**1. Импорт необходимых модулей**

Copy

import spacy

import PyPDF2

from tkinter import \*

import tkinter.filedialog

Этот блок кода импортирует все необходимые модули:

* **spacy** для морфологического и синтаксического анализа текста.
* **PyPDF2** для извлечения текста из PDF-файлов.
* **tkinter** и **tkinter.filedialog** для создания графического пользовательского интерфейса.

**2. Класс Lemma**

Copy

class Lemma:

def \_\_init\_\_(self, lemma,morphology="",role=""):

self.lemma:str = lemma

self.morphology:str = morphology

self.role:str = role

def \_\_iter\_\_(self):

return iter(self.lemma)

def set\_morphology(self, tag:str):

self.morphology = tag

def set\_role(self, role):

self.role = role

Этот класс представляет каждую лексему в тексте с атрибутами:

* **lemma**: Основная часть слова (например, "ходить").
* **morphology**: Морфологическая информация, такая как часть речи, падеж, число и т. д.
* **role**: Синтаксическая роль лексемы в предложении, например, подлежащее, сказуемое и т. д.

**3. Функция process\_text()**

Copy

def process\_text():

*# Извлечение текста из PDF*

with open(file\_path, 'rb') as f:

pdf\_reader = PyPDF2.PdfReader(f)

text = ""

for page in pdf\_reader.pages: text+=page.extract\_text()

*# Морфологический и Синтаксический анализ*

nlp = spacy.load('ru\_core\_news\_sm')

doc = nlp(text)

lemmas\_class: list[Lemma] = []

for token in doc:

lemmas\_class.append(Lemma(token.lemma\_, token.pos\_, token.dep\_))

lemmas\_class.sort(key=lambda x: x.lemma.lower())

*# Вывод результатов*

result\_list.delete(0, END)

for lemma in lemmas\_class:

result\_list.insert(END, f"{lemma.lemma} - {lemma.morphology} - {lemma.role}")

Эта функция выполняет следующие действия:

* Извлекает текст из PDF-файла по пути **file\_path**.
* Загружает предобученную модель обработки естественного языка (NLP) **ru\_core\_news\_sm** из **spacy**.
* Выполняет морфологический и синтаксический анализ текста и для каждой лексемы создает экземпляр класса **Lemma**.
* Сортирует лексемы по алфавиту и отображает их в списке **result\_list**.

**4. Функция open\_file()**

Copy

def open\_file():

global file\_path

file\_path = tkinter.filedialog.askopenfilename(filetypes=[("PDF files", "\*.pdf")])

if file\_path:

*# Извлечение текста из PDF*

with open(file\_path, 'rb') as f:

pdf\_reader = PyPDF2.PdfReader(f)

text = ""

for page in pdf\_reader.pages: text+=page.extract\_text()

*# Отображение извлеченного текста*

text\_input.delete('1.0', END)

text\_input.insert('1.0', text)

Эта функция открывает диалог выбора файла и позволяет пользователю выбрать PDF-файл для обработки. После выбора файла он извлекает текст из файла и отображает его в текстовом поле **text\_input**.

**5. Графический пользовательский интерфейс**  
Весь оставшийся код создает графический пользовательский интерфейс (GUI) с помощью **tkinter**:

* Создается корневое окно **root** с заголовком "Анализатор естественного языка для PDF".
* Добавляется кнопка "Открыть PDF" (**open\_file\_button**), которая вызывается при нажатии и открывает диалог выбора файла.
* Создается текстовое поле **text\_input** для отображения извлеченного текста.
* Добавляется кнопка "Обработать" (**process\_button**), которая запускает обработку текста.
* Создается список **result\_list** для отображения результатов обработки.

**1. Импорт необходимых модулей**

Copy

import difflib

import tkinter as tk

Этот код импортирует два модуля:

* **difflib** для вычисления редакционного расстояния.
* **tkinter** для создания графического пользовательского интерфейса (GUI).

**2. Функция edit\_distance()**

Copy

def edit\_distance(str1, str2):

m = len(str1) + 1

n = len(str2) + 1

t = *[[i + j for j in range(n)] for i in range(m)]*

for i in range(1, m):

c = i - 1

for j in range(1, n):

d = j - 1

t*[i][j]* = min(t*[c][j]* + 1, t*[i][d]* + 1, t*[c][d]* + (str1*[c]* != str2*[d]*))

return t*[m - 1][n - 1]*

Эта функция вычисляет редакционное расстояние между двумя строками **str1** и **str2** с использованием алгоритма Левенштейна. Редакционное расстояние измеряет минимальное количество правок (вставок, удалений или замен) для преобразования одной строки в другую.

**3. Графический интерфейс**

Copy

root = tk.Tk()

root.title("Редакционное расстояние")

Этот код создает корневое окно для графического интерфейса с заголовком "Редакционное расстояние".

**4. Загрузка и обработка текста**

Copy

text = load\_text("index.html")

words = get\_words(text)

Эти функции загружают текст из HTML-файла и разбивают его на список слов.

**5. Вычисление расстояний**

Copy

def calculate\_distances():

input\_word = input\_entry.get()

max\_distance = int(max\_distance\_entry.get())

results = []

for word in words:

distance = edit\_distance(input\_word, word)

if distance <= max\_distance:

results.append((word, distance))

results.sort(key=lambda x: x[1], reverse=True)

output\_text.delete("1.0", tk.END)

for word, distance in results:

output\_text.insert(tk.END, f"{word} ({distance})\n")

Эта функция вызывается при нажатии кнопки "Вычислить". Она получает входное слово и максимальное расстояние от пользователя, вычисляет редакционные расстояния для всех слов в тексте, фильтрует результаты по максимальному расстоянию и отображает отсортированные по расстоянию слова в области вывода.

**6. Сервисные функции**

Copy

def show\_help():

help\_window = tk.Toplevel(root)

help\_window.title("Помощь")

help\_text = tk.Text(help\_window)

help\_text.insert(tk.END, "Помощь\n\nВведите входное слово:\nВведите максимальное расстояние:\nПолучите перечень слов с расстоянием редактирования меньше или равным максимальному.")

help\_text.config(state="disabled")

help\_text.pack()

Эта функция отображает диалоговое окно справки, которое помогает пользователю понять, как использовать приложение.

**7. Визуальные компоненты**  
Остальной код создает визуальные компоненты GUI, такие как метки, поля ввода, кнопки и текстовое поле вывода, и организует их в сетку.

**1. Импорт необходимых модулей**

Copy

import tkinter as tk

from tkinter import filedialog

import spacy

from spacy import displacy

import docx2txt

Этот код импортирует все необходимые модули:

* **tkinter** и **filedialog** для создания графического пользовательского интерфейса (GUI).
* **spacy** для синтаксического анализа текста и визуализации зависимостей.
* **docx2txt** для извлечения текста из DOCX-файлов.

**2. Функция analyze\_syntax()**

Copy

def analyze\_syntax(text):

# Проанализировать текст

doc = nlp(text)

return doc

Эта функция принимает строку текста и выполняет синтаксический анализ с помощью модели spaCy, возвращая объект **spacy.tokens.Doc** с проанализированными зависимостями.

**3. Функция display\_syntax()**

Copy

def display\_syntax(doc):

*# Очистить поле вывода*

text\_box.delete("1.0", tk.END)

displacy.render(doc)

*# Добавить зависимости в поле вывода*

for token in doc:

text\_box.insert("end", f"{token.text} -> {spacy.explain(token.dep\_)} -> {token.head.text}\n")

text\_box.insert("end", "\n")

Эта функция отображает результаты синтаксического анализа в текстовом поле. Она визуализирует зависимости с помощью **spacy.displacy**, а затем отображает детальную информацию о каждом токене, включая его текст, зависимость и управляющий токен.

**4. Функция open\_file()**

Copy

def open\_file():

*# Открыть диалоговое окно "Открыть файл"*

file\_path = filedialog.askopenfilename(filetypes=[("DOCX files", "\*.docx")])

if file\_path:

*# Прочитать текст из файла*

text = docx2txt.process(file\_path)

*# Проанализировать синтаксис текста*

doc = analyze\_syntax(text)

*# Отобразить синтаксический анализ*

display\_syntax(doc)

Эта функция открывает диалоговое окно выбора файла, позволяя пользователю выбрать DOCX-файл для анализа. Затем он извлекает текст из файла с помощью **docx2txt** и передает его функции **analyze\_syntax()**, чтобы получить проанализированный объект **doc**. Наконец, он вызывает **display\_syntax()**, чтобы отобразить результаты в текстовом поле.

**5. Функция show\_help()**

Copy

def show\_help():

# Создать новое окно для руководства

help\_window = tk.Toplevel()

help\_window.title("Руководство")

# Добавить текст руководства

help\_text = "\*\*Руководство по программе\*\*\n\n" \

"1. Нажмите кнопку \"Открыть файл\", чтобы выбрать DOCX файл для анализа.\n" \

"2. Программа проанализирует синтаксис текста в файле.\n" \

"3. Результаты анализа будут отображены в поле вывода.\n" \

"4. Нажмите кнопку \"Помощь\", чтобы открыть это руководство.\n"

help\_label = tk.Label(help\_window, text=help\_text)

help\_label.pack()

Эта функция создает отдельное окно с текстовым полем, в котором отображается краткое руководство по использованию приложения.

**6. Графический пользовательский интерфейс (GUI)**

Copy

# Создать основное окно GUI

root = tk.Tk()

root.title("Анализ синтаксиса spaCy")

# Создать меню

menu\_bar = tk.Menu(root)

root.config(menu=menu\_bar)

# Создать меню "Файл"

file\_menu = tk.Menu(menu\_bar, tearoff=0)

file\_menu.add\_command(label="Открыть файл", command=open\_file)

file\_menu.add\_separator()

file\_menu.add\_command(label="Выход", command=root.destroy)

menu\_bar.add\_cascade(label="Файл", menu=file\_menu)

# Создать меню "Помощь"

help\_menu = tk.Menu(menu\_bar, tearoff=0)

help\_menu.add\_command(label="Помощь", command=show\_help)

menu\_bar.add\_cascade(label="Помощь", menu=help\_menu)

# Создать поле ввода для отображения результатов

text\_box = tk.Text(root)

text\_box.pack()

Этот код создает основное окно GUI с строкой меню, содержащей меню "Файл" и "Помощь". Меню "Файл" содержит команды для открытия файла и выхода из приложения, а меню "Помощь" вызывает функцию **show\_help()** для отображения руководства. В окне также есть текстовое поле для отображения результатов анализа синтаксиса.

**7. Запуск GUI**

Copy

# Запустить основное окно GUI

root.mainloop()

Этот код запускает основной цикл событий GUI, позволяя пользователю взаимодействовать с окном и его элементами.

**1. Импорт необходимых модулей**

Этот код импортирует все необходимые модули:

* **tkinter** и **filedialog** для создания графического пользовательского интерфейса (GUI).
* **spacy** для синтаксического и семантического анализа текста.
* **docx2txt** для извлечения текста из DOCX-файлов.

**2. Функция analyze\_semantics()**

Copy

def analyze\_semantics(text):

*# Проанализировать текст*

doc = nlp(text)

*# Найти отношения между сущностями*

matches = matcher(doc)

*# Добавить отношения в объект документа*

for match in matches:

span = doc[match[1][0]:match[1][1]]

span.set\_extension("RELATION",default=True,force=True)

doc.ents += (span,)

return doc

Эта функция принимает строку текста и выполняет синтаксический и семантический анализ с помощью модели spaCy. Она использует сопоставитель зависимостей для поиска отношений между сущностями, помеченными как существительные, и добавляет эти отношения в объект документа **doc**.

**3. Функция display\_semantics()**

Copy

def display\_semantics(doc):

*# Очистить поле вывода*

text\_box.delete("1.0", tk.END)

*# Добавить отношения в поле вывода*

for ent in doc.ents:

if ent.\_.get("RELATION"):

text\_box.insert("end", f"{ent.text}\n")

Эта функция отображает отношения, найденные в документе **doc**, в текстовом поле.

**4. Функция open\_file()**

Copy

def open\_file():

*# Открыть диалоговое окно "Открыть файл"*

file\_path = filedialog.askopenfilename(filetypes=[("DOCX files", "\*.docx")])

if file\_path:

*# Прочитать текст из файла*

text = docx2txt.process(file\_path)

*# Проанализировать семантику текста*

doc = analyze\_semantics(text)

*# Отобразить семантический анализ*

display\_semantics(doc)

Эта функция открывает диалоговое окно выбора файла, позволяя пользователю выбрать DOCX-файл для анализа. Затем он извлекает текст из файла с помощью **docx2txt** и передает его функции **analyze\_semantics()**, чтобы получить проанализированный объект **doc**. Наконец, он вызывает **display\_semantics()**, чтобы отобразить результаты в текстовом поле.

**5. Функция show\_help()**

Copy

def show\_help():

# Создать новое окно для руководства

help\_window = tk.Toplevel()

help\_window.title("Руководство")

# Добавить текст руководства

help\_text = "\*\*Руководство по программе\*\*\n\n" \

"1. Нажмите кнопку \"Открыть файл\", чтобы выбрать DOCX файл для анализа.\n" \

"2. Программа проанализирует семантику текста в файле.\n" \

"3. Результаты анализа будут отображены в поле вывода.\n" \

"4. Нажмите кнопку \"Помощь\", чтобы открыть это руководство.\n"

help\_label = tk.Label(help\_window, text=help\_text)

help\_label.pack()

Эта функция создает отдельное окно с текстовым полем, в котором отображается краткое руководство по использованию приложения.

**6. Графический пользовательский интерфейс (GUI)**

Этот код создает основное окно GUI с строкой меню, содержащей меню "Файл" и "Помощь". Меню "Файл" содержит команды для открытия файла и выхода из приложения, а меню "Помощь" вызывает функцию **show\_help()** для отображения руководства. В окне также есть текстовое поле для отображения результатов семантического анализа.

**7. Запуск GUI**

Этот код запускает основной цикл событий GUI, позволяя пользователю взаимодействовать с окном и его элементами.

1. Импорт необходимых модулей

Этот код импортирует все необходимые модули:

tkinter для создания графического пользовательского интерфейса (GUI).

spacy для анализа естественного языка.

matcher из spacy для поиска паттернов в документе spacy.

2. Загрузка модели spaCy

Copy

*# Загрузить модель spaCy для русского языка*

nlp = spacy.load("ru\_core\_news\_md")

Этот код загружает предобученную модель spaCy для русского языка.

3. Создание сопоставителя для поиска намерений

Copy

matcher = Matcher(nlp.vocab)

Этот код создает сопоставитель для поиска паттернов, соответствующих намерениям пользователя.

4. Определение паттернов намерений

Copy

patterns = [{"LEMMA": {"IN": [\*responses.keys()]}}]

Для поиска намерений определяются паттерны, которые соответствуют леммам (основным формам слов) намерений из словаря ответов responses.

5. Определение функции analyze\_intent()

Copy

def analyze\_intent(text):

*# Проанализировать текст*

doc = nlp(text)

print(doc)

*# Найти намерение пользователя*

matches = matcher(doc)

*# Получить намерение из первого совпадения (если оно есть)*

intent = ""

if len(matches) > 0 and len(matches[0])>2:

intent = doc[matches[0][1]].lemma\_

return intent

Эта функция принимает ввод текста и выполняет анализ текста с помощью модели spaCy. Затем она использует сопоставитель для поиска совпадений с паттернами намерений и возвращает лемму намерения, если оно найдено.

6. Определение функции generate\_response()

Copy

def generate\_response(intent):

*# Получить ответ из словаря ответов на основе намерения*

response = responses.get(intent, "Я не понимаю вашего запроса.")

return response

Эта функция принимает намерение пользователя и возвращает соответствующий ответ из словаря ответов responses.

7. Определение функции send\_message()

Copy

def send\_message(event):

*# Получить сообщение пользователя*

user\_message = text\_box.get("1.0", "end-1c")

*# Проанализировать намерение пользователя*

intent = analyze\_intent(user\_message)

*# Сгенерировать ответ системы*

response = generate\_response(intent)

*# Очистить поле ввода*

text\_box.delete("1.0", "end")

*# Отобразить ответ системы*

chat\_history.insert("end", f"\*\*Система:\*\* {response}\n")

Эта функция обрабатывает нажатие клавиши Enter в поле ввода сообщения пользователя. Она получает сообщение пользователя, анализирует намерение, генерирует ответ и отображает его в поле вывода чата.

8. Графический пользовательский интерфейс (GUI)

Остальной код создает окно GUI с полем ввода для сообщения пользователя, полем вывода для отображения чата и кнопкой "Отправить" для отправки сообщения.