МИНОБРНАУКИ РОССИИ

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

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информатика»

Тема: Основные управляющие конструкции. Wikipedia API

Студент(ка) гр. 1384 _____ Усачева Д.В.

Преподаватель _____ Шевская Н. В.

Санкт-Петербург

2021

Цель работы.

Ознакомление с управляющими конструкциями языка Python на примере использующей их программы.

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: <u>каждую подзадачу (1, 2, 3) оформите в виде отдельных функций.</u>
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.

Для выполнения задачи были реализованы функции:

1) *is_page_valid(page)* – функция, которая определяет существует ли страница, переданная в качестве аргумента. Если страница существует. Функция возвращает True, в противном случае — False.

2) *language(langg)*- функция, предназначенная для выполнения задачи 1. Она принимает на вход строку *langg(от англ. language)*, являющуюся сокращенным названием языка, и проверяет есть ли такой язык в возможных языках сервиса. Если нет, выводит строку «no results» и возвращает значение *False*. Если язык есть, устанавливает его как язык запросов в текущей программе и возвращает значение *True*.

3) *count_w(name)*- функция, предназначенная для выполнения задачи 2. Она принимает на вход массив *name*, содержащий названия страниц, среди которых необходимо найти максимальное число слов в кратком содержании страницы и название этой страницы. Для сохранения и последующего возвращения имени страницы, содержащей максимальное количество слов, и самого количества мы создаем переменные *tit(от англ. title)(изначально пустая строка)* и *mx(от англ. maximum)(число, равное 0).* В цикле, который исполняется *n*-ное число раз (где *n* –это количество элементов массива *name)*, функция находит количество слов краткого содержания для каждой страницы и записывает это значение в переменную *c_w(от англ. word count).*

Далее при помощи условного оператора мы сравниваем количество слов краткого содержания n-ой страницы- *c_w* и *mx, если c_w* окажется не меньше *mx*, то обновим значение переменной *mx*, заголовок этой страницы *name_n.title* сохраняется в переменную *tit*. По завершении цикла функция возвращает имя страницы, содержащей максимальное количество слов, и само количество.

4) *chain(name)*- функция, предназначенная для выполнения задачи 3. Она принимает на вход массив *name*, содержащий названия страниц, из которых необходимо построить список-цепочку. Элементы списка-цепочки - это цепочки между которыми может быть одна промежуточная страница или не быть промежуточных страниц. Сначала в данной функции создается массив *chain_title*, в котором находится нулевой элемент массива name (название нулевой страницы). Далее в цикле, который исполняется (*i-1)*-ое число раз (где *i* –это количество элементов массива *name)*, проверяется найдется ли *i+1-ый* элемент массива name среди ссылок *i*-ой страницы- *name_i.links*. Если нашелся: в массив *chain_title* добавляется имя *(i+1)*-ой страницы массива *name*. Если нет: при помощи цикла переменная *j* последовательно принимает значения - названия страниц, ссылки на которые содержит страница *name_i* . Функция *is_page_valid(j)* определяет существует ли страница *j*. При возвращении *True* продолжается проверка страницы *j*. Проверяется найдется ли *i+1-ый* элемент массива name среди ссылок страницы *j*. Если найдется, то в массив *chain_title* добавляется имя страницы *j*, а также имя *(i+1)*-ой страницы массива *name*. По завершении работы функция возвращает массив *chain_title*, содержащий построенную цепочку страниц.

5) В основной программе в первую очередь считывается строка, которая сохранится в переменную *name_str***.** Далее эта строка разбивается по

«, » при помощи метода .split() **.** В переменную *lang* записывается последний элемент получившегося массива *name_str*, который позже удаляется из него при помощи среза. В переменную *b(от англ. bool)* записываем значение, возвращаемое при вызове *language(lang).* Если *b=True*, выводятся результаты работы функций *count_w(name_str)* (для вывода которой был использован оператор звёздочки) и *chain(name_str).* В противном случае основная программа выводит «no results» и завершает работу.

Разработанный программный код см. в приложении 1

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 - Пример тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	Айсберг, IBM, english	no results	Проверка работы программы при неверном вводе языка
2.	IBM, Compaq, Кенеха, he		
3.	Айсберг, IBM, ru	115 IBM ['Айсберг', '1959 год', 'IBM']	Проверка работы программы при корректном вводе данных (есть промежуточная страница)

Выводы

Была проведена работа с управляющими конструкциями языка Python, изучена библиотека Wikipedia API, а также написана программа, позволяющая проверить есть ли заданный язык в возможных языках сервиса, если нет, вывести строку «no results, если язык есть, устанавить его как язык запросов в текущей программе, искать и возвращать название страницы, содержащей максимальное количество слов в кратком содержании, и само количество, возвращать список-цепочку с промежуточными страницами, которые содержат ссылку на последующую страницу.

ПРИЛОЖЕНИЕ 1
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Usacheva_Daria_lb1 /src/main.py

```
import wikipedia
from wikipedia import page, languages, set_lang

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

def language(langg):
    if langg not in languages():
        return False
    else:
        set_lang(langg)
        return True

def count_w(name):
    mx = 0
    tit = ''
    for n in range(len(name)):
        name_n = page(name[n])
        c_w = len(name_n.summary.split())
        if c_w >= mx:
            tit = name_n.title
            mx = c_w
    return mx, tit


def chain(name):
    chain_title = [name[0]]
    for i in range(len(name) - 1):
```

```
        name_i = page(name[i])
        if not (name[i + 1] in name_i.links):
            for j in name_i.links:
                if is_page_valid(j):
                    name_i_links = page(j)
                    if name[i + 1] in name_i_links.links:
                        chain_title.append(j)
                        chain_title.append(name[i + 1])
                        break
        else:
            chain_title.append(name[i + 1])
            continue
    return chain_title


name_str = input()
name_str = name_str.split(', ')
lang = name_str[-1]
name_str = name_str[:-1:]
b = language(lang)
if b:
    print(*count_w(name_str))
    print(chain(name_str))
else:
    print("no results")
```