Домашня робота №11/12

Sergiy Ponomarenko

23 лютого 2023 р.

1 Файл README.MD

https://www.edu.goit.global/ru/learn/8016550/5594293/5615318/homework

```
# Домашнє завдання №10


- Записи `Record` у `AddressBook` зберігаються як значення у словнику.
  В якості ключів використовується значення `Record.name.value`.
- `Record` зберігає об'єкт <Name> в окремому атрибуті.
- `Record` зберігає список об'єктів `Phone` в окремому атрибуті.
- `Record` реалізує методи додавання/видалення/редагування об'єктів `Phone`.
- `AddressBook` реалізує метод `add_record`, який додає <Record> у `self.data`.


## Команди
- `hello` --- чат вітається.
- `set birthday` -- встановлює дату народження контакуту у форматі  `DD.MM.YYY`, наприклад `set birthday Sergiy 12.12.1978`.
- `birthday of` -- Виводить на екран дату вказаного контакту, наприклад `birthday of Sergiy`.
- `add` --- чат додає ім'я і телефон, приклад `add Sergiy 0936564532`.
- `chage` --- чат змінює номер для відповідного контакту, приклад `change Sergiy 0936564532 0634564545`.
- `phones` --- чат виводить номери телефонів контакту, приклад `phone Sergiy`.
- `show all`--- чат показує усі контакти та їх номери, приклад `show all`
- `remove` --- чат видаляє запис з вказаним іменем, приклад `remove Sergiy`.
- `good bye`, `good`, `exit` --- чат прощається і завершує роботу і зберігає контакти у файл `contacts.json`.
- `.` --- чат перериває свою роботу без попереджень і зберігає контакти у файл `contacts.json`.
- `save` --- зберігає контакти у файл `.json`, наприклад `save contacts`.
- `load` --- завантажує книгу з контактами з файлу `.json` в чат, наприклад `load contacts`.
- `search` -- здійснює пошук за ключовою фразою, частиною номеру телефона чи дні народження, наприклад `search 123`, або `search Beth`.


```
COMMANDS = {
    "hello": hello,
    "set birthday": set_birthday,
    "birthday of": birthday,
    "add": add,
    "change": change,
    "phones of": phones,
    "show all": show_all,
    "remove": remove,
    "good bye": good_bye,
    "close": good_bye,
    "exit": good_bye,
    "save": save,
    "load": load,
    "search": search,
}
```
```

2 Реалізація

```python
import re
from collections import UserDict
import json
from datetime import datetime
from prettytable import PrettyTable


# ================================== Classes ==================================#


"""
- Записи <Record> у <AddressBook> зберігаються як значення у словнику.
  В якості ключів використовується значення <Record.name.value>.
- <Record> зберігає об'єкт <Name> в окремому атрибуті.
- <Record> зберігає список об'єктів <Phone> в окремому атрибуті.
- <Record> реалізує методи додавання/видалення/редагування об'єктів <Phone>.
- <AddressBook> реалізує метод <add_record>, який додає <Record> у <self.data>.

"""


class Field:
    """Клас є батьківським для всіх полів, у ньому реалізується логіка,
    загальна для всіх полів."""

    def __init__(self, value: str):
        self.__value = value

    @property
    def value(self):
        return self.__value

    @value.setter
    def value(self, value):
        self.__value = value

    def __repr__(self):
        return f"{self.value}"

    def __eq__(self, other):
        return self.value == other.value


class Name(Field):
    """Клас --- обов'язкове поле з ім'ям."""

    pass


class Phone(Field):
    """Клас -- необов'язкове поле з телефоном та таких один записів (Record)
    може містити кілька."""

    def __init__(self, value: str):
        super().__init__(value)
        if not re.match(r"^\d{10}$", value):
            raise ValueError("Phone number must be 10 digits")


```

```python
class Birthday(Field):
    """Клас -- необов'язкове поле з датою народження."""

    def __init__(self, value: str):
        super().__init__(value)
        if value is not None and not re.match(r"^\d{2}\.\d{2}\.\d{4}$", value):
            raise ValueError("Birthday should be in format DD.MM.YYYY")


class Record:
    """Клас відповідає за логіку додавання/видалення/редагування
    необов'язкових полів та зберігання обов'язкового поля Name."""

    records = {}

    # Забороняємо створювати кілька об'єктів з однаковиси полями Name
    def __new__(cls, name: Name, *args, **kwargs):
        if name.value in cls.records:
            return cls.records[name.value]
        return super().__new__(cls)

    def __init__(
        self,
        name: Name,
        phones: list[Phone] = None,
        birthday: Birthday = None,
    ):

        # якщо об'єк було створено, то припинити роботу конструктора
        if name.value in self.records:
            return
        # інакше запустити конструктор
        self.name = name  # Name --- атрибут ля зберігання об'єкту Name
        self.phones = phones or []
        self.birthday = birthday
        # Додаємо в словник об'єктів новий об'єкт
        self.records[name.value] = self

    def add_birthday(self, birthday: Birthday):
        """Метод додає об'єкт день народження до запису."""

        self.birthday = birthday

    def add_phone(self, phone: Phone):
        """Метод додає об'єкт телефон до запису."""

        self.phones.append(phone)

    def remove_phone(self, phone: Phone):
        """Метод видаляє об'єкт телефон із запису."""

        self.phones.remove(phone.value)

    def change_phone(self, old_phone: Phone, new_phone: Phone) -> bool:
        """Метод змінює об'єкт телефон в записі на новий."""

        phones_list_str = (phone.value for phone in self.phones)
        if old_phone.value in phones_list_str:
            idx = phones_list_str.index(old_phone.value)
            self.phones[idx] = new_phone
            return True
        return False

```

```python
    def days_to_birthday(self) -> int:
        """Метод повертає кількість днів до наступного дня народження контакту."""

        if not self.birthday:
            return None
        today = datetime.today()
        dt_birthday = datetime.strptime(self.birthday.value, "%d.%m.%Y")
        next_birthday = dt_birthday.replace(year=today.year)
        if next_birthday < today:
            next_birthday = dt_birthday.replace(year=today.year + 1)
        return (next_birthday - today).days


class AddressBook(UserDict):
    """Клас містить логіку пошуку за записами до цього класу."""

    def add_record(self, record: Record):
        """Метод додає запис до списку контактів."""

        self.data[record.name.value] = record

    def save_contacts(self, filename):
        with open(f"{filename}.json", "w") as f:
            contacts = {}
            for name, record in self.data.items():
                phones = [phone.value for phone in record.phones]
                birthday = record.birthday.value if record.birthday else None
                if birthday:
                    contacts[name] = {"phones": phones, "birthday": birthday}
                else:
                    contacts[name] = {"phones": phones}
            json.dump(contacts, f, ensure_ascii=False, indent=4)

    def load_contacts(self, filename):
        with open(f"{filename}.json", "r") as f:
            data = json.load(f)

        for name, info in data.items():
            phones = [Phone(phone) for phone in info["phones"]]
            birthday = Birthday(info.get("birthday"))
            self.data[name] = Record(
                Name(name), phones=phones, birthday=birthday
            )

    def search(self, search_string):
        results = AddressBook()
        for key in self.data:
            record = self.data[key]
            if (
                search_string in record.name.value
                or (
                    record.birthday.value is not None
                    and search_string in record.birthday.value
                )
                or any(search_string in phone.value for phone in record.phones)
            ):
                results.add_record(record)
        return results

    def iterator(self, page_size=10):
        table = PrettyTable()
        table.field_names = ["Name", "Birthday", "Phones"]
        table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})
```

```
        for i, key in enumerate(self.data, 1):
            record = self.data[key]
            name = record.name.value
            birthday = (
                        record.birthday.value
                        if record.birthday is not None and record.birthday.value
                        else "-"
                    )
            phones = (
                ", ".join(p.value for p in record.phones)
                if record.phones
                else "-"
            )
            table.add_row([name, birthday, phones])
            if i % page_size == 0 or i == len(self.data):
                yield table
                if i != len(self.data):
                    input("Press Enter to continue...")
                table.clear_rows()
        if table._rows:
            yield table


# ================================== Decorator ==================================#


def input_error(func):
    def wrapper(*func_args, **func_kwargs):
        try:
            return func(*func_args, **func_kwargs)
        except KeyError:
            return "Give me a name, please"
        except ValueError as error:
            if "phone" in str(error):
                return "Phone number must be 10 digits"
            elif "Birthday" in str(error):
                return "Birthday should be in format DD.MM.YYYY"
            else:
                return str(error)
        except TypeError:
            return "The contact has no date of birth"
        except FileNotFoundError:
            return "File not found"

    return wrapper


# ================================== handlers ==================================#


def hello(*args):
    return "How can I help you?"


def good_bye(*args):
    contacts.save_contacts("contacts")
    return "Good bye!"


def undefined(*args):
    return "What do you mean?"
```

```python
def show_all(*args):
    """Функция-handler показує книгу контактів."""
    return f"Address book contain {len(contacts)} contacts"


@input_error
def save(*args):
    contacts.save_contacts(args[0])
    return f"File {args[0]} saved"


@input_error
def load(*args):
    contacts.load_contacts(args[0])
    return f"File {args[0]} loaded"


@input_error
def set_birthday(*args):
    """Функція-handler додає день народження до контакту."""

    if not args[0]:
        raise KeyError

    if not args[1]:
        raise ValueError("Birthday should be in format DD.MM.YYYY")

    name = Name(args[0])
    birthday = Birthday(args[1])
    record = Record(name)
    record.add_birthday(birthday)
    contacts.add_record(record)

    return f"I added a birthday {args[1]} to contact {args[0]}"


@input_error
def add(*args):
    """Функція-handler додає телефон до контакту."""

    if not args[0]:
        raise KeyError

    if not args[1]:
        raise ValueError("The phone number must be 10 digits")

    name, phone = Name(args[0]), Phone(args[1])
    record = Record(name)
    record.add_phone(phone)
    contacts.add_record(record)

    return f"I added a phone {args[1]} to contact {args[0]}"


# @input_error
def phones(*args):
    """Функція-handler показує телефонні номери відповідного контакту."""

    table = PrettyTable()
    table.field_names = ["Name", "Phones"]
    table.min_width.update({"Name": 20, "Phones": 40})

    if not args[0]:
```

```
        raise KeyError

    name = Name(args[0])
    phones = Record(name).phones
    phones = ", ".join(phone.value for phone in phones)
    table.add_row([name.value, phones])

    return table


@input_error
def birthday(*args):
    """Функція-handler показує день народження та кількість днів до наступного."""

    table = PrettyTable()
    table.field_names = ["Name", "Birthday", "Days to next Birthday"]
    table.min_width.update(
        {"Name": 20, "Birthday": 12, "Days to next Birthday": 5}
    )

    if not args[0]:
        raise KeyError

    name = Name(args[0])

    birthday = Record(name).birthday

    if birthday:

        days_to_next_birthday = Record(name).days_to_birthday()

        table.add_row([name.value, birthday.value, days_to_next_birthday])

        return table

    return "No such contach founded"


def search(*args):
    return "Here are the found contacts"


@input_error
def remove(*args):
    """Функція-handler видаляє запис з книги."""

    if not args[0]:
        raise KeyError

    name = Name(args[0])

    del contacts[name.value]

    return f"Contact {name.value} was removed"


@input_error
def change(*args):
    """Функція-handler змінює телефон контакту."""

    if not args[0]:
        raise KeyError

```

```
    if not args[1]:
        raise ValueError("Old phone number is required")

    if not args[2]:
        raise ValueError("New phone number is required")

    name = Name(args[0])
    old_phone = Phone(args[1])
    new_phone = Phone(args[2])

    if name.value not in contacts:
        return f"Contact {name.value} not found"

    contact_list = contacts[name.value].phones
    for number in contact_list:
        if number == old_phone:
            idx = contact_list.index(number)
            contact_list[idx] = new_phone
            break
        return f"Phone {old_phone.value} not found for {name.value}"

    return f"Contact {name.value} with phone number {old_phone.value} was updated with new
     ↪  phone number {new_phone.value}"


# ================================ handler loader ==============================#

COMMANDS = {
    "hello": hello,
    "set birthday": set_birthday,
    "birthday of": birthday,
    "add": add,
    "change": change,
    "phones of": phones,
    "show all": show_all,
    "remove": remove,
    "good bye": good_bye,
    "close": good_bye,
    "exit": good_bye,
    "save": save,
    "load": load,
    "search": search,
}


def get_handler(*args):
    """Функція викликає відповідний handler."""
    return COMMANDS.get(args[0], undefined)


# ================================ main function ==============================#


def main():

    table = PrettyTable()
    table.field_names = ["Name", "Birthday", "Phones"]
    table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})

    command_pattern = "|".join(COMMANDS.keys())
    pattern = re.compile(
        r"\b(\.|" + command_pattern + r")\b"
        r"(?:\s+([a-zA-Z0-9\.]+))?"
```

```
        r"(?:\s+(\d{10}|\d{1,2}\.\d{1,2}\.\d{4}(?:\.\d{2})?))?"
        r"(?:\s+(\d{10})?)?",
        re.IGNORECASE,
    )

    while True:

        # waiting for nonempty input
        while True:
            inp = input(">>> ").strip()
            if inp == "":
                continue
            break

        text = pattern.search(inp)

        params = (
            tuple(
                map(
                    # Made a commands to be a uppercase
                    lambda x: x.lower() if text.groups().index(x) == 0 else x,
                    text.groups(),
                )
            )
            if text
            else (None, 0, 0)
        )
        handler = get_handler(*params)
        response = handler(*params[1:])
        if inp.strip() == ".":
            contacts.save_contacts("contacts")
            return
        if params[0] in ("show all", "search"):
            if params[0] == "show all":
                entry = contacts
            elif params[0] == "search":
                entry = contacts.search(params[1])
            for tab in entry.iterator():
                print(tab)

        print(response)
        if response == "Good bye!":
            return


contacts = AddressBook()  # Global variable for storing contacts


# ================================ main programm ============================ #

if __name__ == "__main__":

    main()
```