??????

- Pip, описание модулей
- jsonschema
- Pydantic 2
- Pyinstaller
- Telegram
- Bitcoinlib
- Логгирование
- Docsvision

Pip, ???????? ???????

Альтернативные зеркала

Возможны случаи, когда это потребуется.

```
pip install <package> -i <hostname>
Например
pip install fastapi -i https://pypi.tuna.tsinghua.edu.cn/simple
```

Адрес репозитория	Тип
https://pypi.org/simple	Основной репозиторий
https://pypi.tuna.tsinghua.edu.cn/simple	Вроде очень стабильный репозиторий
https://mirrors.aliyun.com/pypi/simple	Европейский вариант

Настройка pip без необходимости указания сервера
Файл %APPDATA%\pip\pip.ini Если директории нет - создать.

```
[global]
index-url = https://pip.ya.ru/simple
trusted-host = pip.ya.ru
```

Хранение конфигурации

Configparser стандартная библиотека для чтения и записи .ini файлов. Инструкция 1

Jsonschema модуль для проверки соответствия json существующей схеме. Документация

Libsettings модуль на основе jsonschema для чтения конфигурации из json файла и проверки конфигурации на соответствие схеме. Gitverse проекта

```
import logging
from libsettings import Jsettings

logging.basicConfig(level=logging.ERROR,
                    filename='error.log',
                    format="%(levelname)s %(message)s")
```

```
mysettings = Jsettings(settingsfname='mysettings.json',
                           schemafname='myschema.json')
mysettings.load_settings()
```

jsonschema

Используется для валидации json схемы. По умолчанию дополнительно указанные ключи (не существующие в схеме, но присутствующие в документе) не проверяются.

Установка

```
pip install jsonschema
```

Базовое использование

```
from jsonschema import validate


validate(instance=json_to_check, schema=schema)
```

Исключения

jsonschema.exceptions.ValidationError - если документ не соответствует структуре

jsonschema.exceptions.SchemaError - если сама схема некорректна

Пример схемы:

```
schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "age": {"type": "number"},
    },
    "required": ["name"],
}
```

Данная схема определяет json объект с 2 свойствами: name и age. Обязательное свойство name.

Ключевые слова

Для некоторых типов используются дополнительные ключевые слова.

Ключевое слово	Описание
type	Тип. Для корня часто object. string - строка number - число object - объект array - список
$defs	Вложенный шаблон для случая, когда шаблон элемента встречается в нескольких местах.
$ref	Подстановка вложенного шаблона.
$schema	Ссылка на шаблон шаблона. При обновлении версии библиотеки будет использоваться новый шаблон шаблона, что может привести к проблемам. Желательно указывать.

Дополнительные ключевые слова для типов.

Тип array

Ключевое слово	Описание
items	Тип элементов списка. `"scores": { "type": "array", "items": {"type": "number"}, }`
minItems	Минимальное количество элементов

Тип object

Ключевое слово	Описание
required	Список обязательных ключей. `"required": ["name"]`
properties	Определяет ключи объекта и их тип. `"properties": { "name": {"type": "string"} },`

Ключевое слово	Описание
additionalProperties	Если True, то дополнительно указанные ключи приводят к исключению.

Локальные вложенные шаблоны.

Для определения используется переменная $defs, для использования - $ref.

```
schema = {
    "type": "object",
    "properties": {
        "address": {"$ref": "#/$defs/address"},
    },
    "$defs": {
        "address": {
            "type": "object",
            "properties": {
                "street": {"type": "string"},
            }
        },
    },
}
```

Pydantic 2

Описание

Библиотека валидации (проверка на соответствие типов) и трансформации (автоматическое приведение к нужным типам и форматам) данных.

Модели наследуются от класса BaseModel. Модель описывает набор полей, представляющих структуру данных и условия валидации.

Установка

```
pip install pydantic
```

Типизация:

- Простая: указание типа, например, name: str.
- Объект Field(): дополнительные параметры, например, значения по умолчанию, ограничения и т.д.

Внутри класса можно комбинировать способы типизации.

```
from pydantic import BaseModel, Field


class User(BaseModel):
    name: str
    email: str = Field(..., alias='email_address')
```

Валидация:

- Минимальная валидация: встроенные типы Python (например, str, int).
- Валидаторы: например EmailStr для проверки email-адресов. Требуется установка дополнительных зависимостей: pydantic[email] или pydantic[all].
- @field_validator — добавляет логику валидации поля. Вызывается при создании или изменении модели.

```
from pydantic import BaseModel, field_validator


class User(BaseModel):
    age: int
```

```
    @field_validator('age')
    def check_age(cls, value):
        if value < 18:
            raise ValueError('Возраст должен быть больше 18 лет')
        return value
```

- @computed_field — вычисляемое поле на основе данных в модели. Его можно использовать для автоматической генерации значений, а также для валидации.

```
from pydantic import BaseModel, computed_field

class User(BaseModel):
    name: str
    surname: str

    @computed_field
    def full_name(self) -> str:
        return f"{self.name} {self.surname}"
```

- @model_validator - валидация всей модели.

Работает со всей моделью (а не с отдельными полями), может изменять данные перед валидацией (mode='before') или после (mode='after'), полезен для комплексных проверок, где одно поле зависит от другого, может возвращать новую версию модели (если нужно модифицировать данные).

```
@model_validator(mode='before')
def validate_before(cls, data: dict):
    if 'username' not in data:
        data['username'] = "guest_" + str(data.get('id', 0))
    return data
```

```
@model_validator(mode='after')
def validate_after(self):
    if self.age < 18 and self.is_premium:
        raise ValueError("Minors cannot have premium accounts!")
    return self
```

При проверке before передается класс, при after - объект. Можно делать два валидатора: before для подстановки вычисляемых значений и after для финальной проверки

Интеграция с SQLAlchemy:

Для настройки используется параметр ConfigDict с флагом from_attributes=True.

```
from datetime import date
from pydantic import BaseModel, ConfigDict

class User(BaseModel):
    id: int
    name: str = 'John Doe'
    birthday_date: date

    config = ConfigDict(from_attributes=True)
```

Для создания модели Pydantic из объекта ORM используется метод from_orm.

```
user = User.from_orm(orm_instance)
```

Ссылки:

Основа для текста

Pyinstaller

Установка:

```
python -m pip install pyinstaller
```

Использование

```
pyinstaller [параметры] script.py
```

Параметры

Параметр	Описание
--onefile	собирает всё в один .exe файл
--windowed	скрывает консоль (если у вас GUI-приложение). Если нужна консоль, уберите этот флаг.
--icon=ваша_иконка.ico	иконка
--name "МояПрограмма"	имя программы
--dest <путь_к_директории>	директория, в которую будет собираться exe файл

Telegram

При взаимодействии с ботом нужен идентификатор.

Свой идентификатор

- В Telegram напиши боту @userinfobot
- Он ответит твоим `user_id` (число). Это и есть твой `chat_id`.

Для групп или каналов

- Добавь бота в группу/канал.
- Напиши любое сообщение.
- В браузере открой: https://api.telegram.org/bot<YOUR_BOT_TOKEN>/getUpdates
- В JSON-ответе найди `chat":{"id": ... }` — это и есть `CHAT_ID`. Для супергрупп он будет вида `-1001234567890`.

Bitcoinlib

Библиотека для работы с кошельками. Операции, связанные с кошельком:

- + Создание нового кошелька
- Экспорт данных о созданном кошельке
- Импорт существующего кошелька
- + Информация о кошельке
- Транзакция

Создание кошелька. Создается хранилище данных в ~/.bitcoinlib Затем можно проводить операции.

```
def create_wallet():
    # Создаем новый testnet кошелек
    wallet = Wallet.create(
        name='my_testnet_wallet',
        network='testnet'
    )

    print(f"Адрес: {wallet.get_key().address}")
    print(f"Приватный ключ (WIF): {wallet.get_key().wif}")
    print(f"Баланс: {wallet.balance()} satoshi")

    # Получить информацию об адресе
    print(f"Это testnet адрес? {wallet.get_key().address.startswith(('m', 'n', '2', 'tb1'))}")
```

Для получения стартовых btc в сети testnet использовал https://coinfaucet.eu/en/btc-testnet/

Информация о кошельке. Кошелек с данным названием уже установлен в системе.

```
def wallet_info():
    """Полная информация о кошельке (исправленная)"""

    wallet = Wallet('my_testnet_wallet')
    wallet.scan()  # Важно: синхронизируем с сетью

    print("=" * 60)
```

```
print(f"КОШЕЛЁК: {wallet.name}")
print(f"СЕТЬ: {wallet.network.name}")
print("=" * 60)

# Баланс
balance = wallet.balance()
print(f"\n□□БАЛАНС: {balance:,} satoshi")
print(f"         ≈ {balance / 100000000:.8f} BTC")

# UTXOs
utxos = wallet.utxos()
print(f"\n UTXOs: {len(utxos)}")

if utxos:
    utxo_total = 0
    for i, utxo in enumerate(utxos, 1):
        print(f"\n  UTXO #{i}:")
        print(f"    Транзакция: {utxo['txid'][:20]}...:{utxo['output_n']}")
        print(f"    Адрес: {utxo['address']}")
        print(f"    Сумма: {utxo['value']:,} sat")

        if 'confirmations' in utxo:
            confs = utxo['confirmations']
            status = "✓ Подтверждено" if confs > 0 else " Ожидание"
            print(f"    Статус: {status} ({confs} подтверждений)")

        utxo_total += utxo['value']

    print(f"\n  Сумма всех UTXOs: {utxo_total:,} sat")

# Транзакции
transactions = wallet.transactions()
print(f"\n ТРАНЗАКЦИИ: {len(transactions)}")

if transactions:
    for tx in transactions:
        print(f"\n  Транзакция: {tx.txid[:20]}...")
        print(f"    Дата: {tx.date}")

        if tx.confirmations:
```

```
            print(f"    Подтверждений: {tx.confirmations}")
        else:
            print(f"    Статус: Неподтверждена")

        print(f"    Комиссия: {tx.fee} sat")

        # Анализируем сумму
        our_addresses = wallet.addresslist()
        received = 0
        sent = 0

        # Выходы (получение)
        for output in tx.outputs:
            if output.address in our_addresses:
                received += output.value

        # Входы (отправка)
        for input_tx in tx.inputs:
            if input_tx.address in our_addresses:
                sent += input_tx.value

        if received > 0 and sent > 0:
            print(f"    Тип: Перевод")
            print(f"    Изменение баланса: {received - sent:,} sat")
        elif received > 0:
            print(f"    Тип: Получение")
            print(f"    Сумма: +{received:,} sat")
        elif sent > 0:
            net_sent = sent - tx.fee
            print(f"    Тип: Отправка")
            print(f"    Сумма: -{net_sent:,} sat (включая комиссию)")

# Ключи и адреса
print(f"\n КЛЮЧИ И АДРЕСА:")
keys = wallet.keys()
print(f"  Всего ключей: {len(keys)}")

used_addresses = [key.address for key in keys if key.used]
print(f"  Использованных адресов: {len(used_addresses)}")
```

```
    # Показываем первые 5 адресов
    for i, key in enumerate(keys[:5], 1):
        status = " Использован" if key.used else " Не использован"
        print(f"  {i}. {key.address} - {status} ({key.balance} sat)")

    if len(keys) > 5:
        print(f"  ... и ещё {len(keys) - 5} адресов")

    # Сетевая информация
    print(f"\n СЕТЕВАЯ ИНФОРМАЦИЯ:")
    #print(f"  Последний блок: {wallet.last_block}")
    print(f"  ID кошелька: {wallet.wallet_id}")

    return wallet
```

??????????

Встроенный модуль logging. Нужно настроить логгер и использовать его.

Настройка:

```
import logging


if __name__ == '__main__':
    logging.basicConfig(level=logging.ERROR, filename="error.log",filemode="w",
                        format="%(asctime)s %(levelname)s %(message)s")
```

Использование:

Внутри модуля, где настраивался логгер:

```
logging.error("Критическая ошибка в основном модуле")
```

В вызываемом модуле не проводим настройку, только:

```
logger = logging.getLogger(__name__)
logger.error('Wow')
```

Docsvision

Существует webapi

Узнать realm можно через ipconfig на win машине в домене, строка "DNS-суффикс подключения"

Список контроллеров домена: nslookup -type=SRV _ldap._tcp.<realm>

Для работы требуется

```
sudo apt install krb5-user
sudo apt install libkrb5-dev
```

Файл /etc/krb5.conf

```
[libdefaults]
    default_realm = DOMAIN.LOCAL
    dns_lookup_realm = true
    dns_lookup_kdc = true

[realms]
    DOMAIN.LOCAL = {
        kdc = dc01.domain.local
        admin_server = dc01.domain.local
    }
```

Для ручного получения тикета (REALM обязательно прописными!):

```
kinit username@REALM
```

Проверка полученного тикета:

```
klist
```

Модули python

b