Домашня робота №10

Sergiy ponomarenko

23 лютого 2023 р.

https://www.edu.goit.global/ru/learn/8016550/5594293/5615313/homework

```python
import re
from collections import UserDict
import json
from datetime import datetime, date
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


```

```python
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")


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)
```

```
107
108     def remove_phone(self, phone: Phone):
109         """Метод видаляє об'єкт телефон із запису."""
110
111         self.phones.remove(phone.value)
112
113     def change_phone(self, old_phone: Phone, new_phone: Phone) -> bool:
114         """Метод змінює об'єкт телефон в записі на новий."""
115
116         phones_list_str = (phone.value for phone in self.phones)
117         if old_phone.value in phones_list_str:
118             idx = phones_list_str.index(old_phone.value)
119             self.phones[idx] = new_phone
120             return True
121         return False
122
123     def days_to_birthday(self) -> int:
124         """Метод повертає кількість днів до наступного дня народження контакту."""
125
126         if not self.birthday:
127             return None
128         today = datetime.today()
129         dt_birthday = datetime.strptime(self.birthday.value, "%d.%m.%Y")
130         next_birthday = dt_birthday.replace(year=today.year)
131         if next_birthday < today:
132             next_birthday = dt_birthday.replace(year=today.year + 1)
133         return (next_birthday - today).days
134
135
136 class AddressBook(UserDict):
137     """Клас містить логіку пошуку за записами до цього класу."""
138
139     def add_record(self, record: Record):
140         """Метод додає запис до списку контактів."""
141
142         self.data[record.name.value] = record
143
144     def save_contacts(self, filename):
145         with open(f"{filename}.json", "w") as f:
146             contacts = {}
147             for name, record in self.data.items():
148                 phones = [phone.value for phone in record.phones]
149                 birthday = record.birthday.value if record.birthday else None
150                 if birthday:
151                     contacts[name] = {"phones": phones, "birthday": birthday}
152                 else:
153                     contacts[name] = {"phones": phones}
154             json.dump(contacts, f, ensure_ascii=False, indent=4)
155
156     def load_contacts(self, filename):
157         with open(f"{filename}.json", "r") as f:
158             data = json.load(f)
159
160         for name, info in data.items():
161             phones = [Phone(phone) for phone in info["phones"]]
162             birthday = Birthday(info.get("birthday"))
163             self.data[name] = Record(
164                 Name(name), phones=phones, birthday=birthday
165             )
166
167     def search(self, search_string):
168         results = AddressBook()
169         for page in self:
```

```
            for entry in page:
                if (
                    search_string in entry.name.value
                    or (
                        entry.birthday.value is not None
                        and search_string in entry.birthday.value
                    )
                    or any(
                        search_string in phone.value for phone in entry.phones
                    )
                ):
                    results.add_record(entry)
        return results

    def __iter__(self):
        self.index = 0
        self.page_size = 10  # количество записей на странице
        return self

    def __next__(self):
        if self.index >= len(self.data):
            raise StopIteration

        page_entries = []
        for key, value in list(self.data.items())[
            self.index : self.index + self.page_size
        ]:
            page_entries.append(value)
        self.index += self.page_size

        return page_entries


# ================================ 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?"

```

```python

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


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


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 = Name(args[0])
    phone = 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]}"
```

```python


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

    table = PrettyTable()
    table.field_names = ["Name", "Phones"]
    table.min_width["Name"] = 20
    table.min_width["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", "Phones", "Days to next Birthday"]
    table.min_width["Name"] = 20
    table.min_width["Birthday"] = 40
    table.min_width["Days to next Birthday"] = 5

    if not args[0]:
        raise KeyError

    name = Name(args[0])

    birthday = Record(name).birthday

    days_to_next_birthday = Record(name).days_to_birthday()

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

    return table


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]
```

```
359
360     return f"Contact {name.value} was removed"
361
362
363 @input_error
364 def change(*args):
365     """Функція-handler змінює телефон контакту."""
366
367     if not args[0]:
368         raise KeyError
369
370     if not args[1]:
371         raise ValueError("Old phone number is required")
372
373     if not args[2]:
374         raise ValueError("New phone number is required")
375
376     name = Name(args[0])
377     old_phone = Phone(args[1])
378     new_phone = Phone(args[2])
379
380     if name.value not in contacts:
381         return f"Contact {name.value} not found"
382
383     contact_list = contacts[name.value].phones
384     for number in contact_list:
385         if number == old_phone:
386             idx = contact_list.index(number)
387             contact_list[idx] = new_phone
388             break
389         return f"Phone {old_phone.value} not found for {name.value}"
390
391     return f"Contact {name.value} with phone number {old_phone.value} was updated with new
      ↪  phone number {new_phone.value}"
392
393
394 # ================================ handler loader ==============================#
395
396 COMMANDS = {
397     "hello": hello,
398     "set birthday": set_birthday,
399     "birthday of": birthday,
400     "add": add,
401     "change": change,
402     "phones": phones,
403     "show all": show_all,
404     "remove": remove,
405     "good bye": good_bye,
406     "close": good_bye,
407     "exit": good_bye,
408     "save": save,
409     "load": load,
410     "search": search,
411 }
412
413
414 def get_handler(*args):
415     """Функція викликає відповідний handler."""
416     return COMMANDS.get(args[0], undefined)
417
418
419 # ================================ main function ==============================#
420
```

```
421
422 def main():
423
424     table = PrettyTable()
425     table.field_names = ["Name", "Birthday", "Phones"]
426     table.min_width["Name"] = 20
427     table.min_width["Birthday"] = 12
428     table.min_width["Phones"] = 40
429
430     command_pattern = "|".join(COMMANDS.keys())
431     pattern = re.compile(
432         r"\b(\.|" + command_pattern + r")\b"
433         r"(?:\s+([a-zA-Z0-9\.]+))?"
434         r"(?:\s+(\d{10}|\d{1,2}\.\d{1,2}\.\d{4}(?:\.\d{2})?))?"
435         r"(?:\s+(\d{10})?)?",
436         re.IGNORECASE,
437     )
438
439     while True:
440
441         # waiting for nonempty input
442         while True:
443             inp = input(">>> ").strip()
444             if inp == "":
445                 continue
446             break
447
448         text = pattern.search(inp)
449
450         params = (
451             tuple(
452                 map(
453                     # Made a commands to be a uppercase
454                     lambda x: x.lower() if text.groups().index(x) == 0 else x,
455                     text.groups(),
456                 )
457             )
458             if text
459             else (None, 0, 0)
460         )
461         handler = get_handler(*params)
462         response = handler(*params[1:])
463         if inp.strip() == ".":
464             contacts.save_contacts("contacts")
465             return
466         if params[0] in ("show all", "search"):
467             if params[0] == "show all":
468                 entry = contacts
469             elif params[0] == "search":
470                 entry = contacts.search(params[1])
471             num_pages = len(list(entry))
472             for i, page in enumerate(entry):
473                 for entry in page:
474                     name = entry.name.value
475                     birthday = (
476                         entry.birthday.value
477                         if entry.birthday is not None and entry.birthday.value
478                         else "-"
479                     )
480                     phones = (
481                         ", ".join([phone.value for phone in entry.phones])
482                         if entry.phones
483                         else "-"
```

```
                    )
                    table.add_row([name, birthday, phones])
                print(table)
                if i < num_pages - 1:
                    input("Press enter to continue...")
                table.clear_rows()
        print(response)
        if response == "Good bye!":
            return


contacts = AddressBook()  # Global variable for storing contacts

# ================================ Для теста ================================ #

# rec1 = Record(Name("UserA"), [Phone("1111111111"), Phone("2222222222")])
# rec2 = Record(Name("UserB"), [Phone("3333333333"), Phone("5555555555")])
# rec3 = Record(Name("UserC"))
# rec4 = Record(Name("UserD"), [Phone("1212121212"), Phone("3434343434")])
# contacts.add_record(rec1)
# contacts.add_record(rec2)
# contacts.add_record(rec3)

# ========================================================================== #


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

if __name__ == "__main__":

    main()
```