Язык Python. Часть 2

Лекция 3

Простейший пример

```
1 import requests
2
3 response = requests.get('https://docs.python.org/3/library/i
4 print(response)
5 print(type(response))
6 print(dir(response))
```

Что происходит

- Импортируем модуль requests
- Из его пространства имен вызываем функцию get
- Передаем ей URL
- Функция открывает соединение
- И посылает GET-запрос

Что происходит

- Получает ответ
- Разбирает его
- Формирует объект класса `requests.models.Response`
- Возвращает его

Посмотрим на статус

```
1 import requests
2
3 response = requests.get('https://docs.python.org/3/library/i
4 print(response.status_code)
5 print(response.reason)
```

Посмотрим на заголовки

```
1 import requests
2
3 response = requests.get('https://docs.python.org/3/library/i
4 for key, value in response.headers.items():
5     print("header key", key)
6     print("header value", value)
7     print()
```

Наиболее интересные для нас

- Content-length - длина тела запроса
- Content-type - тип данных в теле запроса
- Last-modified - время последнего изменения

Посмотрим на содержимое

```
1 import requests
2
3 response = requests.get('https://docs.python.org/3/library/i
4 print(response.content)
```

Что хранит свойство contents

- Тип данных - bytes
- Можем захотеть преобразовать в строку
- Но только, если Content-type это предполагает
- А если все-таки хотим - нужно определить кодировку

Что хранит свойство contents

- Работу по определению кодировки
- И по самому декодированию
- Берет на себя библиотека
- Плоды ее работы - в свойстве text

Посмотрим на содержимое

```
1 import requests
2
3 response = requests.get('https://docs.python.org/3/library/i
4 print(response.text)
```

Что можем быть в теле ответа

- Бинарные данные
- Текстовое содержимое
  - Оно тоже в бинарном виде
  - Преобразуется с помощью специальных правил
  - Набор правил называется кодировкой

Текстовое содержимое

- Популярная кодировка - UTF-8
- https://en.wikipedia.org/wiki/UTF-8
- Использованная кодировка указывается в заголовках
- Обработкой кодировки занимается requests
- А интерпретацией текста - не занимается

Текстовое содержимое

- HTML
- Просто текст (plain text)
- XML
- JSON
- CSS
- Нас в первую очередь интересуют HTML и JSON

Бинарное содержимое

- Картинки
- Видео
- Аудио
- Архивы файлов
- PDF

Сохраним бинарное содержимое

```
1 import requests
2
3 response = requests.get('https://www.python.org/ftp/python/3
4 with open('Python-3.11.1.tar.xz', 'wb') as f:
5     f.write(response.content)
```

А если содержимое большое ?

- Тогда - проблема
- requests будет читать все тело
- И формировать bytes-объект
- Даже если это 100Gb
- И даже если 1Gb - хотелось бы что-то типа progress-bar

Еще проблемы ?

- Если нет интернета - получим исключение
- Если удаленный узел не работает - тоже
- И если адрес неверный
- А если ошибка в пути ?
- Скорее всего - будет ответ с кодом ошибки

Обработка проблем

- Добавим обработчик исключений
- Добавим проверку статуса
- Используем потоковый режим
- По крайней мере - если ожидаем длинного ответа

Основные исключения

- requests.ConnectionError - проблемы с соединени
- requests.URLRequired - что-то не так с URL
- requests.HTTPError - ошибочный код, преобразованный в исключение
- requests.ConnectTimeout - соединение вроде ест слишком медленное
- https://requests.readthedocs.io/en/latest/api/#excep

Проверка статуса (свойства и методы класса Response)

Работа с длинными данными

- Вызываем `response.get(url, stream=True)`
- Получаем объект Response
- Можем почитать заголовки (header)
- Можем узнать статус
- Но `text` нет
- И `content` - тоже нет

Работа с длинными данными

- Можем получить объект с "файловым" интерфейсом
- И читать данные из него
- С помощью методов вроде `read`
- Прочитанное - например, писать в файл

Работа с длинными данными

- Есть чуть более высокоуровневая альтернатива
- Прямо в response методы `iter_content` и `iter_lines`
- `iter_content` - бинарное чтение
- Задаем количество
- Отличие от `raw.read()` - некоторые преобразования данных

Работа с длинными данными

- `iter_lines` - текстовое чтение
- Удобно для очень длинных текстовых файлов
- Предполагается, что строки - разумной длины
- Иначе рискуем выйти за границы памяти

Работа с длинными данными

- Можем использовать заголовок Content-length
- В нем хранится размер тела ответа
- Можно организовать progress-bar
- Из Content-length знаем общую длину
- Из результатов чтений знаем длину прочитанного

Тонкий нюанс

- Статус идет в начале ответа
- Потом идут заголовки
- И длинное тело
- Которое мы можем не суметь прочитать
- На что может быть масса причин

Тонкий нюанс

- Сервер не прочитал часть данных
- Или сломался интернет
- То есть статус 200 - не гарантия полного успеха
- То есть в потоковом режиме быть готовым к ошибкам чтения
- И их тоже обрабатывать

Ограничения на использование requests

- Формальные и неформальные
- Одиночное скачивание - ok
- Множественное скачивание создает нагрузку на сервер
- Потенциально - значительно бОльшую, чем браузер
- Проявляем понимание, не гонимся за скоростью

Ограничения на использование requests

- Открытость на чтение не дает права на любое использование
- Не надо скачивать все данные ресурса и распространять их как датасет
- Не забываем про пользовательские соглашения
- Особенно если работаем под логином

Статические ресурсы

- Сами данные меняются нечасто
- Хранятся на сервере как файлы
- Можно считать, что есть какой-то каталог в файловой системе сервера
- В нем эти файлы лежат
- Путь в URL - это путь к файлу, начиная от этого каталога

Статические ресурсы

- http://lib.ru/POEEAST/GOMER/gomer01.txt
- На сервере есть некий каталог
- В нем подкаталог POEEAST
- А в нем - GOMER
- А там - файл gomer01.txt
- Не обязательно все буквально так - но это очень популярный вариант

Статические ресурсы: ограничения

- Хорошо подходит для интернет-библиотек
- Для новостных сайтов - уже хуже
- Но как-то еще можно
- Совсем никак - форумы, соцсети и т.п.

Динамические ресурсы

- На стороне сервера работает код
- Код порождает HTML-документ
- Например, страничку дискуссии
- Или заголовки новостей по данной теме
- На основании содержимого баз данных и т.п.

Динамические ресурсы

- Пример: https://www.google.com/search?q=python
- Путь - это идентификатор программы, порождающей ресурс
- А у программы могут быть параметры
- Они передаются как дополнительная часть URL
- В данном случае - это `?q=python`

Динамические ресурсы

- '?' отделяет путь от области параметров
- Область параметров - последовательность пар
- Первый элемент пары - ключ
- Второй элемент пары - значение
- Пары отделяются символом &
- Интерпретация ключей и значений - дело сервера

Параметры запроса

- Ключ и значение отделяются знаком =
- Пары отделяются символом &
- Если хочется спецсимвол сделать частью ключа или значения
- Нужно использовать '%' и ASCII-код
- Интерпретация ключей и значений - дело сервера

Параметры запроса в requests

- Можно сформировать строку запроса
- И включить в нее параметры
- Но это однотипная рутинная работа
- И еще надо заботиться о спецсимволах
- Есть вариант поудобнее

Используем именованный параметр `params`

```
1 import requests
2
3 response = requests.get('https://google.com', params={'q': '
4 print(response.status_code)
```

Параметры запроса в requests

Параметр с несколькими значениями

```
1 import requests
2
3 response = requests.get('https://some-server.com', params={'
4 print(response.url)
```

Смешанный вариант

```
1 import requests
2
3 response = requests.get('https://some-server.com', params={'
4 print(response.url)
```

HTML: Hyper-Text Markup Language

- Язык разметки
- Текст перемежается тегами
- Часто хочется очистить текст от тегов
- Или что-то взять из тегов
- Например, ссылки на другие ресурсы

HTML: Hyper-Text Markup Language

- Хочется отделять полезное содержимое
- От баннеров
- От рекламных вставок
- От элементов навигации

```
<span id="library-index"></span><h
<p>While <a class="reference inter
semantics of the Python language,
describes the standard library tha
describes some of the optional com
in Python distributions.</p>
<p>Python’s standard library is ve
facilities as indicated by the lon
library contains built-in modules
system functionality such as file
```

Выглядит не очень сложно

- Можно написать код, который
  - Найдет все теги
  - Вычленит полезный текст без тегов
  - Извлечет знание о том, где начинаются и заканчиваются абзацы
  - Найдет теги со ссылками

Все сложнее, чем может показаться

- Теги бывают вложенные
- Структура тегов бывает нарушенной
- Переводы строк могут идти в произвольном порядке
- Теговые символы могут встречаться в тексте
  - И для этого есть отдельные конструкции

Beautiful Soup

- Есть специальная библиотека для обработки HTML
- Анализирует HTML-текст
- Порождает структуру вложенных элементов
- Дает возможеность удобного поиска

Beautiful Soup

- https://beautiful-soup-4.readthedocs.io/en/latest/

  https://pypi.org/project/beautifulsoup4/
- https://pypi.org/project/beautifulsoup4
- https://www.crummy.com/software/BeautifulSoup/
- Установка: `python3 -m pip install beautifulsoup4`