

Практикум по курсу «Блокчейн и его приложения»

Разработка смарт-контрактов
на платформе Ethereum + Solidity + Remix
+ MetaMask + клиентский JavaScript

Пинягина О.В.

Казанский университет

2022

Пинягина О.В.

УДК 004.6

ББК 32.973.26 – 018.2

Печатается по решению учебно-методической комиссии
Института вычислительной математики и информационных технологий
протокол № 7 от 31.03.2022 г.

Рецензент:

Доцент кафедры системного анализа и информационных технологий
ИВМиИТ КФУ к.ф.-м.н. Андрианова А.А.

Пинягина О.В.

Практикум по курсу «Блокчейн и его приложения». Разработка смарт-контрактов на платформе Ethereum + Solidity + Remix + MetaMask + клиентский JavaScript / О.В. Пинягина – Казань: Казанский университет, 2022. – 48 с.

Данное учебное пособие разработано для поддержки компьютерных лабораторных занятий и самостоятельной работы студентов по курсу «Блокчейн и его приложения» для студентов, обучающихся по направлениям «Прикладная математика и информатика», «Бизнес-информатика».

В качестве платформы программирования используется комплекс технологий Ethereum + Solidity + Remix + MetaMask + клиентский JavaScript.

© Казанский университет, 2022

© Пинягина О.В. 2022

Оглавление

Предисловие4

Тема 1. Работа с Metamask.....	6
Задание для самостоятельной работы	11
Тема 2. Начинаем изучать Solidity и среду Remix	12
Задание для самостоятельной работы	16
Тема 3. Продолжаем изучать Solidity. Rinkeby и Metamask.....	17
Задание для самостоятельной работы	26
Тема 4. Интерфейс для смарт-контрактов на JavaScript.....	28
Задание для самостоятельной работы	39
<i>Приложение. Контракт для продажи промокодов</i>	<i>41</i>
Литература.....	47
Интернет-ресурсы.....	48

Предисловие

В рамках лабораторных занятий и самостоятельной работы студенты изучают экосистему **Ethereum** и язык программирования **Solidity** для создания смарт-контрактов.

Отдельные задания заключаются в изучении on-line возможностей экосистемы Ethereum и разработке компьютерных программ на следующие темы.

- Работа с кошельком MetaMask в тестовой сети Rinkeby. Установка, создание счетов, получение денег из "крана", перевод денег.
- Основные типы данных в Solidity - числа, строки, булевский тип (обязательно продемонстрировать целочисленное переполнение!).
- Сложные типы данных в Solidity - структуры, перечисления, массивы, хэш-таблицы.
- Модификаторы доступа в Solidity - public, private, internal, external.
- Стандартные методы - конструктор, безымянный метод для приема платежей, get- и set- методы.
- События в Solidity.
- Работа со смарт-контрактами в среде Remix. Компиляция и деплой контрактов.
- Работа со смарт-контрактами в тестовой сети Rinkeby. Деплой контрактов, перевод денег.
- Работа с контрактом через интерфейс Web3.js и клиентские сценарии JavaScript.

После изучения основных технологий и выполнения **простых заданий** студенты будут разрабатывать индивидуальные проекты по созданию контрактов на выбранную тему, например:

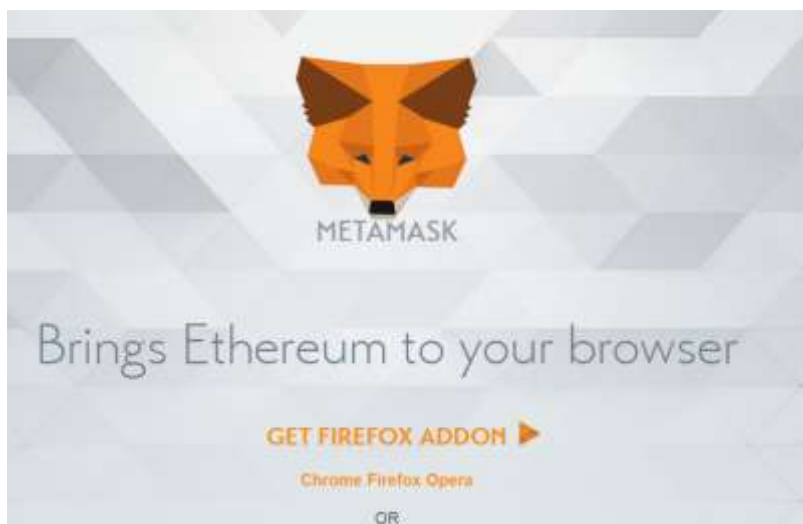
- Смарт-контракт для голосования.
- Смарт-контракт для подтверждения авторских прав.
- Смарт-контракт для продажи цифрового контента.
- Смарт-контракт для приема произвольных платежей.
- Смарт-контракт для оплаты счетов.

- Смарт-контракт для заключения пари.
- Смарт-контракт для хранения дипломов.
- Смарт-контракт для протекции сделки третьей стороной.
- Смарт-контракт для сделки с мультиподписями.
- И т. п.

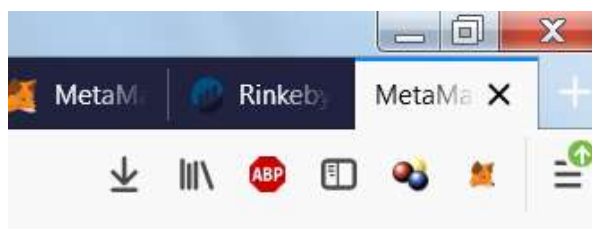
Тема 1. Работа с Metamask

Metamask представляет собой браузерный кошелек для работы с криптовалютой **Ethereum**. С помощью этого кошелька можно как проводить реальные платежи, так и работать в тестовых сетях.

Установите расширение браузера **Metamask** с сайта <https://metamask.io/>



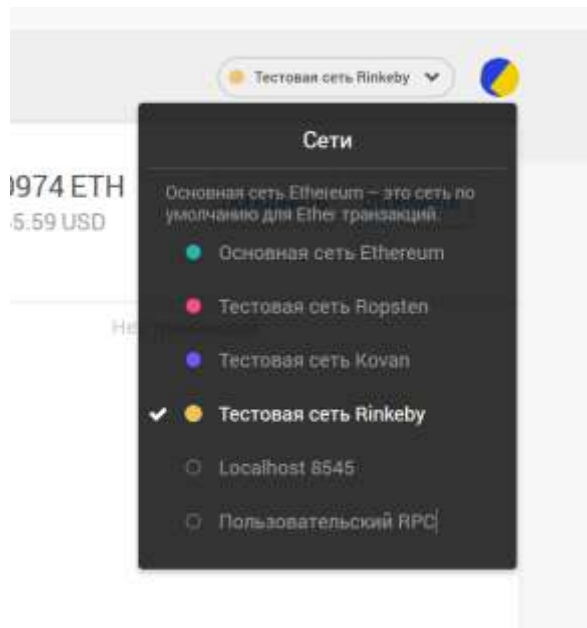
Значок **Metamask** появится в панели инструментов браузера.



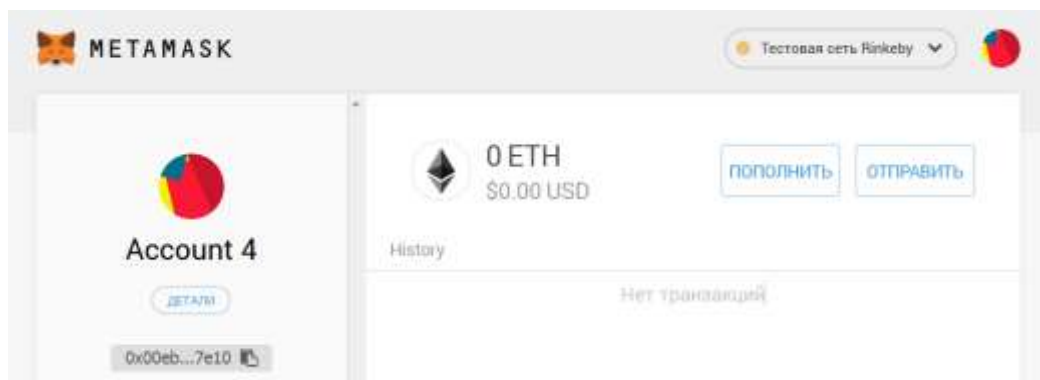
Зарегистрируйтесь в **Metamask**. Сохраните **ключевую фразу** в надежном месте. При регистрации также нужно придумать пароль, который будет использоваться при входе в кошелек с этого компьютера.

Если в дальнейшем вы захотите использовать свои кошельки с другого компьютера, для этого можно будет воспользоваться сохраненной ключевой фразой.

Список в верхнем правом углу окна позволяет выбрать текущую сеть для работы – основную или тестовые. Выберите в качестве текущей сети тестовую сеть **Rinkeby**.



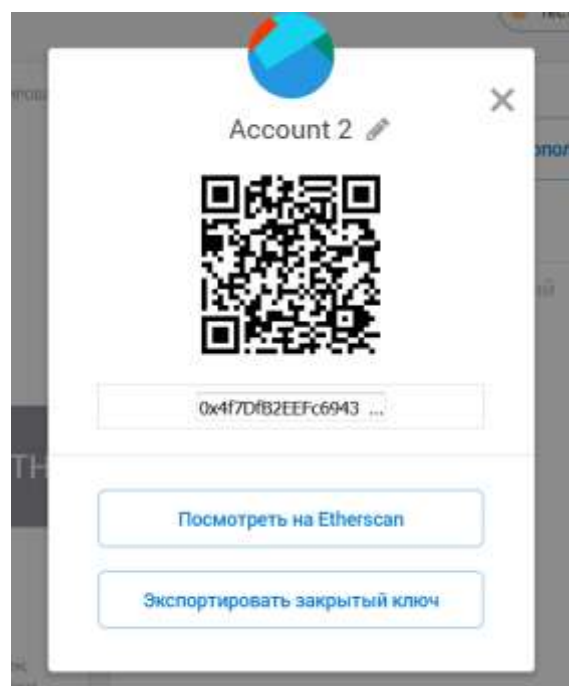
Создайте несколько счетов в своем кошельке.



Каждый кошелек получает уникальный адрес в таком формате:
0x4f7DfB2EEFc69438dE70E9Eb3d424A098c9c3715

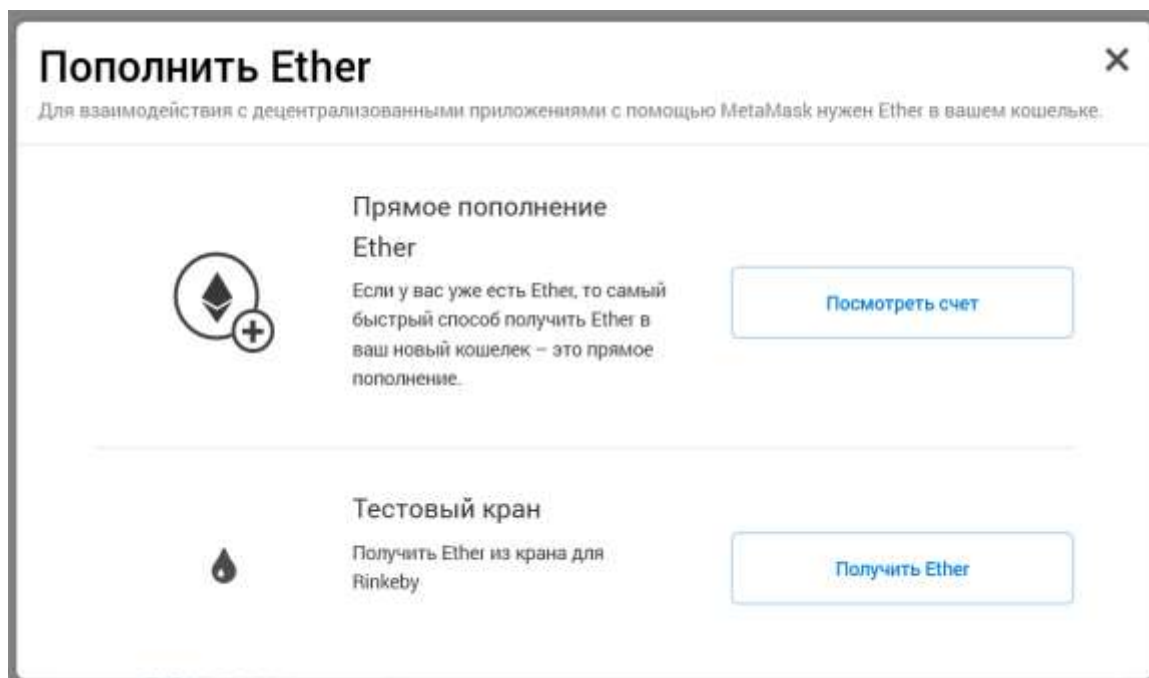
Если в окне счета нажать на кнопку «Детали», то на экран будет выдано служебное окно, в котором можно увидеть QR-код кошелька.

Здесь также можно посмотреть историю операций по данному счету и экспортировать секретные ключи кошелька.

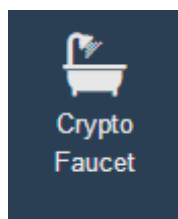


После создания кошелька пустой, баланс на нем равен нулю. Для дальнейшей работы нам понадобится тестовый эфир. Его можно получить через так называемый «кран» по адресу <https://faucet.rinkeby.io/> или <https://www.rinkeby.io/#faucet>

Или можно нажать на кнопку «**Пополнить**» и в открывшемся окне выбрать «**Получить Ether**»



В открывшемся окне нажмите на кнопку с ванной:



На текущий момент порядок получения тестового эфира такой.

В своем аккаунте в сети **Facebook** (последнее время с этим способом проблемы) или в сети **Twitter** опубликуйте сообщение, содержащее номер вашего счета.

Скопируйте ссылку на это сообщение и введите в текстовое поле в следующем окне. Нажмите на кнопку «**Give me Ether**» и выберите количество эфира. Через несколько минут тестовый эфир должен появиться на вашем счете.



Rinkeby Authenticated Faucet

Social network URL containing your Ethereum address... Give me Ether ▾

6 peers 5861322 blocks 9.046266971865328e+56 Ethers 337407 funded

How does this work?

This Ether faucet is running on the Rinkeby network. To prevent malicious actors from exhausting all available funds or accumulating enough Ether to mount long running spam attacks, requests are tied to common 3rd party social network accounts. Anyone having a Twitter or Facebook account may request funds within the permitted limits.

-  To request funds via Twitter, make a [tweet](#) with your Ethereum address pasted into the contents (surrounding text doesn't matter). Copy-paste the [tweets URL](#) into the above input box and fire away!
-  To request funds via Facebook, publish a new [public post](#) with your Ethereum address embedded into the content (surrounding text doesn't matter). Copy-paste the [posts URL](#) into the above input box and fire away!

You can track the current pending requests below the input field to see how much you have to wait until your turn comes.

The faucet is running invisible reCaptcha protection against bots.

Тестовый эфир можно получить и в сети **Ropsten**. Для этого в кошельке Metamask переключитесь на сеть **Ropsten**, выберите счет, нажмите на кнопку «Пополнить», и в следующем окне можно запросить по 1 эфиру:

MetaMask Ether Faucet

faucet

address: 0x81b7e08f65bdf5648606c89998a9cc8164397647
balance: 92914943.57 ether

[request 1 ether from faucet](#)

user

address: 0x38f1cb1f6287256f85af7d8bbe393c763df7780b
balance: 1.00 ether
donate to faucet:

[1 ether](#) [10 ether](#) [100 ether](#)

transactions

0xf3d27e0259e1ea22df494cdf5b3356e5428e7c05b2ef55b9dfb11e5a9992e319

Пинягина О.В.

Проведите несколько переводов эфира между своими счетами либо на счета одноклассников.

Посмотрите историю своих операций на Etherscan.

The screenshot shows the Etherscan website interface. At the top, there is a search bar and navigation links for HOME, BLOCKCHAIN, TOKEN, and MISC. The main content area displays the account overview for the address 0x38F1CB1F6287256F65AF7d80BE393C763dF7780b. The balance is 2.09744095 Ether, and there are 9 transactions. Below this, a table lists the most recent transactions:

TxHash	Block	Age	From	To	Value	[Tax fee]
0xb1467ae5853083...	3651087	33 days 14 hrs ago	0x38f1cb1f6287256f...	0xd309b4e7f89d3...	0.1 Ether	0.0042
0xc20fddc1b9fb17...	3651517	33 days 15 hrs ago	0x38f1cb1f6287256f...	0xaa70a6ca4e72...	0.2 Ether	0.0042
0xf7001d2640a73...	3651453	33 days 15 hrs ago	0x38f1cb1f6287256f...	0xd309b4e7f89d3...	0.2 Ether	0.0042
0x57410757a0004f...	3651444	33 days 15 hrs ago	0x38f1cb1f6287256f...	0x6dc207b8607054...	0.2 Ether	0.0042

Примечание: Для отображения в кошельке MetaMask сумм не только в эфирах, но и в долларах, выберите пункт меню «Настройки» и в секции «Продвинутый»

The screenshot shows the MetaMask mobile application interface. At the top, there is a dropdown menu for the network, currently set to 'Тестовая сеть Rinkeby'. Below this, there is a list of accounts under the heading 'Мои счета'. The accounts are:

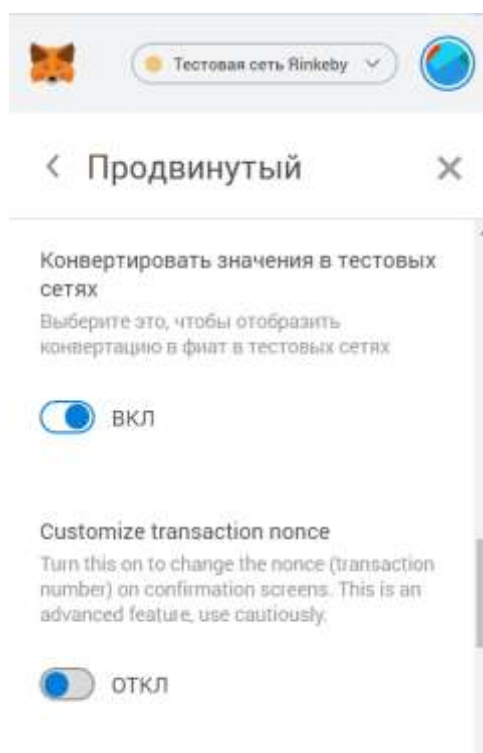
- Account 1: 3.317333 ETH
- Account 2: 39.374292 ETH
- Account 3: 16.909603 ETH

Below the accounts, there are options to '+ Создать счет', 'Импортировать счет', and 'Подключите аппаратный кошелек'. At the bottom, there is a menu with 'Информация и помощь' and 'Настройки'.

The screenshot shows the 'Настройки' (Settings) screen in the MetaMask mobile application. The settings are organized into sections:

- Основное**: Конвертация валюты, основная валюта, язык, идентификаторы блоков.
- Продвинутый**: Получите доступ к функциям разработчика, журналу событий, сбросу учетной записи, настройке тестовых сетей и пользовательский RPC.
- Contacts**: Add, edit, remove, and manage your contacts.

включите конвертирование в тестовых сетях:



Задание для самостоятельной работы

Прочитайте первые 3 главы книги К. Даннена «Введение в Ethereum и Solidity».

По желанию можете установить кошелек Mist и поэкспериментировать с ним. Этот кошелек представляет собой самостоятельное приложение, которое позволяет не только проводить платежи, но и развернуть узел блокчейна на вашем компьютере. Не забывайте, что при запуске кошелька Mist в некоторых режимах запускается синхронизация блокчейна! Она может потребовать много времени и дискового пространства.

Тема 2. Начинаем изучать Solidity и среду Remix

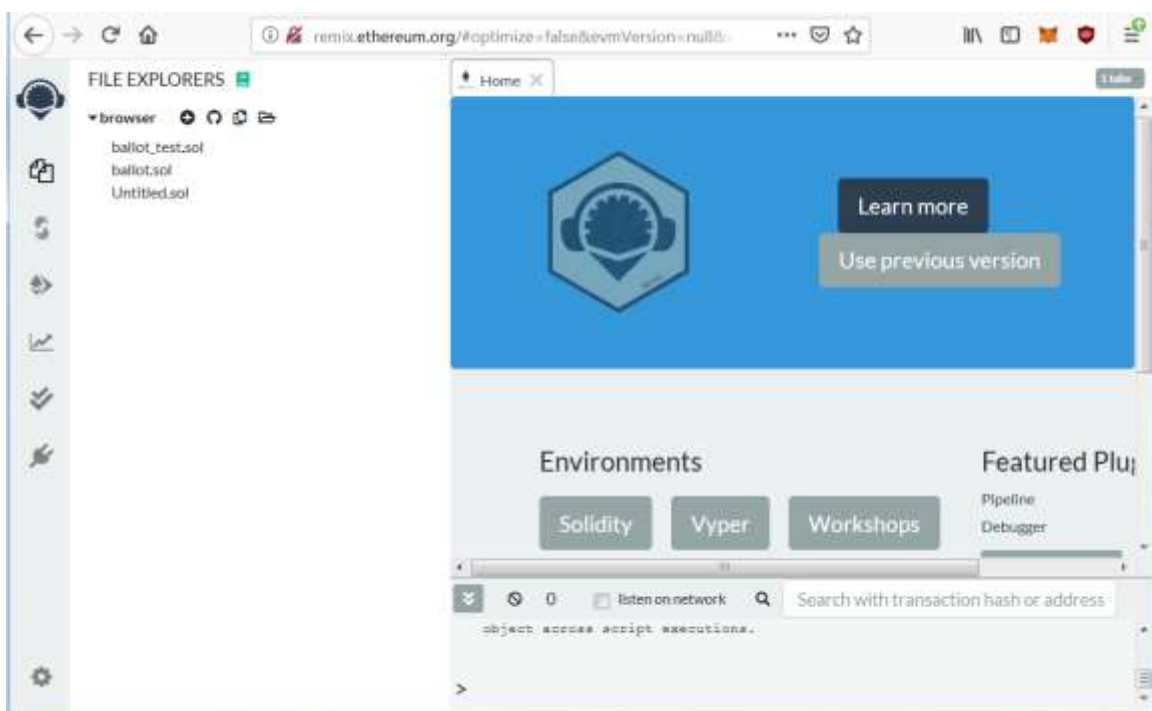
В этом задании мы начнем изучать язык **Solidity** для написания смарт-контрактов и среду **Remix**.

Начнем с того, что откроем документацию Introduction to Smart Contracts по адресу <https://solidity.readthedocs.io/en/v0.5.4/introduction-to-smart-contracts.html> и рассмотрим структуру простого контракта.

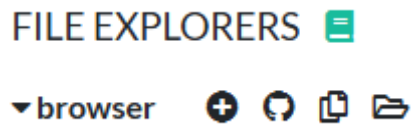
```
pragma solidity >=0.4.0 <0.6.0;  
  
contract SimpleStorage {  
    uint storedData;  
  
    function set(uint x) public {  
        storedData = x;  
    }  
  
    function get() public view returns (uint) {  
        return storedData;  
    }  
}
```

Среда Remix располагается по адресу <http://remix.ethereum.org/>

Загрузим сайт **Remix**, он выглядит примерно так:



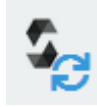
Создадим новый пустой файл (например, test.sol), нажав на кнопку «+»:

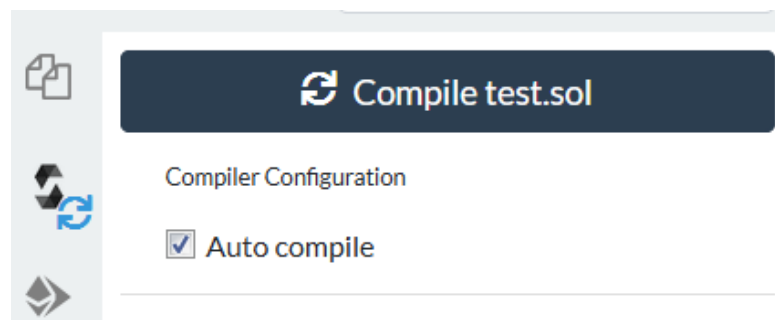


Начнем писать код следующего контракта:

```

1 pragma solidity ^0.5.1;|
2
3 contract MyContract{
4     address creator;
5     uint256 myNumber;
6
7     function MyContract() {
8         creator=msg.sender;
9         myNumber=3;
10    }
11    function getCreator() constant returns (address) {
12        return creator;
13    }
14    function getNumber() constant returns (uint256) {
15        return myNumber;
16    }
17    function setMyNumber(uint256 num) {
18        myNumber=num;
19    }
20    function kill() {
21        if(msg.sender == creator) {
22            suicide(creator);
23        }
24    }
25 }
    
```

Для компиляции следует перейти на вкладку **Solidity compiler**  и нажать на кнопку **Compile** (либо включить автоматическую



компиляцию).

Откомпилируйте программу, отладьте ошибки, если они есть. Ошибки в этом примере точно есть, и связаны они с устаревшими возможностями Solidity (ключевыми словами, семантикой). Устаревший пример приведен **специально**, для тренировки, и взят он из курса «Работа с Ethereum»

<https://www.intuit.ru/studies/courses/3630/872/lecture/32293?page=1>,

Лекция 3.


```
browser/test.sol:11:24: ParserError: The state mutability modifier "constant" was removed in version 0.5.0. Use "view" or "pure" instead.
function getCreator() constant returns (address) {
^-----^




browser/test.sol:14:23: ParserError: The state mutability modifier "constant" was removed in version 0.5.0. Use "view" or "pure" instead.
function getNumber() constant returns (uint256) {
^-----^
```


Исправьте все ошибки.


После того как код программы успешно откомпилирован, следует перейти на вкладку **Deploy & run transactions** выбрать пока в качестве среды выполнения (Environment) виртуальную машину JavaScript.

DEPLOY & RUN TRANSACTIONS

Environment 

Account   

Gas limit 

Value 

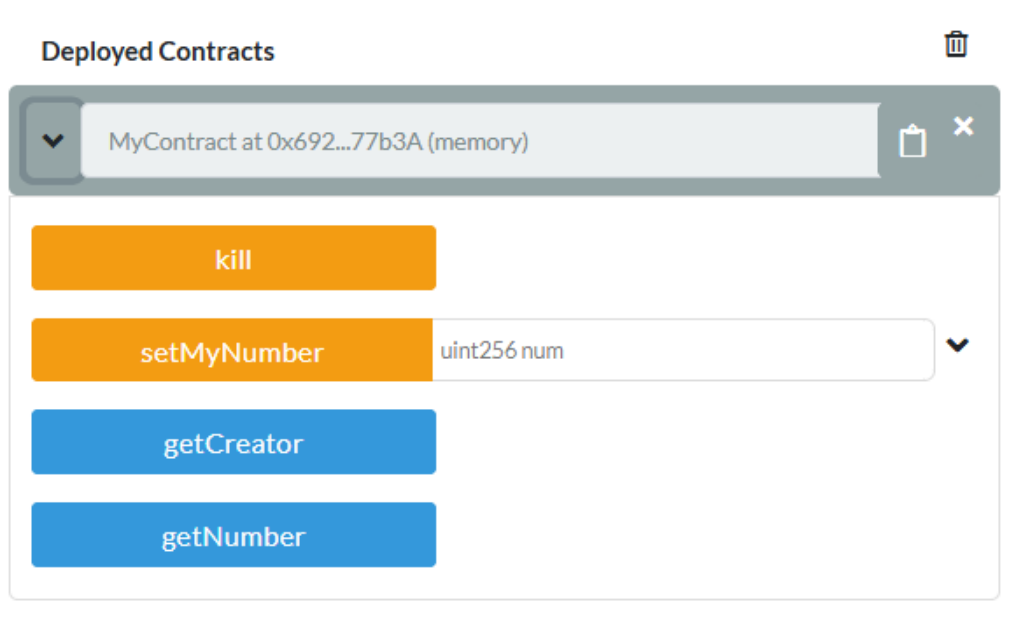


or

Transactions recorded:  

Для запуска контракта следует нажать на кнопку «**Deploy**». Запуск контракта осуществляется от имени условного аккаунта с балансом 100 ETH. Выполнение в рамках виртуальной машины JavaScript не выходит за пределы сеанса браузера.

Запущенный контракт появится ниже в окне:

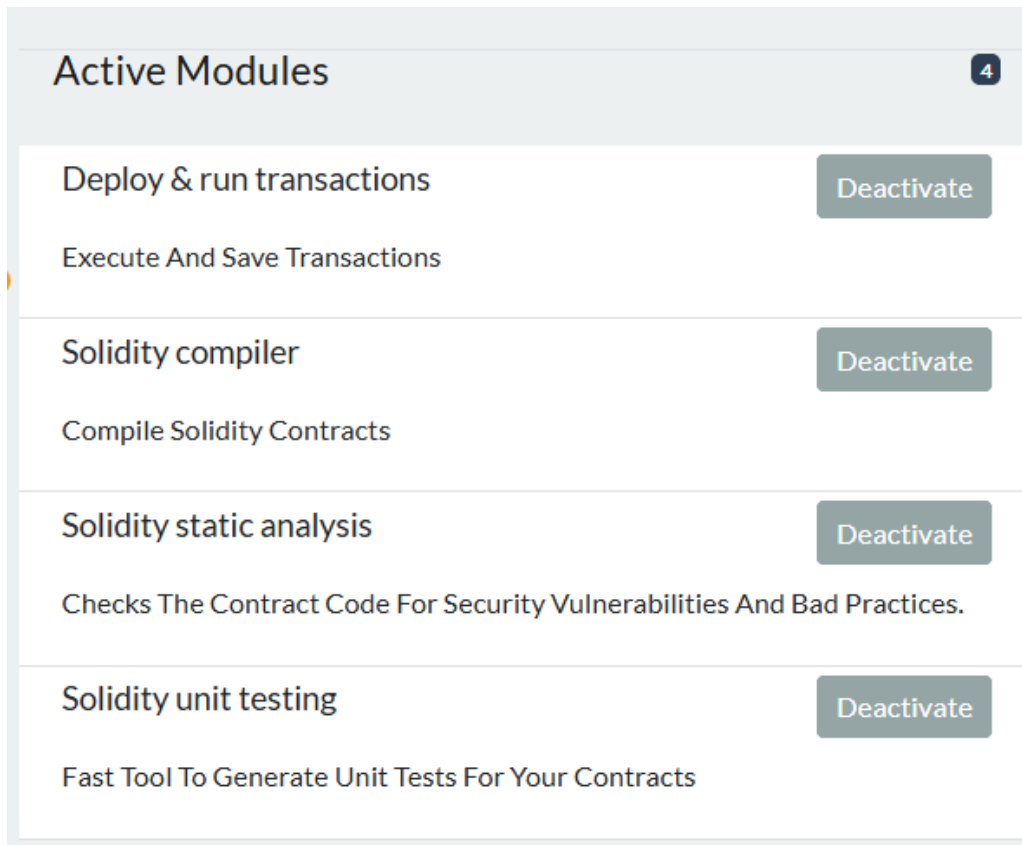


Здесь можно проверить работу всех его функций.





Отметим, что на вкладке Plugin Manager можно включать и выключать отдельные модули:



Задание для самостоятельной работы

Изучите типы данных и модификаторы доступа к данным в языке **Solidity**.

Напишите и отладьте контракт, в котором **используются разнообразные типы данных** и иллюстрируются операции с ними. Обязательно приведите пример **целочисленного переполнения**.

Напишите и отладьте контракт, в котором используются **разнообразные модификаторы доступа**. Проиллюстрируйте доступ к одноименным переменным с разными модификаторами.

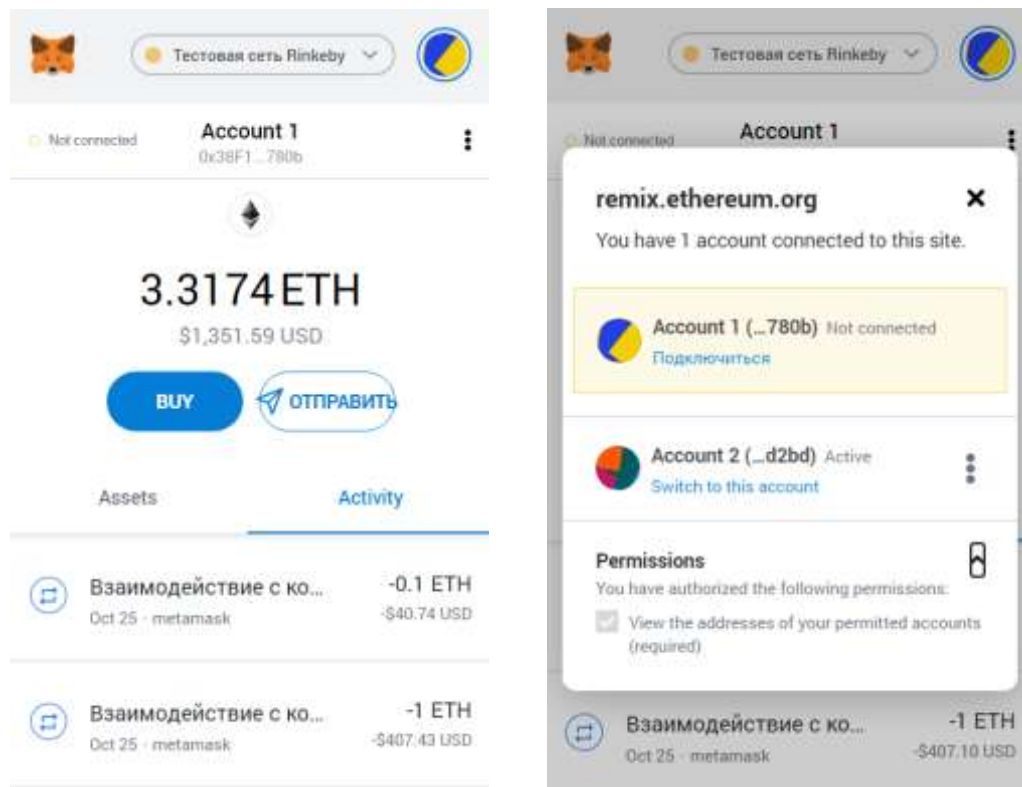
Тема 3. Продолжаем изучать Solidity. Rinkeby* и Metamask

В этом задании мы продолжаем изучать язык **Solidity** для написания смарт-контрактов и среду **Remix**, а также будем запускать свои контракты в глобальной тестовой среде **Rinkeby** с помощью кошельков **MetaMask**.

Важное замечание, октябрь 2020 г.! Далее мы будем выполнять некоторые операции с **разных** счетов кошелька **MetaMask**. Если они пока не подключены к сайту <http://remix.ethereum.org/>, то подключим их сейчас.

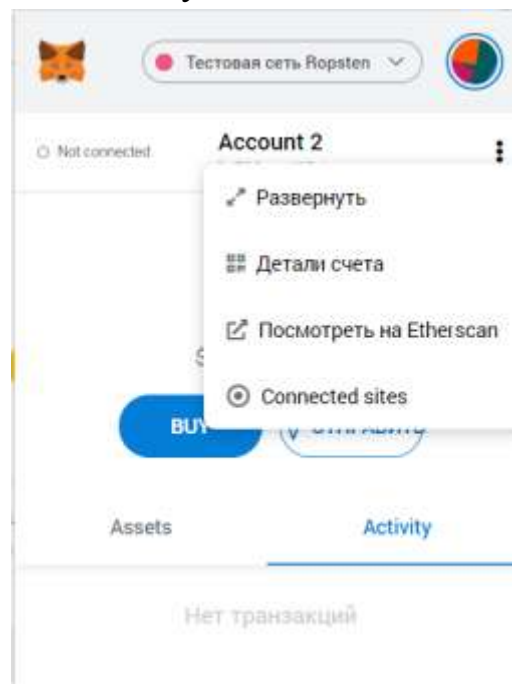
Для этого нужно в браузере перейти на сайт **remix** и открыть счет кошелька **MetaMask** в **свернутом** виде. Под картинкой лисички видим строку «Not connected».

Щелкните мышкой по этой надписи и подключите кошелек к сайту:

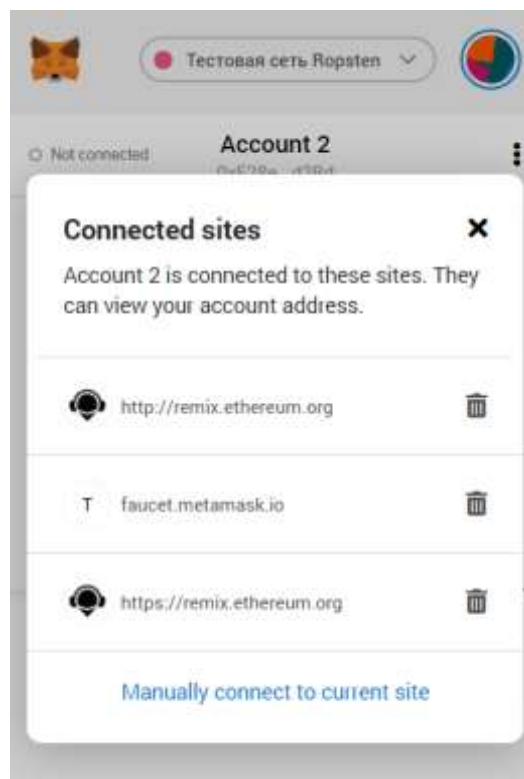


**Примечание: можно использовать вместо Rinkeby любую другую тестовую сеть.*

Также может быть полезным пункт меню «Connected sites»:



Здесь можно посмотреть подключенные сайты, отключиться от них, либо вручную подключиться к текущему сайту.



Безымянная платежная функция для смарт-контрактов

Для того чтобы на контракт можно было перевести эфиры, как на обычный кошелек, требуется создать в нем следующую **безымянную** функцию:

```
function () external payable {
}
```

После этого на данный смарт-контракт можно переводить эфиры.

Если же нужно **запретить** явный перевод эфиров на смарт-контракт, то нужно внутри этой функции данного контракта поместить вызов:

```
revert ();
```

Именованные платежные функции для смарт-контрактов

Часто возможностей безымянной платежной функции бывает недостаточно, тогда можно создать любое количество **именованных** платежных функций. **Следует заметить, что для таких функций невозможно задать возвращаемое значение произвольного типа!** Поэтому в качестве результата вызова функции можно, например, инициировать событие, а события всегда записываются в логи контракта. Рассмотрим следующий пример:

```
16     event PaymentEvent(string message, string returnValue);
17
18     function doPayment() public payable
19     {
20         require(msg.value >= 0.01 ether);
21         emit PaymentEvent("payment was sent", myText);
22     }
```

В строке 16 объявлено событие **PaymentEvent**, оно имеет 2 строковых параметра.

Функция **doPayment** объявлена с модификатором **payable**, поэтому она может принимать платежи.

Вызов функции **require(условие)** проверяет истинность условия и если условие не выполняется, то прекращает выполнение работы. В нашем

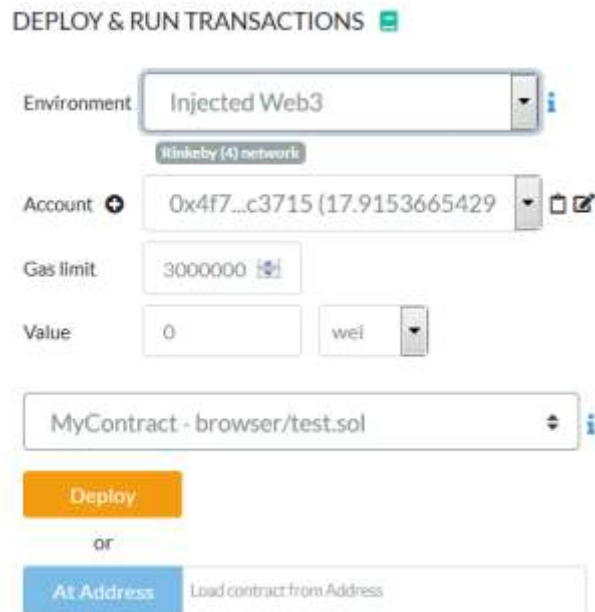
примере предполагаем, что сумма платежа должна быть не менее одной сотой эфира.

Далее команда **emit** инициирует событие **PaymentEvent** с заданными параметрами.

Полностью код контракта следующий:

```
Home | test.sol X
1 pragma solidity ^0.5.1;
2
3 contract MyContract{
4     address payable creator;
5     uint myNumber;
6     string myText;
7
8     constructor() public {
9         creator=msg.sender;
10        myNumber=3;
11        myText="ABC";
12    }
13    function () external payable {
14    }
15
16    event PaymentEvent(string message, string returnValue);
17
18    function doPayment() public payable
19    {
20        require(msg.value >= 0.01 ether);
21        emit PaymentEvent("payment was sent", myText);
22    }
23    function getCreator() public view returns (address) {
24        return creator;
25    }
26    function getNumber() public view returns (uint256) {
27        return myNumber;
28    }
29    function setMyNumber(uint256 num) public {
30        myNumber=num;
31    }
32    function kill() public {
33        if(msg.sender == creator) {
34            selfdestruct(creator);
35        }
36    }
37 }
```

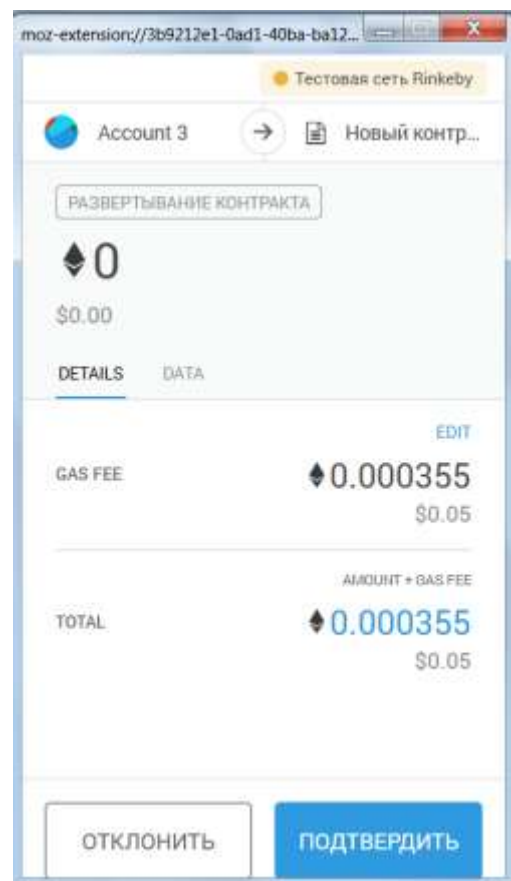
Откомпилируем контракт. Далее для запуска выберем в качестве среды **Injected Web3**, под списком указано, что тестовая сеть **Rinkeby** является текущей сетью на данный момент. Информация о текущей сети берется из **Metamask**.

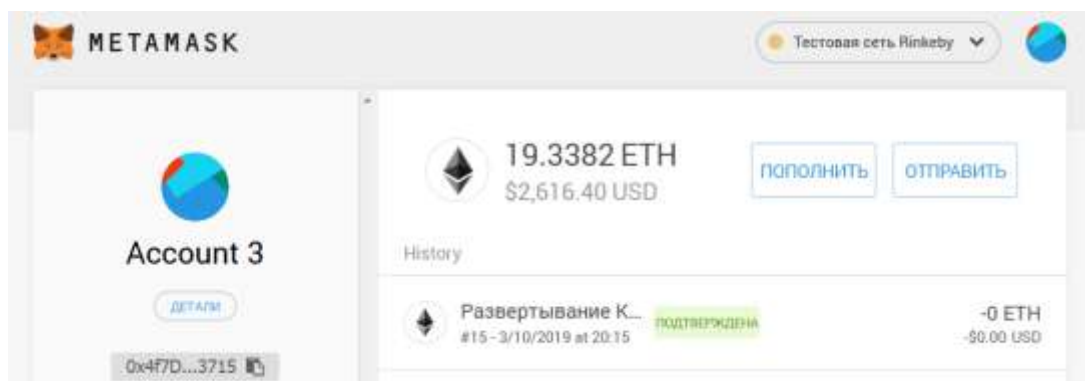


Если информации о текущей сети нет, проверьте, активен ли кошелек **MetaMask**.

При нажатии в среде **Remix** на кнопку **Deploy** появится окно с запросом к **текущему** кошельку (здесь это **Account3**) о запуске контракта.

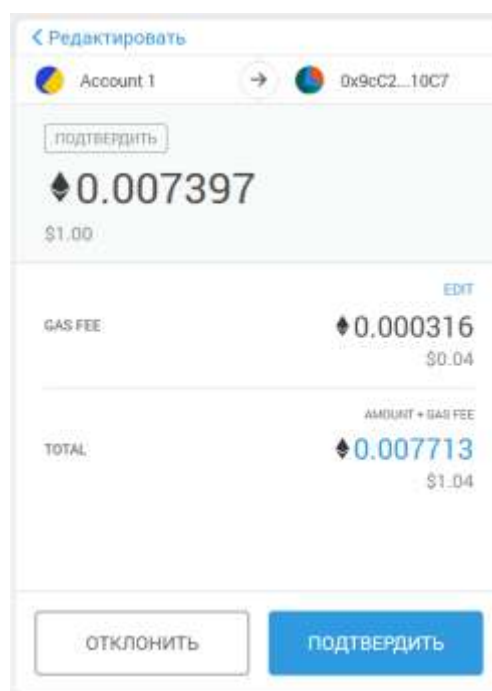
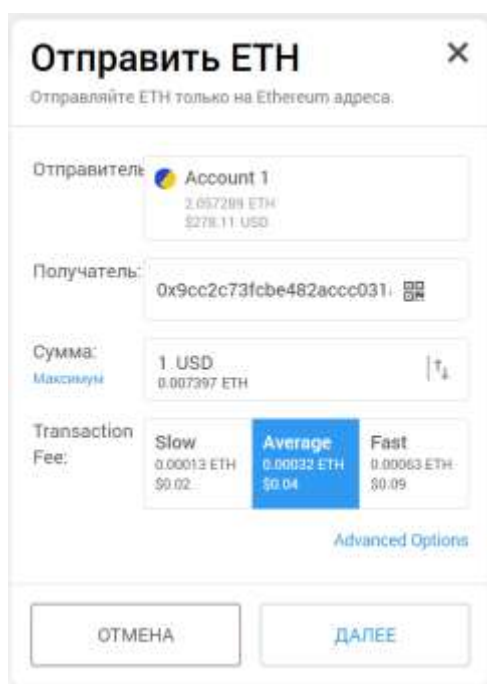
Обратите внимание, что эфиры в данной операции не перечисляются (сумма = 0), но запрашивается платеж в 5 центов за запуск контракта. Подтвердите платеж. Текущий счет стал создателем контракта, в его истории появилась строка о разворачивании контракта (детали всех транзакций можно всегда посмотреть на **EtherScan**).





Далее протестируем основные операции контракта.

1. Попробуем просто перечислить эфиры на счет контракта. Проведем эту транзакцию с другого счета. (Обратите внимание, мы здесь можем переслать и менее 0.01 эфира)



Запрашивается подтверждение, и операция выполняется.

Посмотрим историю контракта на **EtherScan**:

The screenshot shows the Etherscan interface for a contract on the Rinkeby testnet. The contract address is 0x9cC2c73FCBe482ACcc031ad72B27932bf5110C7. The contract overview shows a balance of 0.007397379846057818 Ether and 2 transactions. The transactions table lists the following:

TxHash	Block	Age	From	To	Value	[TxFee]
0x4f7db2eefc69438dE70E9Eb3d424A098c9c3715	4007358	5 mins ago	0x38f1cb18d287256f...	0x9cC2c73fcbe482a...	0.007397379846057 Ether	0.0002194
0x145835f417030e0...	4007292	21 mins ago	0x4f7db2eefc69438...	Contract Creation	0 Ether	0.000356355

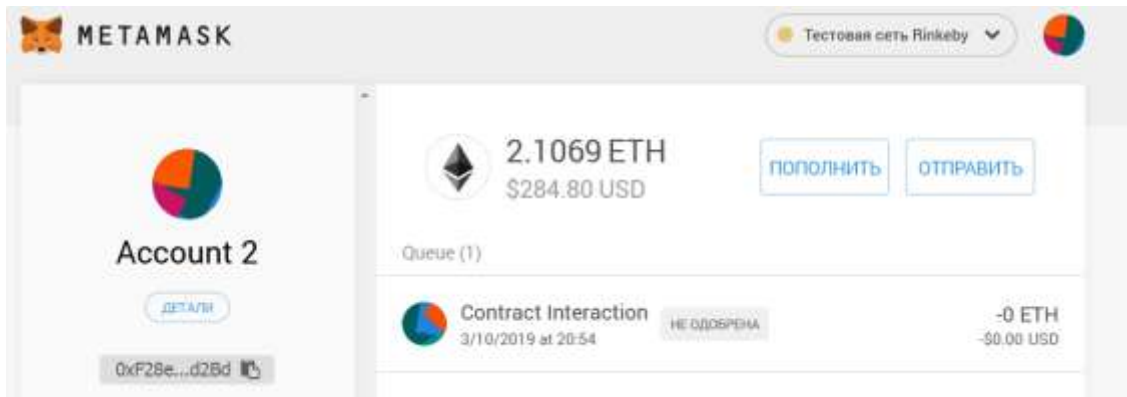
2. Теперь попробуем из среды **Remix** вызвать разные функции контракта.

Вызвать функции **getCreator** и **getNumber** можно бесплатно.

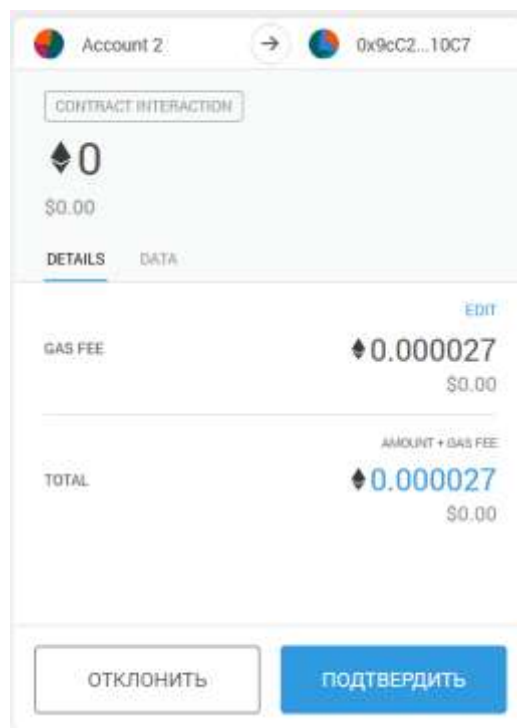
The screenshot shows the 'Deployed Contracts' panel in the Remix IDE. A contract named 'MyContract at 0xD2F...3C4ea (blockchain)' is selected. The interface displays several function buttons: '(fallback)', 'doPayment', 'kill', 'setMyNumber' (with a 'uint256 num' input field), 'getCreator', and 'getNumber'. Below the 'getCreator' button, the output shows the address: 0: address: 0x4f7DfB2EEFc69438dE70E9Eb3d424A098c9c3715. Below the 'getNumber' button, the output shows: 0: uint256: 3.

Перед вызовом функции **setMyNumber** установим в **MetaMask** в качестве активного счета **Account2**.

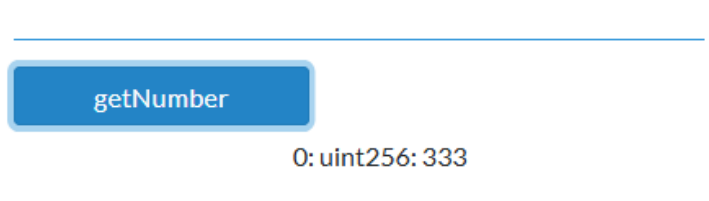
При вызове **setMyNumber** будет запрашиваться оплата этого действия. Если не было всплывающего окна, то переключимся в MetaMask:




Здесь висит неподтвержденная транзакция, подтвердим ее:




После прохождения транзакции функция **getNumber** будет возвращать новое значение:







Для вызова именованной платежной функции **doPayment** сумму, которую требуется перечислить, нужно предварительно указать в поле **Value** (обращайте внимание на единицу измерения рядом с полем Value!):

DEPLOY & RUN TRANSACTIONS 


Environment 

Rinkeby (4) network


Account   



Gas limit 



Value



Историю операций по контракту можно всегда посмотреть на **Etherscan**. Номер контракта можно найти и скопировать здесь:


Deployed Contracts 

▼ MyContract at 0xD2F...3C4ea (blockchain)  

Contract 0xD2Fb5a269C1DD98AeE571F17a4F00dc5e3a3C4ea  

Contract Overview


Balance: 0.02 Ether




More info 

My Name Tag: Not Available

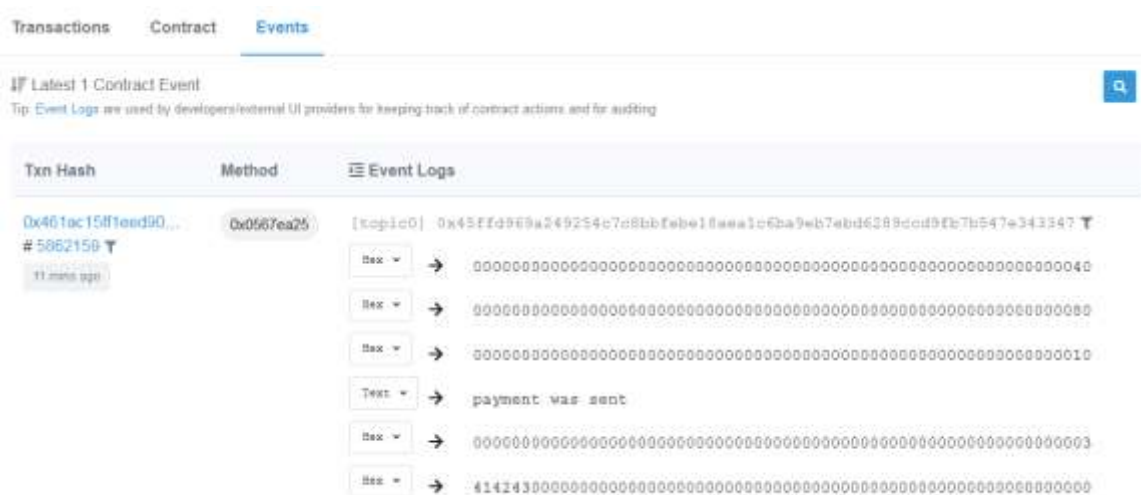
Contract Creator: [0x41fdb2ee6c6943f...](#) at [tx 0xb1fcb8b7eb4222...](#)

Transactions | Contract | Events

⌵ Latest 3 txns 

Txn Hash	Block	Age	From	To	Value	[Txn Fee]
0e461ac15f11ee990...	5862158	4 mins ago	0x29e3d2b5403b2d...	 0xd2fb5a269c1dd98...	0.02 Ether	0.00070201
0ed1b29e8b6bf7edc...	5862142	5 mins ago	0x29e3d2b5403b2d...	 0xd2fb5a269c1dd98...	0 Ether	0.00000602
0xb1fcb8b7eb4222...	5862123	13 mins ago	0x41fdb2ee6c6943f...	 Contract Creation	0 Ether	0.00297222

А на вкладке Events можно посмотреть информацию о событиях:



Если мы захотим уничтожить контракт (а это может выполнить только создатель контракта), весь его баланс будет перечислен на счет создателя контракта. Обратите внимание, что другие счета тоже могут вызывать функцию **kill**, и с них за это будет взиматься плата, но уничтожить контракт они не смогут – у них нет на это прав.

После уничтожения контракта его переменные, в том числе адрес создателя, обнуляются. Тем не менее, он остается в истории **Ethereum** – видимо, навечно. Кстати, его функции можно вызвать, хотя это и не имеет смысла (хотя плата за вызов соответствующих функций, разумеется, берется).

Задание для самостоятельной работы

1. Создайте пример контракта, содержащего:
 - бесплатные и платные функции,
 - безымянную платежную функцию,
 - именованную платежную функцию,
 - объявление и вызов пользовательского события.

Запустите контракт в тестовой среде **Rinkeby**. Проведите платеж с другого счета. Вызовите бесплатные и платные функции. Уничтожьте контракт.

В следующий раз мы будем обращаться к методам контракта из сценариев JavaScript. Поэтому:

2. Изучите основы JavaScript по какому-нибудь базовому учебнику, например: <http://www.wisdomweb.ru/JS/javascript-first.php>
3. Установите на свой компьютер Web-сервер (любой). Здесь даже подойдет отладочный Web-сервер для приложений ASP.Net.

Тема 4. Интерфейс для смарт-контрактов на JavaScript

Для того чтобы представленные примеры корректно выполнялись, нужно установить расширение MetaMask Legacy Web3:

<https://chrome.google.com/webstore/detail/metamask-legacy-web3/dgoegggfhkapjphahmgihfgemkgecdg>

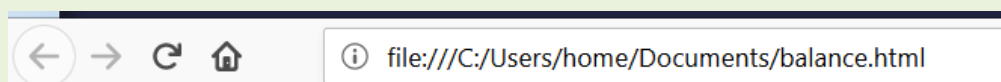
В этом задании мы рассмотрим, как из **HTML**-страниц, а точнее из клиентских сценариев **JavaScript** обращаться к смарт-контрактам.



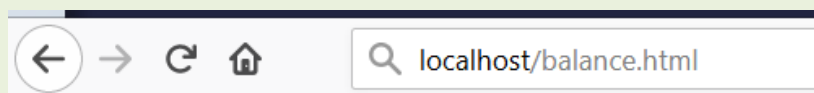
Напрямую из **JavaScript** к сети **Ethereum** обращаться невозможно, нужно использовать «посредника», например, узел **Web3.js**. В нашем случае он встроен в расширение браузера **MetaMask**.

Важное замечание:

для тестирования всех примеров из этого параграфа, которые обращаются к встроенному **Web3.js**, следует просматривать **HTML**-страницу не из файловой системы:



а в локальном режиме web-сервера:



Можете установить любой Web-сервер, или использовать уже установленный.

Например, очень прост и компактен Web-сервер **nginx**. Его даже не требуется устанавливать, достаточно только разархивировать. Если порт 80 занят, в файле конфигурации **nginx.conf** поменяйте порт, например, на 82 (и перезапустите сервер)

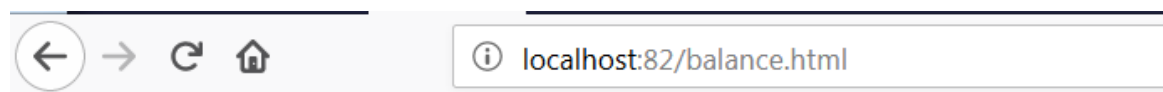
```
server {
    listen    82;
    server_name localhost;
    # итп.
}
```

Тогда сервер будет доступен по ссылке: <http://localhost:82>

Для остановки сервера выполняйте команду **nginx -s stop**

Подробнее см. http://nginx.org/ru/docs/beginners_guide.html

1. Рассмотрим и запустим следующий пример, который получает баланс любого счета по его номеру:



Получение баланса ETH

Введите адрес Ethereum:


```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="UTF-8">
5
6      <script type="text/javascript">
7
8  window.addEventListener('load', function() {
9  // Современные dapp браузеры...
10 if (window.ethereum) {
11     window.web3 = new Web3(ethereum);
12 }
13 // Устаревшие dapp браузеры...
14 else if (window.web3) {
```

```
15     window.web3 = new Web3(web3.currentProvider);
16   }
17   // Non-dapp браузеры...
18   else {
19     console.log('Ваш браузер не поддерживает Ethereum! Установите
расширение MetaMask!');
20   }
21 });
22 function getBalance() {
23   var address, wei, balance
24   address = document.getElementById("address").value
25   try {
26     web3.eth.getBalance(address, function (error, wei) {
27       if (!error) {
28         var balance = web3.fromWei(wei, 'ether');
29         document.getElementById("output").innerHTML = balance + "
ETH";
30       }
31     });
32   } catch (err) {
33     document.getElementById("output").innerHTML = err;
34   }
35 }
35 </script>
37 </head>
38 <body>
39   <h1>Получение баланса ETH</h1>
40   <p>Введите адрес Ethereum:</p>
41   <input type="text" size="50" id="address" />
42   <button type="button" onClick="getBalance();">Получить
баланс</button>
43   <br />
44   <br />
45   <div id="output"></div>
46 </body>
47 </html>
```

В строках 8-20 задается безымянная функция – обработчик события **load** (загрузка страницы). Здесь мы проверяем, встроен ли **web3** в браузер, и выдаем сообщение в случае неудачи. Обратите внимание, как создается объект в новых и старых браузерах.

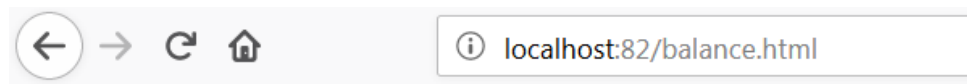
HTML-документ содержит текстовое поле **address** и кнопку. При нажатии на кнопку вызывается функция **getBalance**, заданная в строках 22-35.

В этой функции мы записываем в переменную **address** номер интересующего нас счета, а затем хотим получить его баланс, вызывая функцию **web3.eth.getBalance**.

Второй параметр этой функции – это так называемая *callback*-функция, или функция обратного вызова (она тоже безымянная). Она запускается после того, как произойдет возврат из функции **web3.eth.getBalance**. В

ней выполняется преобразование суммы баланса в эфиры и вывод полученного числа на экран.

Обратите внимание, что в данном примере не проверяется, существует ли на самом деле счет с заданным адресом. Главное – чтобы формат счета был правильным. Для несуществующих счетов будет выдаваться нулевой баланс.

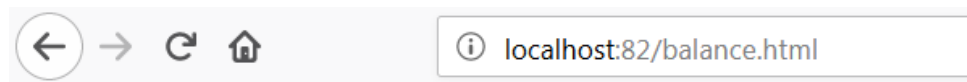


Получение баланса ETH

Введите адрес Ethereum:

Получить баланс

0.04 ETH



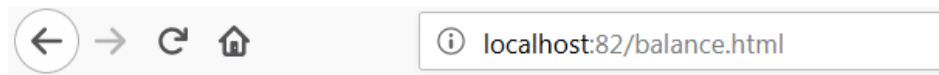
Получение баланса ETH

Введите адрес Ethereum:

Получить баланс

0 ETH

А вот если мы зададим **неправильный** формат счета (например, меньшее или большее количество символов), то будет вызвано исключение и напечатано сообщение об ошибке.



Получение баланса ЕТН

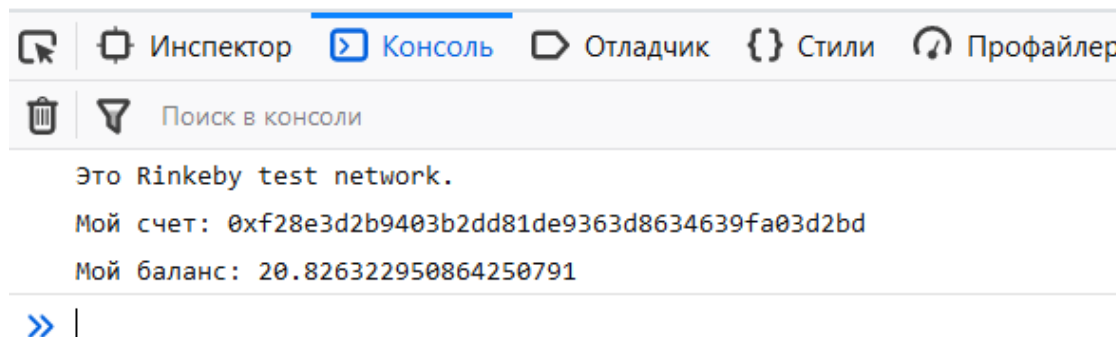
Введите адрес Ethereum:

Error: invalid address

2. Удобным средством отладки является консоль браузера (в *Firefox* это меню «Инструменты – Веб-разработка – Веб-консоль», в *Google Chrome* – «Дополнительные инструменты – Инструменты разработчика»).

В следующем примере мы определяем:

- тип текущей сети,
- номер и
- баланс своего текущего счета (открытого в браузере).



```
1 <HTML>
2 <HEAD>
3 <meta http-equiv="Content-Type" content="text/html;
  charset=utf-8">
4 <TITLE></TITLE>
5 <script language="JavaScript">
6 window.addEventListener('load', async() => {
7     // Современные dapp браузеры...
8     if (window.ethereum) {
9         window.web3 = new Web3(ethereum);
10        // Запрашивается доступ к счетам в Metamask
11        // await ethereum.enable();
12    }
13    // Устаревшие dapp браузеры...
14    else if (window.web3) {
```



```

15     window.web3 = new Web3(web3.currentProvider);
16   }
17   // Non-dapp браузеры...
18   else {
19     console.log ('Ваш браузер не поддерживает Ethereum!
Установите расширение MetaMask!');
20   }
21
22   web3.version.getNetwork(function (err, netId) {
23     switch (netId) {
24       case "1":
25         console.log ('Это основная сеть')
26         break
27       case "2":
28         console.log ('Это устаревшая Morden test network.')
29         break
30       case "3":
31         console.log ('Это ropsten test network.')
32         break
33       case "4":
34         console.log ('Это Rinkeby test network.')
35         break
36       case "42":
37         console.log ('Это Kovan test network.')
38         break
39       default:
40         console.log ('Это неизвестная сеть.')
41     }
42   });
43
44   web3.eth.getAccounts(function (err, acc){
45     if(err){
46       console.log ('error.....',err);
47     }
48     console.log ("Мой счет: "+ acc[0]);
49
50     web3.eth.getBalance(acc[0],function (err, bal){
51       if(err){
52         console.log('error.....',err);
53       }
54       console.log("Мой баланс: "+ web3.fromWei(bal,
'ether'));
55     });
56   });
57 });
58
59 });
60
61 </script>
62
63 </HEAD>
64 <BODY>
65 Тест веб-консоли
66 </BODY>
67 </HTML>

```

Как и ранее, все 3 вызываемые функции `web3.version.getNetwork`, `web3.eth.getAccounts`, `web3.eth.getBalance` требуют назначения `callback`-функций в качестве своих параметров.

В строках 11-12 закомментированы устаревшие действия:

```
// await ethereum.enable();
```

Если нам нужно было в сценарии обращаться к кошельку **Metamask**, ранее требовалось предварительно получить разрешение пользователя. На экран выводилось окно, в котором можно разрешить или запретить доступ. Разумеется, если пользователь не разрешал доступ, сценарий не получал информацию о номере и балансе текущего счета Metamask.

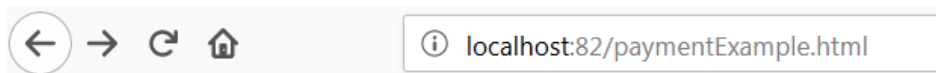
Обратите также внимание, что команда `await` вызывалась в асинхронном обработчике загрузки страницы (строка 6):

```
window.addEventListener('load', async() => {
```

3. Теперь будем тестировать платежную функцию, которую мы написали на предыдущем занятии.

```
16     event PaymentEvent(string message, string returnValue);
17
18     function doPayment() public payable
19     {
20         require(msg.value >= 0.01 ether);
21         emit PaymentEvent("payment was sent", myText);
22     }
```

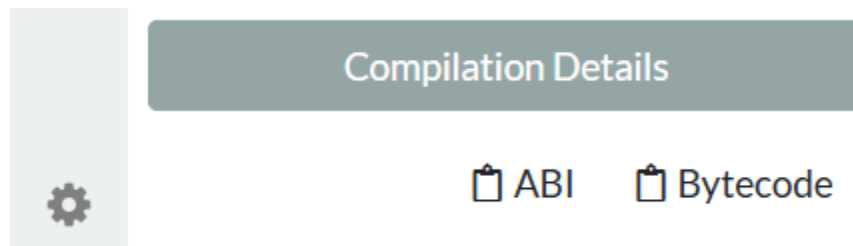
Так выглядит форма для платежа:



Провести платеж в ETH

Сумма платежа (ETH):

Прежде всего, нам будут нужны адрес и ABI (описание структуры) контракта. ABI можно скопировать в Remix на панели компиляции в самом низу:



Создадим соответствующие переменные для ABI и адреса контракта в сценарии:

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="UTF-8">
5  <script type="text/javascript">
6  var abi = [
7      {
8          "anonymous": false,
9          "inputs": [
10         {
11             "indexed": false,
12             "internalType": "string",
13             "name": "message",
14             "type": "string"
15         },
16     ],
17     ...
18 ];
19
20 var address = "0xf567d22C7b32b5D7ecC00C62D471F28a17ACE993";
21
22 window.addEventListener('load', async() => {
23     // Современные dapp браузеры...
24     if (window.ethereum) {
25         window.web3 = new Web3(ethereum);
26         // Запрашивается доступ к счетам в Metamask
27         // await ethereum.enable();
28     }
29     // Устаревшие dapp браузеры...
30     else if (window.web3) {
31         window.web3 = new Web3(web3.currentProvider);
32     }
33     // Non-dapp браузеры...
34     else {
35         console.log('Ваш браузер не поддерживает Ethereum!
36 Установите расширение MetaMask!');
37     }
38 }

```

```

114 });
115
116 function paymentExample() {
117     web3.eth.getAccounts(function(error, acc)
118         {
119             defaultAccount=acc[0];
120             console.log("defaultAccount="+defaultAccount);
121             contract = web3.eth.contract(abi).at(address);
122             console.log(contract);
123             sum = document.getElementById("sum").value;
124             sum = web3.toWei(sum, 'ether');
125             contract.doPayment.sendTransaction(
126                 {
127                     from: defaultAccount,
128                     value: sum
129                 },
130             function(error, data) {
131                 if(error!=null) console.log(error);
132                 console.log("Data="+data);
133             });
134         });
135     }
136
137 </script>
138 </head>
139 <body>
140     <h1>Провести платеж в ЕТН</h1>
141     <p>Сумма платежа (ЕТН):</p>
142     <input type="text" size="50" id="sum" />
143     <button type="button" onClick=" paymentExample();">
оплатить</button>
145
145 </body>
146 </html>

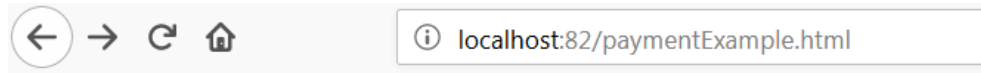
```

В этом сценарии созданная нами функция **paymentExample**:

- получает текущий счет для оплаты (из MetaMask);
- печатает в консоль номер текущего счета;
- создает переменную-контракт по заданному адресу и АБИ;
- печатает в консоль информацию о контракте;
- получает из поля формы сумму платежа (в эфирах) и переводит в самую мелкую единицу wei (1 эфир = 10^{18} wei);
- вызывает функцию контракта **doPayment**; поскольку это платежная функция, она вызывается как **doPayment.sendTransaction**. В эту функцию нужно передать JSON-объект, содержащий, как минимум, номер счета-плательщика и сумму контракта;
- печатает в консоль номер проведенной транзакции.

Наш контракт был запущен от имени **какого-либо** счета, в кошельке MetaMask переключимся к **другому** счету.

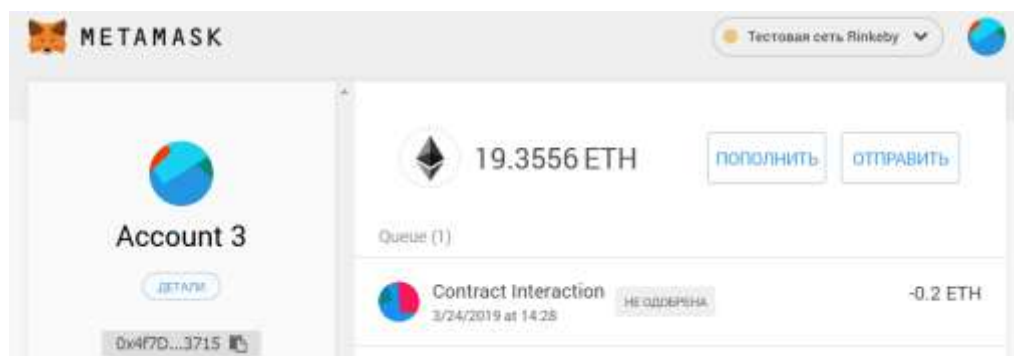
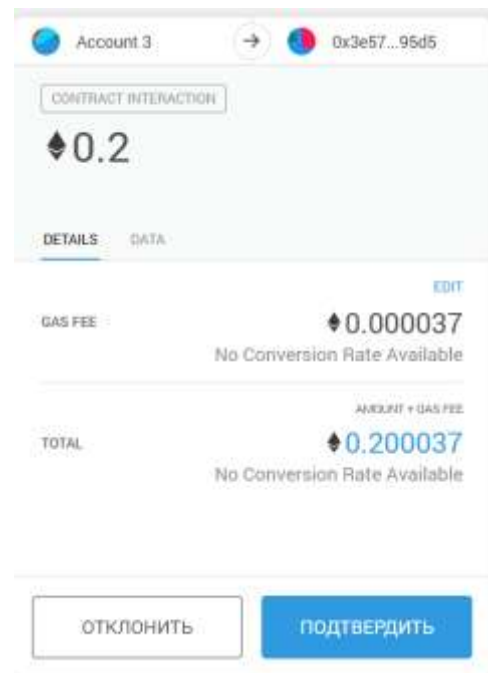
Загрузим эту страницу в браузере и заполним сумму:



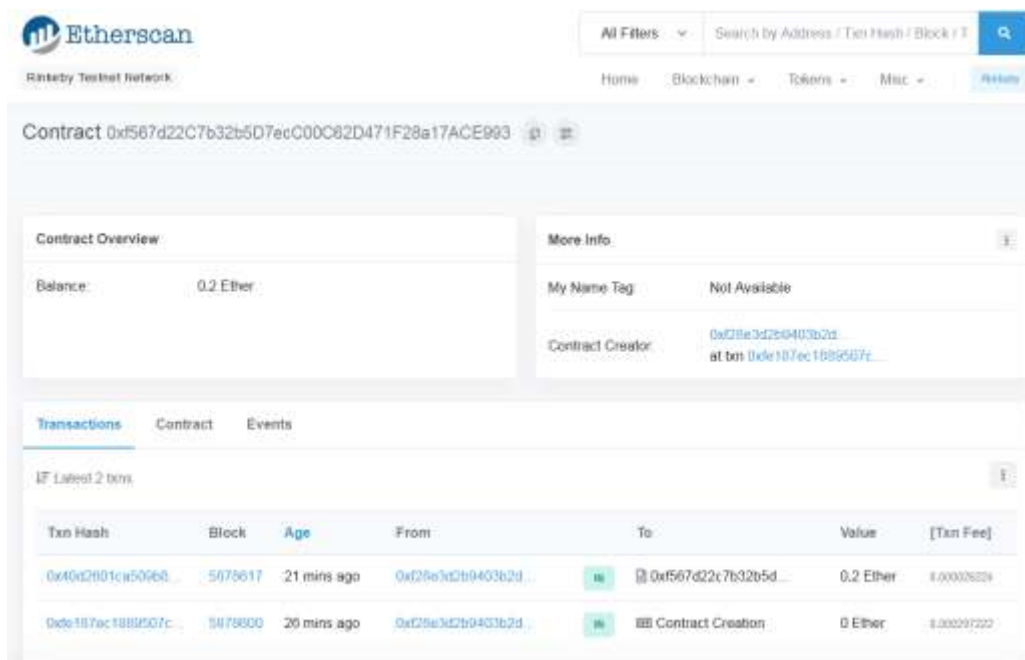
Провести платеж в ETH

Сумма платежа (ETH):

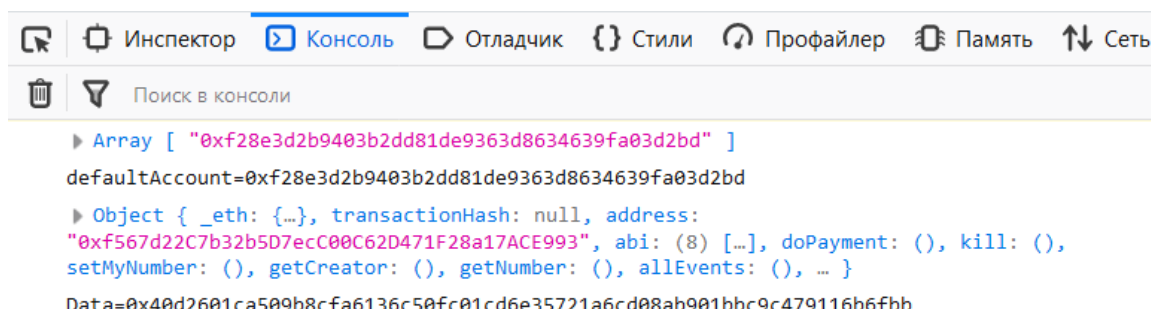
При нажатии на кнопку «оплатить» будет запущена транзакция от имени текущего счета, поэтому ее надо будет подтвердить в MetaMask:



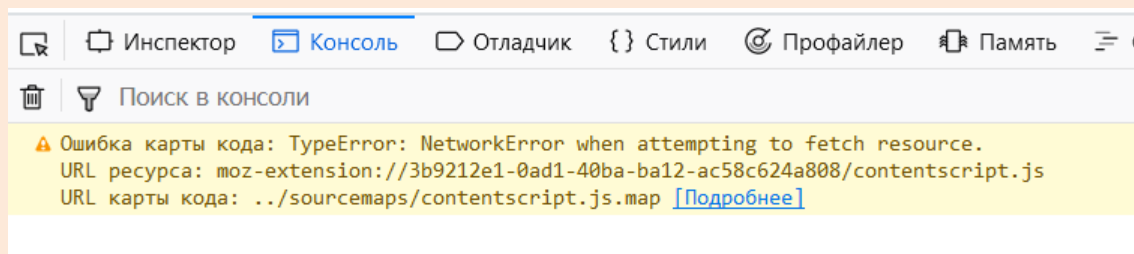
Посмотрим информацию о контракте в Etherscan:



В процессе выполнения данного сценария в консоли браузера (Firefox) были распечатаны следующие данные: текущий счет, контракт и возвращенный из функции **doPayment** номер транзакции:



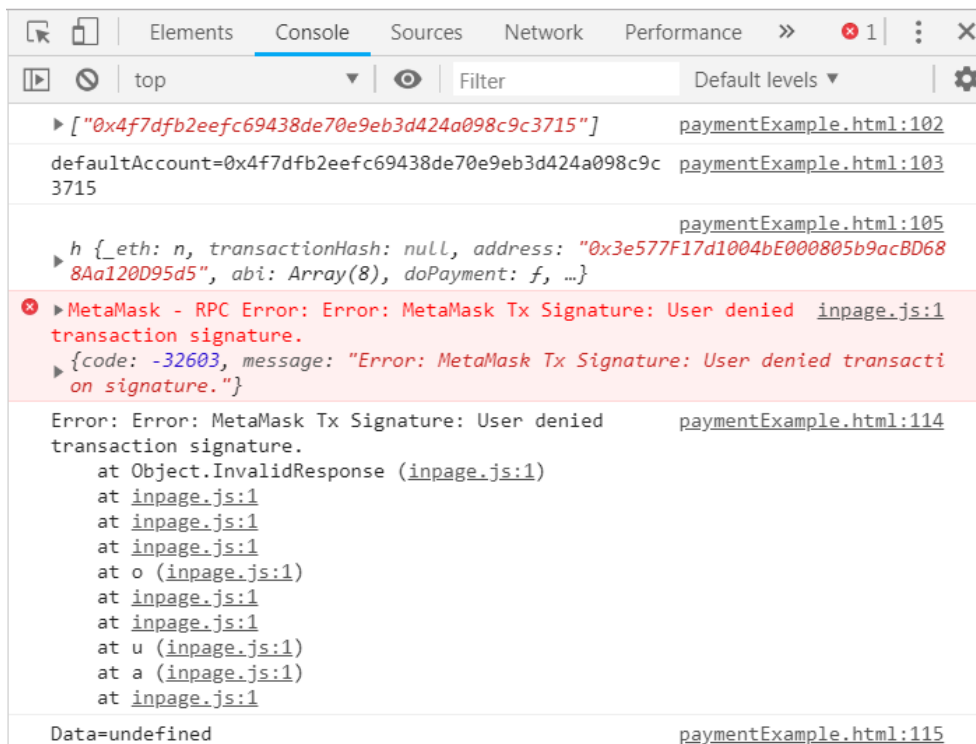
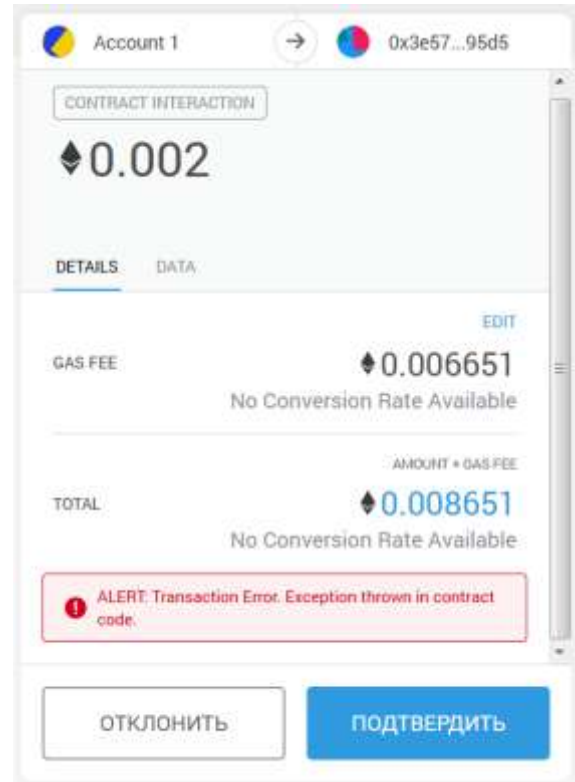
Примечание: при работе в **Firefox** в консоли может выдаваться ошибка:



На выполнение сценария она не влияет. В **Google Chrome** такой ошибки нет.

Теперь попробуем провести платеж на сумму меньше чем 0.01 эфира (на этот случай в методе контракта **doPayment** у нас есть проверка). Такая попытка вызовет ошибку в кошельке MetaMask:

Отклоним эту транзакцию, информация об этом будет отражена в консоли браузера (в примере приведена консоль браузера **Google Chrome**).



Задание для самостоятельной работы

Протестируйте все примеры из данного занятия.

Теперь вы уже можете разрабатывать собственные контракты. Выберите какую-нибудь тему из списка или предложите свою:

- Смарт-контракт для голосования.
- Смарт-контракт для подтверждения авторских прав.
- Смарт-контракт для продажи цифрового контента.
- Смарт-контракт для приема произвольных платежей.
- Смарт-контракт для оплаты счетов.
- Смарт-контракт для заключения пари.
- Смарт-контракт для хранения дипломов.
- Смарт-контракт для протекции сделки третьей стороной.
- Смарт-контракт для сделки с мультиподписями.
- Смарт-контракт для ... тему студент может выбрать сам, но ее нужно предварительно обсудить с преподавателем.

Следует разработать функции минимум для двух ролей – владелец контракта и пользователь контракта. В контракте должны присутствовать сложные типы данных. В контракте желательно использовать платежные функции.

1. Сформулируйте постановку задачи к следующему занятию.
2. До конца семестра разработайте контракт и протестируйте его на платформе Remix + MetaMask + JavaScript.

Приложение. Контракт для продажи промокодов

Цель:– создать контракт для продажи промокодов с web-страницы.

Пользователи:

- администратор,
- покупатель.

Функции администратора:

- создание контракта,
- задание новых промокодов,
- уничтожение контракта.

Функции покупателя:

- оплата промокода,
- получение оплаченного промокода.

Переменные контракта:

- адрес создателя контракта (администратора),
- текущий промокод,
- хэш-таблица оплаченных промокодов (ключ – адрес плательщика, значение – сам промокод).

Задание нового промокода может выполнить администратор в любой момент, старые промокоды не хранятся.

Покупатель **оплачивает** промокод, пересылая сумму не менее 0.01 eth. При этом в хэш-таблицу оплаченных промокодов добавляется новый элемент (или изменяется старый): ключ – адрес плательщика, значение – сам промокод.

После оплаты промокода покупатель может **неограниченное** число раз вызвать функцию получения оплаченного промокода (пока администратор не изменит промокод на новый). При вызове функции получения контракта проверяется, чему равно значение элемента хэш-таблицы с ключом-адресом. Если оно **равно текущему** промокоду, это значит, что с данного адреса поступила оплата этого промокода, **возвращается** значение промокода.

Переводить оплату непосредственно на счет контракта запрещается.

Удалить контракт может только его создатель. При удалении контракта все накопленные на нем эфиры переводятся на счет создателя контракта.

```
pragma solidity ^0.5.1;

contract MyContract{
    address payable creator;
    uint myPromo;
    mapping(address => uint) public orders;

    constructor() public {
        creator=msg.sender;
        myPromo=123;
    }
    function () external payable {
    }

    function getPromo() public view returns(uint)
    {
        require(orders[msg.sender] == myPromo);
        return myPromo;
    }

    event PaymentEvent(string message);

    function doPayment() public payable
    {
        require(msg.value >= 0.01 ether);
        orders[msg.sender]=myPromo;
        emit PaymentEvent("payment was sent");
    }
    function getCreator() public view returns (address) {
        return creator;
    }
    function setPromo(uint promo) public {
        if(msg.sender == creator) {
            myPromo=promo;
        }
    }
}
```

```
function kill() public {
    if(msg.sender == creator) {
        selfdestruct(creator);
    }
}
```

The screenshot shows a web browser at localhost:82/promoExample.html. The page title is "Контракт для продажи промокода". It contains three main sections:

- Сумма платежа (>0.01 ETH):** An input field with "0.02", a "pay" button, and a "get promo" button.
- Смена промокода администратором:** An input field with "777" and a "set promo" button.
- Уничтожение контракта:** A "kill" button.

The browser's developer console is open, displaying the following information:

```

▶ Array [ "0x38f1cb1f6287256f85af7d8bbe393c763df7780b" ]
defaultAccount=0x38f1cb1f6287256f85af7d8bbe393c763df7780b
▶ Object { _eth: {...}, transactionHash: null, address:
"0x61f1c86b0b19c910022b233a239b440853d03e0", abi: (9) [...], doPayment: (), kill: (), setPromo:
(), getCreator: (), getPromo: (), orders: (), ... }
Number=777
    
```

HTML-страница для тестирования контракта

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<script type="text/javascript">
    var abi=[
    {
        "constant": false,
        "inputs": [],
        "name": "doPayment",
        "outputs": [],
```

```
        "payable": true,  
        "stateMutability": "payable",  
        "type": "function"  
    },  
    ...  
];  
  
var address = "0x61f1cC86b0b19C910022B233a239B440853d03e0";
```

```
window.addEventListener('load', async() => {  
    // Современные dapp браузеры...  
    if (window.ethereum) {  
        window.web3 = new Web3(ethereum);  
        // Запрашивается доступ к счетам в Metamask  
        await ethereum.enable();  
    }  
    // Устаревшие dapp браузеры...  
    else if (window.web3) {  
        window.web3 = new Web3(web3.currentProvider);  
    }  
    // Non-dapp браузеры...  
    else {  
        console.log ('Ваш браузер не поддерживает Ethereum!  
Установите расширение MetaMask!');  
    }  
});
```

```
function paymentExample() {  
    web3.eth.getAccounts(function(error, acc) {  
        defaultAccount=acc[0];  
        console.log(acc);  
        console.log("defaultAccount="+defaultAccount);  
        contract = web3.eth.contract(abi).at(address);  
        console.log(contract);  
        sum = document.getElementById("sum").value;  
        sum = web3.toWei(sum, 'ether');  
        contract.doPayment.sendTransaction(  
            {  
                from: defaultAccount,  
                value: sum  
            },  
            function(error, data) {  
                if(error!=null) console.log(error);  
                console.log("Data="+data);  
            }  
        );  
    });  
}
```

```

        });
    });
}

function kill() {
    web3.eth.getAccounts(function(error, acc) {
        defaultAccount=acc[0];
        console.log(acc);
        console.log("defaultAccount="+defaultAccount);
        contract = web3.eth.contract(abi).at(address);
        console.log(contract);
        contract.kill(function(error) {
            console.log("Killed");
        });
    });
}

function setPromo() {
    web3.eth.getAccounts(function(error, acc) {
        defaultAccount=acc[0];
        console.log(acc);
        console.log("defaultAccount="+defaultAccount);
        contract = web3.eth.contract(abi).at(address);
        console.log(contract);
        newPromo = document.getElementById("new").value;
        contract.setPromo(newPromo, function(error, data)
    {
        console.log("Number="+data);
    });
    });
}

function getPromo() {
    web3.eth.getAccounts(function(error, acc) {
        defaultAccount=acc[0];
        console.log(acc);
        console.log("defaultAccount="+defaultAccount);
        contract = web3.eth.contract(abi).at(address);
        console.log(contract);
        contract.getPromo(function(error, data) {
            console.log("Number="+data);
            document.getElementById("newPromo").innerHTML =
data;
        });
    });
}

```

```
}

</script>
</head>
<body>
  <h1>Контракт для продажи промокода</h1>
  <p>Сумма платежа (>0.01 ETH):</p>
  <input type="text" size="50" id="sum" />
  <button type="button"
onClick="paymentExample();">pay</button>
  <button type="button" onClick="getPromo();">get
promo</button>
  <div id="newPromo"></div>
  <hr>
  <p>Смена промокода администратором:</p>
  <input type="text" size="50" id="new" />
  <button type="button" onClick="setPromo();">set
promo</button>
  <p>Уничтожение контракта:</p>
  <button type="button" onClick="kill();">kill</button>
</body>
</html>
```

Литература

- 1. Дрешер Д.** Основы блокчейна: вводный курс для начинающих в 25 небольших главах / Даниэль Дрешер. - М.: ДМК Пресс, 2018. - 312 с.: ил.
- 2. Свон М.** Блокчейн: схема новой экономики / Мелани Свон. - М.: Олимп-бизнес, 2017. - 240 с., ил. ISBN 978-5-9693-0360-7
- 3. Поппер Н.** Цифровое Золото. Невероятная история биткойна или о том, как идеалисты и бизнесмены изобретают деньги заново / Натаниэль Поппер. - М.: Диалектика, 2016. - 75 с. ISBN 978-5-8459-2079-9
- 4. Даннен К.** Введение в Ethereum и Solidity. Основы криптовалют и программирования блокчейнов для начинающих (перевод Dannen Ch. Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain, Programming for Beginners. – Apress, 2017.)

Интернет-ресурсы



MetaMask

<https://metamask.io>



Remix - Solidity IDE

<http://remix.ethereum.org>



Документация по Solidity

<https://solidity.readthedocs.io/en/v0.5.4/>



Учебник по JavaScript

<http://www.wisdomweb.ru/JS/javascript-first.php>