

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("Phone number must be 10 digits")
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("Invalid date format. Please use DD.MM.YYYY")
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 = 10):
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     """Функція строить 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             return "\033[1;31m{}\033[0m".format(str(error).strip("'"))
32         except ValueError as error:
33             return f"\033[1;31m{str(error)}\033[0m"
34         except TypeError as error:
35             return f"\033[1;31m{str(error)}\033[0m"
36         except FileNotFoundError:
37             return "\033[1;31mFile not found\033[0m"
38
39     return wrapper
40
41
42 # ===== handlers =====#
43
44
45 def hello(*args):
46     return "\033[32mHow can I help you?\033[0m"
47
48
49 def good_bye(*args):
50     contacts.save_contacts("contacts")
51     return "\033[32mGood bye!\033[0m"
52
53
54 def undefined(*args):
55     return "\033[32mWhat do you mean?\033[0m"
56
57
58 def show_all(*args):
59     """Функція-handler показує книгу контактів."""
60     return f"Address book contain {len(contacts)} contacts"
61
62
63 @input_error
64 def save(*args):
65     contacts.save_contacts(args[0])
66     return f"File {args[0]} saved"
67
68
69 @input_error
70 def load(*args):
71     contacts.load_contacts(args[0])
72     return f"File {args[0]} loaded"
73
74
75 @input_error
76 def set_birthday(*args):

```

```

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

```

```

140         raise KeyError("Give me a name, please")
141
142     table = PrettyTable()
143     table.field_names = ["Name", "Birthday", "Days to next Birthday"]
144     table.min_width.update(
145         {"Name": 20, "Birthday": 12, "Days to next Birthday": 40}
146     )
147
148     days_to_next_birthday = contacts.data[args[0]].days_to_birthday() or "-"
149     birthday = contacts.get(args[0]).show_birthday() or "-"
150
151     table.add_row([args[0], birthday, days_to_next_birthday])
152
153     return table
154
155     # return "No such contach founded"
156
157 @input_error
158 def search(*args):
159
160     if not args[0]:
161         raise KeyError("Give me a some string, please")
162
163     results = contacts.search(args[0])
164
165     if results:
166         return f'\033[0m{build_table(results)}'
167     return 'By your request found nothing'
168
169
170 @input_error
171 def remove(*args):
172     """Функція-handler видаляє запис з книги."""
173
174     if not args[0]:
175         raise KeyError("Give me a name, please")
176
177     name = Name(args[0])
178
179     del contacts[name.value]
180
181     return f"Contact {name.value} was removed"
182
183
184 @input_error
185 def change(*args):
186     """Функція-handler змінює телефон контакту."""
187
188     if not args[0]:
189         raise KeyError("Give me a name, please")
190
191     if not args[1]:
192         raise ValueError("Old phone number is required")
193
194     if not args[2]:
195         raise ValueError("New phone number is required")
196
197     name = Name(args[0])
198     old_phone = Phone(args[1])
199     new_phone = Phone(args[2])
200
201     if name.value not in contacts:

```



```

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

```

```

265     inp = input(">>> ").strip()
266     if inp == "":
267         continue
268     break
269
270 text = pattern.search(inp)
271
272 params = (
273     tuple(
274         map(
275             # Made a commands to be a uppercase
276             lambda x: x.lower() if text.groups().index(x) == 0 else x,
277             text.groups(),
278         )
279     )
280     if text
281     else (None, 0, 0)
282 )
283 handler = get_handler(*params)
284 response = handler(*params[1:])
285 if inp.strip() == ".":
286     contacts.save_contacts("contacts")
287     return
288 if params[0] == "show all":
289     param = (
290         int(params[1])
291         if params[1] is not None
292         and isinstance(params[1], str)
293         and params[1].isdigit()
294         else 100
295     )
296     for tab in contacts.iterator(param):
297         if tab == "continue":
298             input("Press <Enter> to continue...")
299         else:
300             print(build_table(tab))
301
302     print(f"\033[1;32m{response}\033[0m")
303     if response == "Good bye!":
304         return
305
306 # ===== main programm ===== #
307
308 if __name__ == "__main__":
309
310     main()
311

```