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 ІО-04
Федорко Андрій П.
Залікова книжка № 423

Перевірив:
Пономаренко Артем Миколайович

Київ – 2020

2. Мета лабораторної роботи та загальне завдання
3. Короткі теоретичні відомості, які відображають інформацію про модулі, пакети та елементи ООП, що були використані при написанні лабораторної роботи.
4. Скріншот вашого варіанту
5. Алгоритм (у довільній формі).
6. Роздруківка того фрагменту тексту програми, який написаний індивідуально чорними символами на білому фоні.
7. Скріншот результатів виконання програми з контрольним прикладом
8. Аналіз результатів та висновки.

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.

Відповідно до **номера у списку** вибрати індивідуальне завдання. Написати програму. Забезпечити ввід даних з клавіатури комп'ютера та друк результатів. При виводі використовувати форматування.

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 x з методу класу можна звернутися так: self.x. 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.

23	Створіть клас, який описує українську мову. Клас повинен містити багаторівневий словник з ключами – назвами частин мови. Кожна ключ-частина мови повинен мати значення у вигляді словника другого рівня і т. д. відповідно до класифікації, яка подана у Вікіпедії: https://uk.wikipedia.org/ wiki/Частини_мови. На найнижчому рівні словника повинні знаходитися кортежі зі словами. Методи класу повинні визначати всі характеристики слова як частини мови при його введенні, визначати приклади слів, які відповідають введеній характеристиці частини мови, методи вводу та модифікації слів. Створити об'єкт класу та ввести дані, використовуючи інформацію з Вікіпедії. Програма повинна виводити всі характеристики слова як частини мови при його введенні, виводити приклади слів, які відповідають введеній характеристиці частини мови, вводити та редагувати слова.

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_word
s_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.prope
rty_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", "у", "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", "у", "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:

```
Введіть "help", щоб подивитися більше інформації
> help
Введіть слово, щоб подивитися характеристику.
При введені характеристики, програма виведе приклад слів.
Ключове слово "new" без лапок, щоб додати нові слова у словник
Ключове слово "edit" без лапок, щоб відредагувати існуюче слово нові слова у словник
```

редагування слів

```
> цей
'Помилка: слово не знайдено у словнику.'

> сей
Частина мови – займенник;
за значенням – вказівні;

> edit
Введіть частину мови[числівник]: займенник
Введіть за чим класифікувати слово[за значенням]: за значенням
Введіть до якої характеристики належить[кількісний]: вказівні
Введіть поточне слово[єдин]: сей
Введіть нове слово[один]: цей
Замінити слово 'сей' на 'цей'.
Підтвердити(так/ні): так

> цей
Частина мови – займенник;
за значенням – вказівні;

> сей
'Помилка: слово не знайдено у словнику.'
Введіть слово, щоб подивитися характеристику.
При введені характеристики, програма виведе приклад слів.
```

додавання нових слів

```
> теля
'Помилка: слово не знайдено у словнику.'

> new
Введіть частину мови[прикметник]: іменник
Введіть за чим класифікувати слово[число]: відміна
Введіть до якої характеристики належить[множина]: четверта
Введіть слова розділені пробілом[зелена золотиста променista неймовірна]
>>> теля кошеня порося
Слова 'теля кошеня порося' будуть додані до словника, частина мови – 'іменник', відміна – четверта.
Підтвердити(так/ні): так
> теля
Частина мови – іменник;
   відміна – четверта;
```

```
> кошеня
Частина мови – іменник;
       рід – середній;
    відміна – четверта;
```

виведення слів/характеристик

```
життя
Частина мови – іменник;
       рід – середній;
     число – однина;
   відміна – друга;
відносні
Частина мови – займенник;
Слова, що відповідають характеристиці за значенням – відносні:
що      хто     скільки     який        чийкотрий
середній
Частина мови – іменник;
Слова, що відповідають характеристиці рід – середній:
життя       почуття     право       місто       місце       прислів'я       ягня


знаходжу
Частина мови – дієслово;
       час – теперішній;
працювала
Частина мови – дієслово;
       рід – жіночий;
       час – минулий;
```

розумне виправлення помилок

```
> місцл
'Помилка: слово не знайдено у словнику.'
Можливо ви мали на увазі
::::місце
::::місто

> будь який
'Помилка: слово не знайдено у словнику.'
Можливо ви мали на увазі
будь-який
:::::який

> почутя
'Помилка: слово не знайдено у словнику.'
Можливо ви мали на увазі
::почуття
::::потяг
:::потяги

>
```

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
```