

Домашня робота №11/12

Sergiy Ponomarenko

24 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`
15   ↳ `12.12.1978`.
16 - `birthday of` -- Виводить на екран дату вказаного контакту, наприклад `birthday of Sergiy`.
17 - `add` --- чат додає ім'я і телефон, приклад `add Sergiy 0936564532`.
18 - `change` --- чат змінює номер для відповідного контакту, приклад `change Sergiy 0936564532 0634564545`.
19 - `phones` --- чат виводить номери телефонів контакту, приклад `phone Sergiy`.
20 - `show all N`--- чат показує усі контакти та їх номери, приклад `show all 10`. Необов'язковий параметр `N` -
21   ↳ число записів, що виводяться за одну ітерацію.
22 - `remove` --- чат видаляє запис з вказаним іменем, приклад `remove Sergiy`.
23 - `good bye`, `good`, `exit` --- чат прощається і завершує роботу і зберігає контакти у файл `contacts.json`.
24 - `.` --- чат перериває свою роботу без попереджень і зберігає контакти у файл `contacts.json`.
25 - `save` --- зберігає контакти у файл `.json`, наприклад `save contacts`.
26 - `load` --- завантажує книгу з контактами з файлу `.json` в чат, наприклад `load contacts`.
27 - `search` -- здійснює пошук за ключовою фразою, частиною номеру телефона чи дні народження, наприклад `search`
28   ↳ `123`, або `search Beth`.
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45

```

2 Реалізація класів

```
1 import re
2 from collections import UserDict
3
4 import json
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
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     # Забороняємо створювати кілька об'єктів з однаковими полями Name
74     # Для цього змінюємо метод __new__
75
76     records = {}
77
78     def __new__(cls, name: Name, *args, **kwargs):
79         if name.value in cls.records:
80             return cls.records[name.value]
81         return super().__new__(cls)
82
83     def __init__(
84         self,
85         name: Name,
86         phones: list[Phone] = None,
87         birthday: Birthday = None,
88     ):
89
90         # якщо об'єкт було створено, то припинити роботу конструктора
91         if name.value in self.records:
92             return
93         # інакше запустити конструктор
94         self.name = name # Name --- атрибут для зберігання об'єкту Name
95         self.phones = phones or []
96         self.birthday = birthday
97         # Додаємо в словник об'єктів новий об'єкт
98         self.records[name.value] = self
99
100     def add_birthday(self, birthday: Birthday):
101         """Метод додає об'єкт день народження до запису."""
102
103         self.birthday = birthday
104
105     def add_phone(self, phone: Phone):
106         """Метод додає об'єкт телефон до запису."""
107
108         self.phones.append(phone)
109
110     def remove_phone(self, phone: Phone):
111         """Метод видаляє об'єкт телефон із запису."""
112
113         self.phones.remove(phone)
114
115     def change_phone(self, old_phone: Phone, new_phone: Phone) -> bool:
116         """Метод змінює об'єкт телефон в записі на новий."""
117
118         if old_phone.value in self.phones:
119             idx = self.phones.index(old_phone)
120             self.phones[idx] = new_phone
121             return True
122         return False

```

```

123
124 def days_to_birthday(self) -> int:
125     """Метод повертає кількість днів до наступного дня народження контакту."""
126
127     if not self.birthday:
128         return None
129
130     today = datetime.today()
131     dt_birthday = datetime.strptime(self.birthday.value, "%d.%m.%Y")
132     next_birthday = dt_birthday.replace(year=today.year)
133
134     if next_birthday < today:
135         next_birthday = dt_birthday.replace(year=today.year + 1)
136
137     return (next_birthday - today).days
138
139
140 class AddressBook(UserDict):
141     """Клас містить логіку пошуку за записами до цього класу."""
142
143     def add_record(self, record: Record):
144         """Метод додає запис до списку контактів."""
145
146         self.data[record.name.value] = record
147
148     def save_contacts(self, filename):
149         """Метод зберігає записи до в json файл."""
150
151         with open(f"{filename}.json", "w") as f:
152             contacts = {}
153             for name, record in self.data.items():
154                 phones = [phone.value for phone in record.phones]
155                 birthday = record.birthday.value if record.birthday else None
156                 if birthday:
157                     contacts[name] = {"phones": phones, "birthday": birthday}
158                 else:
159                     contacts[name] = {"phones": phones}
160             json.dump(contacts, f, ensure_ascii=False, indent=4)
161
162     def load_contacts(self, filename):
163         """Метод завантажує записи з json в змінну-екземпляр класу."""
164
165         with open(f"{filename}.json", "r") as f:
166             data = json.load(f)
167
168             for name, info in data.items():
169                 phones = [Phone(phone) for phone in info["phones"]]
170                 birthday = Birthday(info.get("birthday"))
171                 self.data[name] = Record(
172                     Name(name), phones=phones, birthday=birthday
173                 )
174
175     def search(self, search_string):
176         """Метод шукає записи по ключовому слову."""
177
178         results = AddressBook()
179         for key in self.data:
180             record = self.data[key]
181             if (
182                 search_string in record.name.value
183                 or (
184                     record.birthday.value is not None
185                     and search_string in record.birthday.value

```

```

186         )
187         or any(search_string in phone.value for phone in record.phones)
188     ):
189         results.add_record(record)
190     return results
191
192     def iterator(self, n: int):
193         """Метод ітерується по записам і виводить їх частинами по n-штук."""
194
195         data_items = list(self.data.items())
196         for i in range(0, len(data_items), n):
197             data_slice = dict(data_items[i: i + n])
198             yield data_slice
199             if i + n < len(data_items):
200                 yield "continue"
201

```

3 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 стріить 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     for key in data:
14         record = data[key]
15         name = record.name.value
16         birthday = (
17             record.birthday.value
18             if record.birthday is not None and record.birthday.value
19             else "-"
20         )
21         phones = (
22             ", ".join(phone.value for phone in record.phones)
23             if record.phones
24             else "-"
25         )
26         table.add_row([name, birthday, phones])
27     return table
28
29
30 # ===== Decorator =====#
31
32
33 def input_error(func):
34     def wrapper(*func_args, **func_kwargs):
35         try:
36             return func(*func_args, **func_kwargs)
37         except KeyError:
38             return "Give me a name, please"
39

```

```

39     except ValueError as error:
40         if "phone" in str(error):
41             return "Phone number must be 10 digits"
42         elif "Birthday" in str(error):
43             return "Birthday should be in format DD.MM.YYYY"
44         else:
45             return str(error)
46     except TypeError:
47         return "The contact has no date of birth"
48     except FileNotFoundError:
49         return "File not found"
50
51     return wrapper
52
53
54 # ===== handlers =====#
55
56
57 def hello(*args):
58     return "How can I help you?"
59
60
61 def good_bye(*args):
62     contacts.save_contacts("contacts")
63     return "Good bye!"
64
65
66 def undefined(*args):
67     return "What do you mean?"
68
69
70 def show_all(*args):
71     """Функція-handler показує книгу контактів."""
72     return f"Address book contain {len(contacts)} contacts"
73
74
75 @input_error
76 def save(*args):
77     contacts.save_contacts(args[0])
78     return f"File {args[0]} saved"
79
80
81 @input_error
82 def load(*args):
83     contacts.load_contacts(args[0])
84     return f"File {args[0]} loaded"
85
86
87 @input_error
88 def set_birthday(*args):
89     """Функція-handler додає день народження до контакту."""
90
91     if not args[0]:
92         raise KeyError
93
94     if not args[1]:
95         raise ValueError("Birthday should be in format DD.MM.YYYY")
96
97     name = Name(args[0])
98     birthday = Birthday(args[1])
99     record = Record(name)
100     record.add_birthday(birthday)
101     contacts.add_record(record)

```

```

102     return f"I added a birthday {args[1]} to contact {args[0]}"
103
104
105
106 @input_error
107 def add(*args):
108     """Функція-handler додає телефон до контакту."""
109
110     if not args[0]:
111         raise KeyError
112
113     if not args[1]:
114         raise ValueError("The phone number must be 10 digits")
115
116     name, phone = Name(args[0]), Phone(args[1])
117     record = Record(name)
118     record.add_phone(phone)
119     contacts.add_record(record)
120
121     return f"I added a phone {args[1]} to contact {args[0]}"
122
123
124 @input_error
125 def phones(*args):
126     """Функція-handler показує телефонні номери відповідного контакту."""
127
128     table = PrettyTable()
129     table.field_names = ["Name", "Phones"]
130     table.min_width.update({"Name": 20, "Phones": 55})
131
132     if not args[0]:
133         raise KeyError
134
135     name = Name(args[0])
136     phones = Record(name).phones
137     phones = ", ".join(phone.value for phone in phones)
138     table.add_row([name.value, phones])
139
140     return table
141
142
143 @input_error
144 def birthday(*args):
145     """Функція-handler показує день народження та кількість днів до наступного."""
146
147     table = PrettyTable()
148     table.field_names = ["Name", "Birthday", "Days to next Birthday"]
149     table.min_width.update(
150         {"Name": 20, "Birthday": 12, "Days to next Birthday": 40}
151     )
152
153     if not args[0]:
154         raise KeyError
155
156     name = Name(args[0])
157
158     birthday = Record(name).birthday
159
160     if birthday:
161
162         days_to_next_birthday = Record(name).days_to_birthday()
163
164         table.add_row([name.value, birthday.value, days_to_next_birthday])

```

```

165         return table
166
167     return "No such contach founded"
168
169
170
171 def search(*args):
172     return "Here are the found contacts"
173
174
175 @input_error
176 def remove(*args):
177     """Функція-handler видаляє запис з книги."""
178
179     if not args[0]:
180         raise KeyError
181
182     name = Name(args[0])
183
184     del contacts[name.value]
185
186     return f"Contact {name.value} was removed"
187
188
189 @input_error
190 def change(*args):
191     """Функція-handler змінює телефон контакту."""
192
193     if not args[0]:
194         raise KeyError
195
196     if not args[1]:
197         raise ValueError("Old phone number is required")
198
199     if not args[2]:
200         raise ValueError("New phone number is required")
201
202     name = Name(args[0])
203     old_phone = Phone(args[1])
204     new_phone = Phone(args[2])
205
206     if name.value not in contacts:
207         return f"Contact {name.value} not found"
208
209     contact_list = contacts[name.value].phones
210     for number in contact_list:
211         if number == old_phone:
212             idx = contact_list.index(number)
213             contact_list[idx] = new_phone
214             break
215     return f"Phone {old_phone.value} not found for {name.value}"
216
217     return f"Contact {name.value} with phone number {old_phone.value} was updated with new
    ↪ phone number {new_phone.value}"
218
219
220 # ===== handler loader =====#
221
222 COMMANDS = {
223     "hello": hello,
224     "set birthday": set_birthday,
225     "birthday of": birthday,
226     "add": add,

```



```

227     "change": change,
228     "phones of": phones,
229     "show all": show_all,
230     "remove": remove,
231     "good bye": good_bye,
232     "close": good_bye,
233     "exit": good_bye,
234     "save": save,
235     "load": load,
236     "search": search,
237 }
238
239
240 def get_handler(*args):
241     """Функція викликає відповідний handler."""
242     return COMMANDS.get(args[0], undefined)
243
244
245 # ===== main function =====#
246
247
248 def main():
249
250     table = PrettyTable()
251     table.field_names = ["Name", "Birthday", "Phones"]
252     table.min_width.update({"Name": 20, "Birthday": 12, "Phones": 40})
253
254     command_pattern = "|".join(COMMANDS.keys())
255     pattern = re.compile(
256         r"\b(\.|" + command_pattern + r")\b"
257         r"(?:\s+([a-zA-Z0-9\.\+]))?"
258         r"(?:\s+(\d{10}|\d{1,2}\.\d{1,2}\.\d{4}(?:\.\d{2}))?)?"
259         r"(?:\s+(\d{10}))?",
260         re.IGNORECASE,
261     )
262
263     while True:
264
265         # waiting for nonempty input
266         while True:
267             inp = input(">>> ").strip()
268             if inp == "":
269                 continue
270             break
271
272         text = pattern.search(inp)
273
274         params = (
275             tuple(
276                 map(
277                     # Made a commands to be a uppercase
278                     lambda x: x.lower() if text.groups().index(x) == 0 else x,
279                     text.groups(),
280                 )
281             )
282             if text
283             else (None, 0, 0)
284         )
285         handler = get_handler(*params)
286         response = handler(*params[1:])
287         if inp.strip() == ".":
288             contacts.save_contacts("contacts")
289         return

```

```

290     if params[0] in ("show all", "search"):
291         param = (
292             int(params[1])
293             if params[1] is not None
294             and isinstance(params[1], str)
295             and params[1].isdigit()
296             else 10
297         )
298         if params[0] == "show all":
299             entry = contacts
300         elif params[0] == "search":
301             entry = contacts.search(params[1])
302         for tab in entry.iterator(param):
303             if tab == "continue":
304                 input("Press <Enter> to continue...")
305             else:
306                 print(build_table(tab))
307
308     print(response)
309     if response == "Good bye!":
310         return
311
312
313 contacts = AddressBook() # Global variable for storing contacts
314
315
316 # ===== main programm ===== #
317
318 if __name__ == "__main__":
319
320     main()

```