Моделирование работы смарт-контрактов в блокчейне Ethereum

Цель работы: Использовать Python для симуляции работы простого смарт-контракта.

Описание работы:

Создать простую модель смарт-контракта для проведения платежей (например, передача средств между двумя пользователями). Использовать библиотеку web3.py для взаимодействия с тестовой сетью Ethereum. Визуализировать результат транзакций и изменения балансов.

Устанавливает библиотеку Web3 для работы с Ethereum блокчейном, подключается к сети через Infura провайдер и проверяет статус соединения. Это базовый шаг для любых операций с блокчейном - чтения балансов, отправки транзакций или работы со смарт-контрактами.

```
!pip install Web3
!pip install Web3.isConnected#
```

```
Collecting Web3
  Downloading web3-7.13.0-py3-none-any.whl.metadata (5.6 kB)
Collecting eth-abi>=5.0.1 (from Web3)
  Downloading eth_abi-5.2.0-py3-none-any.whl.metadata (3.8 kB)
Collecting eth-account>=0.13.6 (from Web3)
  Downloading eth_account-0.13.7-py3-none-any.whl.metadata (3.7 kB)
Collecting eth-hash>=0.5.1 (from eth-hash[pycryptodome]>=0.5.1->Web3)
  Downloading eth_hash-0.7.1-py3-none-any.whl.metadata (4.2 kB)
Collecting eth-typing>=5.0.0 (from Web3)
  Downloading eth_typing-5.2.1-py3-none-any.whl.metadata (3.2 kB)
Collecting eth-utils>=5.0.0 (from Web3)
  Downloading eth_utils-5.3.1-py3-none-any.whl.metadata (5.7 kB)
Collecting hexbytes>=1.2.0 (from Web3)
  Downloading hexbytes-1.3.1-py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: aiohttp>=3.7.4.post0 in /usr/local/lib/python3.12/dist-packages (from Web3) (3.12.15)
Requirement already satisfied: pydantic>=2.4.0 in /usr/local/lib/python3.12/dist-packages (from Web3) (2.11.7)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.12/dist-packages (from Web3) (2.32.4)
Requirement already satisfied: typing-extensions>=4.0.1 in /usr/local/lib/python3.12/dist-packages (from Web3) (4.15.0)
Collecting types-requests>=2.0.0 (from Web3)
  Downloading types_requests-2.32.4.20250809-py3-none-any.whl.metadata (2.0 kB)
Requirement already satisfied: websockets<16.0.0,>=10.0.0 in /usr/local/lib/python3.12/dist-packages (from Web3) (15.0.1)
Collecting pyunormalize>=15.0.0 (from Web3)
  Downloading pyunormalize-16.0.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.pos
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.post0->Web
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.post0->Web3)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.post0->We
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.post0->
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.post0->Web
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp>=3.7.4.post0->We
Collecting parsimonious<0.11.0,>=0.10.0 (from eth-abi>=5.0.1->Web3)
  Downloading parsimonious-0.10.0-py3-none-any.whl.metadata (25 kB)
Collecting bitarray>=2.4.0 (from eth-account>=0.13.6->Web3)
  Downloading bitarray-3.7.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (34 kB)
Collecting eth-keyfile<0.9.0,>=0.7.0 (from eth-account>=0.13.6->Web3)
  Downloading eth_keyfile-0.8.1-py3-none-any.whl.metadata (8.5 kB)
Collecting eth-keys>=0.4.0 (from eth-account>=0.13.6->Web3)
  Downloading eth_keys-0.7.0-py3-none-any.whl.metadata (13 kB)
Collecting eth-rlp>=2.1.0 (from eth-account>=0.13.6->Web3)
  Downloading eth_rlp-2.2.0-py3-none-any.whl.metadata (3.3 kB)
Collecting rlp>=1.0.0 (from eth-account>=0.13.6->Web3)
  Downloading rlp-4.1.0-py3-none-any.whl.metadata (3.2 kB)
Collecting ckzg>=2.0.0 (from eth-account>=0.13.6->Web3)
  Downloading ckzg-2.1.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (887 bytes)
Collecting pycryptodome<4,>=3.6.6 (from eth-hash[pycryptodome]>=0.5.1->Web3)
  Downloading pycryptodome-3.23.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Collecting cytoolz>=0.10.1 (from eth-utils>=5.0.0->Web3)
  Downloading cytoolz-1.0.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.6 kB)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2.4.0->We
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2.4.0->Web
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic>=2.4.0->
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->Web3) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->Web3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.23.0->Web3)
Requirement already satisfied: toolz>=0.8.0 in /usr/local/lib/python3.12/dist-packages (from cytoolz>=0.10.1->eth-utils>=5
Requirement already satisfied: regex>=2022.3.15 in /usr/local/lib/python3.12/dist-packages (from parsimonious<0.11.0,>=0.1
```

Этот код представляет собой базовый скрипт для работы с блокчейном Ethereum через библиотеку Web3.py. Он подключается к тестовой сети Ropsten через сервис Infura, проверяет статус подключения и получает балансы Ether для двух указанных адресов кошельков.

Код также содержит закомментированный пример реализации отправки транзакции, который показывает как создать, подписать и отправить перевод средств между кошельками. Скрипт служит основой для разработки более сложных

взаимодействий с блокчейном, таких как отслеживание балансов, отправка транзакций и работа со смарт-контрактами в тестовой среде перед переходом на основную сеть.

```
from web3 import Web3

# Подключаемся к тестовой сети (например, Ropsten)
# Замените 'YOUR_INFURA_PROJECT_ID' на ваш реальный ID
infura_url = "https://ropsten.infura.io/v3/YOUR_INFURA_PROJECT_ID"
web3 = Web3(Web3.HTTPProvider(infura_url))

# Проверяем соединение
if web3.is_connected():
    print('Connected to Ethereum network')
else:
    print('Failed to connect to Ethereum network')

# Указываем адреса отправителя и получателя
# Замените '0xSenderAddress' и '0xReceiverAddress' на реальные адреса
sender = "0xSenderAddress"  # Замените на адрес отправителя
receiver = "0xReceiverAddress"  # Замените на адрес получателя

# Проверка балансов (требует подключения к сети и действительных адресов)
try:
    sender_balance = web3.eth.get_balance(sender)
    receiver_balance = web3.eth.get_balance(receiver)
    print(f"Баланс отправителя: {web3.fromWei(sender_balance, 'ether')} ETH")
    print(f"Бас отправителя: {web3.fromWei(sender_balance, 'ether')} ETH")
    print(f"Баланс получателя: {web3.fromWei(receiver_balance, 'ether')} ETH")
except Exception as e:
    print(f"Ошибка при получении баланса: {e}")

# Симуляция перевода средств (требует дальнейшей реализации с подписанием транзакции)
# Код для подписания и отправки транзакции
# Это более сложный шаг, который требует закрытого ключа отправителя и создания подписанной транзакции
print("\nСимуляция перевода средств (требуется реализация)")
# Пример: how to send a simple transaction (requires private key management)
# from eth_account import Account
# account = Account.from_key("YOUR_PRIVATE_KEY") # Replace with your private key
# nonce = web3.eth.get_transaction_count(account.address)
# tx = {
#     'nonce': nonce,
#     'to': receiver,
#     'value': web3.toWei(0.001, 'ether'), # Amount to send
#     'gas': 21000,
#     'gasPrice': web3.toWei('50', 'gwei')
# }
# signed_tx = account.sign_transaction(tx)
# tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
# print(f"Транзакция отправлена: {tx_hash.hex()}")
```

```
Failed to connect to Ethereum network
Ошибка при получении баланса: ENS name: '0xSenderAddress' is invalid.

Симуляция перевода средств (требуется реализация)
```

Выводы:

Модель линейной регрессии позволяет спрогнозировать рост объема транзакций. Можно использовать другие модели для повышения точности прогнозирования. Комментарии:

Студенты могут экспериментировать с разными моделями машинного обучения и улучшать свои прогнозы.

Анализ безналичных платежей в России с помощью Python

Цель работы: Проанализировать данные о количестве безналичных платежей в России.

Описание работы:

Скачать статистику платежных систем в России (например, из открытых источников). Использовать библиотеку pandas для обработки данных о платежах. Построить графики с использованием matplotlib для наглядного представления трендов в платежах.

Генерация файла payment_data.csv к заданию

Этот код создает демонстрационные данные о платежах и сохраняет их в CSV-файл:

Что делает код: Создает таблицу с данными о платежах за разные годы (2018-2021) с разделением по типам оплаты (карта и мобильный). Для каждого года и типа платежа указывается сумма. Данные сохраняются в файл payment_data.csv для

последующего анализа, построения графиков или обработки в аналитических системах.

```
import pandas as pd
import io

# Create sample data for payment_data.csv
data = {
    'Year': [2018, 2018, 2019, 2019, 2020, 2020, 2021, 2021],
    'Payment_Type': ['Card', 'Mobile', 'Card', 'Mobile', 'Card', 'Mobile', 'Card', 'Mobile'],
    'Amount': [1000, 500, 1200, 600, 1500, 700, 1800, 800]
}
df_sample = pd.DataFrame(data)

# Save the sample data to a CSV file
csv_data = df_sample.to_csv('payment_data.csv', index=False)
```

Этот код создает и обучает модель линейной регрессии для прогнозирования объема транзакций на основе временных данных.

Что делает код: Берет данные о количестве транзакций за 5 лет (2016-2020), разделяет их на обучающую и тестовую выборки, обучает модель линейной регрессии и строит прогноз. Затем визуализирует результаты на графике, где красные точки показывают реальные значения, а синяя линия - прогноз модели.

Цель кода: Продемонстрировать базовый пример машинного обучения для прогнозирования временных рядов, показать как работает линейная регрессия для выявления тренда и сделать простой прогноз будущих значений на основе исторических данных.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

# Пример данных
years = np.array([2016, 2017, 2018, 2019, 2020]).reshape(-1, 1)
transactions = np.array([100, 150, 200, 250, 300])

# Разделение на обучающие и тестовые данные
X_train, X_test, y_train, y_test = train_test_split(years, transactions, test_size=0.2)

# Модель линейной регрессии
model = LinearRegression()
model.fit(X_train, y_train)

# Прогнозирование
predictions = model.predict(X_test)

# Визуализация
plt.scatter(X_test, y_test, color='red')
plt.plot(X_test, predictions, color='blue')
plt.title('Прогноз объема транзакций')
plt.xlabel('Год')
plt.ylabel('Объем транзакций')
plt.show()
```

Выводы:

Анализ показывает рост безналичных транзакций в России. Можно сделать вывод о переходе к более цифровым формам платежей, особенно в период пандемии. Комментарии:

Студенты могут использовать эти данные для анализа тенденций на рынке и разработки прогнозов на будущее.

Прогнозирование объемов транзакций с использованием машинного обучения

Цель работы: Использовать машинное обучение для прогнозирования объемов транзакций в платежных системах России.

Описание работы:

Собрать данные о транзакциях за несколько лет. Применить алгоритмы машинного обучения, такие как линейная регрессия или модель ARIMA для прогнозирования. Визуализировать результаты прогнозирования.

Эти примеры демонстрируют, как с помощью Python можно проводить анализ данных, моделировать процессы и прогнозировать результаты в области платежных систем и блокчейн-технологий.

Анализ данных транзакций в платежных системах с визуализацией

Описание: Использование Python для анализа транзакционных данных в платежных системах. Пример данных может включать транзакции по дням, суммы платежей, частоту использования.

Решение:

Этот код создает демонстрационные данные о финансовых транзакциях и сохраняет их в CSV-файл.

Что делает код: Генерирует таблицу с датами транзакций (с 1 по 4 января 2023 года) и соответствующими суммами платежей. Некоторые даты содержат несколько транзакций. Данные сохраняются в файл transactions.csv для последующего анализа.

Структура данных:

Дата транзакции (в правильном datetime формате)

Сумма транзакции (разные значения от 50 до 300)

Файл предназначен для анализа ежедневных транзакций, вычисления статистик (средний чек, общая выручка по дням) и построения временных графиков финансовой активности.

```
import pandas as pd
import io

# Create sample data for transactions.csv
data = {
    'date': pd.to_datetime(['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-03', '2023-01-03', '2023-01-03', '2023-01-04
    'amount': [100, 50, 200, 150, 75, 120, 300]
}
df_transactions = pd.DataFrame(data)

# Save the sample data to a CSV file
df_transactions.to_csv('transactions.csv', index=False)
```

Этот код выполняет анализ и визуализацию данных о финансовых транзакциях.

Что делает код: Загружает данные из CSV-файла с транзакциями, группирует транзакции по датам и подсчитывает количество операций в каждый день. Затем создает столбчатую диаграмму, которая наглядно показывает активность транзакций по дням.

Цель кода: Проанализировать распределение транзакционной активности во времени, выявить дни с наибольшим и наименьшим количеством операций, и визуализировать эту информацию для быстрого понимания паттернов активности клиентов или пользователей системы.

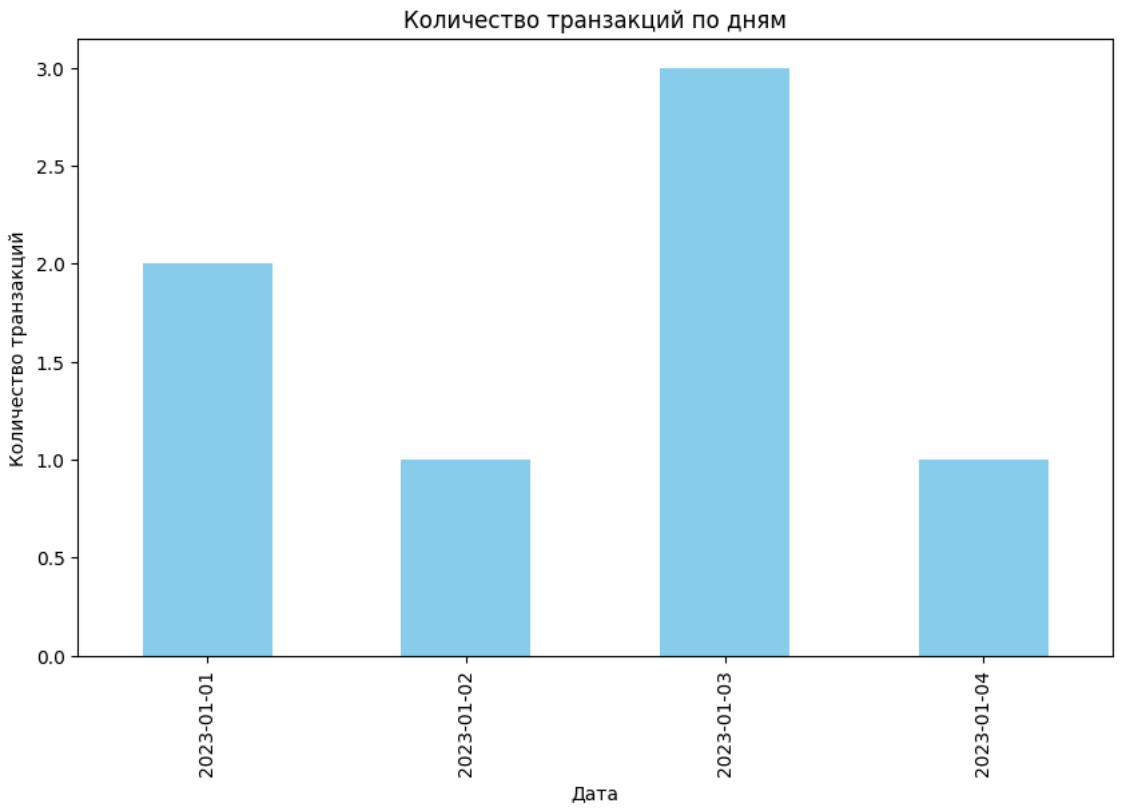
```
import pandas as pd
import matplotlib.pyplot as plt

# Загрузка данных о транзакциях
data = pd.read_csv('transactions.csv')

# Анализ частоты транзакций
daily_transactions = data.groupby('date')['amount'].count()

# Визуализация частоты транзакций
plt.figure(figsize=(10, 6))
daily_transactions.plot(kind='bar', color='skyblue')
plt.title('Количество транзакций по дням')
plt.xlabel('Дата')
```

```
plt.ylabel('Количество транзакций')
plt.show()
```



Выводы: После анализа данных можно сделать выводы о пиковых днях и времени активности пользователей, что помогает оптимизировать работу платежной системы.

Предсказание аномальных транзакций в платежных системах с помощью машинного обучения

Описание: Создание модели для выявления аномалий в транзакциях на основе их характеристик (сумма, время, место).

Решение:

Этот код использует алгоритм Isolation Forest для обнаружения аномалий в данных о транзакциях.

Что делает код: Генерирует 1000 случайных двумерных транзакций, обучает модель Isolation Forest для выявления выбросов, помечает аномалии (10% данных) и визуализирует результаты на точечной диаграмме, где цветом выделяются нормальные и аномальные транзакции.

Цель кода: Продемонстрировать применение машинного обучения для обнаружения подозрительных операций, мошеннических транзакций или статистических выбросов в финансовых данных, что является ключевой задачей в системах фрод-мониторинга и финансовой безопасности.

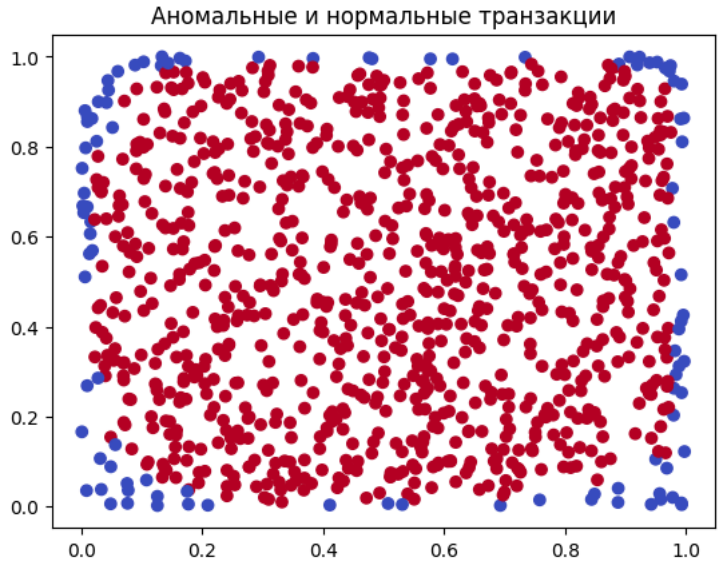
```
from sklearn.ensemble import IsolationForest
import numpy as np

# Генерация случайных данных о транзакциях
data = np.random.rand(1000, 2)

# Модель для выявления аномалий
model = IsolationForest(contamination=0.1)
model.fit(data)

# Определение аномалий
predictions = model.predict(data)

# Визуализация результатов
plt.scatter(data[:, 0], data[:, 1], c=predictions, cmap='coolwarm')
plt.title('Аномальные и нормальные транзакции')
plt.show()
```



Выводы: Использование Python для построения модели выявления аномалий помогает обнаружить подозрительные транзакции, что повышает безопасность платежных систем.

Создание распределенной сети платежных каналов на Python Описание: Имитация работы платежной системы на основе технологии платежных каналов (например, Lightning Network), позволяющей проводить транзакции без участия третьей стороны. Решение:

Этот код представляет собой упрощенную реализацию платежного канала — технологии, которая позволяет проводить мгновенные транзакции между двумя сторонами без постоянной записи каждой операции в блокчейн.

Суть работы: Класс PaymentChannel создает виртуальный канал для расчетов между двумя участниками (A и B). У каждого есть начальный баланс. Метод send_payment() выполняет перевод средств от A к B, но только если у отправителя достаточно средств. Это гарантирует, что транзакции будут безопасными и не приведут к отрицательному балансу.

Как используется: В примере создается канал, где у стороны A есть 100 условных единиц, у стороны B — 50. Затем выполняется перевод 20 единиц от A к B. После операции балансы обновляются: у A остается 80, у B становится 70.

Зачем это нужно: Такая система полезна для микротранзакций, где скорость важнее немедленного подтверждения в блокчейне. Например, для оплаты поездок в такси, покупок в играх или чаевых. Финальный расчет в блокчейне происходит только при закрытии канала, что экономит время и комиссии.

Аналоги в реальном мире: Lightning Network в Bitcoin или Raiden Network в Ethereum работают по похожему принципу, позволяя проводить тысячи транзакций в секунду без перегрузки основной сети.

```
class PaymentChannel:
    def __init__(self, balance_a, balance_b):
        self.balance_a = balance_a
        self.balance_b = balance_b

    def send_payment(self, amount):
        if self.balance_a >= amount:
            self.balance_a -= amount
            self.balance_b += amount
            return True
        else:
            return False

# Пример использования
channel = PaymentChannel(balance_a=100, balance_b=50)
channel.send_payment(20)

print(f"Остаток у стороны A: {channel.balance_a}")
print(f"Остаток у стороны B: {channel.balance_b}")
```

```
Остаток у стороны A: 80
Остаток у стороны B: 70
```

Выводы: Моделирование работы платежных каналов позволяет понять, как можно оптимизировать процесс транзакций и уменьшить комиссии за проведение платежей.

Визуализация блоков в блокчейне и их транзакций

Описание: Визуализация структуры блокчейна и транзакций внутри каждого блока с использованием Python и библиотек для визуализации графов.

Решение:

Этот код создает и визуализирует упрощенную структуру блокчейна с помощью графа.

Что делает код: Создает направленный граф, добавляет 5 блоков (от Block 1 до Block 5) и соединяет их последовательными связями, имитируя цепочку блоков в блокчейне. Каждый блок связан с последующим, что представляет хеш-связь между блоками в реальной блокчейн-системе.

Визуализация: Граф отображается с помощью spring layout, где блоки показаны в виде узлов светло-голубого цвета со своими названиями, а стрелки между ними указывают направление связи от более раннего блока к более позднему.

Назначение: Демонстрирует базовую концепцию блокчейна как цепочки связанных блоков, где каждый новый блок ссылается на предыдущий, обеспечивая неизменность и целостность всей цепочки данных.

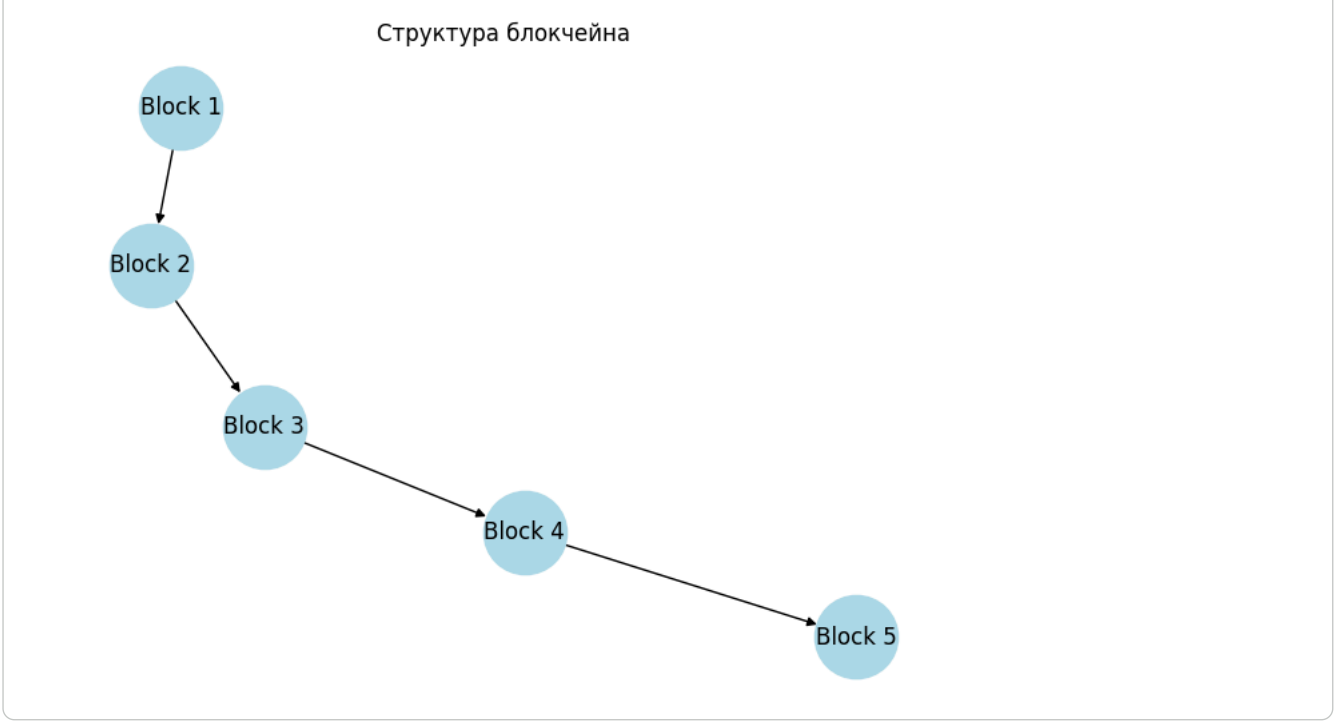
```
import networkx as nx
import matplotlib.pyplot as plt

# Создание графа блокчейна
G = nx.DiGraph()

# Добавление блоков
for i in range(1, 6):
    G.add_node(f"Block {i}", label=f"Block {i}")

# Добавление связей между блоками
for i in range(1, 5):
    G.add_edge(f"Block {i}", f"Block {i+1}")

# Визуализация блоков и связей
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='lightblue', node_size=2000)
plt.title("Структура блокчейна")
plt.show()
```



Эти примеры помогут студентам освоить применение Python для решения задач, связанных с платежными системами и блокчейн-технологиями, с акцентом на анализ данных, безопасность и моделирование.