

Домашня робота №11/12

Sergiy Ponomarenko

26 лютого 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

```
1 # Домашнє завдання №10
2
3
4 - Записи `Record` у `AddressBook` зберігаються як значення у словнику.
5 В якості ключів використовується значення `Record.name.value`.
6 - `Record` зберігає об'єкт <Name> в окремому атрибуті.
7 - `Record` зберігає список об'єктів `Phone` в окремому атрибуті.
8 - `Record` реалізує методи додавання/видалення/редагування об'єктів `Phone`.
9 - `AddressBook` реалізує метод `add_record`, який додає <Record> у `self.data`.
10
11
12 ## Команди
13 - `hello` --- чат вітається.
14 - `set birthday` -- встановлює дату народження контакту у форматі `DD.MM.YYY`, наприклад `set birthday Sergiy`
   ↳ 12.12.1978`.
15 - `birthday of` -- Виводить на екран дату вказаного контакту, наприклад `birthday of Sergiy`.
16 - `add` --- чат додає ім'я і телефон, приклад `add Sergiy 0936564532`.
17 - `chage` --- чат змінює номер для відповідного контакту, приклад `change Sergiy 0936564532 0634564545`.
18 - `phones` --- чат виводить номери телефонів контакту, приклад `phone Sergiy`.
19 - `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.
20 - `remove` --- чат видаляє запис з вказаним іменем, приклад `remove Sergiy`.
21 - `good bye`, `good`, `exit` --- чат прощається і завершує роботу і зберігає контакти у файл `contacts.json`.
22 - `.` --- чат перериває свою роботу без попереджень і зберігає контакти у файл `contacts.json`.
23 - `save` --- зберігає контакти у файл `.json`, наприклад `save contacts`.
24 - `load` --- завантажує книгу з контактами з файлу `.json` в чат, наприклад `load contacts`.
25 - `search` -- здійснює пошук за ключовою фразою, частиною номеру телефона чи дні народження, наприклад `search`
   ↳ 123`, або `search Beth`.
26
27
28 ```
29 COMMANDS = {
30     "hello": hello,
31     "set birthday": set_birthday,
32     "birthday of": birthday,
33     "add": add,
34     "change": change,
35     "phones of": phones,
36     "show all": show_all,
37     "remove": remove,
38     "good bye": good_bye,
39     "close": good_bye,
40     "exit": good_bye,
41     "save": save,
42     "load": load,
43     "search": search,
44 }
45 ```
```

3 Реалізація класів

```
1 import re
2 from collections import UserDict
3
4 import pickle
5 from datetime import datetime
6
7
8 # ===== Classes =====#
9
10
11 """
12 - Записи <Record> у <AddressBook> зберігаються як значення у словнику.
13 В якості ключів використовується значення <Record.name.value>.
```

```

14 - <Record> зберігає об'єкт <Name> в окремому атрибуті.
15 - <Record> зберігає список об'єктів <Phone> в окремому атрибуті.
16 - <Record> реалізує методи додавання/видалення/редагування об'єктів <Phone>.
17 - <AddressBook> реалізує метод <add_record>, який додає <Record> у <self.data>.
18
19 """
20
21
22 class Field:
23     """Клас є батьківським для всіх полів, у ньому реалізується логіка,
24     загальна для всіх полів."""
25
26     def __init__(self, value: str):
27         self.__value = value
28         self.value = value
29
30     @property
31     def value(self):
32         return self.__value
33
34     def __eq__(self, other):
35         return self.value == other.value
36
37
38 class Name(Field):
39     """Клас --- обов'язкове поле з ім'ям."""
40
41     @Field.value.setter
42     def value(self, value):
43         self.__value = value
44
45
46 class Phone(Field):
47     """Клас -- необов'язкове поле з телефоном та таких один записів (Record)
48     може містити кілька."""
49
50     @Field.value.setter
51     def value(self, value):
52         if not bool(re.match(r"\d{10}", value)):
53             raise ValueError("\033[31mPhone number must be 10 digits\033[0m")
54         self.__value = value
55
56
57 class Birthday(Field):
58     """Клас -- необов'язкове поле з датою народження."""
59
60     @Field.value.setter
61     def value(self, value):
62         try:
63             date = datetime.strptime(value, "%d.%m.%Y")
64         except (TypeError, ValueError):
65             raise ValueError("\033[31mInvalid date format. Please use DD.MM.YYYY\033[0m")
66         if date > datetime.today():
67             raise ValueError("Date cannot be in the future")
68         self.__value = date
69
70
71 class Record:
72     """Клас відповідає за логіку додавання/видалення/редагування
73     необов'язкових полів та зберігання обов'язкового поля Name."""
74
75     def __init__(
76         self,

```

```

77     name: Name,
78     phones: list[Phone] = None,
79     birthday: Birthday = None,
80 ):
81
82     self.name = name # Name --- атрибут для зберігання об'єкту Name
83     self.phones = phones or []
84     self.birthday = birthday
85
86     def add_birthday(self, birthday: Birthday):
87         """Метод додає об'єкт день народження до запису."""
88
89         self.birthday = birthday
90
91     def add_phone(self, phone: Phone):
92         """Метод додає об'єкт телефон до запису."""
93
94         self.phones.append(phone)
95
96     def remove_phone(self, phone: Phone):
97         """Метод видаляє об'єкт телефон із запису."""
98
99         self.phones.remove(phone)
100
101     def change_phone(self, old_phone: Phone, new_phone: Phone) -> bool:
102         """Метод змінює об'єкт телефон в записі на новий."""
103
104         for phone in self.phones:
105             if phone == old_phone:
106                 self.phones.remove(phone)
107                 self.phones.append(new_phone)
108                 return True
109             return False
110
111     def show_phones(self):
112
113         phones = ", ".join(phone.value for phone in self.phones) or "-"
114         return phones
115
116     def show_birthday(self):
117
118         birthday = getattr(self.birthday, "value", None) or "-"
119         return birthday
120
121     def days_to_birthday(self) -> int:
122         """Метод повертає кількість днів до наступного дня народження контакту."""
123
124         if not self.birthday:
125             return None
126
127         today = datetime.today()
128         dt_birthday = datetime.strptime(self.birthday.value, "%d.%m.%Y")
129         next_birthday = dt_birthday.replace(year=today.year)
130
131         if next_birthday < today:
132             next_birthday = dt_birthday.replace(year=today.year + 1)
133
134         return (next_birthday - today).days
135
136
137 class AddressBook(UserDict):
138     """Клас містить логіку пошуку за записами до цього класу."""
139

```

```

140 def add_record(self, record: Record):
141     """Метод додає запис до списку контактів."""
142
143     self.data[record.name.value] = record
144
145 def save_contacts(self, filename):
146     with open(filename, "wb") as file:
147         pickle.dump(list(self.data.items()), file)
148
149 def load_contacts(self, filename):
150     with open(filename, "rb") as file:
151         data = pickle.load(file)
152         self.clear()
153         self.update(data)
154
155 def search(self, search_string):
156     """Метод шукає записи по ключовому слову."""
157
158     results = AddressBook()
159     for key in self.data:
160         record = self.data[key]
161         if (
162             search_string in record.name.value
163             or (
164                 getattr(record.birthday, "value", False)
165                 and search_string in record.birthday.value
166             )
167             or any(search_string in phone.value for phone in record.phones)
168         ):
169             results.add_record(record)
170     return results
171
172 def iterator(self, n: int):
173     """Метод ітерується по записам і виводить їх частинами по n-штук."""
174
175     data_items = list(self.data.items())
176     for i in range(0, len(data_items), n):
177         data_slice = dict(data_items[i: i + n])
178         yield data_slice
179         if i + n < len(data_items):
180             yield "continue"

```

4 Реалізація основної програми

```

1 import re
2 from prettytable import PrettyTable
3 from botmodule import Name, Phone, Birthday, Record, AddressBook
4
5 # ===== Tables decoration =====#
6
7
8 def build_table(data):
9     """Функция строит PrettyTable для заданного списка записей."""
10    table = PrettyTable()
11    table.field_names = ["Name", "Birthday", "Phones"]
12    table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})
13    data = AddressBook(data)

```

```

14     for key in data:
15         record = data[key]
16         name = record.name.value
17         birthday = record.show_birthday()
18         phones = record.show_phones()
19         table.add_row([name, birthday, phones])
20     return table
21
22
23 # ===== Decorator =====#
24
25
26 def input_error(func):
27     def wrapper(*func_args, **func_kwargs):
28         try:
29             return func(*func_args, **func_kwargs)
30         except KeyError as error:
31             if "name" in str(error):
32                 return "\033[31mGive me a name, please\033[0m"
33         except ValueError as error:
34             return f'\033[31m{str(error)}\033[0m'
35         except TypeError as error:
36             return f'\033[31m{str(error)}\033[0m'
37         except FileNotFoundError:
38             return "\033[31mFile not found\033[31m"
39
40     return wrapper
41
42
43 # ===== handlers =====#
44
45
46 def hello(*args):
47     return "\033[32mHow can I help you?\033[0m"
48
49
50 def good_bye(*args):
51     contacts.save_contacts("contacts")
52     return "\033[32mGood bye!\033[0m"
53
54
55 def undefined(*args):
56     return "\033[32mWhat do you mean?\033[0m"
57
58
59 def show_all(*args):
60     """Функція-handler показує книгу контактів."""
61     return f"\033[32mAddress book contain {len(contacts)} contacts\033[0m"
62
63
64 @input_error
65 def save(*args):
66     contacts.save_contacts(args[0])
67     return f"File {args[0]} saved"
68
69
70 @input_error
71 def load(*args):
72     contacts.load_contacts(args[0])
73     return f"File {args[0]} loaded"
74
75
76 @input_error

```

```

77 def set_birthday(*args):
78     """Функція-handler додає день народження до контакту."""
79
80     if not args[0]:
81         raise KeyError("Give me a name, please")
82     if not args[1]:
83         raise ValueError("Give me a date, please")
84
85     name, birthday = Name(args[0]), Birthday(args[1])
86
87     if name.value in contacts.data:
88         record = contacts.data[name.value]
89     else:
90         record = Record(name)
91         contacts.add_record(record)
92     record.add_birthday(birthday)
93
94     return f"I added a birthday {args[1]} to contact {args[0]}"
95
96
97 @input_error
98 def add(*args):
99     """Додає телефонний номер в контакт по імені."""
100
101     if not args[0]:
102         raise KeyError("Give me a name, please")
103
104     if not args[1]:
105         raise ValueError("Give me a phone, please")
106
107     name, phone = Name(args[0]), Phone(args[1])
108
109     if name.value in contacts.data:
110         record = contacts.data[name.value]
111     else:
112         record = Record(name)
113         contacts.add_record(record)
114     record.add_phone(phone)
115
116     return f"I added a phone {args[1]} to contact {args[0]}"
117
118
119 @input_error
120 def phones(*args):
121     """Функція-handler показує телефонні номери відповідного контакту."""
122
123     if not args[0]:
124         raise KeyError("Give me a name, please")
125
126     table = PrettyTable()
127     table.field_names = ["Name", "Phones"]
128     table.min_width.update({"Name": 20, "Phones": 55})
129
130     phones = contacts.show_phones(Name(args[0])) or "-"
131     table.add_row([args[0], phones])
132
133     return table
134
135
136 @input_error
137 def birthday(*args):
138     """Функція-handler показує день народження та кількість днів до наступного."""
139

```

```

140     if not args[0]:
141         raise KeyError("Give me a name, please")
142
143     table = PrettyTable()
144     table.field_names = ["Name", "Birthday", "Days to next Birthday"]
145     table.min_width.update(
146         {"Name": 20, "Birthday": 12, "Days to next Birthday": 40}
147     )
148
149     days_to_next_birthday = contacts.data[args[0]].days_to_birthday() or "-"
150     birthday = contacts.get(args[0]).show_birthday() or "-"
151
152     table.add_row([args[0], birthday, days_to_next_birthday])
153
154     return table
155
156     # return "No such contach founded"
157
158
159 def search(*args):
160     return "Here are the found contacts"
161
162
163 @input_error
164 def remove(*args):
165     """Функція-handler видаляє запис з книги."""
166
167     if not args[0]:
168         raise KeyError
169
170     name = Name(args[0])
171
172     del contacts[name.value]
173
174     return f"Contact {name.value} was removed"
175
176
177 @input_error
178 def change(*args):
179     """Функція-handler змінює телефон контакту."""
180
181     if not args[0]:
182         raise KeyError
183
184     if not args[1]:
185         raise ValueError("Old phone number is required")
186
187     if not args[2]:
188         raise ValueError("New phone number is required")
189
190     name = Name(args[0])
191     old_phone = Phone(args[1])
192     new_phone = Phone(args[2])
193
194     if name.value not in contacts:
195         return f"Contact {name.value} not found"
196
197     contact_list = contacts[name.value].phones
198     for number in contact_list:
199         if number == old_phone:
200             idx = contact_list.index(number)
201             contact_list[idx] = new_phone
202             break

```



```

203         return f"Phone {old_phone.value} not found for {name.value}"
204
205     return f"Contact {name.value} with phone number {old_phone.value} was updated with new
    ↪ phone number {new_phone.value}"
206
207
208 # ===== handler loader =====#
209
210 COMMANDS = {
211     "hello": hello,
212     "set birthday": set_birthday,
213     "birthday of": birthday,
214     "add": add,
215     "change": change,
216     "phones of": phones,
217     "show all": show_all,
218     "remove": remove,
219     "good bye": good_bye,
220     "close": good_bye,
221     "exit": good_bye,
222     "save": save,
223     "load": load,
224     "search": search,
225 }
226
227
228 def get_handler(*args):
229     """Функція викликає відповідний handler."""
230     return COMMANDS.get(args[0], undefined)
231
232
233 # ===== main function =====#
234
235 contacts = AddressBook() # Global variable for storing contacts
236
237
238 def main():
239
240     table = PrettyTable()
241     table.field_names = ["Name", "Birthday", "Phones"]
242     table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})
243
244     command_pattern = "|".join(COMMANDS.keys())
245     pattern = re.compile(
246         r"\b(\.|" + command_pattern + r")\b"
247         r"(?:\s+([a-zA-Z0-9\.\+])?)?"
248         r"(?:\s+(\d+|\d{1,2}\.\d{1,2}\.\d{4}(?:\.\d{2})?)?)?"
249         r"(?:\s+(\d+)?)?",
250         re.IGNORECASE,
251     )
252
253     load("contacts")
254     while True:
255         # waiting for nonempty input
256         while True:
257             inp = input(">>> ").strip()
258             if inp == "":
259                 continue
260             break
261
262         text = pattern.search(inp)
263
264         params = (

```

```

265         tuple(
266             map(
267                 # Made a commands to be a uppercase
268                 lambda x: x.lower() if text.groups().index(x) == 0 else x,
269                 text.groups(),
270             )
271         )
272         if text
273         else (None, 0, 0)
274     )
275     handler = get_handler(*params)
276     response = handler(*params[1:])
277     if inp.strip() == ".":
278         contacts.save_contacts("contacts")
279         return
280     if params[0] in ("show all", "search"):
281         param = (
282             int(params[1])
283             if params[1] is not None
284             and isinstance(params[1], str)
285             and params[1].isdigit()
286             else 100
287         )
288         if params[0] == "show all":
289             entry = contacts
290         elif params[0] == "search":
291             entry = contacts.search(params[1])
292         for tab in entry.iterator(param):
293             if tab == "continue":
294                 input("Press <Enter> to continue...")
295             else:
296                 print(build_table(tab))
297
298     print(response)
299     if response == "Good bye!":
300         return
301
302
303 # ===== main programm ===== #
304
305 if __name__ == "__main__":
306
307     main()

```