

Домашня робота №11/12

Sergiy Ponomarenko

23 лютого 2023 р.

1 Файл README.MD

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

```
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 - `change` --- чат змінює номер для відповідного контакту, приклад `change Sergiy 0936564532 0634564545`.
18 - `phones` --- чат виводить номери телефонів контакту, приклад `phone Sergiy`.
19 - `show all` --- чат показує усі контакти та їх номери, приклад `show all`.
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 ```
```

2 Реалізація

```
1 import re
2 from collections import UserDict
3 import json
4 from datetime import datetime
5 from prettytable import PrettyTable
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
29     @property
30     def value(self):
31         return self.__value
32
33     @value.setter
34     def value(self, value):
35         self.__value = value
36
37     def __repr__(self):
38         return f"{self.value}"
39
40     def __eq__(self, other):
41         return self.value == other.value
42
43
44 class Name(Field):
45     """Клас --- обов'язкове поле з ім'ям."""
46
47     pass
48
49
50 class Phone(Field):
51     """Клас -- необов'язкове поле з телефоном та таких один записів (Record)
52     може містити кілька."""
53
54     def __init__(self, value: str):
55         super().__init__(value)
56         if not re.match(r"^\d{10}$", value):
57             raise ValueError("Phone number must be 10 digits")
58
59
```

```

60 class Birthday(Field):
61     """Клас -- необов'язкове поле з датою народження."""
62
63     def __init__(self, value: str):
64         super().__init__(value)
65         if value is not None and not re.match(r"^\d{2}\.\d{2}\.\d{4}$", value):
66             raise ValueError("Birthday should be in format DD.MM.YYYY")
67
68
69 class Record:
70     """Клас відповідає за логіку додавання/видалення/редагування
71     необов'язкових полів та зберігання обов'язкового поля Name."""
72
73     records = {}
74
75     # Забороняємо створювати кілька об'єктів з однаковими полями Name
76     def __new__(cls, name: Name, *args, **kwargs):
77         if name.value in cls.records:
78             return cls.records[name.value]
79         return super().__new__(cls)
80
81     def __init__(
82         self,
83         name: Name,
84         phones: list[Phone] = None,
85         birthday: Birthday = None,
86     ):
87
88         # якщо об'єк було створено, то припинити роботу конструктора
89         if name.value in self.records:
90             return
91         # інакше запустити конструктор
92         self.name = name # Name --- атрибут для зберігання об'єкту Name
93         self.phones = phones or []
94         self.birthday = birthday
95         # Додаємо в словник об'єктів новий об'єкт
96         self.records[name.value] = self
97
98     def add_birthday(self, birthday: Birthday):
99         """Метод додає об'єкт день народження до запису."""
100
101         self.birthday = birthday
102
103     def add_phone(self, phone: Phone):
104         """Метод додає об'єкт телефон до запису."""
105
106         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 key in self.data:
170             record = self.data[key]
171             if (
172                 search_string in record.name.value
173                 or (
174                     record.birthday.value is not None
175                     and search_string in record.birthday.value
176                 )
177                 or any(search_string in phone.value for phone in record.phones)
178             ):
179                 results.add_record(record)
180         return results
181
182     def iterator(self, page_size=10):
183         table = PrettyTable()
184         table.field_names = ["Name", "Birthday", "Phones"]
185         table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})

```

```

186     for i, key in enumerate(self.data, 1):
187         record = self.data[key]
188         name = record.name.value
189         birthday = (
190             record.birthday.value
191             if record.birthday is not None and record.birthday.value
192             else "-"
193         )
194         phones = (
195             ", ".join(p.value for p in record.phones)
196             if record.phones
197             else "-"
198         )
199         table.add_row([name, birthday, phones])
200         if i % page_size == 0 or i == len(self.data):
201             yield table
202             if i != len(self.data):
203                 input("Press Enter to continue...")
204             table.clear_rows()
205     if table._rows:
206         yield table
207
208
209 # ===== Decorator =====#
210
211
212 def input_error(func):
213     def wrapper(*func_args, **func_kwargs):
214         try:
215             return func(*func_args, **func_kwargs)
216         except KeyError:
217             return "Give me a name, please"
218         except ValueError as error:
219             if "phone" in str(error):
220                 return "Phone number must be 10 digits"
221             elif "Birthday" in str(error):
222                 return "Birthday should be in format DD.MM.YYYY"
223             else:
224                 return str(error)
225         except TypeError:
226             return "The contact has no date of birth"
227         except FileNotFoundError:
228             return "File not found"
229
230     return wrapper
231
232
233 # ===== handlers =====#
234
235
236 def hello(*args):
237     return "How can I help you?"
238
239
240 def good_bye(*args):
241     contacts.save_contacts("contacts")
242     return "Good bye!"
243
244
245 def undefined(*args):
246     return "What do you mean?"
247
248

```

```

249 def show_all(*args):
250     """Функція-handler показує книгу контактів."""
251     return f"Address book contain {len(contacts)} contacts"
252
253
254 @input_error
255 def save(*args):
256     contacts.save_contacts(args[0])
257     return f"File {args[0]} saved"
258
259
260 @input_error
261 def load(*args):
262     contacts.load_contacts(args[0])
263     return f"File {args[0]} loaded"
264
265
266 @input_error
267 def set_birthday(*args):
268     """Функція-handler додає день народження до контакту."""
269
270     if not args[0]:
271         raise KeyError
272
273     if not args[1]:
274         raise ValueError("Birthday should be in format DD.MM.YYYY")
275
276     name = Name(args[0])
277     birthday = Birthday(args[1])
278     record = Record(name)
279     record.add_birthday(birthday)
280     contacts.add_record(record)
281
282     return f"I added a birthday {args[1]} to contact {args[0]}"
283
284
285 @input_error
286 def add(*args):
287     """Функція-handler додає телефон до контакту."""
288
289     if not args[0]:
290         raise KeyError
291
292     if not args[1]:
293         raise ValueError("The phone number must be 10 digits")
294
295     name, phone = Name(args[0]), Phone(args[1])
296     record = Record(name)
297     record.add_phone(phone)
298     contacts.add_record(record)
299
300     return f"I added a phone {args[1]} to contact {args[0]}"
301
302
303 # @input_error
304 def phones(*args):
305     """Функція-handler показує телефонні номери відповідного контакту."""
306
307     table = PrettyTable()
308     table.field_names = ["Name", "Phones"]
309     table.min_width.update({"Name": 20, "Phones": 40})
310
311     if not args[0]:

```

```

312         raise KeyError
313
314     name = Name(args[0])
315     phones = Record(name).phones
316     phones = ", ".join(phone.value for phone in phones)
317     table.add_row([name.value, phones])
318
319     return table
320
321
322 @input_error
323 def birthday(*args):
324     """Функція-handler показує день народження та кількість днів до наступного."""
325
326     table = PrettyTable()
327     table.field_names = ["Name", "Birthday", "Days to next Birthday"]
328     table.min_width.update(
329         {"Name": 20, "Birthday": 12, "Days to next Birthday": 5}
330     )
331
332     if not args[0]:
333         raise KeyError
334
335     name = Name(args[0])
336
337     birthday = Record(name).birthday
338
339     if birthday:
340
341         days_to_next_birthday = Record(name).days_to_birthday()
342
343         table.add_row([name.value, birthday.value, days_to_next_birthday])
344
345         return table
346
347     return "No such contach founded"
348
349
350 def search(*args):
351     return "Here are the found contacts"
352
353
354 @input_error
355 def remove(*args):
356     """Функція-handler видаляє запис з книги."""
357
358     if not args[0]:
359         raise KeyError
360
361     name = Name(args[0])
362
363     del contacts[name.value]
364
365     return f"Contact {name.value} was removed"
366
367
368 @input_error
369 def change(*args):
370     """Функція-handler змінює телефон контакту."""
371
372     if not args[0]:
373         raise KeyError
374

```

```

375     if not args[1]:
376         raise ValueError("Old phone number is required")
377
378     if not args[2]:
379         raise ValueError("New phone number is required")
380
381     name = Name(args[0])
382     old_phone = Phone(args[1])
383     new_phone = Phone(args[2])
384
385     if name.value not in contacts:
386         return f"Contact {name.value} not found"
387
388     contact_list = contacts[name.value].phones
389     for number in contact_list:
390         if number == old_phone:
391             idx = contact_list.index(number)
392             contact_list[idx] = new_phone
393             break
394         return f"Phone {old_phone.value} not found for {name.value}"
395
396     return f"Contact {name.value} with phone number {old_phone.value} was updated with new
397     ↪ phone number {new_phone.value}"
398
399 # ===== handler loader =====#
400
401 COMMANDS = {
402     "hello": hello,
403     "set birthday": set_birthday,
404     "birthday of": birthday,
405     "add": add,
406     "change": change,
407     "phones of": phones,
408     "show all": show_all,
409     "remove": remove,
410     "good bye": good_bye,
411     "close": good_bye,
412     "exit": good_bye,
413     "save": save,
414     "load": load,
415     "search": search,
416 }
417
418
419 def get_handler(*args):
420     """Функція викликає відповідний handler."""
421     return COMMANDS.get(args[0], undefined)
422
423
424 # ===== main function =====#
425
426
427 def main():
428
429     table = PrettyTable()
430     table.field_names = ["Name", "Birthday", "Phones"]
431     table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})
432
433     command_pattern = "|".join(COMMANDS.keys())
434     pattern = re.compile(
435         r"\b(\.|" + command_pattern + r")\b"
436         r"(?:\s+([a-zA-Z0-9\.\+])?)?"

```



```

437     r"(?:\s+(\d{10}|\d{1,2}\.\d{1,2}\.\d{4}(?:\.\d{2})?))?"
438     r"(?:\s+(\d{10})?)?",
439     re.IGNORECASE,
440 )
441
442 while True:
443
444     # waiting for nonempty input
445     while True:
446         inp = input(">>> ").strip()
447         if inp == "":
448             continue
449         break
450
451     text = pattern.search(inp)
452
453     params = (
454         tuple(
455             map(
456                 # Made a commands to be a uppercase
457                 lambda x: x.lower() if text.groups().index(x) == 0 else x,
458                 text.groups(),
459             )
460         )
461         if text
462         else (None, 0, 0)
463     )
464     handler = get_handler(*params)
465     response = handler(*params[1:])
466     if inp.strip() == ".":
467         contacts.save_contacts("contacts")
468         return
469     if params[0] in ("show all", "search"):
470         if params[0] == "show all":
471             entry = contacts
472         elif params[0] == "search":
473             entry = contacts.search(params[1])
474         for tab in entry.iterator():
475             print(tab)
476
477     print(response)
478     if response == "Good bye!":
479         return
480
481
482 contacts = AddressBook() # Global variable for storing contacts
483
484
485 # ===== main programm ===== #
486
487 if __name__ == "__main__":
488
489     main()

```