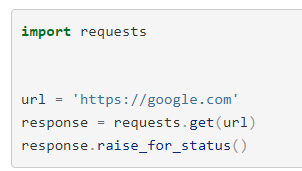
**Тема занятия №22: Сетевое программирование и веб-запросы. Парсинг данных**

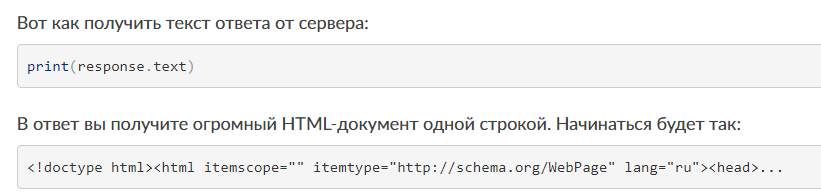
**1. Requests**

**Requests** — это библиотека, с помощью которой можно делать запросы в интернет. Само слово “requests” переводится как запросы. Вот как сделать запрос на сайт google.com:



Функция requests.get() делает запрос к сайту google.com, а в переменной response теперь ответ от сайта. Ответ — это не строка и не число. Это сложный объект, в котором есть много всего. Подробнее об этом можно посмотреть в слайдах.

Строчка response.raise\_for\_status() нужна для того, чтобы проверить, понял вас сервер или нет. Если сервер вернёт 404 «Ресурс не найден», то в response не будет странички сайта google.com, а будет только “Ошибка 404”. Если не вызвать raise\_for\_status, программа подумает, что всё в порядке, что вы так и хотели: отправить запрос на страницу, которой нет. Обязательно вызывайте .raise\_for\_status() после каждого запроса в интернет, иначе вы рискуете потратить кучу времени на поиск ошибки, которую замалчивает библиотека requests.



Именно это получает ваш браузер, когда вы заходите на сайт google.com, и из этого он уже отрисовывает красивую страничку.

Но стоит понимать, что requests может скачивать не только HTML-документы, а буквально всё, что есть в интернете.

Вот как скачать картинку:



Другие типы HTTP запросов, такие как : PUT, DELETE, HEAD и OPTIONS так же очень легко выполнить:



Часто вам может понадобится отправить какие-то данные в строке запроса URL. Если вы настраиваете URL вручную, эти данные будут представлены в нем в виде пар ключ/значение после знака вопроса. Например, httpbin.org/get?key=val. Requests позволяет передать эти аргументы в качестве словаря, используя аргумент params. Если вы хотите передать key1=value1 и key2=value2 ресурсу httpbin.org/get, вы должны использовать следующий код:



Содержимое ответа (response)

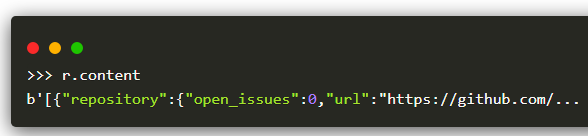
Мы можем прочитать содержимое ответа сервера. Рассмотрим снова тайм-лайн GitHub:



Requests будет автоматически декодировать содержимое ответа сервера. Большинство кодировок unicode декодируются без проблем.

Когда вы делаете запрос, Requests делает предположение о кодировке, основанное на заголовках HTTP. Эта же кодировка текста, используется при обращение к r.text. Можно узнать, какую кодировку использует Requests, и изменить её с помощью r.encoding.

Бинарное содержимое ответа



Содержимое ответа в JSON

Если вы работаете с данными в формате JSON, воспользуйтесь встроенным JSON декодером:



Если декодирование в JSON не удалось, r.json() вернет исключение. Например, если ответ с кодом 204 (No Content), или на случай если ответ содержит не валидный JSON, попытка обращения к r.json() будет возвращать ValueError: No JSON object could be decoded.

Следует отметить, что успешный вызов r.json() не указывает на успешный ответ сервера. Некоторые серверы могут возвращать объект JSON при неудачном ответе (например, сведения об ошибке HTTP 500). Такой JSON будет декодирован и возвращен. Для того, чтобы проверить успешен ли запрос, используйте r.raise\_for\_status() или проверьте какой r.status\_code.

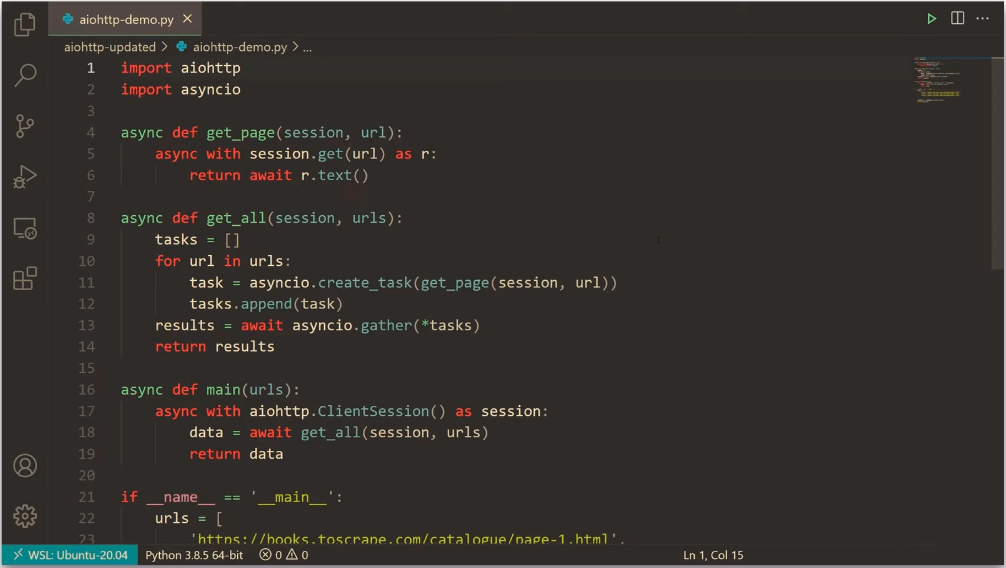
**1. AioHttp**

**Aiohttp** — асинхронный HTTP-клиент/сервер для модуля asyncio. Это библиотека языка Python, которая нужна для выполнения клиентских запросов и создания веб-сервера с потоковой выдачей и веб-сокетами.

Простым языком, Aiohttp — это технология, которая помогает приложению (сервер) и его пользователям (клиенты) быстрее и обмениваться данными. От того, насколько хорошо реализована эта связь, зависит производительность вашего веб-проекта.

Есть несколько способов реализовать HTTP-клиент для проектов на Python, но Aiohttp — самое популярное решение, быстрое и легко масштабируемое, которое используется практически по умолчанию.

Асинхронность нужна для выполнения нескольких операций, установления соединений, одновременно, без ожидания завершения предыдущих. Использование такого подхода увеличивает скорость работы веб-сервисов в несколько раз, причина низкой производительности которых, обычно, не проведение сложных вычислений, а именно ожидание ввода/вывода.



Используем asyncio.gather

Gather - как ни банально с английского собирать. Метод gather собирает коллекцию корутин и запускает их разом (тоже условно конечно). То есть, в отличии от предыдущего случая, мы в цикле создаем корутины, а потом их запускаем.

Было:

[cоздали корутину] -> [запустили корутину] -> [cоздали корутину] -> [запустили корутину] ->

[cоздали корутину] -> [запустили корутину] ->[cоздали корутину] -> [запустили корутину]

А стало:

[cоздали корутину] -> [создали корутину] ->[cоздали корутину] -> [создали корутину] ->

[запустили корутину] -> [запустили корутину] -> [запустили корутину] -> [запустили корутину]

