



Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа № 2

«Основы работы с TON: создание кошелька и взаимодействие со смарт-контрактом»

по курсу «Распределенные системы обработки информации и
блокчейн-технологии»

Калуга, 2024

Содержание

Содержание	2
Цели и задачи на лабораторную работу.....	3
Теоретическая часть.....	3
Выводы	7
Практическая часть	9
Установка библиотек	9
Создание кошелька	9
Развёртывание кошелька	11
Отправка тонкоинов.....	13
Работа со смарт-контрактом	14
Требования к представлению результатов и отчёту.....	17
Задания на лабораторную работу	18
Контрольные вопросы	20

Цели и задачи на лабораторную работу

Целью лабораторной работы является изучение основных принципов работы кошельков в блокчейне TON и принципов взаимодействия со смарт-контрактами, а также освоение практических навыков работы с блокчейном на языке Python.

Основные задачи лабораторной работы:

- Изучить основные принципы работы кошельков в TON.
- Получить практические навыки работы с библиотеками `tonsdk`, `tontools` и `pytonlib` на языке Python.
- Создать и развернуть кошелек.
- Выполнить переводы тонкоинов
- Освоить взаимодействие со смарт-контрактом с помощью библиотек.
- Выполнить задания согласно варианту.

Теоретическая часть

The Open Network (TON) - это децентрализованная и открытая интернет-платформа, состоящая из нескольких частей: TON Blockchain, TON DNS, TON Storage и TON Sites. Блокчейн TON - это основа, соединяющая базовую инфраструктуру TON в единую экосистему TON.

Блокчейн TON задуман как распределенный суперкомпьютер, или «суперсервер», предназначенный для предоставления различных продуктов и услуг, способствующих развитию децентрализованного видения нового интернета. TON стремится к совместимости между разными блокчейнами, работая в безопасной и масштабируемой системе. Он может обрабатывать миллионы транзакций в секунду.

Кошелек позволяет пользователям управлять своими активами на блокчейне TON. Он состоит из нескольких основных частей:

- **Адрес кошелька** - криптографический хэш-код кошелька.
- **Публичный ключ** - это открытый ключ, который используется для проверки подлинности транзакций и получения средств. Публичный ключ служит для идентификации пользователя в сети TON. Любой может получить публичный ключ, так как он используется для отправки средств на кошелек пользователя. Например, вы хотите отправить токены на чей-то кошелек, вы используете публичный ключ этого кошелька для подтверждения, что средства отправляются правильному получателю.

- **Приватный ключ** - это секретный ключ, который используется для подписи транзакций и доступа к вашим активам. Приватный ключ является самым важным элементом безопасности в кошельке. Он позволяет пользователю подписывать транзакции и подтверждать право собственности на активы. Когда вы хотите отправить токены с вашего кошелька, вы используете приватный ключ для подписания транзакции. Это гарантирует, что только владелец приватного ключа может отправить средства.

- **Мнемоническая фраза** - это фраза из 24 случайных слов, предназначенная для восстановления доступа к кошельку, без которой кошелек не получится восстановить.

Кошелек в TON является особым смарт-контрактом в сети блокчейн, так как он выполняет функции, которые характерны для смарт-контрактов.

- **Автоматическое выполнение.** Кошелек в TON программируется таким образом, чтобы выполнять определенные действия на основе предопределенных условий. Например, он может выполнять автоматические транзакции, проверять балансы и на основании этого выполнять определенные действия.

- **Хранение активов.** Кошелек в TON может хранить различные активы, такие как токены, NFT и т.д.. Он может отслеживать балансы и обеспечивать их безопасное хранение.

TON-кошелек может принимать одно из следующих состояний:

1. **Uninit (Создан)** - состояние кошелька, который был только что создан, но еще не активирован.
2. **Active (Активирован)** - состояние активного кошелька, который может отправлять и получать транзакции.
3. **Frozen (Заморожен)** - состояние кошелька, в котором нельзя отправлять транзакции, но можно получать. Обычно устанавливается для безопасности или при необходимости временно выключить кошелек.
4. **Deactivated (Отключен)** - состояние кошелька, в котором нельзя отправлять и получать транзакции. Может быть установлено, например, при утере кошелька или подозрении на взлом.
5. **Overloaded (Перегружен)** - состояние перегруженного кошелька, который превысил свою емкость и временно не может выполнять операции.
6. **Forgotten (Забыт)** - состояние кошелька, который долгое время не используется и запоминается блокчейном для оптимизации, но не доступен для использования до активации.

Кошелёк в TON работает по следующему принципу:

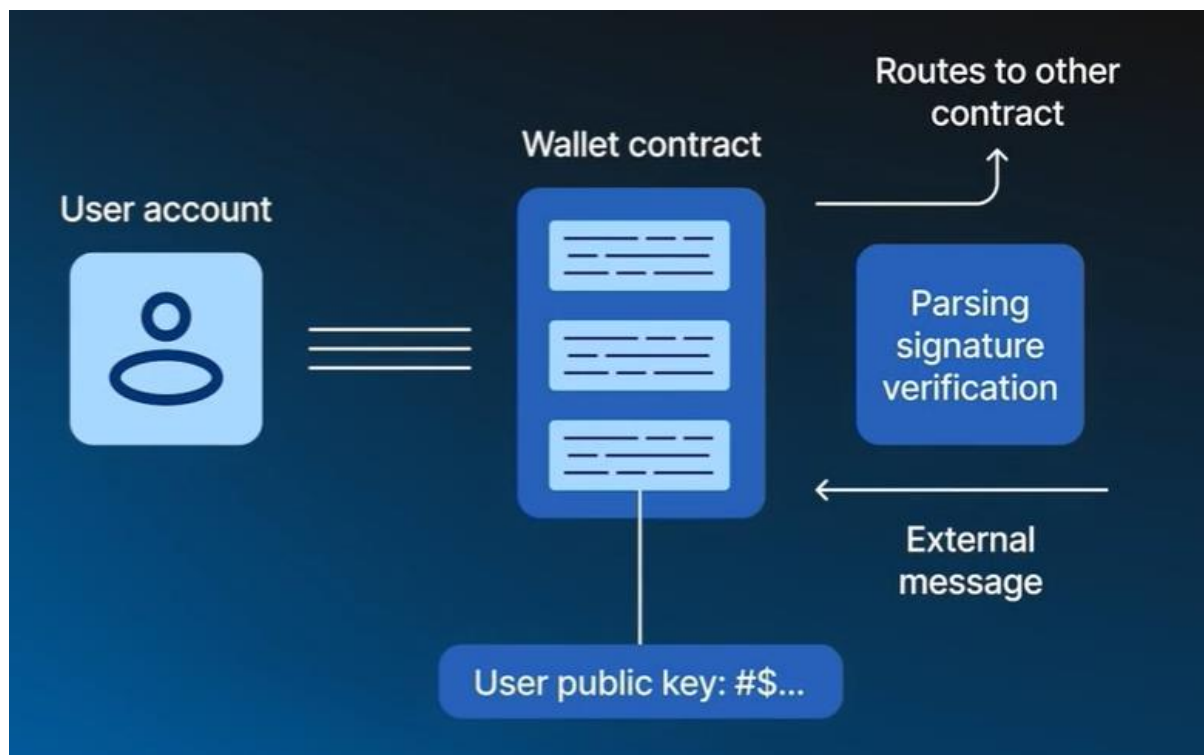


Рисунок 1 - схема работы кошелька TON

Кошелёк создаётся вместе с аккаунтом пользователя, и хранит публичный ключ пользователя. При поступлении на контракт кошелька сообщений, он проверяет подпись с помощью публичного ключа в сообщении, обновляет свой собственный счётчик для избежания повторной обработки одного и того же сообщения, и либо отправляет, либо принимает, например, тонкоины. Стоит отметить, что токен в TON тоже является контрактом, одним из полей которого является его владелец. Поэтому, одним из этапов перевода тонкоинов является изменение поля владельца.

Чтобы защитить сеть от DoS-атак типа «отказ в обслуживании», все контракты должны платить за свое функционирование в сети. Эта оплата состоит из множества частей, включающих аренду, стоимость исполнения, маршрутизацию сообщений и не только. Рассмотрим подробнее наиболее важные составляющие:

- Каждый раз при выполнении контракта вы работаете со **стоимостью газа**. Исполнение любой инструкции в виртуальной машине TON (TVM) можно оценить в абстрактных единицах — **в газе**. На уровне сети установлен такой параметр, как **цена газа**, который определяет, сколько TON вы должны заплатить за каждую инструкцию. Чем дольше работает ваша программа, тем выше будет стоимость газа, и она будет вычитаться из баланса контракта. Когда баланс вашего контракта опустится до нуля, его исполнение прекратится, и связанная транзакция завершится неудачей. Плата за газ гарантирует, что вы не увеличите нагрузку на сеть TON без оплаты этой нагрузки.

- TON взимает оплату — «аренду» — за каждый бит данных контракта каждую секунду его работы в сети, и эта сумма вычитается из баланса контракта.

- Маршрутизация сообщений

Выводы

- The Open Network (TON) представляет собой децентрализованную и открытую интернет-платформу, состоящую из нескольких частей: TON Blockchain, TON DNS, TON Storage и TON Sites. Блокчейн TON является основой, соединяющей базовую инфраструктуру TON в единую экосистему.
- В блокчейне TON основной действующей единицей является смарт-контракт, поскольку кошелёк и тонкоины являются особыми смарт-контрактами.
- Кошелек в TON позволяет пользователям управлять своими активами на блокчейне. Он состоит из адреса кошелька (криптографический хэш-код), публичного ключа (для проверки подлинности транзакций и получения средств), приватного ключа (для подписи транзакций и доступа к активам) и мнемонической фразы (для восстановления доступа).
- Для работы с кошельком в TON необходимы: адрес кошелька, мнемоническая фраза, публичных и закрытый ключи.
- Кошелек в TON является особым смарт-контрактом, выполняющим автоматическое выполнение транзакций и хранение активов. Он может принимать различные состояния: Uninit (Создан), Active (Активирован), Frozen (Заморожен), Deactivated (Отключен), Overloaded (Перегружен) и Forgotten (Забыт).
- Для защиты от DoS-атак блокчейн берёт оплату за «газ», ренту и маршрутизацию сообщений.
- При нулевом балансе смарт-контракт, в том числе и кошелёк, замораживаются.

Практическая часть

Установка библиотек

```
tonsdk: pip install tonsdk  
pytonlib от psylopunk: pip install ton  
pytonlib от toncenter: pip install pytonlib
```

При работе с библиотекой tonlib от psylopunk необходимо поменять бинарный файл, поскольку данная библиотека давно не обновлялась. Сделать это можно 2 способами: скомпилировать в официальной репозитории TON, или взять из репозитория tonlib от toncenter:

<https://github.com/toncenter/pytonlib/tree/main/pytonlib/distlib>

Затем файл можно скопировать в рабочую директорию, и указать при инициализации путь к нему.

```
client = TonlibClient  
    (config='https://ton.org/testnet-global.config.json',  
     ls_index=11, verbosity_level=3)  
  
client.enable_unaudited_binaries()  
await client.init_tonlib  
    (cdll_path='tonlibjson.amd64.dll')
```

Далее рассмотрим примеры создания, развёртывания кошелька и отправки с него тонкоинов на примере библиотек tonsdk и tonlib от toncenter.

Создание кошелька

Первым делом, для работы с TON, необходимо создать кошелёк. Создадим кошелёк с помощью библиотеки tonsdk.

```
from tonsdk.contract.wallet import Wallets, WalletVersionEnum  
mnemonics, public_key, private_key, wallet=  
Wallets.create(version=WalletVersionEnum.v3r2, workchain=0)  
if __name__ == "main":  
    print(mnemonics)  
    print(wallet.address.to_string(True, True, True, True))
```

Запустим код, и получим следующий результат:

```
PS D:\Desktop\TON lessons> d;; cd 'd:\Desktop\TON lessons'; & 'C:\Users\Admin\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Admin\.vscode\extensions\ms-python.python-2023.6.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '61191' '--' 'd:\Desktop\TON lessons\1 lesson\wallet\wallet-creation.py'
['disorder', 'ritual', 'detail', 'evil', 'famous', 'search', 'cause', 'youth', 'dust', 'laptop', 'spirit', 'course', 'trash', 'enrich', 'divorce', 'pelican', 'flip', 'snack', 'track', 'example', 'engine', 'private', 'book', 'hand']
kQDPqai-70BZlgy91KwoyGv9gk3J6H9Yw1BifXoDMCQtYiIM
PS D:\Desktop\TON lessons> d;; cd 'd:\Desktop\TON lessons'; & 'C:\Users\Admin\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Admin\.vscode\extensions\ms-python.python-2023.6.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '61211' '--' 'd:\Desktop\TON lessons\1 lesson\wallet\wallet-creation.py'
['citizen', 'naive', 'treat', 'mercy', 'result', 'brave', 'arena', 'lens', 'add', 'hip', 'concert', 'prize', 'issue', 'until', 'radar', 'noodle', 'urge', 'mammal', 'opera', 'tool', 'flush', 'beef', 'innocent', 'rule']
kQCCyVl6AFCjgP5hWtp-cwD8MagikKPXOh5hZ7RZvfjeXW1e
PS D:\Desktop\TON lessons>
```

В массиве записана мнемоника кошелька. Мнемоника — это пароль, который состоит из 12 отдельных слов. Мнемоническая фраза используется для восстановления кошелька. Использование мнемонической фразы гарантирует безопасность кошелька в сети TON. Под мнемоникой располагается адрес кошелька - уникальный идентификатор, по которому блокчейн TON понимает, куда, например, отправлять тонкоины. Скопируйте адрес кошелька куда-нибудь, он пригодится позже.

При повторном запуске будет создан другой кошелек, поэтому если мы хотим работать с одним кошельком, нам надо запомнить мнемонику и изменить код:

```
from tonsdk.contract.wallet import Wallets, WalletVersionEnum

mnemonics, public_key, private_key, wallet=
Wallets.from_mnemonics(mnemonics=mnemonics,
version=WalletVersionEnum.v3r2, workchain=0)

if __name__ == "main":
    print(mnemonics)
    print(wallet.address.to_string(True, True, True, True))
```

Если запустить данный код, то метод вернёт тот же самый адрес кошелька.

```
['citizen', 'naive', 'treat', 'mercy', 'result', 'brave', 'arena', 'lens', 'add', 'hip', 'concert', 'prize', 'issue', 'until', 'radar', 'noodle', 'urge', 'mammal', 'opera', 'tool', 'flush', 'beef', 'innocent', 'rule']
kQCCyVl6AFCjgP5hWtp-cwD8MagikKPXOh5hZ7RZvfjeXW1e
PS D:\Desktop\TON lessons> d;; cd 'd:\Desktop\TON lessons'; & 'C:\Users\Admin\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\Admin\.vscode\extensions\ms-python.python-2023.6.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '61469' '--' 'd:\Desktop\TON lessons\1 lesson\wallet\wallet-creation.py'
['citizen', 'naive', 'treat', 'mercy', 'result', 'brave', 'arena', 'lens', 'add', 'hip', 'concert', 'prize', 'issue', 'until', 'radar', 'noodle', 'urge', 'mammal', 'opera', 'tool', 'flush', 'beef', 'innocent', 'rule']
kQCCyVl6AFCjgP5hWtp-cwD8MagikKPXOh5hZ7RZvfjeXW1e
PS D:\Desktop\TON lessons>
```

Развёртывание кошелька

Для того, чтобы развернуть наше приложение с кошельком TON, воспользуемся библиотекой `pytonlib` от `toncenter`.

Библиотека `pytonlib` - асинхронная, это надо иметь ввиду при написании программы.

```
from pytonlib import TonlibClient
import requests
from pathlib import Path
import asyncio

async def main():
    #ссылка на конфигурационный файл для тестовой сети
    url = " https://ton.org/testnet-global.config.json"
    #Получение конфигурационного файла в формате JSON
    config =requests.get(url).json()
    # Директория, в которой будет храниться ключ
    keystore_dir = '/tmp/ton_keystore'
    Path(keystore_dir).mkdir(parents=True, exist_ok=True)
    # создание клиента для работы с кошельком
    client = TonlibClient(ls_index=0, config
    = config,keystore=keystore_dir)
```

Далее, для развёртывания программы необходимо получить инициализирующее сообщений. Это можно сделать с помощью метода `create_init_external_message()`. Данный метод является частью библиотеки `tonsdk`, его можно вызвать из переменной `wallet`, объявленной в предыдущем примере.

```
query = wallet.create_init_external_message()
message = query['message'].to_boc(False)
print(message)
```

Метод `create_init_external_message()` возвращает словарь, содержащий информацию об адресе, сообщении и т.д. Нас интересует поле `'message'`. Выведем его:

Отправка тонкоинов

Процедура отправки тонкоинов, по сути, ничем не отличается от процедуры развёртывания кошелька. Разница только в том, что при отправке используется метод `wallet.create_transfer_message()`.

Перед тем, как отправить тонкоины, нам необходимо узнать `seqno` - порядковый номер последней транзакции, отправленной кошельком. Каждый раз, когда кошелек отправляет транзакцию, `seqno` увеличивается.

Однако, может появиться такая ошибка:

```
pytonlib.tonlibjson.LiteServerTimeout: LITE_SERVER_NETWORKadnl query timeout
```

Дело в том, что сервера в `testnet` могут быть загружены, и поэтому не отвечают. Можно попробовать поменять параметры `ls_index` (отвечает за адрес сервера) и указать время ожидания при инициализации клиента в библиотеке `toncenter`, что было ранее.

```
async def get_seqno(client: TonlibClient, address: str):
    data = await client.raw_run_method(method='seqno',
    stack_data=[], address=wallet_address)
    return int(data['stack'][0][1], 16)
```

После получения `seqno` можно отправлять сообщение:

```
#Создание сообщения для отправки тонов
seqno = await get_seqno(client, wallet_address)
transfer_query= wallet.create_transfer_message
(to_addr='Адрес кошелька, на который будет выполнен перевод',
amount=to_nano(0.01, 'ton'),
seqno=seqno,
payload='Hello world')
transfer_msg= transfer_query['message'].to_boc(False)
print(transfer_msg)

await client.raw_send_message(transfer_msg)
```

Обратите внимание, что для перевода используются нанотоны, т.е. 10^{-9} тона. Для удобства можно импортировать функцию `to_nano`:

```
from tonsdk.utils import to_nano
```

После выполнения кода в `tonscan` отразится данная транзакция

Address	kQCXsaHkjHjOM7BFtWS4QTrG7q6gfc-zxr4FkgWLi6MaRER				
Balance	1,995406552 TON				
Last activity	15 seconds ago				
State	● Active				
Contract Type	wallet v3 r2				

TRANSACTIONS					
Age	From	To	Value		
15 seconds ago	0QCD39VS5jcpthL8vMjEXrzGaRcCVYt... Og8xqPvC	IN	0QCAxsaHkjHjOM7BFtWS4QTrG7q6gfc-... Li6MaUzU	0.0096 TON	▼
15 seconds ago	0QCAxsaHkjHjOM7BFtWS4QTrG7q6gfc-... Li6MaUzU	OUT	0QCD39VS5jcpthL8vMjEXrzGaRcCVYt... Og8xqPvC	0.01 TON	▼
16 hours ago	kQCES0TZYqcVkgoguhib8iMEo4cvaEw... gnN8ftBF	IN	0QCAxsaHkjHjOM7BFtWS4QTrG7q6gfc-... Li6MaUzU	2 TON	▼

Проверяем в кошельке, на который перевели тоны:

7 minutes ago	0QCAxsaHkjHjOM7BFtWS4QTrG7q6g- Li6MaUzU	IN	0QCD39VS5jcpthL8vMjEXrzGaRcCV... Og8xqPvC	0.01 TON	▲
Status	Timestamp	Hash	Logical time	Fee	Message
OK	06.10.2024, 14:36:48	UrPvxgTtpBKE88BH6Y1tC1dbtm0JrRJyJVoVf04dYIk=	26652279000003	0.000133364 TON	Hello world

Работа со смарт-контрактом

Смарт-контракты в TON - это программы, которые хранятся и выполняются непосредственно на блокчейне. Они обеспечивают сквозную децентрализацию и устойчивость к ошибкам и вредоносным атакам, проверяя результаты своих исполнений множеством узлов сети. В отличие от централизованных систем, где результаты исполнения контролируются одним субъектом, в TON каждый узел сети может независимо проверять корректность взаимодействия с системой и выполнение вычислений.

Создадим и развернём простейший смарт-контракт:

```
() recv_internal(int msg_value, cell in_msg, slice
in_msg_body) impure {
    slice cs = in_msg.begin_parse();
    int flags = cs~load_uint(4);
```

```

slice sender_address = cs~load_msg_addr();
int op = in_msg_body~load_uint(32);
if (op == 1) {
    slice ds = get_data().begin_parse();
    int counter_value = ds~load_uint(32);
    set_data(
begin_cell().store_uint(counter_value + 1, 32)
.store_slice(sender_address).end_cell()
    );
    return ();
}
return ();
}

(int, slice) get_contract_storage_data() method_id {
    slice ds = get_data().begin_parse();
    return (
        ds~load_uint(32), ;; counter_value
        ds~load_msg_addr() ;; the most recent sender
    );
}

```

Данный код написан на языке FunC и представляет собой реализацию смарт-контракта счётчика для TON. Этот контракт позволяет увеличивать значение счётчика при получении определённого типа сообщения и предоставляет метод для получения текущего значения счётчика и адреса отправителя последнего сообщения, которое увеличило счётчик.

Рассмотрим функции кода смарт-контракта:

1. **recv_internal** — это процедура, которая обрабатывает входящие сообщения, проверяет тип сообщения (op) и, если он равен 1, увеличивает значение счётчика на 1 и сохраняет адрес отправителя в хранилище данных контракта. Параметр op сообщает о том, что сообщение внутреннее.

2. **get_contract_storage_data** — это функция, которая возвращает текущее значение счётчика и адрес отправителя последнего сообщения, увеличившего счётчик.

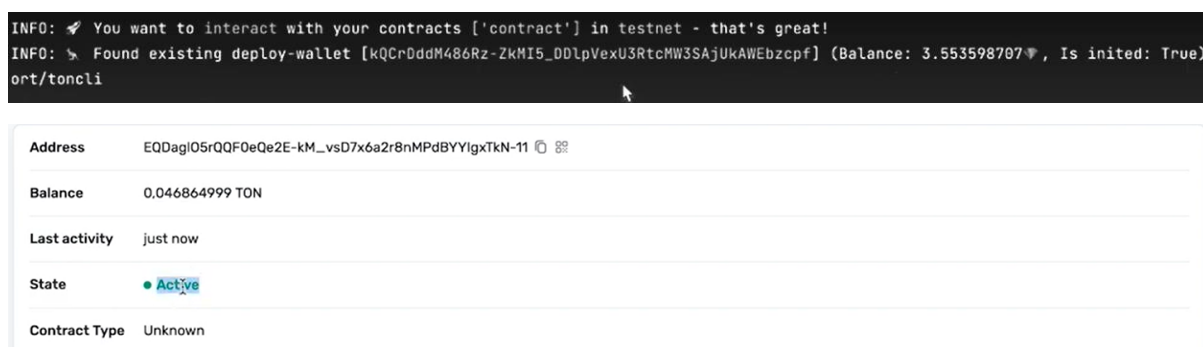
Первым делом, для работы со смарт-контрактом скачайте приложенный файл code.func. Далее развернём его. Введём команды:

```
toncli start wallet -n counter
```

В созданной директории func меняем файл code.func на имеющийся, а затем:

```
toncli deploy -n testnet
```

В итоге выведется сообщение, что смарт-контракт развёрнут с адресом смарт-контракта. Скопируем его и вставим в tonscan, чтобы проверить, что тонкоины перевелись, а контракт развёрнут.



The image shows a terminal window at the top with the following output:

```
INFO: 🚀 You want to interact with your contracts ['contract'] in testnet - that's great!  
INFO: 🔍 Found existing deploy-wallet [kQCrDddM486Rz-ZkMI5_ODlpVexU3RtcMW3SAjUKAWEBzcpf] (Balance: 3.553598707, Is inited: True)  
ort/toncli
```

Below the terminal is a screenshot of the wallet details page on tonscan.com. The details are as follows:

Address	EQDagI05rQQF0eQe2E-kM_vsD7x6a2r8nMPdBYyIgxTkN-11
Balance	0.046864999 TON
Last activity	just now
State	Active
Contract Type	Unknown

Далее, проверим, что счётчик смарт-контракта увеличен на 1. Для этого необходимо запустить его GET-метод, описанный в теле контракта. Это делается с помощью метода `raw_run_method`.

```
client.raw_run_method(address, method='get_contract_storage_data', stack_data=[])
```

В результате мы получим список, где в секции `stack` будет увеличившееся на единицу значение:



The image shows a JSON response from the contract:

```
{ '@type': 'smc.runResult', 'gas_used': 505, 'stack': [['num', '0x1']], [ 'cell', { 'bytes': 'te6ccKEBAQEAJAAQ4APQYzg/wHmeXSmmrtR95TTzjk47332drjsVpg8xsLrtDLCUzr', 'obje
```


Требования к представлению результатов и отчёту

Результатами лабораторной работы являются:

- 2 созданных и развёрнутых кошелька на платформе TON
- Выполнение переводов между кошельками согласно варианту
- Созданный и развёрнутый смарт-контракт
- Оформленный отчёт

Отчёт должен содержать:

- Титульный лист
- Цели и задачи
- Процесс выполнения работы со скриншотами и их описанием
- Выводы
- Ответы на контрольные вопросы

Задания на лабораторную работу

Задание 1. Создать и развернуть TON-кошелёк. Использовать библиотеки согласно варианту:

1. tonsdk и pytonlib от toncenter
2. tonsdk и pytonlib от psylopunk
3. tonsdk и tontools
4. pytonlib от toncenter и pytonlib от psylopunk
5. pytonlib от toncenter и tontools
6. pytonlib от psylopunk и tontools

Задание 2. Перевести тоны между созданными ранее кошельками. Перевести тоны обратно. При разработке программы необходимо сделать проверку статуса проведённой транзакции. При переводе дополнительно выполнить задачи согласно варианту:

1. Перевести с первого кошелька, разработанного при помощи tonsdk, случайное количество тонов в пределах 0.1 - 0.3 тона. В сообщении указать баланс кошелька. Перевести со второго кошелька хэш последней транзакции.

2. Перевести с первого кошелька, разработанного при помощи tonsdk, случайное количество тонов в пределах 0.1 - 0.3, указав в сообщении публичный ключ кошелька. Перевести со второго кошелька случайное количество тонов в пределах 0.1 - 0.3 тона, указав историю транзакций на этом кошельке.

3. Перевести с первого кошелька, разработанного при помощи tonsdk, случайное количество тонов в пределах 0.1 - 0.3, указав в сообщении первые 3 слова мнемонической фразы. Перевести со второго кошелька случайное количество тонов в пределах 0.1 - 0.3 тона, указав баланс кошелька после поступления на него тонов.

4. Перевести с первого кошелька, разработанного при помощи tonsdk, случайное количество тонов в пределах 0.1 - 0.3, указав в сообщении

последние 3 транзакции. Перевести со второго кошелька случайное количество тонов в пределах 0.1 - 0.3 тона, указав первую транзакцию первого кошелька.

5. Перевести с первого кошелька, разработанного при помощи tonsdk, случайное количество тонов в пределах 0.1 - 0.3 тона. В сообщении указать хэш и баланс кошелька. Перевести со второго кошелька случайное количество тонов в пределах 0.1 - 0.3 тона, указав баланс кошелька и последнюю транзакцию.

6. Перевести случайное количество тонов в пределах 0.1-0.3 тона, прикрепив id последней транзакции. Перевести со второго кошелька случайное количество тонов в пределах 0.1 - 0.3 тона, указав последние 2 транзакции на этом кошельке.

Задание 3. Создать и развернуть смарт-контракт. Увеличить значение счётчика до 5.

Контрольные вопросы

1. Что такое платформа TON?
2. Для чего платформа TON задумана как распределённый суперкомпьютер?
3. Из каких основных составляющих состоит кошелёк на платформе TON?
4. Какой вид имеет адрес кошелька?
5. В чём отличие между публичным и приватным ключами?
6. Для чего нужна мнемоническая фраза?
7. Является ли кошелёк в платформе TON смарт-контрактом? Если да, то почему?
8. Какие состояния есть у кошелька в TON?
9. За что блокчейн TON берёт плату? Для чего это нужно?
10. Опишите алгоритм запуска в работу кошелька в TON
11. Опишите принцип взаимодействия со смарт-контрактами.
12. Чем отличаются библиотеки `pytonlib` от `psyloupunk` и `pytonlib` от `toncenter`?