Домашня робота №11/12

Sergiy Ponomarenko

17 березня 2023 р.

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

1 Вигляд роботи програми

```
>>> hello
How can I help you?
>>> search S
+----------------------+-------------+----------------------------------------+
|         Name         |   Birthday  |                 Phones                 |
+----------------------+-------------+----------------------------------------+
|        Sergiy        |  12.02.1569 |               2341241323               |
|        Skirt         |  02.02.1991 |               9431287135               |
|        Snack         |      -      |               3897016452               |
|        Spade         |      -      |               8249367105               |
|        Sable         |  19.07.2008 |               7201634985               |
+----------------------+-------------+----------------------------------------+
Here are the found contacts
>>> show all
+----------------------+-------------+----------------------------------------+
|         Name         |   Birthday  |                 Phones                 |
+----------------------+-------------+----------------------------------------+
|        Roast         |  01.01.1990 |               7219542064               |
|        Skirt         |  02.02.1991 |               9431287135               |
|        Chord         |  03.03.1992 |               5871693280               |
|        Tasty         |  04.04.1993 |               3798124615               |
|        Gloom         |  05.05.1994 |               8024793119               |
|        Fluke         |  06.06.1995 |               2146518723               |
|        Tease         |      -      |               2159834871               |
|        Bumpy         |      -      |               5019478326               |
|        Brisk         |      -      |               3576012389               |
|        Plush         |      -      |               6782934015               |
+----------------------+-------------+----------------------------------------+
Press <Enter> to continue...
+----------------------+-------------+----------------------------------------+
|         Name         |   Birthday  |                 Phones                 |
+----------------------+-------------+----------------------------------------+
|        Haste         |      -      |               8304695127               |
|        Snack         |      -      |               3897016452               |
|        Blunt         |      -      |               1629405837               |
|        Spade         |      -      |               8249367105               |
|        Tonic         |      -      |               5983721046               |
|        Chive         |  16.04.2005 |               2708193541               |
|        Flair         |  17.05.2006 |               6859247310               |
|        Quell         |  18.06.2007 |               4681739250               |
|        Sable         |  19.07.2008 |               7201634985               |
|        Vigor         |  20.08.2009 |               8352406179               |
+----------------------+-------------+----------------------------------------+
Address book contain 30 contacts
>>>
```

2 Файл README.MD

```
# Домашнє завдання №12


- 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 N`--- чат показує усі контакти та їх номери, приклад `show all 10`. Необов'язковий параметр `N` - число записів, що виводяться за одну ітерацію.
- `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`.
- `save` --- зберігає контакти у файл `.json`, наприклад `save contacts`.
- `load` --- завантажує книгу з контактами з файлу в чат, наприклад `load contacts`.
- `search` -- здійснює пошук за ключовою фразою, частиною номеру телефона чи дні народження, наприклад `search 123`, або `search Beth`.
- `export` -- Експортує дані в формат `csv`, приклад `export somefile`.
- `import` -- Імпортує дані з формату `csv`, приклад `impott somefile`.


```
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,
    "export": export_to_csv,
    "import": import_from_csv,
}
```
```

3 Реалізація класів

```python
import re
from collections import UserDict
import pickle
import csv
from datetime import datetime, timedelta
import calendar

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


```

```python
"""
- 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
        self.value = value

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

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


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

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


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

    @Field.value.setter
    def value(self, value):
        if not bool(re.match(r"\d{10}", value)) or len(value) > 10:
            raise ValueError("Phone number must be 10 digits")
        self.__value = value


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

    @Field.value.setter
    def value(self, value):
        try:
            date = datetime.strptime(value, "%d.%m.%Y")
        except (TypeError, ValueError):
            raise ValueError("Invalid date format. Please use DD.MM.YYYY")
        if date > datetime.today():
            raise ValueError("Date cannot be in the future")
        self.__value = date


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

```python

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

        self.name = name  # Name --- атрибут ля зберігання об'єкту Name
        self.phones = phones or []
        self.birthday = birthday

    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)

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

        for phone in self.phones:
            if phone == old_phone:
                self.phones.remove(phone)
                self.phones.append(new_phone)
                return True
            return False

    def show_phones(self):

        phones = ", ".join(phone.value for phone in self.phones) or "-"
        return phones

    def show_birthday(self):

        birthday = getattr(self.birthday, "value", None) or "-"
        return birthday

        name = getattr(self.name, "value", False)
        return name

    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(filename, "wb") as file:
            pickle.dump(list(self.data.items()), file)

    def load_contacts(self, filename):
        with open(filename, "rb") as file:
            data = pickle.load(file)
            self.clear()
            self.update(data)

    def search(self, search_string):
        """Метод шукає записи  по ключовому слову."""

        results = AddressBook()
        for key in self.data:
            record = self.data[key]
            if (
                search_string in record.name.value
                or (
                    getattr(record.birthday, "value", False)
                    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 get_birthdays_per_week(self):
        today = datetime.today().replace(hour=0, minute=0, second=0, microsecond=0)
        week_start = today - timedelta(days=today.weekday())
        week_end = week_start + timedelta(days=6)

        birthdays = {}

        for name, record in self.items():

            if record.birthday is None:
                continue
            birthday = record.birthday.value

            user_birthday = datetime.strptime(birthday, "%d.%m.%Y").replace(
                year=today.year
            )

            if user_birthday < week_start - timedelta(days=2):
                user_birthday = user_birthday.replace(year=today.year + 1)

            if (
                week_start - timedelta(days=2)
                <= user_birthday
                <= week_end - timedelta(days=2)
            ):
```

```
                if user_birthday.weekday() < 5:
                    weekday = user_birthday.strftime("%A")
                else:
                    weekday = "Monday"
                if weekday not in birthdays:
                    birthdays[weekday] = []
                birthdays[weekday].append(name)

        return birthdays

    def export_to_csv(self, filename):
        """Експорт записів із AddressBook в CSV-файл"""

        with open(filename, "w", newline="") as csvfile:
            fieldnames = ["name", "birthday", "phones"]
            writer = csv.DictWriter(csvfile, fieldnames=fieldnames, delimiter="|")
            writer.writeheader()
            for record in self.data.values():
                writer.writerow(
                    {
                        "name": record.name.value,
                        "phones": record.show_phones(),
                        "birthday": record.show_birthday(),
                    }
                )

    def import_from_csv(self, filename):
        """Імпорт записів із CSV-файлу в AddressBook"""

        with open(filename, newline="") as csvfile:
            reader = csv.DictReader(csvfile, delimiter="|")
            for row in reader:
                name = row["name"]
                phones = [
                    Phone(phone.strip())
                    for phone in row["phones"].split(",")
                    if phone.strip() and row["phones"] != "-"
                ]
                birthday = Birthday(row["birthday"]) if row["birthday"] != "-" else None
                record = Record(name=Name(name), phones=phones, birthday=birthday)
                self.add_record(record)

    def iterator(self, n: int = 10):
        """Метод ітерується по записам і виводить їх частинами по n-штук."""

        data_items = list(self.data.items())
        for i in range(0, len(data_items), n):
            data_slice = dict(data_items[i : i + n])
            yield data_slice
            if i + n < len(data_items):
                yield "continue"
```

4 Реалізація основної програми

```
import re
from pathlib import Path
from prettytable import PrettyTable
```

```python
from botmodule import Name, Phone, Birthday, Record, AddressBook

# ============================== Tables decoration ==============================#


def build_table(data):
    """Функция строит PrettyTable для заданного списка записей."""
    table = PrettyTable()
    table.field_names = ["Name", "Birthday", "Phones"]
    table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})
    data = AddressBook(data)
    for key in data:
        record = data[key]
        name = record.name.value
        birthday = record.show_birthday()
        phones = record.show_phones()
        table.add_row([name, birthday, phones])
    return table


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


def input_error(func):
    def wrapper(*func_args, **func_kwargs):
        try:
            return func(*func_args, **func_kwargs)
        except KeyError as error:
            return "\033[1;31m{}\033[0m".format(str(error).strip("'"))
        except ValueError as error:
            return f"\033[1;31m{str(error)}\033[0m"
        except TypeError as error:
            return f"\033[1;31m{str(error)}\033[0m"
        except FileNotFoundError:
            return "\033[1;31mFile not found\033[0m"

    return wrapper


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


def hello(*args):
    return "\033[32mHow can I help you?\033[0m"


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


def undefined(*args):
    return "\033[32mWhat do you mean?\033[0m"


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


@input_error
def load(*args):
```

```
    file_path = Path(args[0])
    if file_path.exists():
        contacts.load_contacts(args[0])
        return f"File {args[0]} loaded"
    else:
        raise FileNotFoundError


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

    if not args[0] or args[0].isdigit():
        raise KeyError("Give me a name, please")
    if not args[1]:
        raise ValueError("Give me a date, please")

    name, birthday = Name(args[0]), Birthday(args[1])

    if name.value in contacts.data:
        record = contacts.data[name.value]
    else:
        record = Record(name)
        contacts.add_record(record)
    record.add_birthday(birthday)

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


@input_error
def add(*args):
    """Добавляет телефонный номер в контакт по имени."""

    if not args[0]:
        raise KeyError("Give me a name, please")

    if not args[1]:
        raise ValueError("Give me a phone, please")

    name, phone = Name(args[0]), Phone(args[1])

    if name.value in contacts.data:
        record = contacts.data[name.value]
    else:
        record = Record(name)
        contacts.add_record(record)
    record.add_phone(phone)

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


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

    if not args[0]:
        raise KeyError("Give me a name, please")

    if args[0] not in contacts.keys():
        raise ValueError("Contact does not in AddressBook")

    table = PrettyTable()
    table.field_names = ["Name", "Phones"]
```

```
    table.min_width.update({"Name": 20, "Phones": 55})

    phones = contacts.get(args[0]).show_phones() or "-"
    table.add_row([args[0], phones])

    return f"\033[0m{table}"


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

    if not args[0]:
        raise KeyError("Give me a name, please")

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

    if args[0] not in contacts.keys():
        raise ValueError("Contact does not in AddressBook")

    days_to_next_birthday = contacts.data[args[0]].days_to_birthday() or "-"
    birthday = contacts.get(args[0]).show_birthday() or "-"

    table.add_row([args[0], birthday, days_to_next_birthday])

    return f"\033[0m{table}"


@input_error
def search(*args):

    if not args[0]:
        raise KeyError("Give me a some string, please")

    results = contacts.search(args[0])

    if results:
        return f"\033[0m{build_table(results)}"
    return "By your request found nothing"


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

    if not args[0]:
        raise KeyError("Give me a name, please")

    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("Give me a name, please")
```

```
193
194     if not args[1]:
195         raise ValueError("Old phone number is required")
196
197     if not args[2]:
198         raise ValueError("New phone number is required")
199
200     name = Name(args[0])
201     old_phone = Phone(args[1])
202     new_phone = Phone(args[2])
203
204     if name.value not in contacts:
205         return f"Contact {name.value} not found"
206
207     contact_list = contacts[name.value].phones
208     for number in contact_list:
209         if number == old_phone:
210             idx = contact_list.index(number)
211             contact_list[idx] = new_phone
212             break
213         return f"Phone {old_phone.value} not found for {name.value}"
214
215     return f"Contact {name.value} with phone number {old_phone.value} was updated with new
      ↪  phone number {new_phone.value}"
216
217
218 @input_error
219 def export_to_csv(*args):
220     if not args[0]:
221         raise TypeError("Set file name, please")
222     contacts.export_to_csv(args[0])
223     return f"File {args[0]} exported to csv"
224
225
226 @input_error
227 def import_from_csv(*args):
228     contacts.import_from_csv(args[0])
229     return f"File {args[0]} imported from csv"
230
231
232 def show_all(*args):
233     """Функция-handler показує книгу контактів."""
234
235     number_of_entries = (
236         int(args[0])
237         if args[0] is not None and isinstance(args[0], str) and args[0].isdigit()
238         else 100
239     )
240     for tab in contacts.iterator(number_of_entries):
241         if tab == "continue":
242             input("\033[1;32mPress <Enter> to continue...\033[0m")
243         else:
244             print(build_table(tab))
245
246     return f"Address book contain {len(contacts)} contacts"
247
248
249 def help_commands(*args):
250     """Функція показує перелік всіх команд."""
251
252     table = PrettyTable()
253     table.field_names = ["Command"]
254     table.min_width.update({"Command": 20})
```

```
255
256     for command in COMMANDS:
257         table.add_row([command])
258
259     return f"\nPlease type folowed commands:\n\033[0m{table}"
260
261
262 def week_birthdays(*args):
263
264     table = PrettyTable()
265     table.field_names = ["Weekday", "Peoples"]
266     table.min_width.update({"Weekday": 20, "Peoples": 55})
267     birthdays = contacts.get_birthdays_per_week()
268
269     for weekday, names in birthdays.items():
270         table.add_row([weekday, ", ".join(names)])
271
272     return f"\033[0m{table}"
273
274
275 # ================================ handler loader ============================#
276
277 COMMANDS = {
278     "help": help_commands,
279     "hello": hello,
280     "set birthday": set_birthday,
281     "birthday of": birthday,
282     "add": add,
283     "change": change,
284     "phones of": phones,
285     "show all": show_all,
286     "remove": remove,
287     "good bye": good_bye,
288     "close": good_bye,
289     "exit": good_bye,
290     "save": save,
291     "load": load,
292     "search": search,
293     "export": export_to_csv,
294     "import": import_from_csv,
295     "near birthdays": week_birthdays,
296 }
297
298 command_pattern = "|".join(COMMANDS.keys())
299 pattern = re.compile(
300     r"\b(\.|" + command_pattern + r")\b"
301     r"(?:\s+([a-zA-Z0-9\.]+))?"
302     r"(?:\s+(\d{1,2}\.\d{1,2}\.\d{4}|\d{1,})?)?"
303     r"(?:\s+(\d+)?)?",
304     re.IGNORECASE,
305 )
306
307
308 def get_handler(*args):
309     """Функція викликає відповідний handler."""
310     return COMMANDS.get(args[0], undefined)
311
312
313 contacts = AddressBook()  # Global variable for storing contacts
314
315
316 def wait_for_input(prompt):
317     while True:
```

```
        inp = input(prompt).strip()
        if inp == "":
            continue
        break
    return inp


def parse_command(command):
    text = pattern.search(command)

    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)
    )

    return params


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


def main():
    print("\033[1;32mHello, I'm an assistant. Type help if you need.\033[0m")
    load("contacts")
    while True:
        command = wait_for_input(">>> ")

        if command.strip() == ".":
            contacts.save_contacts("contacts")
            return
        params = parse_command(command)
        handler = get_handler(*params)
        response = handler(*params[1:])
        print(f"\033[1;32m{response}\033[0m")
        if response == "Good bye!":
            return


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

if __name__ == "__main__":

    main()
```