently implement the functionality for an individual donor to requestRefund(). However, I cannot efficiently iterate through a hashtable. Thus, it is not possible to automatically refund donations if for example a Kickstarter campaign's goal is not met. In that scenario, a donor must individually request their refund.

   If I stored donors and donations in arrays instead, to get the index of a donor so I can use that index to get their donation would require an inefficient O(n) search. However, to iterate through donors to automatically refund donations would require only traversing the array I created rather than the entire hashtable of addresses.

   Because I wanted to enable a donor to request a refund at will, I chose to store my donors and donations data in a mapping rather than in arrays. Donation requests should be more common than a Kickstarter campaign failing to reach its goal and requiring refunds.

2. The events that are defined are Launch and Cashout, emitted by the Crowdfunding’s constructor and the cashout() function respectively. The events transmit all information that can be used to reconstruct a campaign.

   Other functions such as donate(), requestRefund(), or viewReward() do not have events defined for them because I do think that individual events that can happen hundreds of times are not worth transmitting. Only events that can happen once such as starting and ending a campaign are noteworthy.

4 Conclusion

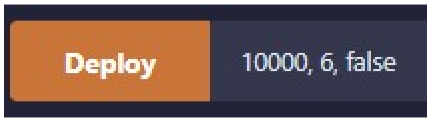
I am proud of the work I did for this project, which I will put on my GitHub page. To the best of my ability, I implemented mechanisms of both crowdfunding websites I researched. I added the appropriate require to each function and as my video and image demonstrations show, I tested every function against multiple inputs and conditions.

This project required not only Solidity knowledge, but also design creativity. Smart contracts have the disadvantage that the more code and the more complicated logic they require, the higher the gas costs required to execute and verify them. That required me to implement the logic of my service in as little code as possible, which forced me to skip cosmetic additions. If I implement the service in a situation without this gas cost, I would add these cosmetic additions.

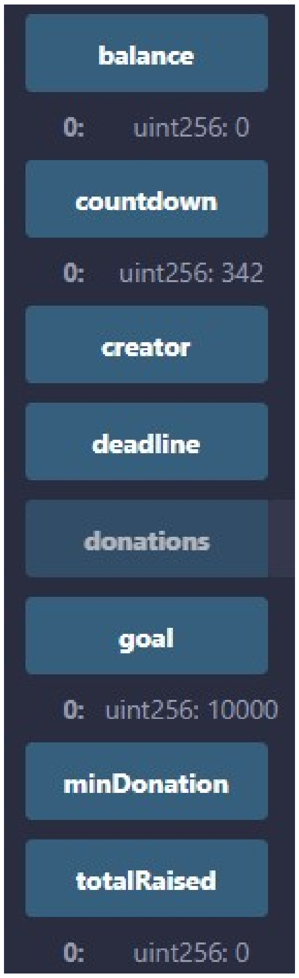
Service Demonstration

GoFundMe Campaign

1A. Deploy (false: Kickstarter) campaign, with (goal: 10000) and (duration: 6) minutes

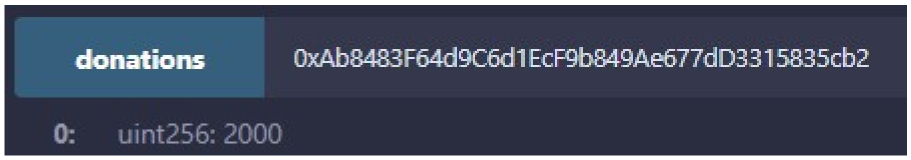


1B. Before any donation, totalRaised and balance are 0. There is 342 seconds or ~5 minutes left until the campaign ends

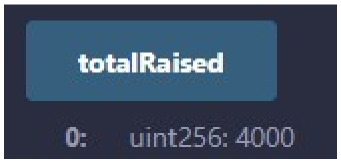


```
balance
0:  uint256: 0
countdown
0:  uint256: 342
creator
deadline
donations
goal
0:  uint256: 10000
minDonation
totalRaised
0:  uint256: 0
```

1C. Accounts (..cb2) and (..2cb) donor 2000 each, so now totalRaised is now 4000

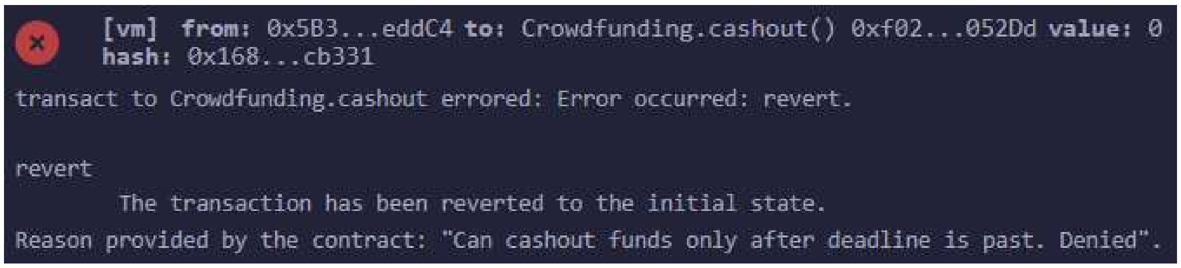


```
donations  0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
0:  uint256: 2000
```



```
totalRaised
0:  uint256: 4000
```

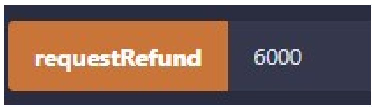
1D. Creator tries to cashout before deadline, but is denied.



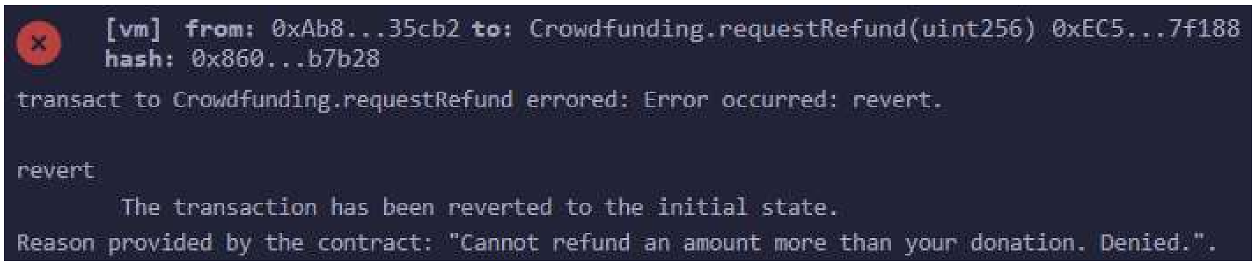
```
[vm] from: 0x5B3...eddC4 to: Crowdfunding.cashout() 0xf02...052Dd value: 0
hash: 0x168...cb331
transact to Crowdfunding.cashout errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Can cashout funds only after deadline is past. Denied".
```

1E. Accounts (..cb2) donated 2000, but requestRefund(6000), so is denied.



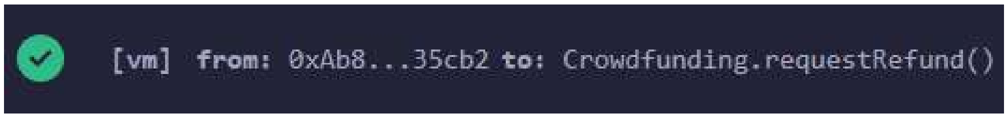
```
requestRefund  6000
```



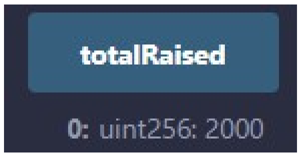
```
[vm] from: 0xAb8...35cb2 to: Crowdfunding.requestRefund(uint256) 0xEC5...7f188
hash: 0x860...b7b28
transact to Crowdfunding.requestRefund errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Cannot refund an amount more than your donation. Denied.".
```

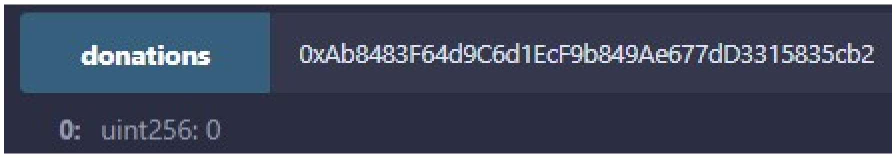
1F. Account (..cb2) requestRefund(2000), which is granted. totalRaised tracker is reduced and their donation receipt is zeroed.



```
[vm] from: 0xAb8...35cb2 to: Crowdfunding.requestRefund()
```

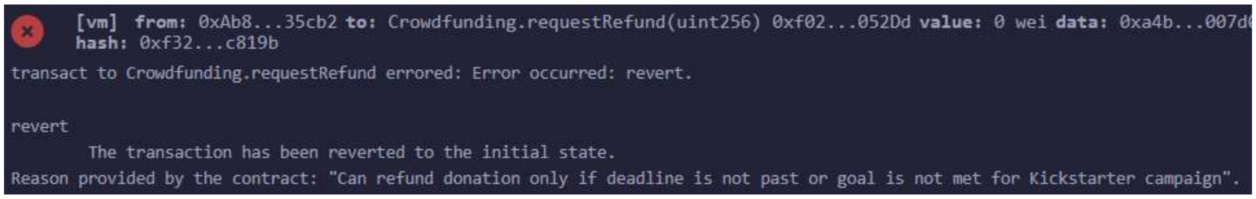


```
totalRaised
0: uint256: 2000
```



```
donations  0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
0: uint256: 0
```

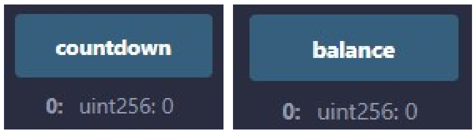
1G. requestRefund is denied because deadline is past



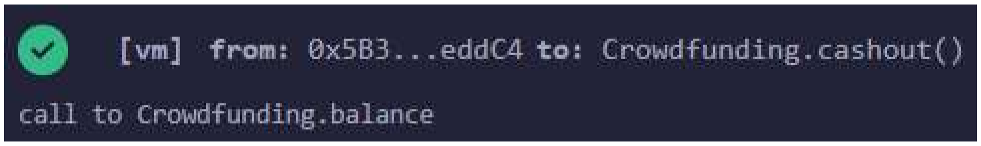
```
[vm] from: 0xAb8...35cb2 to: Crowdfunding.requestRefund(uint256) 0xf02...052Dd value: 0 wei data: 0xa4b...007d
hash: 0xf32...c819b
transact to Crowdfunding.requestRefund errored: Error occurred: revert.

revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Can refund donation only if deadline is not past or goal is not met for Kickstarter campaign".
```

1H. After the countdown is 0, creator cashout(), which sets balance to 0

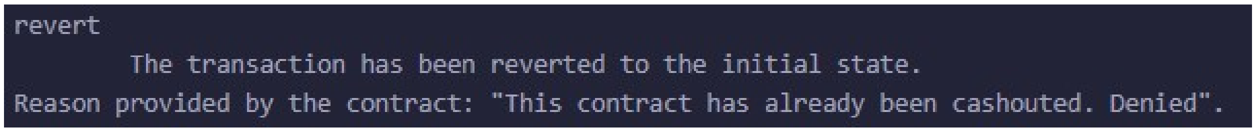


```
countdown          balance
0: uint256: 0      0: uint256: 0
```



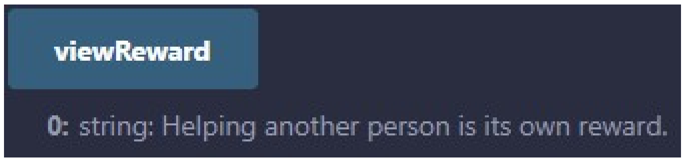
```
[vm] from: 0x5B3...eddC4 to: Crowdfunding.cashout()
call to Crowdfunding.balance
```

1I. Creator’s attempt to cashout() multiple times and double-spent is denied.



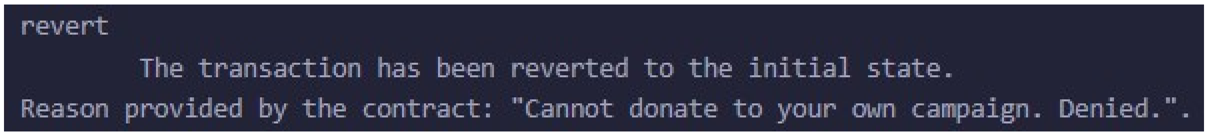
```
revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "This contract has already been cashouted. Denied".
```

1J. On GoFundMe campaigns, this message is the reward for a donor.



0: string: Helping another person is its own reward.

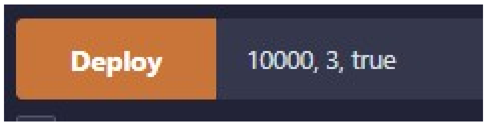
1K. For both campaign types, creator cannot donate to their own campaign.



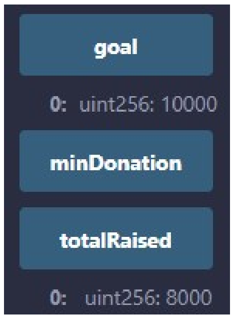
```
revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Cannot donate to your own campaign. Denied.".
```

Kickstarter Campaign

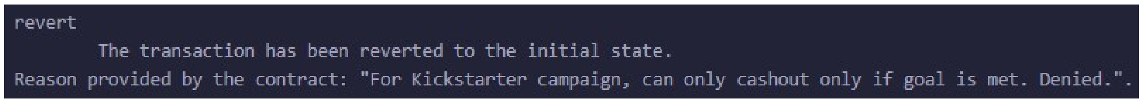
2A. Deploy (true: Kickstarter) campaign, with (goal: 10000) and (duration: 3) minutes



2B. Goal of 10000 is unmet, so cashout is denied. Donors should requestRefund()

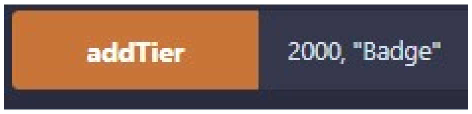


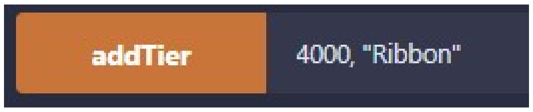
```
0: uint256: 10000
0: uint256: 8000
```

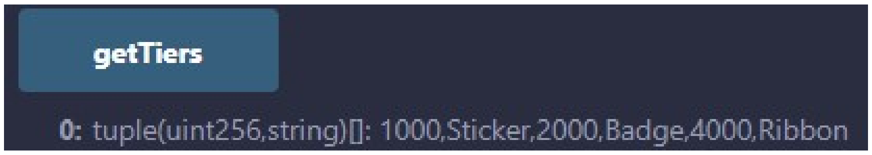


```
revert
	The transaction has been reverted to the initial state.
Reason provided by the contract: "For Kickstarter campaign, can only cashout only if goal is met. Denied.".
```

2C. Add tiers with increasingly higher minAmounts: (1000, “Sticker”), (2000, “Badge”), (4000, “Ribbon”).

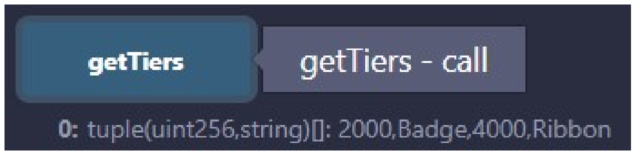




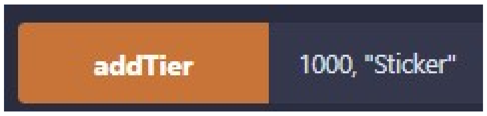


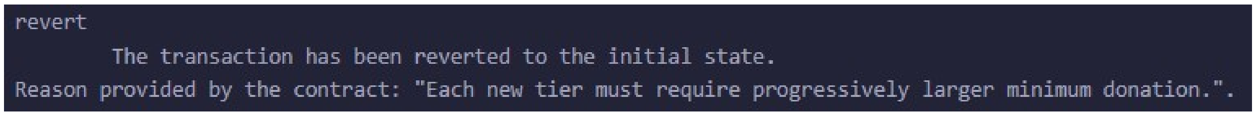
```
0: tuple(uint256,string)[]: 1000,Sticker,2000,Badge,4000,Ribbon
```

2D. After adding Tier (4000, “Ribbon”), new tiers require minAmount > 4000. Attempt to add Tier with minAmount = 1000 < 4000 was denied.



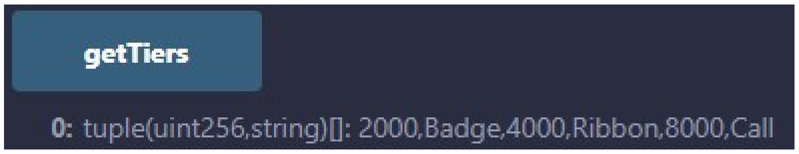
```
0: tuple(uint256,string)[]: 2000,Badge,4000,Ribbon
```



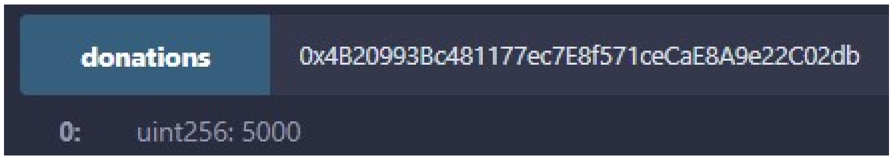


```
revert
        The transaction has been reverted to the initial state.
Reason provided by the contract: "Each new tier must require progressively larger minimum donation.".
```

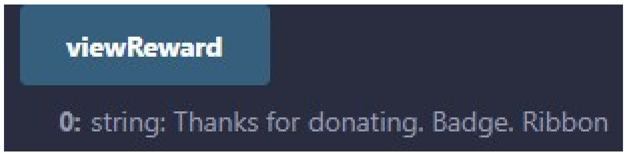
2E. Donation of 5000 > 4000 gets rewards of Tiers with minAmounts == 2000, 4000. Donation of 20000 > 8000 gets additional reward from Tier with minAmount == 8000



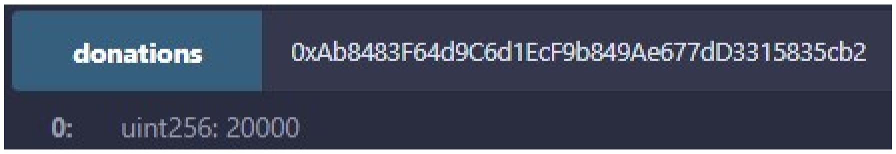
```
0: tuple(uint256,string)[]: 2000,Badge,4000,Ribbon,8000,Call
```



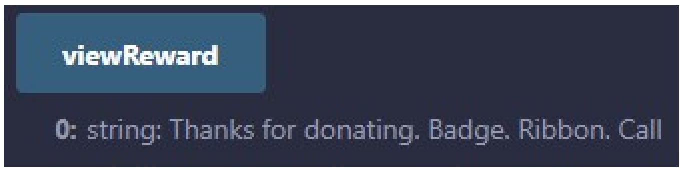
```
0:    uint256: 5000
```



```
0: string: Thanks for donating. Badge. Ribbon
```



```
0:    uint256: 20000
```



```
0: string: Thanks for donating. Badge. Ribbon. Call
```

Code (Crowdfunding.sol)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/**
 * @title Crowdfunding
 * @dev Implement Kickstarter && GoFundMe features
 * @custom:dev-run-script Crowdfunding.sol
 */

contract Crowdfunding {
    address public creator;

    // Goal set by creator. Affects outcome of only Kickstarter campaigns
    uint256 public goal;

    // Closing datetime computed as (block.timestamp + duration)
    uint256 public deadline;

    bool isKickstarter;

    // Default set to 500 Wei (~$1.50). Donation of less than this amount is
refused
    uint256 public minDonation = 500;

    // Balance that crownfunder can cashout. For Kickstarter campaigns,
totalRaised
    // must meet goal metric to cashout; else, donors can requestRefund
    uint256 public totalRaised;

    event Launch(
        address indexed creator,
        uint goal,
        uint start,
        uint end,
        bool isKickstarter
    );

    event Cashout(
        address indexed creator,
        uint goal,
        uint deadline,
        uint cashedDate,
        bool isKickstarter,
        uint paid
    );

    struct Tier {
        uint256 minAmount;
        string  reward;
    }
    Tier[] tiers;

    // {donor: donationTotal}. Use to conduct refunds
    mapping(address => uint256) public donations;

    // Constructor accepts only literals so 2000000000000000000 rather than 2
ether or 2 * 10**18
    constructor(uint256 _goal, uint256 durationInMinutes, bool _isKickstarter) {
        creator = msg.sender;
        goal = _goal;
        deadline = block.timestamp + 60 * durationInMinutes;
        isKickstarter = _isKickstarter;

        emit Launch(creator, goal, block.timestamp, deadline, isKickstarter);
    }

    modifier onlyCreator() {
        require(msg.sender == creator, "To call this function, you must be the
creator/crowdfunder.");
        _;
    }

    modifier onlyDonor() {
        require(
            donations[msg.sender] > 0, "To call this function, you must have
donated.");
        _;
    }

    // For Kickstarer project, add tier, with that minimum [amount] and [reward].
    // Require new tier's minAmount > previously added tier's so each higher
tier
    // requires progressively larger donations
    function addTier(uint256 amount, string memory reward) public onlyCreator {
        require(isKickstarter, "GoFundMe projects do not have tiers and rewards.
Denied.");
        require(block.timestamp < deadline, "The deadline for this campaign is
past. Denied");
```

```
        require(amount >= minDonation,
        string(abi.encodePacked(
            "All tiers must have minimum donation amount of ", minDonation, "
wei."))
        );

        require ((tiers.length == 0) || (amount > tiers[tiers.length -
1].minAmount),
                "Each new tier must require progressively larger minimum
donation.");

        tiers.push(Tier(amount, reward));
    }

    function getTiers() public view returns (Tier[] memory) {
        return tiers;
    }

    // View amount in contract that creator can cashout at campaign's end
    function balance() public view returns (uint256) {
        return address(this).balance;
    }

    // View time remaining until campaign's end as string "min_ m sec_"
    function countdown() public view  returns (uint256) {
        if (block.timestamp >= deadline) {
            return 0;
        }
        return deadline - block.timestamp;
    }

    function donate() public payable {
        require(block.timestamp < deadline, "The deadline for this campaign is
past.");

        // Stops creator bypassing requirement that goal is reached to access
cashout
        require(msg.sender != creator, "Cannot donate to your own campaign.
Denied.");

        require(
            msg.value >= minDonation,
            string(abi.encodePacked("The minimum donation is ", minDonation, "
wei. Donate more!"))
        );
        // Tally donation for that donor and for total
        donations[msg.sender] += msg.value;
        totalRaised += msg.value;
    }

    // Donor to get [amount] paid back to them
    // Allowed only if deadline is not past or goal is not met for Kickstarter
campaign
    function requestRefund(uint256 amount) public onlyDonor {
        require(block.timestamp < deadline || (isKickstarter && (totalRaised <
goal)),
        "Can refund donation only if deadline is not past or goal is not met for
Kickstarter campaign");

        require(donations[msg.sender] >= amount, "Cannot refund an amount more
than your donation. Denied.");

        // On refund, zeroing donation record prevents double-spending
        payable(msg.sender).transfer(amount);
        totalRaised -= amount;
        donations[msg.sender] -= amount;
    }

    // Overloaded: Donor to request complete refund if no amount is specified
    function requestRefund() public onlyDonor {
        requestRefund(donations[msg.sender]);
    }

    // Creator should cashout at campaign's end. If goal is met, pay to creator
from balance.
    // If isKickstarter and goal is not met, each donor should request refund
    function cashout() public onlyCreator {
        require(block.timestamp >= deadline,
        "Can cashout funds only after deadline is past. Denied");

        require(!isKickstarter || (totalRaised >= goal),
        "For Kickstarter campaign, can only cashout only if goal is met.
Denied.");

        require(address(this).balance >= totalRaised,
        "This contract has already been cashouted. Denied");

        payable(creator).transfer(address(this).balance);

        uint paid = (!isKickstarter || totalRaised >= goal) ? totalRaised : 0;
        emit Cashout(creator, goal, deadline, block.timestamp, isKickstarter,
paid);
    }


    // Donor to view what rewards they earned based on their donation total
    function viewReward() public view onlyDonor returns (string memory) {
        require(block.timestamp > deadline,
        "Expect to get your reward only after the campaign has ended.");

        string memory rewards;

        if (isKickstarter) {

            require(totalRaised >= goal,
            "This Kickstarter campaign failed to meet its goal. You should
request a refund.");

            rewards = "Thanks";
            for (uint256 i = 0; i < tiers.length; i++) {
                if (donations[msg.sender] >= tiers[i].minAmount) {
                    rewards = string(abi.encodePacked(rewards, ". ",
tiers[i].reward));
                }
            }
        }
        else {
            rewards = "Helping another person is its own reward.";
        }

        return rewards;
    }
}
```