Парсерные программы для сайтов с ранжированными списками приемных комиссий ВУЗов

Руководство программиста

Содержание

1 Введение

1.1 Назначение

Данное руководство посвящено программированию ПО «Парсерные программы для сайтов с ранжированными списками приемных комиссий ВУЗов» на языке Python.

1.2 Общие сведения

Данное ПО предназначено для получения csv-таблиц с ранжированными списками приемных комиссий ВУЗов.
В процессе работы программ создается запрос по заданному url-адресу веб-страницы с таблицей, извлекаются данные и формируется csv-файл с полученными данными в указанной папке.

Осуществляется автоматизация следующих процессов:

— получение таблиц с заданной веб-страницы;

— загрузка полученной таблицы в файл;

— сохранение файла в заданную пользователем папку;

— получение таблиц с множества ссылок, загрузка в файлы, систематизация файлов по папкам;

— фильтрация данных в таблицах;

— реализация взаимодействия пользователя с программой через простой и понятный интерфейс;

— после успешного завершения работы программы вызывается соответствующее информационное сообщение;

— в случае возникновения ошибки вызывается соответствующее информационное сообщение;

— при попытке закрыть программу вызывается соответствующее уточняющее информационное сообщение.

При разработке программ использовались следующие библиотеки:

— requests - исполнение HTTP-запросов;
— pandas - работа с данными, размещенными в таблице;
— pathlib - работа с директориями;
— BeautifulSoup - извлечение данных из файлов HTML и XML;
— tkinter - создание оконных интерфейсов;
— datetime - получение даты и времени системы;
— os - работа с установленной ОС и файловой системой ПК;
— fake_useragent - замена пользовательского агента;

2 Программа «parser»

2.1 Интерфейс (оконное приложение)

Механизм работы приложения следующий:

1. Пользователь вводит адрес сайта, с которого необходимо спарсить данные, в текстовое поле под надписью «Введите адрес ссылки».
2. Пользователь вводит название файла, который создаст парсер и загрузит в него полученные данные, в текстовое поле под надписью «Введите название файла».
3. Пользователь вводит адрес папки, куда парсер должен сохранить созданный файл с данными, в текстовое поле под надписью «Введите путь».
4. Пользователь нажимает на кнопку «Спарсить».
5. После успешного завершения работы программы всплывает информационное окно с сообщением «Файл с данными сформирован!». Файлы с таблицами будут находится в папке, которую указал пользователь. В случае некорректного ввода каких-либо данных будет выведено соответствующее сообщение об ошибке.

Для реализации оконного приложения использованы методы библиотеки Tkinter.

Окно приложения содержит три элемента Label (надписи), триэлемента Entry (текстовые поля ввода) и один элемент Button (кнопка):

Определение оконного приложения, его размеров и размеров отступов по горизонтали и вертикали:

```
1   window = Tk()
2   window.title('Parser' )
3   window.geometry('600x300' )
4   frame = Frame(
5     window,
6     padx=10,
7     pady=10)
```

Определение надписи Label «Введите адрес ссылки»:

```
1   url_lb = Label(
2       frame,
3       text ="Введите адрес ссылки")
4   url_lb.grid(row=4,column=2)
```

Аналогично надписи Label «Введите название файла» и «Введите путь».

Размещение текстового поля Entry для ввода url-адреса:

```
1   url_tf = Entry(
2   frame)
3   url_tf.grid(row=5, column=2, pady = 4)
```

Аналогично текстовые поля Entry для ввода названия файла и для ввода адреса папки.

Размещение кнопку Button «Спарсить», при нажатии на которую вызвается функция парсинга:

```
1   pars_btn = Button(
2   frame,
3   text='Спарсить' ,
4   command=parser)
5   pars_btn.grid(row=7, column=2)
```

Вызов окна для взаимодействия с пользователем:

```
1   window.mainloop()
```

При закрытии окна вызывается информационное сообщение:

```
1   def on_closing(): # оконное приложение
2         if messagebox.askokcancel("Выход", "Вы хотите выйти?"):
3               window.destroy()
```

```
4   window.protocol("WM_DELETE_WINDOW", on_closing)
```

2.2 Функция формирования названия файла и адреса папки name_addres()

При первом запуске программы пользователь должен ввести адрес папки, куда нужно сохранить сформированный файл, и имя этого файла. Они автоматически сохраняются в файле file.txt. При последующих вызовах парсера, если пользователь ничего не ввел, имя файла и адрес директории извлекаются из файла. Если пользователь ввел хотя бы одно из этих данных, они обновляются в файле.

Также к названию файла добавляются текущие дата и время.

Реализация:

```
1  def name_addres(): # Формирование названия и адреса
2       s = "/Users/mariibykova/Desktop/file.txt"
3       infile = open(s, 'r+') #считали файл file.txt
4       lines = infile.readlines()
5       if lines==[]:
6             lines=["",""]
7       name1 = lines[0].strip() #записанно в файле file.txt
8       addr1 = lines[1].strip() #путь, записанный в файле file.txt
9       infile.close()
10      infile = open(s,'w')
11      infile.close()
12      infile = open(s, 'r+') #очистили файл file.txt
13      name = name_tf.get()   #название введено пользователем
14      adres = adres_tf.get() #путь, введенный пользователем
15      if name == "": #если пользователь не ввел название
16         name = name1
17      else:
18         name1=name
19      if adres == "":#если пользователь не ввел путь
20         adres = addr1
21      else:
22         addr1=adres
23      lines = [name1, addr1]
24      for  line in lines: #сохраняем данные в файл file.txt
25         infile.write(line + '\n')
```

```
26      return lines
27  # получение текущей даты и времени
28  real_data = datetime.now()
29  data = "___" + str(real_data.day) + "_" + str(real_data.month) +
    "_" + str(real_data.year) + "___"
30  data +=  str(real_data.hour) + "_" + str(real_data.minute) + «_»
    + str(real_data.second) + «___"
31  # выгрузка имени и адреса
32  a = []
33  a = name_addres()
34  name = a[0]
35  adres= a[1]
```

2.3 Парсерная функция parser()

Парсерная функция получает адрес url-ссылки, делает запрос и получает HTML-код страницы. После этого по тегу «table» находится таблица, по тегу «th» определяются столбцы таблицы, по которым определяется размер будущей таблицы. Дальше по тегу «tr» вычленяется строка таблицы, которая делится на ячейки (тег «td»). Содержимое каждой ячейки сохраняется в массив данных, который потом выгружается в фрейм. Полученный фрейм экспортируется в csv-файл. Если на странице несколько таблиц, они делятся на разные файлы. После успешного завершения работы выводится информационное сообщение. Реализация:

```
1   UserAgent().chrome
2   s1 = '.csv' # строка с расширением
3   page = requests.get(url, headers={'User-Agent':
    UserAgent().chrome})
4   soup = BeautifulSoup(page.text, 'lxml') # получаем html-код
5   table1 = soup.find('table') # поиск первой таблицы
6   k = 0 # счетчик таблиц
7   mydata1 = pd.DataFrame() # определяем фрейм данных
8   add = []
9   rows = [] # список для строк таблиц
10  for table1 in soup.find_all('table',): # пока на странице есть
    таблицы
11          headers = [] # пустой список для колонок таблицы
12          for i in table1.find_all('th'): # пока есть колонки
13              title = i.text
```

```
14              headers.append(title) # добавляем колонку в список
15          mydata = pd.DataFrame(columns = headers) #
определяем в фрейме столбцы
16          mydata3 = pd.DataFrame(columns = headers)
17          k = 0
18          row = ""
19          for j in table1.find_all('tr')[1:]: # строка внутри тега <tr>
20              row_data = j.find_all('td') # элементы внутри тега <td>
21              row = [i.text for i in row_data] #получаем и сохраняем
22              rows.append(row) # добавляем строку в массив
23          k +=1
24          # записываем все в фрейм
25          mydata = pd.DataFrame(rows, columns=[""]*max(len(i) for i
            in rows))
26          add.append(mydata)
27          s = name+s1 # формирование строки с названием
28          mydata.to_csv(s, index=False) # экспорт данных
29    # если на странице больше одной таблицы, то делим их в
разные файлы
30          s = ''
31          k = 1
32          if (len(add)>1):
33              for i in add:
34                  s =name+str(k)+s1
35                  k += 1
36                  i.to_csv(s, index=False)
37    # сообщение об успешном завершении работы парсера
38    messagebox.showinfo('ParserINFO', f'Файл с данными
      сформирован!')
39    url_tf.delete(0, END)
40    name_tf.delete(0, END)
41    adres_tf.delete(0, END)
```

Проверка возможных ошибок:

```
1       if (url == ""):
2           messagebox.showinfo('ParserINFO', f'Не введен адрес
ссылки!')
3           return
4       if (name == ""):
```

```
5           messagebox.showinfo('ParserINFO', f'Не введено
название файла!’)
6           return
7       if (adres == ""):
8           messagebox.showinfo('ParserINFO', f'Не введен путь!')
9           return
```

3 Программа «parser_all»

3.1 Интерфейс (оконное приложение)

Механизм работы приложения следующий:

1. Пользователь вводит адрес директории, куда парсер должен сохранить созданный архив данными, в текстовое поле под надписью «Введите путь».

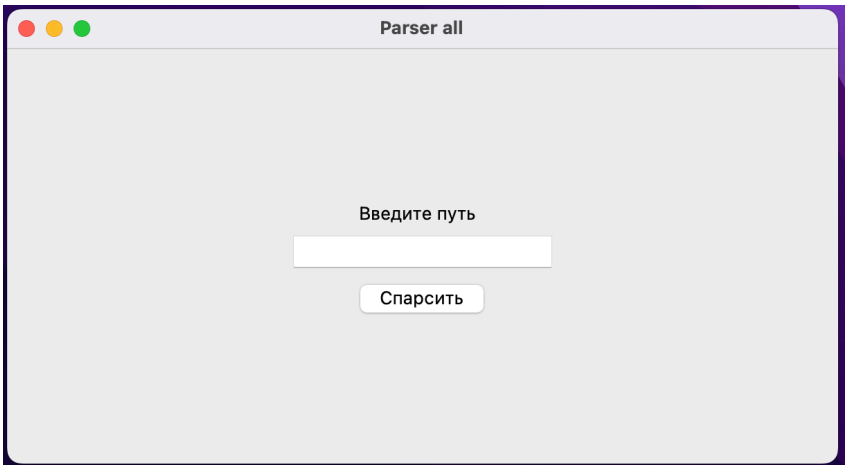
2. Пользователь нажимает на кнопку «Спарсить».

3. После успешного завершения работы программы всплывает информационное окно с сообщением «Файл с данными сформирован!». Файлы с таблицами будут находится в папке, которую указал пользователь. В случае некорректного ввода будет выведено соответствующее сообщение об ошибке.

Набор ссылок для парсера находится в файле file1.txt.

Для реализации оконного приложения использованы методы библиотеки Tkinter.

Окно приложения содержит один элемент Label (надпись), элемент Entry (текстовое поле ввода) и один элемент Button (кнопка):



Определение оконного приложения, его размеров и размеров отступов по горизонтали и вертикали:

```
1   window = Tk()
2   window.title('Parser all' )
3   window.geometry('600x300' )
4   frame = Frame(
5     window,
6     padx=10,
7     pady=10)
```

Определение надписи Label «Введите путь»:

```
adres_lb = Label( # надпись
    frame,
    text="Введите путь  ",)
adres_lb.grid(row=1, column=1)
```

Размещение текстового поля Entry для ввода url-адреса:

```
url_tf = Entry(
frame)
url_tf.grid(row=2, column=1, pady = 4)
```

Размещение кнопку Button «Спарсить», при нажатии на которую вызвается функция парсинга:

```
pars_btn = Button(
frame,
text='Спарсить' ,
command=parser)
pars_btn.grid(row=3, column=1)
```

Вызов окна для взаимодействия с пользователем:

```
window.mainloop()
```

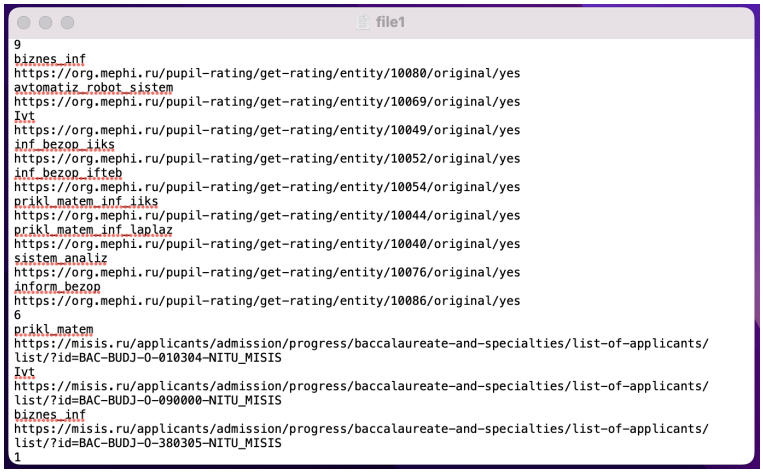
При закрытии окна вызывается информационное сообщение:

```
def on_closing(): # оконное приложение
    if messagebox.askokcancel("Выход", "Вы хотите выйти?"):
        window.destroy()
window.protocol("WM_DELETE_WINDOW", on_closing)
```

3.2 Выгрузка данных - функция parser12()

Данная функция получает адрес директории, в которую нужно сохранить папку со спарсенными данными. Имя папки представляет собой дату и время формирования этого архива. Внутри папки создаются папки по каждому ВУЗу, в них сохраняются сформированные csv-файлы.

Список всех url-ссылок, названий направлений и коды ВУЗов находятся в файле file1.txt. Структура файла:



После успешного завершения работы выводится информационное сообщение. Реализация:

```
1   def parser12():
2       adres = adres_tf.get() #путь, введенный пользователем
3       # получение текущей даты и времени
4       real_data = datetime.now()
5       data = str(real_data.day) + "_" + str(real_data.month) + "_" +
      str(real_data.year) + "___"
6       data +=  str(real_data.hour) + "_" + str(real_data.minute) + «_"
       + str(real_data.second) + «___"
7       path = os.path.join(adres, data)
8       os.chdir(adres) # переходим в нужную директорию
9       os.mkdir(data)  # создаем в ней папку
10      adres+= "/"+data +"/" # дополняем путь
11      s = «/Users/mariibykova/Desktop/file1.txt"
12      infile = open(s, 'r+') #считали файл file1.txt
13      lines = infile.readlines()
14      i =0
15      os.chdir(adres) # переход в директорию по адрес
16      while(i<len(lines)): # проход по файлу
# если встретили цифру - смена университета, иначе парсим
 17         if(lines[i].strip() == "1"):
 18                 os.mkdir("sgtu")
 19                 adres1 = adres+"/sgtu"
 20                 i+=1
 21          elif (lines[i].strip()=="4"):
 22                 os.mkdir("sgu")
 23                 adres1 = adres+"/sgu"
```

```
24              i+=1
25                      < . . . >
26      else:
27              name = lines[i].strip()
28              url = lines[i+1].strip()
29              parser(name, url,adres1)
30              i +=2
31  # сообщение, что успешно завершена работа парсера
32  messagebox.showinfo('ParserINFO', f'Файл с данными
сформирован!')
33  adres_tf.delete(0, END)
```

3.3 Парсерная функция parser()

Парсерная функция работает так же, как и в программе «parser» (см. п. 2.3).

4 Программа «parser_snils»

Программа «parser_snils» создана на основе программы «parser_all» с добавлением проверки наличия СНИЛСа абитуриента среди СНИЛСов из файла sinls.txt. Таким образом, программа сохраняет в файлы только те таблицы, в которых есть хотя бы один абитуриент, СНИЛС которого встречается в файле snils.txt.

4.1 Глобаланые переменные

Программа загружает в массив СНИЛСы с файла snils.txt. Так формат таблиц в каждом ВУЗе разный, объявляется переменная key, которая для каждого ВУза будет хранить номер столбца со СНИЛС.

4.2 Интерфейс (оконное приложение)

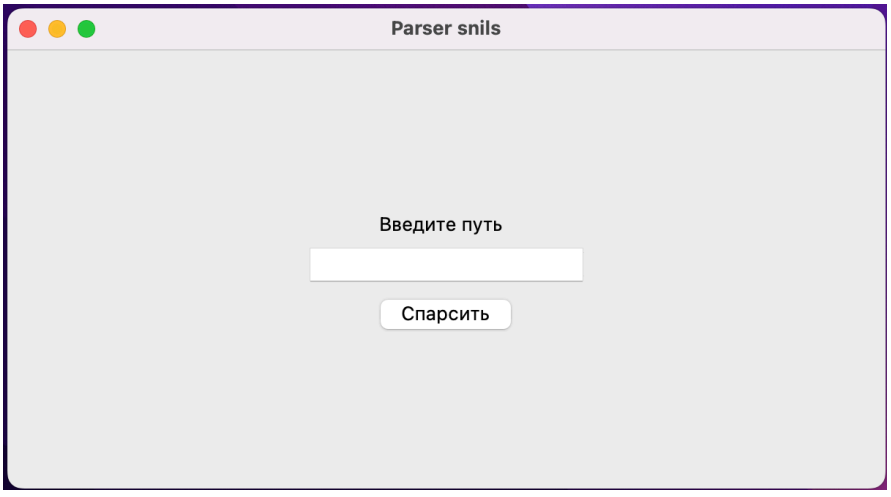
Механизм работы приложения следующий:

1. Пользователь вводит адрес директории, куда парсер должен сохранить созданный архив данными, в текстовое поле под надписью «Введите путь».
2. Пользователь нажимает на кнопку «Спарсить».
3. После успешного завершения работы программы всплывает информационное окно с сообщением «Файл с данными сформирован!». Файлы с таблицами будут находится в папке, которую указал пользователь. В случае некорректного ввода будет выведено соответствующее сообщение об ошибке.

Набор ссылок для парсера находится в файле file1.txt.

Для реализации оконного приложения использованы методы библиотеки Tkinter.

Окно приложения содержит один элемент Label (надпись), элемент Entry (текстовое поле ввода) и один элемент Button (кнопка):



Определение оконного приложения, его размеров и размеров отступов по горизонтали и вертикали:

```
1  window = Tk()
2  window.title(‘Parser snils' )
3  window.geometry('600x300' )
4  frame = Frame(
5    window,
6    padx=10,
7    pady=10)
```

Определение надписи Label «Введите путь»:

```
1  adres_lb = Label( # надпись
2       frame,
3       text="Введите путь  ",)
4  adres_lb.grid(row=1, column=1)
```

Размещение текстового поля Entry для ввода url-адреса:

```
1  url_tf = Entry(
2  frame)
3  url_tf.grid(row=2, column=1, pady = 4)
```

Размещение кнопку Button «Спарсить», при нажатии на которую вызвается функция парсинга:

```
1  pars_btn = Button(
2  frame,
3  text='Спарсить' ,
4  command=parser)
5  pars_btn.grid(row=3, column=1)
```

Вызов окна для взаимодействия с пользователем:

```
1  window.mainloop()
```

При закрытии окна вызывается информационное сообщение:

```
1  def on_closing(): # оконное приложение
2        if messagebox.askokcancel("Выход", "Вы хотите выйти?"):
3              window.destroy()
4  window.protocol("WM_DELETE_WINDOW", on_closing)
```

4.3 Выгрузка данных - функция parser12()

Функция работает так же, как и для программы «parser_all». Но для каждого ВУЗа дополнительно указывается значение переменной key.

4.4 Парсерная функция parser()

Функции передается дополнительный аргумент - значение переменной key.

Также перед выгрузкой данных в фрейм проводится проверка наличие спарсенного СНИЛСа среди СНИЛСов из файла «snils.txt». Реализация:

```
1    # если строка непустая проверяем снилс в данных
2    if (len(row)>=key+1):
3    for i in range(len(row[key])):
4          if (row[key][i] in "0123456789"):
5                str1234 += str(row[key][i])
6    str1234 = str1234.replace(" ", "")
7    # если снилс есть, то сохраняем строку таблицы в фрейм
8    if (str1234 in snils):
9          rows.append(row)
```

В остальном, функция реализована так же, как в программе «parser_all» (см. п. 3.3)