МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информатика»

Тема “Основные управляющие конструкции. Wikipedia API”

Студентка гр. 1384 _____ Логинова А. Ю.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург

2021

Цель работы.

Применить знания основных конструкций Python в разработке, научиться работать с модулем Wikipedia.

Задание.

Используя вышеописанные инструменты, напишите программу, которая принимает на вход строку вида *название_страницы_1, название страницы_2, ... название_страницы_n, сокращенная_форма_языка* и делает следующее:

1. Проверяет, есть ли такой язык в возможных языках сервиса, если нет, выводит строку "no results" и больше ничего не делает. В случае, если язык есть, устанавливает его как язык запросов в текущей программе и выполняет еще два действия:

2. Ищет максимальное число слов в кратком содержании страниц "название_страницы_1", "название страницы_2", ... "название_страницы_n", выводит на экран это максимальное количество и название страницы (т.е. её **title**), у которой оно обнаружилось. Считается, что слова разделены пробельными символами. Если максимальных значений несколько, выведите последнее.

3. Строит список-цепочку из страниц и выводит полученный список на экран. Элементы списка-цепочки - это страницы "название_страницы_1", "название страницы_2", ... "название_страницы_n", между которыми может быть одна промежуточная страница или не быть промежуточных страниц.

Предположим, нам на вход поступила строка *(данный пример актуализирован к состоянию страниц wikipedia на 2021 год)*:

Айсберг, IBM, ru

В числе ссылок страницы с названием "Айсберг", есть страница с названием , которая содержит ссылку на страницу с названием "1959 год", у

которой есть ссылка на страницу с названием "IBM" -- это и есть цепочка с промежуточным звеном в виде страницы "1959 год".

Гарантируется, что существует или одна промежуточная страница или ноль: т.е. в числе ссылок первой страницы можно обнаружить вторую.

Цепочка должна быть кратчайшей, т.е. если существуют две цепочки, одна из которых содержит промежуточную страницу, а вторая нет, стройте цепочку без промежуточного элемента.

Пример входных данных:

```
Айсберг, IBM, ru
```

Пример вывода:

```
115 IBM
['Айсберг', '1959 год', 'IBM']
```

Первая строка содержит решение подзадачи №2, вторая - №3.

Важное уточнение: каждую подзадачу (1, 2, 3) оформите в виде отдельных функций.

Функции должны быть "чистыми". Мы с этим определением ближе познакомимся в *разделе №3 на лекциях*, на данный момент следует выполнить требования:

1. Ваши функции не должны выводить что-либо на экран (только возвращать результат)

2. Ваши функции не должны изменять глобальные переменные (те переменные, которые существуют вне функции, то есть во внешней программе)

3. Ваши функции не должны изменять и свои аргументы, которые передаются в функцию (лучше возвращать измененную копию аргумента).

Выполнение работы.

На вход программе подается строка, которая делится на подстроки элементом “, ” и присваивается переменной *input_data*. Последним элементом получившегося списка является сокращенная форма языка, которая присваивается переменной *language* и проверяется на наличие в Wikipedia с помощью функции *is_language_valid*, в которую помещается переменная *lang*. Из функции возвращается *True* или *False*. Если такой язык существует, он устанавливается как язык запросов в текущей программе, если нет, то программа завершает работу, выводя на экран *no results*.

Список названий страниц *input_data* передается в функцию *get_max_summary*, где вычисляется максимальная длина слов в кратком содержании страниц. Функция возвращает переменные *max_summary*, *name*, которым были присвоены максимальная длина краткого содержания и название страницы с данной длиной. Данные выводятся на экран через переменные *max_s, title*. Работа функции заключается в переборе каждого названия страницы, вычислении его размера и отборе максимального.

Далее список передается в функцию *add_missed_titles*, откуда возвращается новый дополненный список с недостающими промежуточными названиями страниц *new_titles_list*, и выводится на экран. Данная функция реализована с помощью цикла *while*, который проходится по всем элементам списка с помощью индекса *i* и пропускает последний элемент, т.к. работа функции основана на сравнении элемента со следующим элементом, соответственно, последний элемент не с чем сравнивать. В новый список последний элемент добавляется после прохождения всех итераций цикла.

В цикле в новый список добавляется *i*-ый элемент. Если в списке ссылок этого *i*-ого элемента находится ссылка на следующий элемент исходного списка, дальнейшие действия пропускаются, т.к. найден кратчайший путь и между *i* и *i+1* элементом ничего не должно быть. Если ссылки на следующий

элемент нет, находится первая промежуточная ссылка среди всех ссылок *i*-ого элемента, которая и добавляется в новый список.

Разработанный программный код приводится в приложении А.

Тестирование.

№ п/п	Входные данные	Выходные данные	Комментарии
1	Айсберг, IBM, ru	115 IBM [“Айсберг”, “1959 год”,”IBM”]	Программа работает корректно
2	Айсберг, IBM, russian	no results	Программа не найдет язык, если в нем есть ошибка или была указана полная форма языка.

Выводы.

Были изучены и исследованы основные конструкции Python, функции модуля Wikipedia. Была разработана программа, считывающая информацию пользователя с клавиатуры, для обработки которой использовались условные операторы if/else, были написаны функции для отлавливания несуществующих страниц и языков, а также находилась взаимосвязь между страницами из Wikipedia.

ПРИЛОЖЕНИЕ А.

Исходный код программы.

Название файла: main.py

```
import wikipedia


def is_page_valid(title):
    try:
        wikipedia.page(title)
    except Exception:
        return False
    return True


def is_language_valid(lang):
    if lang in wikipedia.languages():
        wikipedia.set_lang(language)
        return True
    return False


def get_max_summary(titles):
    max_summary, name = 0, 0
    for title in titles:
        if not is_page_valid(title):
            return 'page "{}" not found'.format(title)
        lengths = len(wikipedia.page(title).summary.split())
        if max_summary <= lengths:
            max_summary = lengths
            name = title
    return max_summary, name


def add_missed_titles(titles):
    new_titles_list = []
    i = 0
    while i + 1 < len(titles):
        new_titles_list.append(titles[i])
        links = wikipedia.page(titles[i]).links
        if titles[i + 1] not in links:
            for link in links:
                new_links = wikipedia.page(link).links
                if titles[i + 1] in new_links:
                    new_titles_list.append(link)
                    break
        i += 1
    new_titles_list.append(titles[-1])
    return new_titles_list


input_data = input().split(', ')
language = input_data.pop(-1)
if not is_language_valid(language):
    print('no results')
```

```
else:
    data = get_max_summary(input_data)
    max_s, title = data
    print(max_s, wikipedia.page(title).title)
    print(add_missed_titles(input_data))
```