Домашня робота №10

Sergiy ponomarenko

20 лютого 2023 р.

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

```python
import re
from collections import UserDict

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

    # def __repr__(self):
    #     return f"{self.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."""

    pass


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: Phone):
        # якщо об'єк було створено, то припинити роботу конструктора
        if name.value in self.records:
            return
        self.name = name  # Name
        self.phones = []  # list[str]
        self.phones.extend([phone.value for phone in phones])
        # Додаємо в словник об'єктів новий об'єкт
        self.records[name.value] = self

    def add_phone(self, phone: Phone):
        self.phones.append(phone.value)

    def remove_phone(self, phone: Phone):
        self.phones.remove(phone.value)

    def change_phone(self, old_phone: Phone, new_phone: Phone) -> bool:
        if old_phone.value in self.phones:
            idx = self.phones.index(old_phone.value)
            self.phones[idx] = new_phone.value
            return True
        return False


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

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

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

    def __str__(self):
        items = [
            f"{k}: {', '.join(str(i) for i in v)}" for k, v in self.items()
        ]
        if items:
            return "\n".join(items)
        return "Empty"


# ================================ 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:
            return "Give me a phone, please"

```

```
107     return wrapper
108
109
110 # ================================== handlers ================================#
111
112
113 def hello(*args):
114     return "How can I help you?"
115
116
117 def good_bye(*args):
118     return "Good bye!"
119
120
121 def undefined(*args):
122     return "What do you mean?"
123
124
125 def show_all(*args):
126     return contacts
127
128
129 @input_error
130 def add_contact(*args):
131
132     if not args[0]:
133         raise KeyError
134
135     if not args[1]:
136         raise ValueError
137
138     name = Name(args[0])
139     phone = Phone(args[1])
140     record = Record(name)
141     record.add_phone(phone)
142     contacts.add_record(record)
143
144     return f"I added a nomber {args[1]} to contact {args[0]}"
145
146
147 @input_error
148 def get_phones(*args):
149
150     if not args[0]:
151         raise KeyError
152
153     name = Name(args[0])
154
155     phones = Record(name).phones
156
157     if phones:
158         return f"{name.value}: " + ", ".join(
159             f"{element}" for element in phones
160         )
161
162
163 @input_error
164 def remove_contact(*args):
165
166     if not args[0]:
167         raise KeyError
168
169     name = Name(args[0])
```

```
170
171     del contacts[name.value]
172
173     return f"Contact {name.value} was removed"
174
175
176 @input_error
177 def change_contact(*args):
178
179     if not args[0]:
180         raise KeyError
181
182     if not args[1]:
183         raise ValueError("Old phone number is required")
184
185     if not args[2]:
186         raise ValueError("New phone number is required")
187
188     name = Name(args[0])
189     old_phone = Phone(args[1])
190     new_phone = Phone(args[2])
191
192     if name.value not in contacts:
193         return f"Contact {name.value} not found"
194
195     contact_list = contacts[name.value]
196     for number in contact_list:
197         if number == old_phone.value:
198             idx = contact_list.index(number)
199             contact_list[idx] = new_phone.value
200             break
201         return f"Phone {old_phone.value} not found for {name.value}"
202
203     return f"Contact {name.value} with phone number {old_phone.value} was updated with new
      ↪  phone number {new_phone.value}"
204
205
206 # ================================ handler loader ==============================#
207
208
209 def get_handler(*args):
210     COMMANDS = {
211         "hello": hello,
212         "add": add_contact,
213         "change": change_contact,
214         "phones": get_phones,
215         "show all": show_all,
216         "remove": remove_contact,
217         "good bye": good_bye,
218         "close": good_bye,
219         "exit": good_bye,
220     }
221     return COMMANDS.get(args[0], undefined)
222
223
224 # ================================ main function ==============================#
225
226
227 def main():
228
229     pattern = re.compile(
230         r"\b(\.|hello|add|remove|clear number|change|phones|show all|good
          ↪  bye|close|exit)\b"
```

```
        r"(?:\s+([a-zA-Z]+))?"
        r"(?:\s+(\d{10}))?"
        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() == ".":
            return
        print(response)
        if response == "Good bye!":
            return


contacts = AddressBook()  # Global variable for storing contacts


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

if __name__ == "__main__":

    main()
```