Лабораторная работа №6

Разработка программ, передающих и получающих данные по протоколу HTTP.

Цель работы: научиться разбирать URL-адрес и строку запроса, разрабатывать программы, передающие и получающие данные по протоколу HTTP, с помощью стандартных и сторонних модулей

Задание на лабораторную работу:

1. Изучить теоретические сведения.
2. Написать программу в соответствии с вариантом.

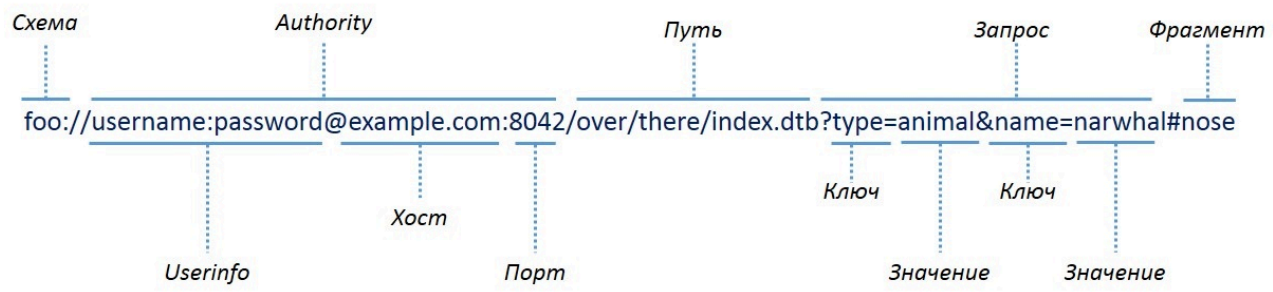
Теоретические сведения

В состав стандартной библиотеки Python входит множество модулей, позволяющих работать практически со всеми протоколами Интернета и решающих в том числе наиболее часто встречающиеся задачи:

- разбор URL-адреса и строки запроса на составляющие,
- преобразование гиперссылок,
- обмен данными по протоколу HTTP.

URI (*англ.* **U**niform **R**esource **I**dentifier) - унифицированный (единообразный) идентификатор ресурса.

Пример URI приведен на рисунке. Обязательными атрибутами являются: имя, схема, путь, и Userinfo в случае авторизации.



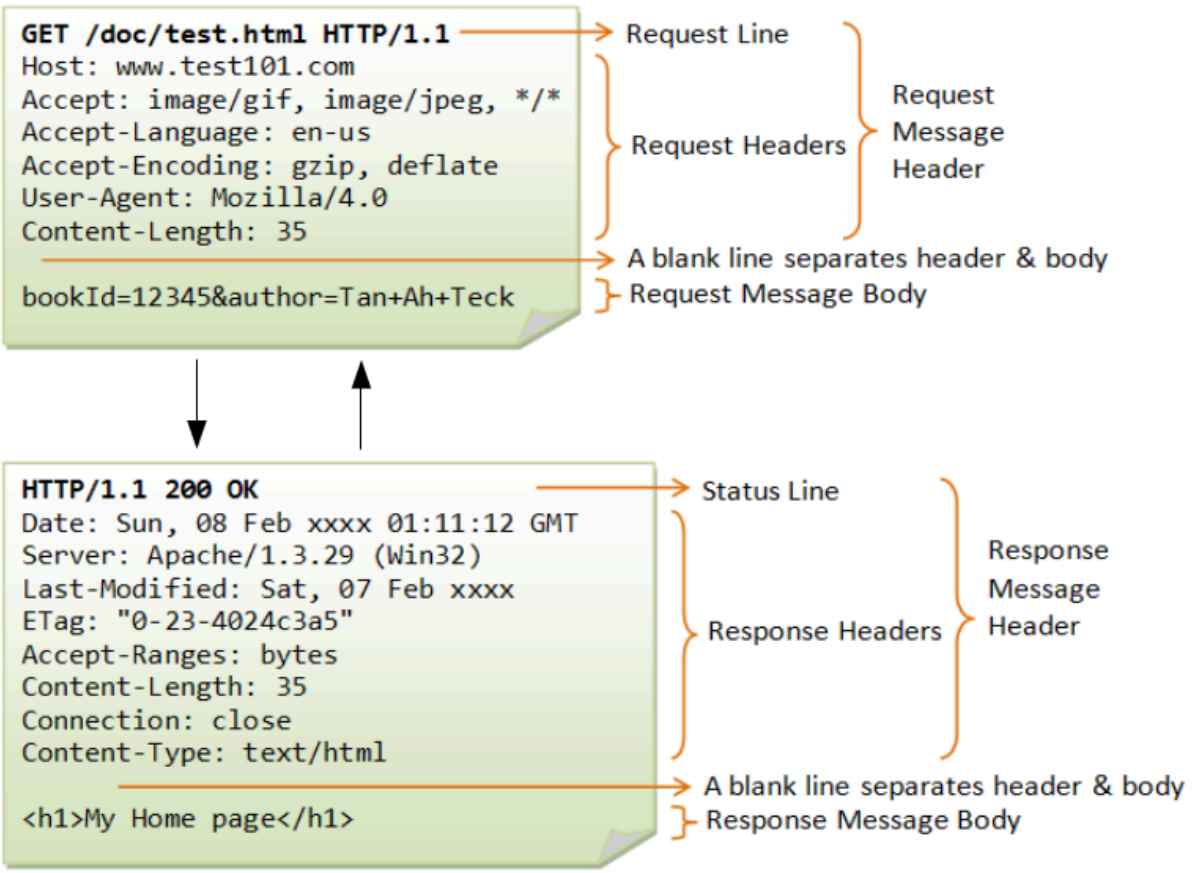
Как HTTP-запрос (*англ.* Request), так и HTTP-ответ (*англ.* Response) имеют следующий формат:

1. Стартовая строка (*англ.* Starting Line) — определяет тип сообщения (обязательна);
2. Заголовки (*англ.* Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения (могут отсутствовать);
3. Пустая строка;
4. Тело сообщения (*англ.* Message Body) — непосредственно данные сообщения (например, код HTML-страницы или файл).

Основные HTTP-заголовки и их предназначение:

- GET - заголовок запроса при передаче данных методом GET;
- POST - заголовок запроса при передаче данных методом POST;
- Host - название домена;
- Accept - MIME-типы, поддерживаемые Web-браузером;
- Accept-Language - список поддерживаемых языков в порядке предпочтения;
- Accept-Charset - список поддерживаемых кодировок;
- Accept-Encoding - список поддерживаемых методов сжатия;
- Content-Type - тип передаваемых данных;
- Content-Length - длина передаваемых данных при методе POST;
- Cookie - информация об установленных cookies;

- Last-Modified - дата последней модификации файла;
- Location - перенаправление на другой URL-адрес;
- Pragma - заголовок, запрещающий кэширование документа в протоколе HTTP/1.0;
- Cache-Control - заголовок, управляющий кэшированием документа в протоколе HTTP/1.1;
- Referer - URL-адрес, с которого пользователь перешел на наш сайт;
- Server - название и версия программного обеспечения Web-сервера;
- User-Agent - информация об используемом Web-браузере.



Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP. Он представляет собой целое число из трёх десятичных цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза

• 1xx: Informational (информационные)
• 2xx: Success (успешно):
  – 200 OK («хорошо»)
  – 201 Created («создано»)
  – 202 Accepted («принято»)
• 3xx: Redirection (перенаправление)
• 4xx: Client Error (ошибка клиента):
  – 400 Bad Request («неверный запрос»);
  – 404 Not Found («не найдено»)
• 5xx: Server Error (ошибка сервера):
  – 500 Internal Server Error («внутренняя ошибка сервера»)
  – 501 Not Implemented («не реализовано»)
  – 502 Bad Gateway («ошибочный шлюз»)

– 504 Gateway Timeout («шлюз не отвечает»)

Разобрать URL-адрес на составляющие позволяет **функция urlparse()** стандартного модуля **urllib.parse**.

Возвращаемое значение представляет собой именованный кортеж, что означает, что к его элементам можно получить доступ по индексу или как к именованным атрибутам:

Атрибут	Индекс	Обозначение	Значение по умолчанию
scheme	0	Спецификатор схемы URL	параметр схемы
netloc	1	Часть сетевого расположения	пустой строкой
path	2	Иерархический путь	пустой строкой
params	3	Параметры для последнего элемента пути	пустой строкой
query	4	Компонент запроса	пустой строкой
fragment	5	Идентификатор фрагмента	пустой строкой
username		Имя пользователя	None
password		Пароль	None
hostname		Имя хоста (нижний регистр)	None
port		Номер порта как целое число, если присутствует	None

!!!! urlparse распознает netloc, только если он правильно введен с помощью '//'. Иначе предполагается, что ввод является относительным URL-адресом и, таким образом, начинается с компонента пути.

```
>>> urlparse('www.cwi.nl/%7Eguido/Python.html')
ParseResult(scheme='', netloc='',
path='www.cwi.nl/%7Eguido/Python.html', params='', query='',
fragment='')
>>> urlparse('help/Python.html')
ParseResult(scheme='', netloc='', path='help/Python.html', params='',
query='', fragment='')
```

!!!! Обратите внимание на то, что ***значение параметра <Запрос>*** **(query)** возвращается в виде строки. Строка запроса является составной конструкцией, содержащей пары *параметр=значение*. Все специальные символы внутри названия параметра и значения кодируются последовательностями **%nn**. Например, для параметра str, имеющего значение *"Строка"* в кодировке Windows-1251, строка запроса будет выглядеть так:

```
str=%Dl%F2%F0%EE%EA%E0
```

Если строка запроса содержит несколько пар *параметр=значение*, то они разделяются символом **&**. Добавим параметр v со значением 10:

```
str=%Dl%F2%F0%EE%EA%E0&v=10
```

В строке запроса может быть несколько параметров с одним названием, но разными значениями,- например, если передаются значения нескольких выбранных пунктов в списке с множественным выбором:

```
str=%Dl%F2%F0%EE%EA%EG&v=10&v=20
```

Выполнить обратную операцию (собрать URL-адрес из отдельных значений) позволяет функция **urlunparse** (<Последовательность>).

Вместо функции urlparse() можно воспользоваться функцией **urlsplit** (<URL-адрес>[, <Схема> [, <Разбор якоря>]]).

Ее отличие от urlparse() проявляется в том, что она не выделяет из интернет-адреса параметры.

Выполнить обратную операцию (собрать URL-адрес из отдельных значений) позволяет функция **urlunsplit** (<Последовательность>).

Модуль **http.client** определяет классы, реализующие клиентскую сторону протоколов HTTP и HTTPS.

АВТОМАТИЧЕСКИЙ СБОР ДАННЫХ В ИНТЕРНЕТЕ (Web Scraping, веб-скрейпинг) - получения веб-данных путем извлечения их со страниц веб-ресурсов.

Автоматический сбор данных с веб-страниц можно упростить с помощью модулей Python:

- **webbrowser** поставляется вместе с Python и предназначен для открытия браузера на определенной веб-странице.
- **requests** загружает файлы и веб-страницы из Интернета.
- **BeautifulSoup** предназначен для синтаксического анализа кода HTML-языка, на котором написаны веб-страницы.
- **Selenium** запускает веб-браузер и управляет его работой; способен заполнять формы и имитировать щелчки мышью в браузере.

Функция **open () модуля webbrowser** запускает браузер с использованием указанного URL-адреса.

Пример программы с использованием модуля webbrowser

Написать программу, которая открывает (указывает) в приложении Google Maps адрес, содержащийся в буфере обмена.

Если загрузить в браузер страницу http://maps.google.com/ и выполнить поиск по интересующему почтовому адресу, то URL-адрес, отображаемый в адресной строке браузера, будет выглядеть примерно так:

```
Поиск адреса: 870 Valencia St, San Francisco, CA

https://www.google.com/maps/place/870+Valencia+St/@37.7590311,
122.4215096,17z/data=!3m1!4b1!4m2!3m1!1s0x808f7e3dadc07a37:0xc86b0b2bb93b73d8
```

URL-адрес содержит почтовый адрес, но, кроме него, включает также дополнительный текст. Веб-сайты часто вставляют дополнительные данные в URL-адреса, которые используются для отслеживания привычек посетителей или адаптации сайта к запросам пользователей. Если исключить эти данные и выполнить в браузере переход по упрощенному адресу
https://www.google.com/maps/place/870+Valencia+St+San+Francisco+CA/,
то по-прежнему отображается нужная страница. Следовательно, достаточно настроить программу на страницу https://www.google.com/maps/place/*ваша_адресная_строка*, где *ваша_адресная_строка* — это почтовый адрес, в соответствии с которым должна быть открыта карта.

```
import webbrowser, pyperclip
# Получение почтового адреса из буфера обмена.
address = pyperclip.paste()
webbrowser.open('https://www.google.com/maps/place/' +  address)
```

requests - это HTTP-библиотека Python, основная цель разработки которой - сделать HTTP-запросы более простыми и удобными.

Примеры разных видов запросов (GET, POST, HEAD, PUT, DELETE):

```
>>> r = requests.get('https://api.github.com/events') # GET-запрос
>>> r = requests.post('https://httpbin.org/post', data =
{'key':'value'})
>>> r = requests.head('https://httpbin.org/get')
```

```
>>> r = requests.put('https://httpbin.org/put', data =
{'key':'value'})
>>> r = requests.delete('https://httpbin.org/delete')
```

Метод **raise_for_status()** — это эффективный способ гарантированной остановки программы в случае неудачной загрузки.
Данный метод возбуждает исключение, если в процессе загрузки файла произошла ошибка, и не совершает никаких действий в случае успешной загрузки.
Можно обернуть строку кода raise_for_status() конструкцией try/except, чтобы обработать эту ошибку, не допуская аварийной остановки программы.

```
res= requests.get ('http://invalid')
try:
    res.raise_for_status ()
except Exception as exc:
    print('Возникла проблема: %s' % (exc))
#в результате выведет:
Возникла проблема: 404 Client Error: Not Found
```

Целесообразно всегда вызывать метод raise_for_status() после функции requests.get(). Это позволяет убедиться в том, что загрузка действительно была осуществлена, прежде чем предоставить программе возможность дальнейшего выполнения.

Ответ на запрос GET содержит информацию. Она находится в теле сообщения и называется пейлоад (payload). Используя атрибуты и методы библиотеки Response, можно получить пейлоад в различных форматах.

Для того, чтобы получить содержимое запроса в байтах, необходимо использовать **.content**

```
>>> r = requests.get('https://api.github.com/events') # GET-запрос
>>> r.content
b'[{"repository":{"open_issues":0,"url":"https://github.com/...
```

Однако, зачастую требуется конвертировать полученную информацию в строку в кодировке UTF-8. response делает это при помощи **.text**.
В случае успешного запроса загруженная страница сохраняется в виде строки в переменной text объекта Response. Т.е. можно прочитать содержимое ответа сервера.

```
>>>res = requests.get
('http://www.gutenberg.org/cache/epub/1112/pg1112.txt')
В переменной res.text хранится полный текст пьесы в виде одной
длинной строки; результат вызова len(res.text) говорит о том, что эта
строка содержит более 178 000 символов.
>>>len(res.text)
178981
>>>print (res.text [: 250])
Вызов print (res.text [: 250]) отображает лишь первые 250 символов.
```

Записать веб-страницу в файл можно с помощью цикла for по возвращаемому значению метода **iter_content ()** объекта Response. Метод iter_content () возвращает порции содержимого на каждой стадии цикла. Каждая порция данных — это данные байтового типа, и нужно самому указать, сколько байтов должна содержать каждая порция.

Алгоритм процесса загрузки и сохранения файла:

1. Вызов функции requests.get () для загрузки файла.
2. Вызов функции open () с аргументом 'wb' для создания нового файла в режиме записи двоичных данных.
3. Цикл по возвращаемому значению метода iter_content () объекта Response.
4. Вызов метода write () на каждой итерации для записи содержимого файла.
5. Вызов метода close () для закрытия файла.

Пример1: скачать содержимое страницы
http://www.gutenberg.org/cache/epub/1112/pg1112.txt в файл 1.txt порциями по 100000 байт.

```
>>> import requests
>>> res =
requests.get('http://www.gutenberg.org/cache/epub/1112/pg1112.txt’)
>>> res.raise_for_status()
>>> playFile = open(‘1.txt', ‘wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)
100000
78983
>>> playFile.close()
```

Метод write () возвращает количество байтов, записанных в файл (по умолчанию 1 байт). В примере первая порция включала 100000 байт, для оставшейся части файла понадобилось только 78983 б.
В рабочем каталоге появился файл RomeoAndJuliet.txt.

Пример2:

```
>>> import requests
>>> url = 'https://learn.python.ru/media/projects/sl1_Cj4bKxp.png'
>>> im = requests.get(url, stream=True)
>>> with open('new.png', 'wb') as f:
...       f.write(im.content)
```

По умолчанию stream=False, это сделано для того, чтобы препятствовать загрузке больших объемов данных, т.к. мы можем не знать размеры скачиваемых объектов. После выполнения данного скрипта, мы увидим новое изображение под названием new.png в директории, из которой у нас запущен скрипт.

-

Задания

Задание 1

B1-7: Дан список адресов. Заменить все названия доменов (netloc) на 'www.test.com', используя метод `._replace()`. Метод возвращает новый объект ParseResult, заменяя указанные поля новыми значениями.

```
Например, список:
//www.cwi.nl:80/%7Eguido/Python.html
http://stackoverflow.com/search?q=question
http://www.example.org/default.html?ct=32&op=92&item=98
Получили:
//www.admin.ru/%7Eguido/Python.html
http://www.admin.ru/search?q=question
http://www.admin.ru/default.html?ct=32&op=92&item=98
```

B8-13: Дан список URL-адресов. Из этого списка составить новый список, который может содержать только один url из одного уникального домена.

```
Например, дан список
urls = ['http://www.mail.ru/1.html',
'https://pythonru.com/uroki/biblioteka-pygame-chast-1-vvedenie',
'https://pythonru.com/uroki/biblioteka-pygame-chast-2-rabota-so-sprajtami',
'https://news.mail.ru/politics/51550058/?frommail=1',
'news.mail.ru/society/51554979/?frommail=1']
Этот пример списка содержит несколько URL-адресов в одном домене:
https://pythonru.com
news.mail.ru
В новый список должен попасть только один из этих URL.
```

Задание 2

Дан список URL-адресов, вывести все значения параметров запроса.

Список:

```
['https://news.mail.ru/politics/51550058/?fr=1',
'http://example.com/?q=abc&p=123',
'https://learn.python.ru/media?name1=muuuuu&name2=999&name3=goodby']
Должны получить:  1 abc 123 muuuuu 999 goodby
```

Задание 3

Дан файл, в котором находятся 5 адресов страниц, содержащих изображения (для примера можно использовать изображения с сайта https://www.hdwallpapers.in).

Написать программу для открытия всех ссылок в отдельных вкладках браузера.

Сохранить все эти изображения в папку, указанную пользователем. Организовать блочную загрузку файлов посредством метода iter_content по 8 Кб.

Задание 4

Написать программу, которая будет анализировать сайт ggpk.by и выводить сообщение есть ли расписание на следующий день.

Задание 5

Написать программу, которая выполняет парсинг страницы. Вывести на экран информацию.

B1	https://www.gastronom.ru/section/shashliky	названия статей и их ссылки, названия пунктов меню

B2	https://7745.by/podarki/podarki-dlya-podrostkov	названия всех категорий подарков и их ссылки, текст статьи «Что подарить?»
B3	https://imarket.by/	Названия товаров из раздела «Лучшее на сегодня» и их цену, названия статей из раздела «Советы и обзоры»
B4	https://sputnik.by/education/	вывести заголовки новости, ссылки на страницу с подробностями, дату публикации новости
B5	https://stopgame.ru/review/izumitelno	названия обзоров, ссылки на страницу с подробностями, количество комментариев
B6	https://grodno.in/afisha/kino/	названия фильмов, ссылку на страницу с подробностями, дату (до которой они идут)
B7	https://tech.onliner.by/	вывести заголовки новости, ссылки на страницу с подробностями, дату и время публикации новости
B8	https://abit.grsu.by/	вывести названия разделов («Нормативные документы», «Итоги приема прошлых лет» и т.д.), названия всех специальностей из раздела «В программисты я б пошел - пусть меня научат!», ссылки на страницу с подробностями
B9	https://afisha.relax.by/all/grodno/	вывести заголовки разделов, названия фильмов, ссылки на страницу с подробностями
B10	https://vse-kursy.by/read/130-15-luchshih-shkol-angliyskogo-yazyka-v-grodno.html	названия всех школ, ссылки на страницу с подробностями, названия пунктов меню
B11	https://grodno.in/pool/	названия бассейнов и их адрес, текст статьи «Что нужно взять с собой в бассейн»
B12-13	https://museums.by/muzei/muzei-g-grodno-i-grodnenskoy-oblasti/	названия музеев, краткую информацию о них, ссылки на страницу с подробностями

Контрольные вопросы:

1. Что такое URI, URL?
2. Из каких частей состоят HTTP-запрос и ответ.
3. Перечислите основные функции модуля urllib.parse
4. Для чего предназначен метод raise_for_status()
5. Для чего предназначены модули webbrowser, requests, BeautifulSoup