Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Програмування**

**Лабораторна робота №6**

«Основи об’єктно-орієнтованого програмування. Модулі та пакети»

Виконав:

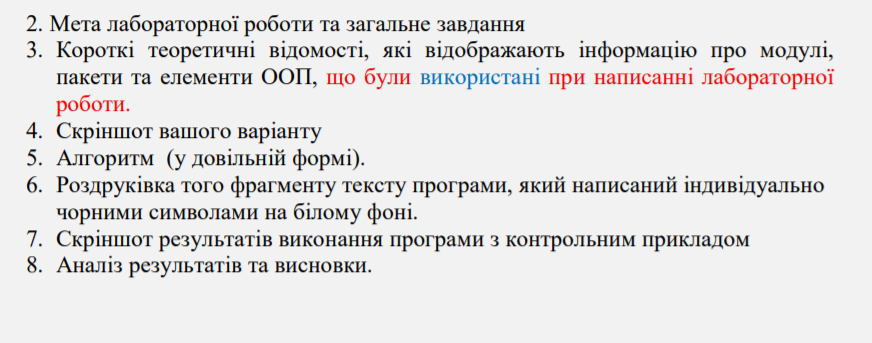
студент групи ІO-04

Федорко Андрій П.

Залікова книжка № 423

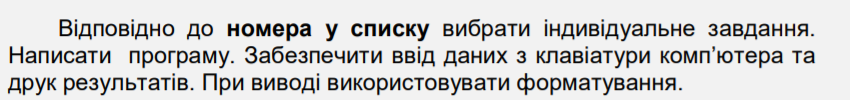
Перевірив:

Пономаренко Артем Миколайович

****

Мета: вивчити способи створення та підключення модулів та пакетів. Основи ООП. Методи і атрибути класів та робота з ними. Побудова програми у стилі ООП.

Завдання: 1. Вивчити матеріал лекцій 18, 19, 20 та 21. 2. Виконати індивідуальне завдання лабораторної роботи, вибране відповідно до варіанту.

****

**Теоретичні основи:**

Перевірити існування атрибута дозволяє функція hasattr (, ).

Імпортування модуля виконується тільки при першому виклику інструкції import (або from). При кожному виклику інструкції import перевіряється наявність об'єкта модуля в словнику modules з модуля sys. Якщо посилання на модуль перебуває в цьому словнику, то модуль повторно імпортуватися не буде.

Для імпортування тільки деяких визначених ідентифікаторів з модуля можна скористатися інструкцією from. Формат інструкції: from import

from <назва пакета>.<назва модуля> import <>

Пакетом називають каталог з модулями, у якому розташований файл ініціалізації \_\_init\_\_.py. Файл ініціалізації може бути порожнім або містити код, який буде виконаний при першому доступі до пакета. У будь-якому разі він обов'язково повинен бути присутнім всередині каталогу з модулями.

Об'єктно-орієнтоване програмування (ООП) – це спосіб організації програми, що дозволяє використовувати той самий код багаторазово. На відміну від функцій і модулів, ООП дозволяє не тільки розділити програму на фрагменти, але й описати предмети реального світу у вигляді зручних сутностей – об'єктів, а також організувати зв'язки між цими об'єктами. Основною «цеглинкою» ООП є клас. Клас – це складний тип даних, що включає набір змінних і функцій для керування значеннями, що зберігаються в цих змінних. Змінні називають атрибутами, а функції – методами. Клас є фабрикою об'єктів, тобто дозволяє створити необмежену кількість екземплярів, заснованих на цьому класі.

основні концепції ООП: інкапсуляція, спадкування

Визначення класу й створення екземпляра класу Клас описують за допомогою ключового слова class за наступною схемою: class [([, ... , ])]: [""" Рядок документування """]

Інструкція створює новий об'єкт і присвоює посилання на нього ідентифікатору, зазначеному після ключового слова class. Це означає, що назва класу повинна повністю відповідати правилам іменування змінних. Після назви класу в круглих дужках можна вказати один або кілька базових класів через кому. Якщо ж клас не успадковує базові класи, то круглі дужки можна не вказувати. Всі вирази всередині інструкції class виконуються при створенні класу, а не його екземпляра. Створення атрибута класу аналогічно створенню звичайної змінної. Метод всередині класу створюється так само, як і звичайна функція, за допомогою інструкції def. Методам класу в першому параметрі, який обов'язково слід указати явно, автоматично передають посилання на екземпляр класу. Загальноприйнято цей параметр називати ім’ям self (не обов'язково). Доступ до атрибутів і методів класу всередині обумовленого методу проводиться через змінну self за допомогою точкової нотації – до атрибута х з методу класу можна звернутися так: self.х. Щоб використовувати атрибути й методи класу, необхідно створити екземпляр класу згідно з наступним синтаксисом: = ((]) При доступі до методів класу використовують такий формат: .((]) При виклику методу не потрібно передавати посилання на екземпляр класу як параметр, як у визначенні методу всередині класу. Посилання на екземпляр класу інтерпретатор передає автоматично. Доступ до атрибутів класу здійснюється аналогічно: .

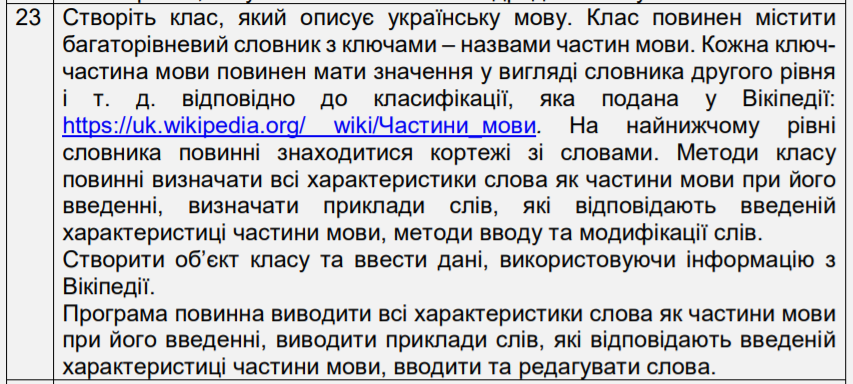
При створенні екземпляра класу інтерпретатор автоматично викликає метод ініціалізації \_\_init\_\_(). В інших мовах програмування такий метод прийнято називати конструктором класу. Формат методу: def\_\_init\_\_(self[, [, ... , ]]):

Інкапсуляція Розглянемо концепцію розробки, що одержала назву «інкапсуляція». Ідея інкапсуляції полягає в тому, щоб сховати логіку функціонування від зовнішнього доступу, а користувачеві даного фрагмента коду надати тільки інтерфейс для його використання. 1. У цьому випадку ви можете звертатися до цього фрагмента з різних областей програми, що заощаджує розмір коду. 2. Якщо необхідно модифікувати інкапсульований фрагмент, то це не вплине на працездатність всієї програми. В Python інкапсуляція виконується за допомогою методів класу.

Спадкування Спадкування є найголовнішим поняттям ООП. Припустимо, у нас є клас (наприклад, Class1). За допомогою спадкування ми можемо створити новий клас (наприклад, Class2), у якому буде реалізований доступ до всіх атрибутів і методів класу Class1. УВАГА! Конструктор базового класу автоматично не викликається, якщо він перевизначений у підкласі. Щоб викликати однойменний метод з базового класу, можна також скористатися функцією super(). Формат функції: super([, ])

За допомогою функції super() інструкцію Class1.\_\_init\_\_(self) #Викликаємо конструктор базового класу можна записати так: super().\_\_init\_\_() #Викликаємо конструктор базового класу або так: super(Class2, self).\_\_init\_\_() # Викликаємо конструктор базового класу

При використанні функції super() не потрібно явно передавати вказівник self у викликуваний метод. Крім того, у першому параметрі функції super() вказують похідний клас (підклас), а не базовий. Пошук ідентифікатора буде проводитися у всіх базових класах. Результатом пошуку стане перший знайдений ідентифікатор у ланцюжку спадкування.

****

**Алгоритм (у довільній формі):**<https://github.com/Ap3lsin4k/words-aspart-of-speech>

**Source code:**

*# language\_interactor.py*from language\_entity import LanguageEntity  
from repositories.correct\_typo\_repository import CorrectTypo  
from repositories.word\_classifier\_repository import WordClassifierRepository  
from repositories.word\_of\_same\_category\_repository import WordsOfSameCategoryRepository  
  
  
class UkrainianLanguageInteractor():  
 def \_\_init\_\_(self, nested\_dictionary):  
 self.\_\_dictionary = LanguageEntity(nested\_dictionary)  
 self.\_\_classifier = WordClassifierRepository(self.\_\_dictionary)  
 self.\_\_words\_same\_category = WordsOfSameCategoryRepository(self.\_\_dictionary)  
  
 *# WORD CLASSIFIER* def classify(self, input\_word):  
 self.\_\_classifier.for\_each\_part\_of\_speech(self.\_\_classifier.make\_response\_model, input\_word)  
 return self.\_\_classifier.result  
  
 *# SHOW EXAMPLES OF WORDS FOR GIVEN PROPERTY, SHOW CLASS OF WORDS WITH SAME PROPERTY* def get\_examples(self, property\_name):  
 self.\_\_words\_same\_category.for\_each\_part\_of\_speech(self.\_\_words\_same\_category.find\_words\_in\_category\_of\_properties, property\_name)  
 res = [self.\_\_words\_same\_category.result, self.\_\_words\_same\_category.bm]  
 return res  
  
 def modify(self, bookmark, old\_word, new\_word):  
 modifiable = list(self.\_\_dictionary.get\_words\_for\_property(bookmark))  
 index = modifiable.index(old\_word)  
 modifiable[index] = new\_word  
 self.\_\_dictionary[bookmark.get\_part\_of\_speech()][bookmark.category\_name][bookmark.property\_name] = tuple(modifiable)  
  
 def update(self, param):  
 self.\_\_dictionary.update(param)  
  
 def construct\_close\_matches(self, typo):  
 suggestion = set()  
 CorrectTypo().get\_close\_matches(suggestion, self.\_\_dictionary, typo)  
 return suggestion

*# language\_entity.py*from repositories.language\_extend\_behaviour import LanguageExtendBehaviour  
  
  
class LanguageEntity(LanguageExtendBehaviour):  
  
 def \_\_init\_\_(self, nested\_dictionary):  
 super().\_\_init\_\_()  
 self.update(nested\_dictionary)  
  
 def get\_part\_of\_speech(self, bookmark):  
 return self[bookmark.get\_part\_of\_speech()]  
  
 def get\_properties(self, bookmark):  
 return self[bookmark.get\_part\_of\_speech()][bookmark.category\_name]  
  
 def get\_words\_for\_property(self, bookmark):  
 return self[bookmark.get\_part\_of\_speech()][bookmark.category\_name][bookmark.property\_name]

*# bookmark\_entity.py*class Bookmark:  
 def \_\_init\_\_(self, part\_of\_speech=None, category\_name=None, property\_name=None):  
 self.\_\_part\_of\_speech = part\_of\_speech  
 self.category\_name = category\_name  
 self.property\_name = property\_name  
  
 def get\_part\_of\_speech(self):  
 if self.\_\_part\_of\_speech is not None:  
 return self.\_\_part\_of\_speech  
 else:  
 raise ValueError(**"Cannot get Bookmark.part\_of\_speech key name because it was not set to a value. "  
 "Please specify before using."**)

*#\_\_init\_\_.py*from presentation.ua\_lang\_controller import Controller  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 c = Controller()  
 while True:  
 c.execute()

from bookmark\_entity import Bookmark  
from language\_interactor import UkrainianLanguageInteractor  
from presentation.ua\_lang\_presenter import UkrainianLanguagePresenter  
  
ua\_lang = UkrainianLanguageInteractor({  
 **'іменник'**: {  
 **'рід'**: {  
 **'чоловічий'**:  
 (**'хлопець'**, **"потяг"**, **"каменяр"**),  
 **'жіночий'**:  
 (**'дівчина'**, **"нехворощ"**, **"любов"**),  
 **'середній'**:  
 (**"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**, **"ягня"**),  
 **'спільний'**:  
 (**"ледащо"**, **"сирота"**, **"нероба"**, **"розбишака"**, **"бідолаха"**, **"староста"**)  
  
 },  
 **'число'**: {  
 **'однина'**:  
 (**'хлопець'**, **"дівчина"**, **"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**, **"потяг"**, **"ледащо"**,  
 **"сирота"**, **"нероба"**, **"нехворощ"**, **"любов"**),  
 **'множина'**:  
 (**"потяги"**, **"двері"**, **"штани"**, **"ножиці"**)  
 },  
 **'відміна'**: {  
 **'перша'**:  
 (**"дівчина"**, **"ледащо"**, **"сирота"**, **"нероба"**, **"розбишака"**, **"бідолаха"**, **"староста"**),  
 **'друга'**:  
 (**"хлопець"**, **"потяг"**, **"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**, **"каменяр"**),  
 **'третя'**:  
 (**"нехворощ"**, **"любов"**, **"мати"**),  
 **'четверта'**:  
 (**"ягня"**,)  
 }  
 },  
 **'числівник'**: {  
 **'за значенням'**: {  
 **"кількісний"**:  
 (**"п'ять"**, **"двісті двадцять"**, **"шість"**, **"тридцять три"**, **"сорок вісім"**),  
 **"порядковий"**:  
 (**"четвертий"**, **"сьомий"**, **"десятий"**, **"сто двадцять перший"**)  
 }  
 },  
 **'займенник'**: {  
 **'за значенням'**: {  
 **"власні (особові)"**:  
 (**"я"**, **"ти"**, **"він"**, **"вона"**, **"воно"**),  
 **"зворотні"**:  
 (**"себе"**,),  
 **"питальні"**:  
 (**"що?"**, **"хто?"**, **"скільки?"**, **"який?"**, **"чий?" "котрий?"**),  
 **"відносні"**:  
 (**"що"**, **"хто"**, **"скільки"**, **"який"**, **"чий" "котрий"**),  
 **"присвійні"**:  
 (**"мій"**, **"твій"**, **"наш"**, **"ваш"**, **"його"**, **"її"**, **"їхній"**, **"свій"**),  
 **"вказівні"**:  
 (**"оцей"**, **"сей"**, **"той"**, **"стільки"**, **"такий"**, **"отакий"**),  
 **"означальні"**: (**"весь"**, **"всякий"**, **"сам"**, **"кожний"**, **"самий"**, **"інший"**),  
 **"неозначені"**: (**"абихто"**, **"абищо"**, **"будь-який"**, **"скільки-небудь"**),  
 **"заперечні"**: (**"ніщо"**, **"ніякий"**, **"нічий"**, **"аніхто"**, **"аніщо"**, **"аніякий"**)  
 }  
 },  
 **'дієслово'**: {  
 **'вид'**: {  
 **"доконаний"**:  
 (**"заробив"**, **"перевів"**, **"співають"**, **"бажає"**),  
 **"недоконаний"**:  
 (**"заробляв"**, **"переводить"**, **"співали"**, **"бажала"**),  
 },  
 **'рід'**: {  
 **"чоловічий"**:  
 (**"працював"**,),  
 **"жіночий"**:  
 (**"працювала"**,),  
 **"середній"**:  
 (**"працювало"**,)  
 },  
 **'число'**: {  
 **'однина'**:  
 (**"малював"**, **"малювала"**, **"малювало"**),  
 **'множина'**:  
 (**"малювали"**,),  
 },  
 **'спосіб'**: {  
 **'дійсний'**:  
 (**"працюю"**, **"сказали"**, **"летить"**, **"прийде"**),  
 **'умовний'**:  
 (**"ходила б"**, **"вживав би"**, **"поспішали б"**),  
 **'наказовий'**:  
 (**"напишіть"**, **"ходіймо"**, **"хай зайде"**)  
 },  
 **'час'**: {  
 **'минулий'**:  
 (**"знаходив"**, **"знаходила"**, **"знаходили"**, **"працював"**, **"працювала"**, **"працювало"**),  
 **'теперішній'**:  
 (**"знаходжу"**, **"знаходимо"**, **"знаходите"**),  
 **'майбутній'**:  
 (**"знайду"**, **"знайдемо"**)  
 },  
 }  
})  
presenter = UkrainianLanguagePresenter()  
  
  
class Controller:  
  
 def execute(self):  
 command = input(**"> "**)  
 if command == **'help'** or command == **'"help"'** or command == **'help()'**:  
 self.\_\_print\_manual()  
 elif **'new'** in command:  
 self.extend\_dictionary()  
 elif **'edit'** in command:  
 self.edit\_dictionary()  
 else:  
 self.\_\_make\_request(command)  
 print()  
  
 def \_\_print\_manual(self):  
 print(**"Введіть слово, щоб подивитися характеристику."**)  
 print(**"При введені характеристики, програма виведе приклад слів."**)  
 print(**"Ключове слово** \"**new**\" **без лапок, щоб додати нові слова у словник."**)  
 print(**"Ключове слово** \"**edit**\" **без лапок, щоб відредагувати існуюче слово нові слова у словник."**)  
  
 def \_\_make\_request(self, word\_might\_be\_typo):  
 try:  
 presenter.print\_properties(ua\_lang.classify(word\_might\_be\_typo))  
 except (KeyError, ValueError) as msg:  
 presenter.error\_messages.append(str(msg))  
 try:  
 presenter.print\_words\_as\_examples(\*ua\_lang.get\_examples(word\_might\_be\_typo))  
 presenter.error\_messages.clear()  
 except (KeyError, ValueError) as msg:  
 if str(msg) not in presenter.error\_messages:  
 presenter.error\_messages.append(str(msg))  
 presenter.print\_error()  
 presenter.print\_suggestions\_to\_typo(ua\_lang.construct\_close\_matches(word\_might\_be\_typo))  
  
 def extend\_dictionary(self):  
 part\_of\_speech = input(**'Введіть частину мови[прикметник]: '**)  
 category\_name = input(**'Введіть за чим класифікувати слово[число]: '**)  
 property\_name = input(**'Введіть до якої характеристики належить[множина]: '**)  
  
 print(**"Введіть слова розділені пробілом[зелена золотиста промениста неймовірна]"**)  
 words = input(**">>> "**)  
  
 print(**"Слова '{}' будуть додані до словника, частина мови — '{}', {} — {}."** .format(words, part\_of\_speech, category\_name, property\_name))  
 command = input(**"Підтвердити(так/ні): "**)  
 if command.lower() in (**"так"**, **"т"**, **"y"**, **"yes"**):  
 inp = {part\_of\_speech: {category\_name: {property\_name: tuple(words.split())}}}  
 ua\_lang.update(inp)  
 else:  
 print(**"Слово(а) не були додані до словника"**, words)  
  
 def edit\_dictionary(self):  
 part\_of\_speech = input(**'Введіть частину мови[числівник]: '**)  
 category\_name = input(**'Введіть за чим класифікувати слово[за значенням]: '**)  
 property\_name = input(**'Введіть до якої характеристики належить[кількісний]: '**)  
 old\_word = input(**"Введіть поточне слово[єдин]: "**)  
 new\_word = input(**"Введіть нове слово[один]: "**)  
 print(**"Замінити слово '{}' на '{}'."**.format(old\_word, new\_word))  
 command = input(**"Підтвердити(так/ні): "**)  
 if command.lower() in (**"так"**, **"т"**, **"y"**, **"yes"**):  
 bm = Bookmark(part\_of\_speech, category\_name, property\_name)  
 ua\_lang.modify(bm, old\_word, new\_word)  
 else:  
 print(**"Скасовано"**)

class UkrainianLanguagePresenter:  
 error\_messages = []  
  
 def \_\_init\_\_(self):  
 print(**'Введіть "help", щоб подивитися більше інформації'**)  
  
 @staticmethod  
 def print\_properties(result):  
 print(**'Частина мови — {};'**.format(tuple(result.keys())[0]))  
  
 for category\_of\_property in result.values():  
 for property, property\_name in category\_of\_property.items():  
 print(**'{:>10} — {};'**.format(property, property\_name))  
  
 @staticmethod  
 def print\_words\_as\_examples(words, bookmark):  
 print(**'Частина мови — {};'**.format(bookmark.get\_part\_of\_speech()))  
  
 print(**'Слова, що відповідають характеристиці {} — {}:'**.format(bookmark.category\_name, bookmark.property\_name))  
 print(**"**\t\t**"**.join(words))  
  
 @staticmethod  
 def print\_suggestions\_to\_typo(most\_similiar\_words):  
 if not most\_similiar\_words:  
 return  
  
 print(**"Можливо ви мали на увазі"**)  
 for suggestion in most\_similiar\_words:  
 print(**"{::>9}"**.format(suggestion))  
  
 def print\_error(self):  
 for msg in self.error\_messages:  
 print(msg)  
 self.error\_messages.clear()

from bookmark\_entity import Bookmark  
  
  
class ResponseModel:  
 bookmark: Bookmark  
 words: tuple

class LanguageExtendBehaviour(dict):  
  
 def update(self, E=None, \*\*F): *# known special case of dict.update* if hasattr(E, **"keys"**):  
 for part\_of\_speech in E.keys():  
 self.\_\_initialize\_part\_of\_speech(E, part\_of\_speech)  
 else:  
 raise NotImplementedError  
  
 def \_\_initialize\_part\_of\_speech(self, in\_first\_lvl\_dict, part\_of\_speech):  
 self.\_\_assign\_if\_key\_does\_not\_exist(self, part\_of\_speech)  
 self.\_\_try\_update\_each\_category(self[part\_of\_speech], in\_first\_lvl\_dict[part\_of\_speech])  
  
 def \_\_try\_update\_each\_category(self, dictionary\_set\_by\_reference, in\_categories\_dict):  
 if hasattr(in\_categories\_dict, **"keys"**):  
 self.\_\_update\_each\_category(dictionary\_set\_by\_reference, in\_categories\_dict)  
 else:  
 raise TypeError(**"Expected to get dictionary with category properties as keys, but got {}"** .format(type(in\_categories\_dict)))  
  
 def \_\_update\_each\_category(self, dictionary\_set\_by\_reference, second\_lvl\_dict):  
 for category in second\_lvl\_dict.keys():  
 self.\_\_assign\_if\_key\_does\_not\_exist(dictionary\_set\_by\_reference, category)  
 self.\_\_try\_update\_each\_property(dictionary\_set\_by\_reference[category], second\_lvl\_dict[category])  
  
 def \_\_try\_update\_each\_property(self, dict\_reference, in\_dictionary):  
 if not isinstance(in\_dictionary, dict):  
 raise TypeError(**"Expected to get dictionary, but got {}"** .format(type(in\_dictionary)))  
  
 self.\_\_update\_each\_property(dict\_reference, in\_dictionary)  
  
 def \_\_update\_each\_property(self, dict\_reference, in\_dictionary):  
 for property\_name in in\_dictionary:  
 self.\_\_push\_back\_words\_to\_property(dict\_reference, property\_name, in\_dictionary[property\_name])  
  
 def \_\_push\_back\_words\_to\_property(self, dict\_reference, property\_key, new\_words):  
 self.\_\_assign\_if\_key\_does\_not\_exist(dict\_reference, property\_key, default\_value=tuple())  
 dict\_reference[property\_key] = dict\_reference[property\_key] + new\_words  
  
 def \_\_assign\_if\_key\_does\_not\_exist(self, dict\_ref, key, default\_value=None):  
 if default\_value is None:  
 default\_value = dict()  
  
 if not isinstance(dict\_ref, dict):  
 raise TypeError(**"Expected to get dictionary, but got {}"**.format(type(dict\_ref)))  
  
 if key not in dict\_ref:  
 dict\_ref[key] = default\_value

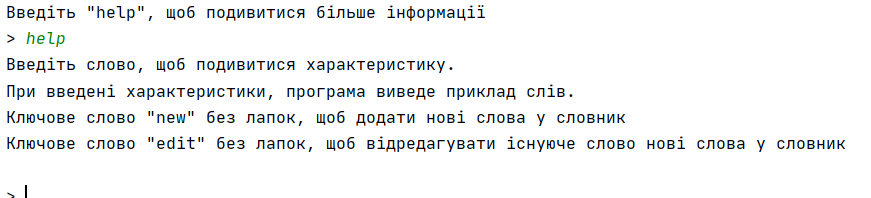
from bookmark\_entity import Bookmark  
from language\_entity import LanguageEntity  
from repositories.dictionary\_surfer\_common import DictionarySurferRepository  
  
  
class WordClassifierRepository(DictionarySurferRepository):  
  
 def \_\_init\_\_(self, dictionary\_entity: LanguageEntity):  
 super().\_\_init\_\_(dictionary\_entity)  
  
 def make\_response\_model(self, part\_of\_speech, input\_word):  
 self.result = {part\_of\_speech: {}}  
  
 for category\_of\_property, properties in self.dictionary[part\_of\_speech].items():  
 bookmark = Bookmark(part\_of\_speech, category\_of\_property)  
 self.\_\_classify\_word\_by\_property(bookmark, input\_word)  
  
 if len(self.result[part\_of\_speech]) == 0:  
 self.result = None  
  
 def \_\_save\_property\_of\_word\_to\_presentable\_format(self, bookmark):  
 self.result[bookmark.get\_part\_of\_speech()].update({bookmark.category\_name: bookmark.property\_name})  
  
 def \_\_classify\_word\_by\_property(self, bookmark, input\_word):  
 for bookmark.property\_name in self.dictionary.get\_properties(bookmark):  
 words\_tuple = self.dictionary.get\_words\_for\_property(bookmark)  
 if input\_word in words\_tuple:  
 self.\_\_save\_property\_of\_word\_to\_presentable\_format(bookmark)

from language\_entity import LanguageEntity  
from repositories.dictionary\_surfer\_common import DictionarySurferRepository  
  
  
class WordsOfSameCategoryRepository(DictionarySurferRepository):  
  
 def \_\_init\_\_(self, dictionary\_entity: LanguageEntity):  
 super().\_\_init\_\_(dictionary\_entity)  
 self.result = None  
 self.bm = None  
  
 def find\_words\_in\_category\_of\_properties(self, part\_of\_speech, property\_name):  
 self.for\_each\_category\_of\_property(self.\_\_save\_examples\_for\_given\_property, part\_of\_speech, property\_name)  
  
 def \_\_save\_examples\_for\_given\_property(self, bm, property\_name):  
 if property\_name in self.dictionary.get\_properties(bm):  
 self.bm = bm  
 self.bm.property\_name = property\_name  
 self.result = self.dictionary.get\_words\_for\_property(bm)

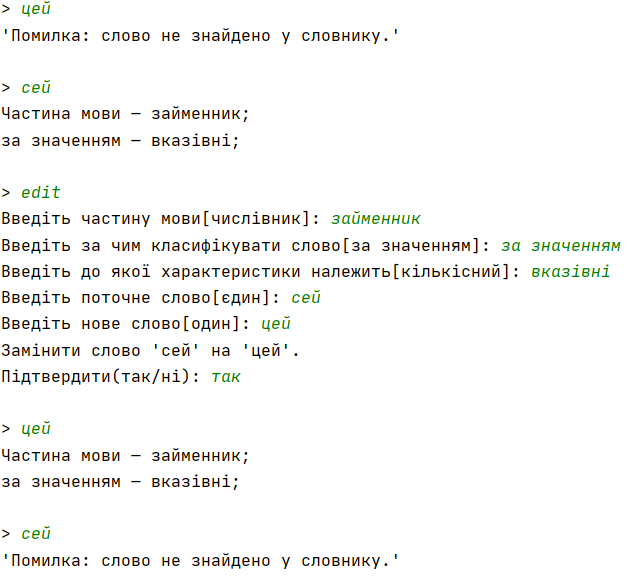
from bookmark\_entity import Bookmark  
from language\_entity import LanguageEntity  
  
  
class DictionarySurferRepository:  
  
 def \_\_init\_\_(self, dictionary\_entity: LanguageEntity):  
 self.dictionary = dictionary\_entity  
 self.result = None  
  
 def for\_each\_part\_of\_speech(self, handle\_func, input\_word\_or\_property):  
 if input\_word\_or\_property is None or input\_word\_or\_property == **""**:  
 raise ValueError(**"Помилка: рядок не може бути пустий"**)  
  
 self.result = None  
 for part\_of\_speech, categories\_of\_properties\_dict in self.dictionary.items():  
 handle\_func(part\_of\_speech, input\_word\_or\_property)  
  
 if self.result is not None:  
 return  
 if self.result is None:  
 raise KeyError(**"Помилка: слово не знайдено у словнику."**)  
  
 def for\_each\_category\_of\_property(self, handle\_func, part\_of\_speech, property\_name):  
 for category\_of\_property, properties in self.dictionary[part\_of\_speech].items():  
 bm = Bookmark(part\_of\_speech, category\_of\_property, property\_name)  
 handle\_func(bm, property\_name)

import difflib  
  
  
class CorrectTypo:  
 def get\_close\_matches(self, out\_result, container, typo):  
 out\_result.update(difflib.get\_close\_matches(typo, container))  
 if hasattr(container, **"values"**):  
 for inner in container.values():  
 self.get\_close\_matches(out\_result, inner, typo)

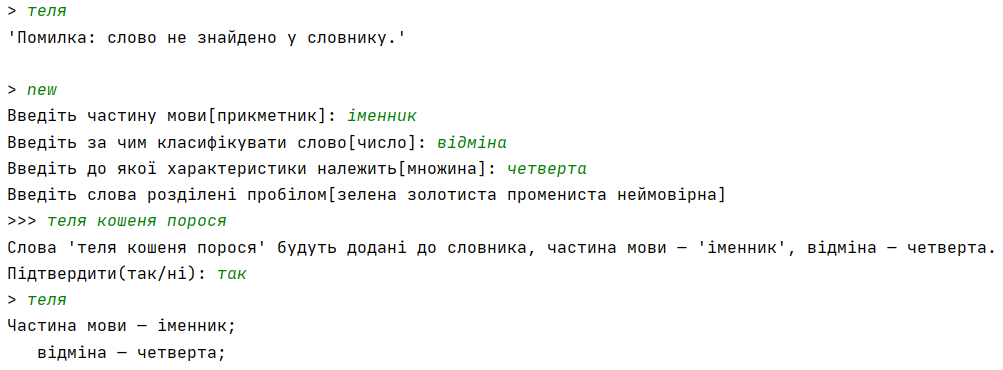
**Скріншот результатів виконання завдання:**

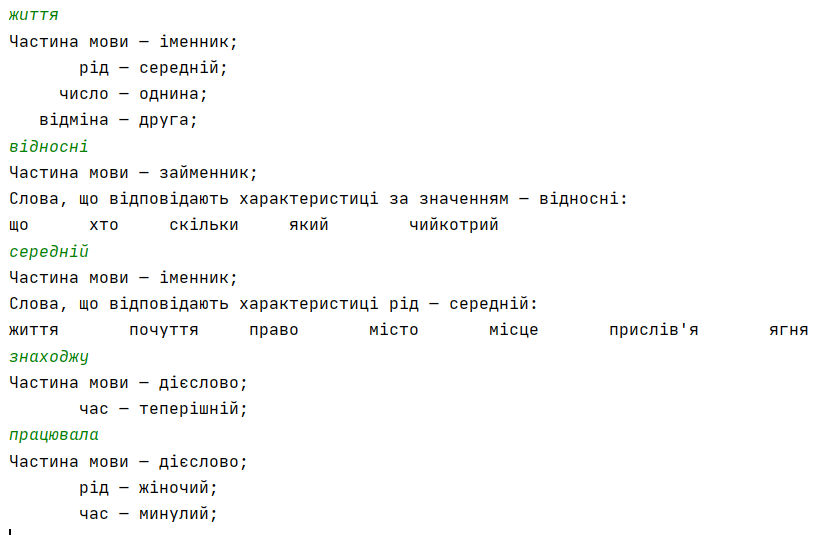
****

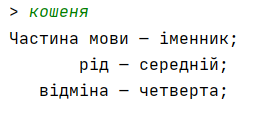
*редагування слів*

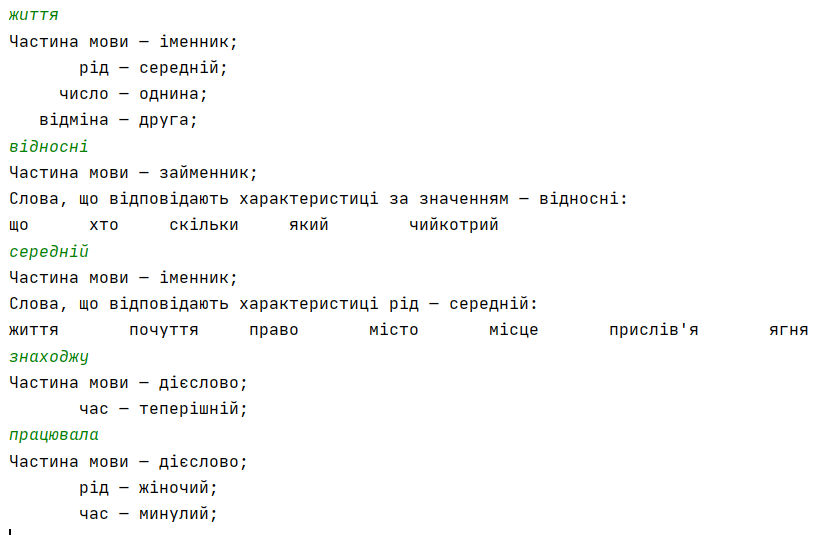
****

****

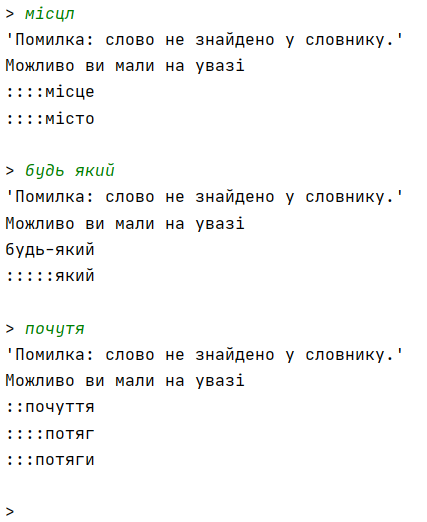
*додавання нових слів*****

*****виведення слів/характеристик*



****

*розумне виправлення помилок*

****

**Висновок**

<https://github.com/Ap3lsin4k/words-as-part-of-speech>

Автоматичні тести і в той же час документація по використанюю коду.

from bookmark\_entity import Bookmark  
from language\_interactor import UkrainianLanguageInteractor  
import pytest  
  
  
@pytest.fixture  
def use\_cases() -> UkrainianLanguageInteractor:  
 return UkrainianLanguageInteractor({  
 **'іменник'**: {  
 **'рід'**: {  
 **'середній'**:  
 (**"почуття"**,)  
 },  
 },  
 })  
  
  
def test\_extend\_with\_new\_words(use\_cases):  
 with pytest.raises(KeyError):  
 use\_cases.classify(**"добрий"**)  
  
  
def test\_extend\_should\_fail(use\_cases):  
 with pytest.raises(TypeError):  
 use\_cases.update({**"прикметник"**: {**"рід"**: (**"чоловічий"**,)}})  
  
  
def test\_002\_extend\_should\_fail(use\_cases):  
 with pytest.raises(TypeError):  
 use\_cases.update({**"прикметник"**: (**"рід"**,)})  
  
  
def test\_newly\_added\_piece\_of\_information(use\_cases):  
 with pytest.raises(KeyError):  
 use\_cases.classify(**"веселий"**)  
 use\_cases.update({**"прикметник"**: {**"рід"**: {**"чоловічий"**: (**"веселий"**,)}}})  
 result = use\_cases.classify(**"веселий"**)  
 assert result is not None  
 assert result == {**'прикметник'**: {**'рід'**: **'чоловічий'**}}  
  
  
def test\_2\_newly\_added\_words\_at\_the\_same\_time(use\_cases):  
 with pytest.raises(KeyError):  
 use\_cases.classify(**"гартувати"**)  
 with pytest.raises(KeyError):  
 use\_cases.classify(**"думати"**)  
 use\_cases.update({**"дієслово"**: {**"час"**: {**"теперішній"**: (**"гартувати"**, **"думати"**)}}})  
 result = use\_cases.classify(**"гартувати"**)  
 assert result is not None  
 assert result == {**"дієслово"**: {**"час"**: **"теперішній"**}}  
 result = use\_cases.classify(**"думати"**)  
 assert result is not None  
  
  
def test\_extend\_for\_two\_words\_but\_with\_different\_property(use\_cases):  
 with pytest.raises(KeyError):  
 use\_cases.classify(**"гартувати"**)  
 with pytest.raises(KeyError):  
 use\_cases.classify(**"думав"**)  
 use\_cases.update({**"дієслово"**: {**"час"**: {**"теперішній"**: (**"гартувати"**,), **"минулий"**: (**"думав"**,)}}})  
  
 result = use\_cases.classify(**"гартувати"**)  
 assert result is not None  
 assert result == {**"дієслово"**: {**"час"**: **"теперішній"**}}  
 result = use\_cases.classify(**"думав"**)  
 assert result == {**"дієслово"**: {**"час"**: **"минулий"**}}  
  
  
def test\_extend\_two\_parts\_of\_speech\_at\_the\_same\_time(use\_cases):  
 use\_cases.update({**"прикметник"**: {**"рід"**: {**"чоловічий"**: (**"веселий"**,)}},  
 **"дієслово"**: {**"час"**: {**"минулий"**: (**"думав"**,)}}})  
 result = use\_cases.classify(**"думав"**)  
 assert result is not None  
 result = use\_cases.classify(**"веселий"**)  
 assert result is not None  
  
  
def test\_show\_property\_when\_extended(use\_cases):  
 with pytest.raises(KeyError):  
 result\_none = use\_cases.get\_examples(**"минулий"**)  
 use\_cases.update({**"дієслово"**: {**"час"**: {**"минулий"**: (**"думав"**,)}}})  
 result = use\_cases.get\_examples(**"минулий"**)[0]  
 assert len(result) == 1  
 assert result[0] == **"думав"**def test\_update\_should\_not\_clean\_dict(use\_cases):  
 result\_before\_update = use\_cases.classify(**"почуття"**)  
 with pytest.raises(KeyError):  
 result\_none = use\_cases.classify(**"село"**)  
 use\_cases.update({**'іменник'**: {**'рід'**: {**'середній'**: (**'село'**,)}}})  
 result = use\_cases.classify(**'село'**)  
  
 assert **'рід'** in result\_before\_update[**'іменник'**]  
 assert **'середній'** == result\_before\_update[**'іменник'**][**'рід'**]  
 assert result == {**'іменник'**: {**'рід'**: **'середній'**}}  
  
  
def test\_should\_fail\_edit\_if\_key\_does\_not\_exist():  
 interactor = UkrainianLanguageInteractor({**'noun'**: {**'grammatical number'**: {**'plural'**: (**"travellers"**,)}}})  
 bm = Bookmark(**'noun'**, **'grammaticalnumber'**, **'plural'**)  
 with pytest.raises(KeyError):  
 interactor.modify(bm, **'travellers'**, **'travelers'**)  
  
 bm = Bookmark(**'noun'**, **'grammatical number'**, **'plural'**)  
 with pytest.raises(ValueError):  
 interactor.modify(bm, **'doed'**, **'did'**)  
  
  
*# advanced modification*def test\_edit\_newly\_added\_word\_in\_tuple():  
 interactor = UkrainianLanguageInteractor({**'noun'**: {**'grammatical number'**: {**'plural'**: (**"travellers"**,)}}})  
 result\_before\_update = interactor.get\_examples(**"plural"**)[0]  
 assert result\_before\_update == (**'travellers'**,)  
  
 bm = Bookmark(**'noun'**, **'grammatical number'**, **'plural'**)  
 interactor.modify(bm, **'travellers'**, **'travelers'**)  
 result = interactor.get\_examples(**"plural"**)[0]  
 assert result == (**'travelers'**,)  
  
 interactor.update({**'verb'**: {**'tense'**: {**'past'**: (**"doed"**,)}}})  
 result\_before\_modification = interactor.get\_examples(**"past"**)[0]  
 assert result\_before\_modification == (**"doed"**,)  
  
 interactor.modify(Bookmark(**'verb'**, **'tense'**, **'past'**),  
 **'doed'**, **'did'**)  
  
 result = interactor.get\_examples(**'past'**)[0]  
 assert result == (**'did'**,)

from language\_interactor import UkrainianLanguageInteractor  
import pytest  
  
ua = UkrainianLanguageInteractor({  
 **'іменник'**: {  
 **'рід'**: {  
 **'чоловічий'**:  
 (**'хлопець'**, **"потяг"**)  
 ,  
 **'жіночий'**:  
 (**'дівчина'**,),  
 **'середній'**:  
 (**"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**)  
  
 },  
 **'число'**: {  
 **'однина'**:  
 (**'хлопець'**, **"дівчина"**, **"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**, **"потяг"**),  
 **'множина'**:  
 (**"потяги"**, **"двері"**, **"штани"**, **"ножиці"**)  
 }  
 },  
 **'числівник'**: {  
 **'за значенням'**: {  
 **"кількісний"**:  
 (**"п'ять"**, **"двісті двадцять"**, **"шість"**, **"тридцять три"**, **"сорок вісім"**),  
 **"порядковий"**:  
 (**"четвертий"**, **"сьомий"**, **"десятий"**, **"сто двадцять перший"**)  
 }  
 }  
 })  
  
  
def test\_000\_characterize\_noun():  
 result = ua.classify(**"хлопець"**)  
 assert (**'іменник'** in result)  
 assert result[**'іменник'**] == {**'рід'**: **'чоловічий'**, **'число'**: **"однина"**}  
  
 result = ua.classify(**'дівчина'**)  
 assert result == {**'іменник'**: {**'рід'**: **'жіночий'**, **'число'**: **'однина'**}}  
  
  
def test\_001\_characterize\_numbers():  
 result = ua.classify(**"п'ять"**)  
 assert result == {**'числівник'**: {**'за значенням'**: **'кількісний'**}}  
  
  
def test\_should\_raise\_error\_for\_empty\_input():  
 with pytest.raises(ValueError):  
 ua.classify(**""**)  
 ua.classify(None)

from bookmark\_entity import Bookmark  
from language\_interactor import UkrainianLanguageInteractor  
import pytest  
  
  
@pytest.fixture  
def use\_cases() -> UkrainianLanguageInteractor:  
 return UkrainianLanguageInteractor({  
 **'іменник'**: {  
 **'рід'**: {  
 **'середній'**:  
 (**"почуття"**,)  
 },  
 **'відміна'**: {  
 **'перша'**:  
 (**"дівчина"**, **"ледащо"**, **"сирота"**, **"нероба"**, **"розбишака"**, **"бідолаха"**, **"староста"**),  
 **'друга'**:  
 (**"хлопець"**, **"потяг"**, **"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**, **"каменяр"**),  
 **'третя'**:  
 (**"нехворощ"**, **"любов"**, **"мати"**),  
 **'четверта'**:  
 (**"ягня"**,)  
 }  
 },  
  
 })  
  
  
def test\_should\_suggest\_the\_word\_itself\_when\_perfect\_match(use\_cases):  
 result = use\_cases.construct\_close\_matches(**'іменник'**)  
 assert **'іменник'** in result  
  
  
def test\_should\_suggest\_if\_typo\_at\_second\_level(use\_cases):  
 result = use\_cases.construct\_close\_matches(**'род'**)  
 assert **'рід'** in result  
 result = use\_cases.construct\_close\_matches(**'відмінок'**)  
 assert **'відміна'** in result  
  
  
def test\_should\_suggest\_when\_multiple\_parts\_of\_speech(use\_cases):  
 use\_cases.update({**'числівник'**: {  
 **'за значенням'**: {  
 **"кількісний"**:  
 (**"п'ять"**, **"двісті двадцять"**, **"шість"**, **"тридцять три"**, **"сорок вісім"**),  
 **"порядковий"**:  
 (**"четвертий"**, **"сьомий"**, **"десятий"**, **"сто двадцять перший"**)  
 }  
 }})  
  
 result = use\_cases.construct\_close\_matches(**'іменик'**)  
 assert **'іменник'** in result  
  
 result = use\_cases.construct\_close\_matches(**'числівник'**)  
 assert **'числівник'** in result  
  
 result = use\_cases.construct\_close\_matches(**'кількістний'**)  
 assert **'кількісний'** in result  
  
  
def test\_should\_suggest\_multiple\_words(use\_cases):  
 suggested = use\_cases.construct\_close\_matches(**"місцо"**)  
  
 assert **'місто'** in suggested  
 assert **'місце'** in suggested  
  
  
def test\_should\_not\_contain\_duplicates(use\_cases):  
 result = use\_cases.construct\_close\_matches(**"почутя"**)  
 assert **'почуття'** in result

import pytest  
from language\_interactor import UkrainianLanguageInteractor  
  
ua = UkrainianLanguageInteractor({  
 **'іменник'**: {  
 **'рід'**: {  
 **'чоловічий'**:  
 (**'хлопець'**, **"потяг"**)  
 ,  
 **'жіночий'**:  
 (**'дівчина'**,),  
 **'середній'**:  
 (**"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**)  
  
 },  
 **'число'**: {  
 **'однина'**:  
 (**'хлопець'**, **"дівчина"**, **"життя"**, **"почуття"**, **"право"**, **"місто"**, **"місце"**, **"прислів'я"**, **"потяг"**),  
 **'множина'**:  
 (**"потяги"**, **"двері"**, **"штани"**, **"ножиці"**)  
 }  
 },  
  
 })  
  
  
def test\_property\_example():  
 examples, bm = ua.get\_examples(**"середній"**)  
  
 assert examples[0] == **'життя'** assert examples[1] == **'почуття'** assert examples[2] == **'право'** assert bm.get\_part\_of\_speech() == **'іменник'** assert bm.category\_name == **'рід'** assert bm.property\_name == **'середній'** examples = ua.get\_examples(**'однина'**)[0]  
 assert len(examples) > 6