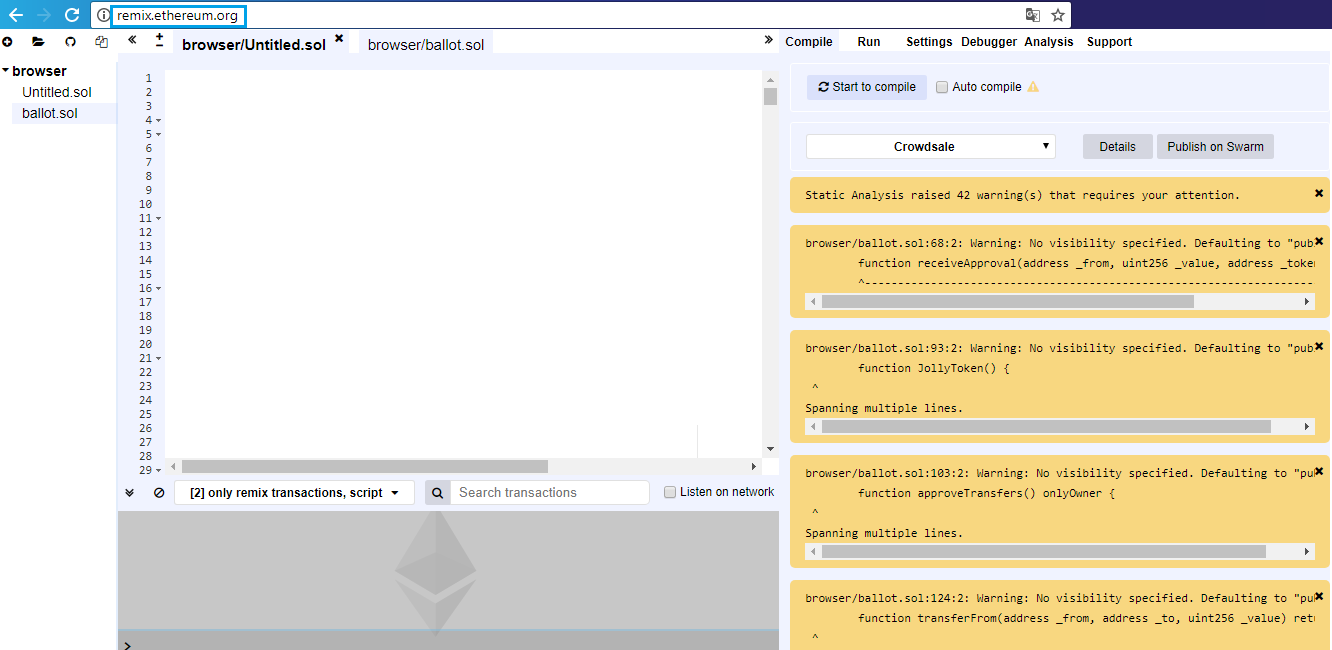
# Exercises: Reentrancy Attack on a Smart Contract

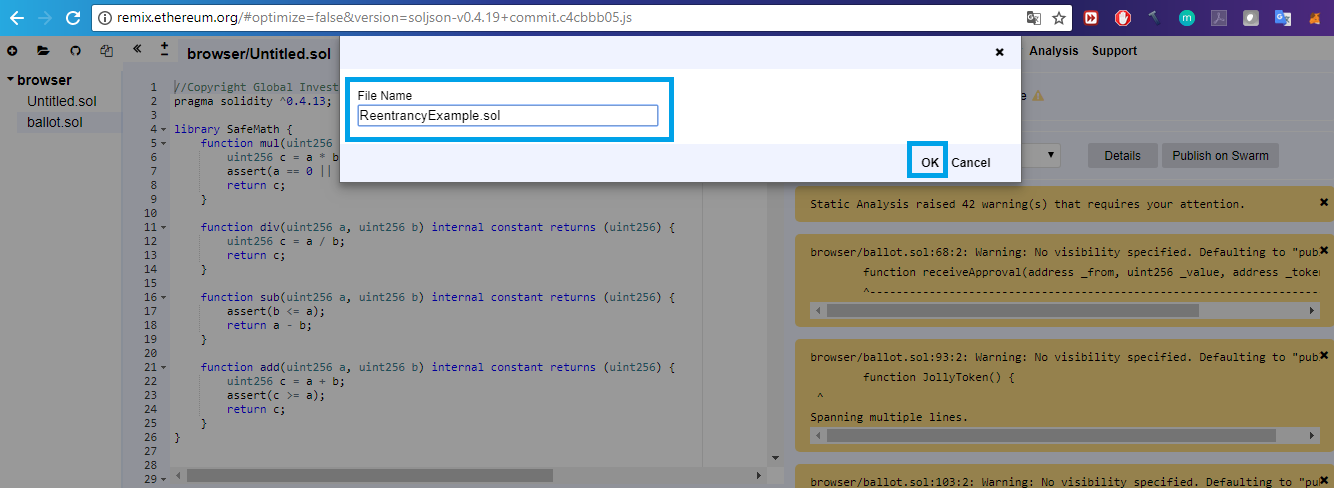
This document describes the **exercise assignments** for the ["Blockchain Academy" course @ Software University](https://softuni.bg/courses/programming-fundamentals). In this lesson we learned the **basics of Solidity** programming language. The goal of this exercise is to get practical skills in writing simple smart contracts in Solidity, publishing and testing contracts in the Remix IDE.

## Create and Deploy HoneyPot Contract

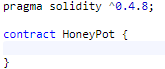
* 1. Go to <https://remix.ethereum.org>



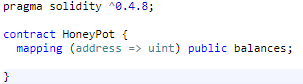
* 1. Create new file and click “OK”.



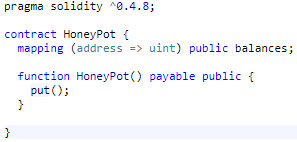
* 1. Create the contract **HoneyPot**. Define the solidity version you want to compile the contract.



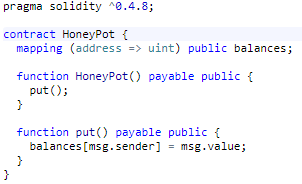
* 1. The following code maps addresses to a value and store it in a public variable called balances. It allows to check the **HoneyPot** balance for an address.



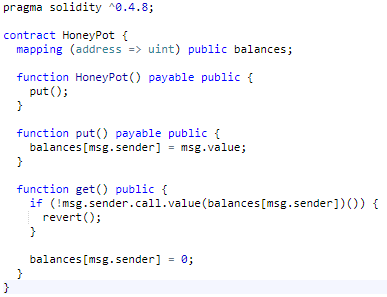
* 1. This is the constructor of the contract. The function with the same name as the contracts is the called constructor. It will be called when deploying the contract.



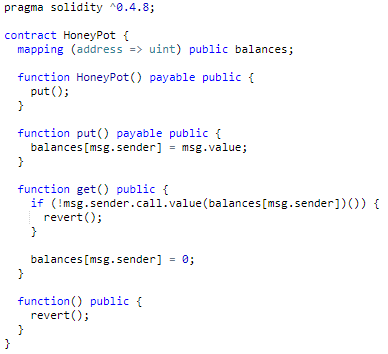
* 1. The **put()** function below is where the storage of the ether value happens in the contract. Note that **msg.sender** here is the address from the sender of the transaction.



* 1. The purpose of **get()** function is to let addresses to withdraw the value of ether they have in the **HoneyPot** balances.



* 1. The unnamed function is called **fallback** function. It will only throw an exception if it is called.

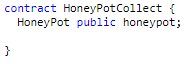


## Create and Deploy HoneyPotCollect Contract

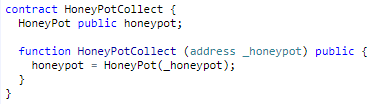
* 1. Create the contract **HoneyPotCollect**.

2.7.png

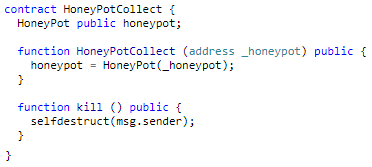
* 1. Define **HoneyPot** variable.



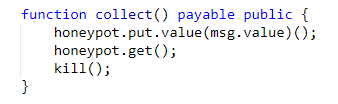
* 1. Then we define the constructor function. This is the function that is called when **HoneyPotCollect** is created. Note that we pass an address to this function. This address will be the **HoneyPot** contract address.



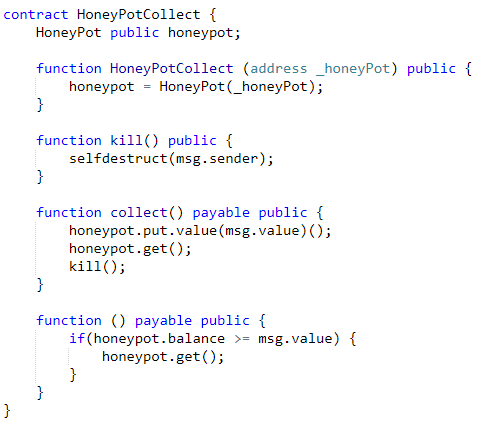
* 1. The **kill()** function destroys the **HoneyPotCollect** and send all ether containing in it to the address that calls the kill function.



* 1. The **collect()** function is the one that will set the **reentrancy attack** in motion. It puts some ether in **HoneyPot** and right after it gets it. The payable term here tells the **Ethereum Virtual Machine** that it permits to receive ether. Invoke this function with also some ether.

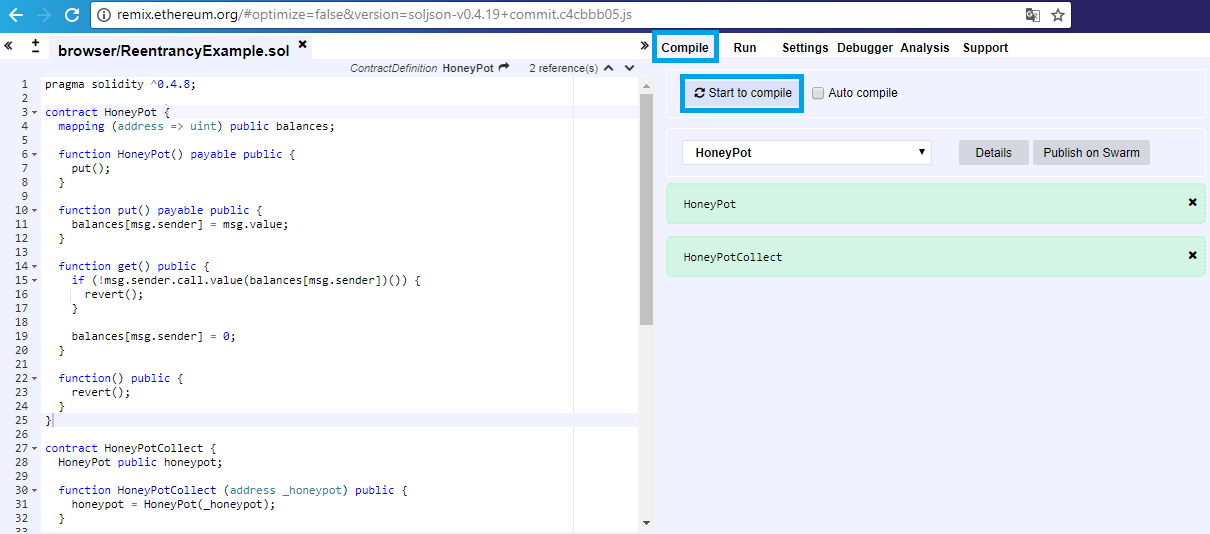


* 1. The last function is the **fallback** function. This unnamed function is called whenever the HoneyPotCollect contract receives ether.

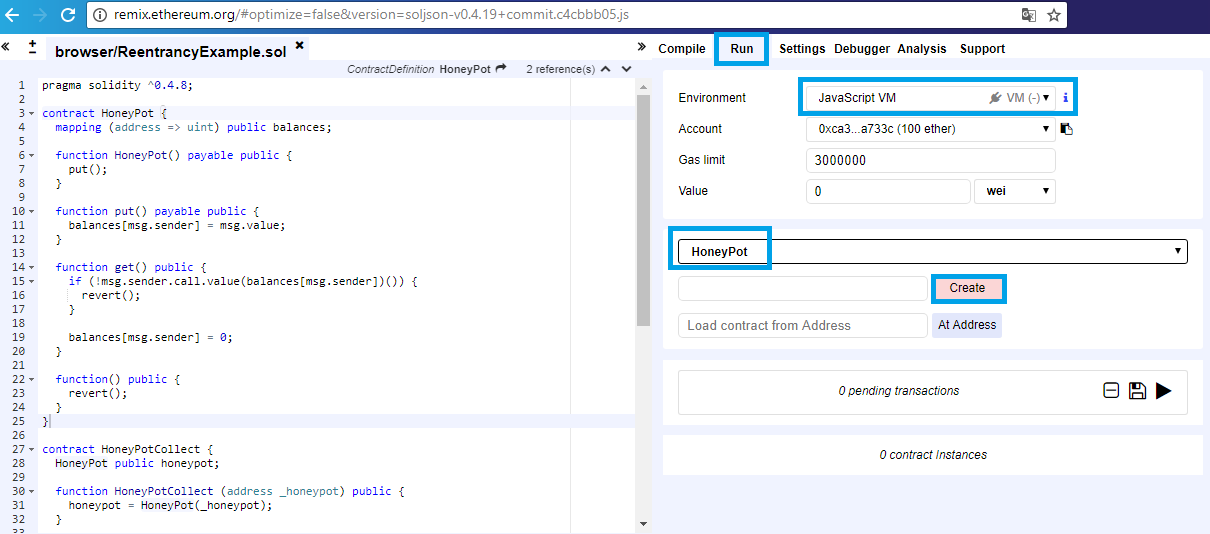


## Exercise

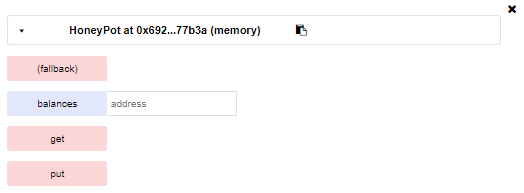
* 1. Then **“Start to compile”** the contacts.



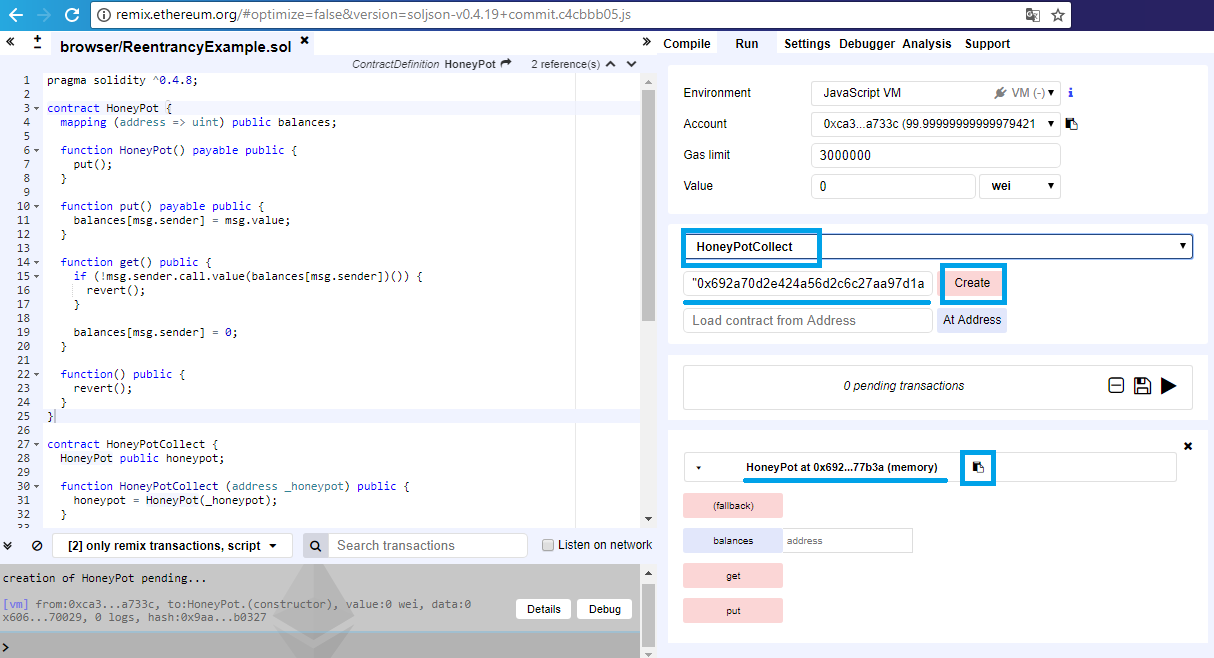
* 1. Go to **“Run”** tab and choose for environment **“JavaScript VM”**. Firstly choose the **HoneyPot** contract and create it.



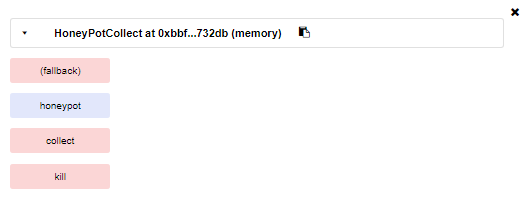
* 1. It should look like the following window.



* 1. Then choose **HoneyPotCollect** contract, copy the **HoneyPot** contract address and paste it to create the contact.



* 1. It should look like the following window.



After deploying HoneyPotCollect, call **collect()** and sending with it some ether and debug the function.

**HoneyPot** **get()** function sends ether to the address that called it only if this contract has any ether as balance. When **HoneyPot** sends ether to **HoneyPotCollect** the fallback function is triggered. If the **HoneyPot** balance is more than the value that it was sent to, the fallback function calls **get()** function once again and the cycle repeats.

Recall that within the **get()** function the code that sets the balance to zero comes only after sending the transaction. This tricks the **HoneyPot** contract into sending money to the **HoneyPotCollect** address repeatedly until **HoneyPot** is depleted of almost all its ether.

# What to Submit?

Create a **zip file** (e.g. your-username-honeypotcollect-attack.zip) with the source code and an image with an address with at least 100+ ethers.

Submit your **zip** file as **homework** at the course Web site.