

1. Общее представление о WEB

Цель занятия

В результате обучения на этой неделе:

- вы узнаете основные протоколы транспортного уровня;
- поймете, как функционирует взаимодействие в интернете;
- научитесь пользоваться библиотекой requests;
- научитесь с ее помощью извлекать информацию из интернета.

План занятия

1. [Основы организации компьютерных сетей и модель TCP/IP](#)
2. [Библиотека requests](#)
3. [Практика по requests](#)

Используемые термины

IP-адрес — уникальный адрес, который присвоен компьютеру.

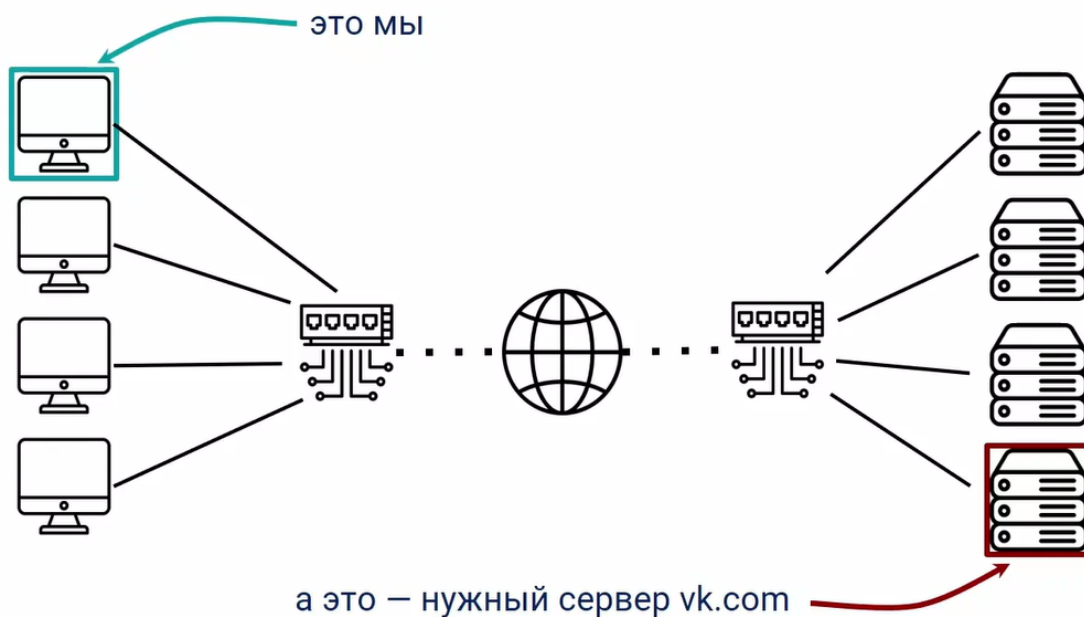
Порт — число, определяющее программу или процесс-получатель на компьютере.

Конспект занятия

1. Основы организации компьютерных сетей и модель TCP/IP

Модели OSI и TCP/IP похожи, но имеют некоторые различия.

Любую сеть можно охарактеризовать следующим образом:



Допустим, мы хотим загрузить нашу страницу VK. Как работают эти соединения и что происходит в то время, пока загружается страница в браузере, как раз и описывают модели OSI и TCP/IP.

Модель OSI

Эта модель более теоретическая, более стандартизированная. У модели 7 уровней:

7	Прикладной	Доступ к сетевым службам
6	Представление	Представление / кодирование
5	Сеансовый	Управление сеансом связи
4	Транспортный	Соединение

3	Сетевой	Логическая адресация и IP
2	Канальный	Физическая адресация
1	Физический	Кабель, сигналы

Самый первый, физический, отвечает за сигналы, которые передаются по кабелям. На 7-м уровне мы говорим непосредственно о программе, через которую используем сетевое соединение — будь то браузер, мессенджер, онлайн-игра или служба обновления ПО.

Каждый уровень решает свою задачу и работает со своим типом данных. Рассмотрим их подробнее на примере отправки данных.

Модель OSI: отправка

При отправке мы идем сверху вниз нашей таблицы:




7	прикладной	доступ к сетевым службам
6	представления	представление/кодирование
5	сеансовый	управление сеансом связи
4	транспортный	соединение
3	сетевой	логическая адресация и IP
2	канальный	физическая адресация
1	физический	кабель, сигналы

1. Определяемся, что отправить.

2. Разбиваем: блоки→пакеты→кадры.
3. Отправляем.

Модель OSI: прием

Если мы говорим про прием, то все идет в обратную сторону:



7	прикладной	доступ к сетевым службам
6	представления	представление/кодирование
5	сеансовый	управление сеансом связи
4	транспортный	соединение
3	сетевой	логическая адресация и IP
2	канальный	физическая адресация
1	физический	кабель, сигналы

1. Получаем биты.
2. Собираем их: биты→кадры→пакеты→блоки→данные.
3. Отображаем (почтовый редактор, браузер,...).

Модель OSI довольно теоретическая. Теория и практика различаются, иногда довольно значительно. К модели OSI претензии в основном в том, что она очень детальная и слишком предписывающая. В реальном мире некоторые уровни можно объединить и сделать проще.

Модель TCP/IP

Похожа на OSI, но упрощена — 4 уровня вместо 7:

4	Приложение	Обработка
3	Транспортный	Соединение
2	Сетевой	Выход в интернет
1	Доступ к сети	Кабель, сигналы

Модель TCP/IP названа по двум основным протоколам:

- TCP = Transmission Control Protocol.
- IP = Internet Protocol.

Зачем нужен IP-адрес

IP-адрес — это уникальный адрес, который присвоен компьютеру.

Согласно модели OSI, IP — это уровень 3:

- часть IP указывает на подсеть, к которой принадлежит компьютер;
- другая часть — на сам компьютер.

За определение, в нашей подсети компьютер или нет, отвечает маска подсети.

Как устроен IP-адрес

IP-адреса существуют в двух разных вариантах — четвертая и шестая версии.

IPv4:

- 4 группы десятичных чисел в диапазоне 0–255.
- Разделены точкой.
- Такой адрес занимает 32 бита памяти.
- Например: 192.168.1.1.

Всем «умным» устройствам нужны IP-адреса. И некоторым компьютерам и устройствам в наше время уже просто не хватает уникальных адресов.

IPv6:

- 8 групп шестнадцатеричных чисел от 0 до ffff (65535).
- Разделены двоеточием.
- 128 бит памяти.
- 2001:db8:85a3:8d3:1319:8a2e:370:7348

IP и маска подсети

Представим, что у нашего компьютера IP 192.168.2.1 маска подсети — /24. Это число означает, что первые 24 бита в бинарной записи IP-адреса останутся такими, какие они есть. Перейдем в бинарную систему и умножим побитово:

IP 11000000 10101000 00000010 00000001 перевели IP в двоичную систему

Маска 11111111 11111111 11111111 00000000 первые 24 бита — единицы

Результат 11000000 10101000 00000010 префикс подсети

Префикс переводим обратно в десятичную систему. Для нас получается, что все IP-адреса, начинающиеся на 192.168.2.xxx, принадлежат нашей подсети.

Порты

Список одновременно работающего ПО:

- почтовый клиент;
- браузер (много вкладок, много сеансов связи);
- онлайн-игра;
- хранилище файлов (DropBox, iCloud, etc.);
- обновление ОС.

Проблема в том, что мы одновременно можем или принимать данные, или отправлять данные. Одновременно принимать и отправлять данные мы не можем. Как организовать обмен данными и работу с сетью с учетом этого ограничения? Здесь на помощь приходят порты.

Порт — число, определяющее программу или процесс-получатель на компьютере. Каждой программе или процессу-получателю присвоен определенный порт.

Диапазон портов: 0–65535.

Некоторые порты зарезервированы специальной организацией IANA (Internet Assigned Numbers Authority), которая также распределяет IP-адреса.

Распределение портов

Порт	Назначение
0–1023	Общеизвестные
25	Получение почты
110	Отправка почты
80	http, https
1024– 49151	Зарегистрированные / пользовательские
2195, 2197	push-нотификации в Apple
3306	Базы данных MySQL
8000	Локальный сервер для веб-разработки
49152–65535	Динамические / частные

Что можно сделать с портом

- Открыть или закрыть для приема / передачи данных.
- Задать — назначить программе использовать конкретный порт.

- Пробросить — перенаправить на другой порт.
- Просканировать — проверить, не занят ли он чем-то.
- Заблокировать — выбрасывать / переадресовывать данные, поступающие на этот порт.

2. Библиотека requests

Клиент-серверная архитектура

Клиент-серверная архитектура сейчас является де-факто стандартом индустрии.

Пример. Что будет, если в поисковике вы нажмете кнопку «Найти»? Какие процессы происходят под капотом? Мы здесь присутствуем как клиент, а поисковик — как сервер. У сервера есть какая-то база данных, к которой он обращается за результатом.



клиент

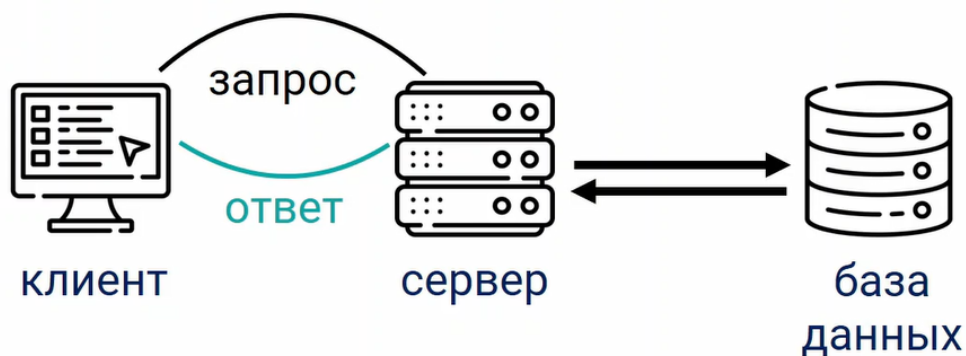


сервер



база
данных

Мы, как клиент, отправляем на сервер запрос. Получив запрос, сервер обрабатывает его и отправляется в базу данных. Получает ответ из базы данных, возвращает некоторый ответ:



Разберем запрос и посмотрим, из чего он состоит.

HTTP-запрос

Запрос (request) = Заголовки (headers) + Тело (body)

Для чего нужны заголовки:

- Описывают, что будет в теле запроса:
 - название устройства, с которого был произведен запрос;
 - операционная система, на которой он работает;
 - временная зона;
 - иногда кодировка.
- Уточняют запрос — например, какой тип данных мы хотим получить в ответе.

Тело запроса — это данные, связанные с запросом.

В примере с поисковиком телом запроса могут быть те слова, которые мы написали в поисковой строке.

Тела запроса может и не быть. Заголовки будут почти всегда. Большинство из них присваивается браузером, почтовым клиентом или другой программой.

Виды запросов

Запрос	Назначение
GET	Получить данные (например, страницу сайта)
POST	Отправить данные (например, заполненную на сайте форму)
PUT	Создать новый ресурс
OPTIONS	Посмотреть, какие запросы можно отправить

Зачем нужен requests

Это самый простой и понятный способ автоматизировать создание и отправку запросов на Python (3+ млн инсталляций).

Установка:

```
pip3 install requests
```

Посмотрим, как используется requests.

Использование: GET

Попробуем получить содержимое страницы sports.ru:

```
import requests

page = requests.get('https://sports.ru')

print(page.status_code)  # 200 OK

print(page.text)  # содержимое страницы
```

Если статус начинается на 3, значит, вы были перенаправлены на какую-то другую страницу. Если статус начинается на 4 — проблема со стороны клиента (например, ошибка 404, страница не найдена). Все ошибки на 5 — ошибки со стороны сервера.

Использование: POST

Меняется метод (post вместо get), добавляется обязательный атрибут data, содержащий тело запроса:

```
import requests

page = requests.post('https://httpbin.org', data={'key': 'value'})
```

Использование: заголовки

Все заголовки передаются внутри одного словаря, который передается в параметр headers:

```
import requests

url = 'https://api.github.com/some/endpoint'

headers = {'user-agent': 'my-app/0.0.1'}

r = requests.get(url, headers=headers)
```

3. Практика по requests

Попрактикуемся в использовании библиотеки requests и написании кода с ее помощью. Начнем с простого примера, а потом попробуем скачать целую электронную книгу.

Работаем на сайте httpbin.org. Этот сайт работает как некоторая песочница, которая собирает разные методы и дает нам ими воспользоваться.

Начнем с запроса Get. Получим картинку: Get/Image.

Воспроизведем пример в Python:

```
import requests

base_url = "https://httpbin.org/image"
response = requests.get(base_url)

print(response.status_code)
```

Ошибка 406 — мы не можем обработать контент, который получим.

Поэтому получим не картинку, а, например, кэш:

```
import requests

base_url = "https://httpbin.org/cache"
response = requests.get(base_url)

print(response.status_code)
```

Теперь сделаем цикл:

```
import requests

base_url = "https://httpbin.org/cache"
response = requests.get(base_url)

if response.status_code == 200:
    print("request OK")
else:
    print("error")
```

Распечатаем заголовки:

```
import requests

base_url = "https://httpbin.org/cache"
response = requests.get(base_url)

if response.status_code == 200:
    print("request OK")
    for header in response.headers:
        print(f"{header}: {response.headers[header]}")
else:
    print("error")
```

Теперь распечатаем тело ответа и получим словареподобную структуру, которая содержит все наши ответы:

```
import requests

base_url = "https://httpbin.org/cache"
response = requests.get(base_url)

if response.status_code == 200:
    print("request OK")
    for header in response.headers:
        print(f"{header}: {response.headers[header]}")
    print(response.text)
else:
    print("error")
```

Попрактикуемся в других типах запросов. У нас есть POST-запрос, который позволяет перенаправиться на какую-то другую ссылку:

```
import requests

base_url = "https://httpbin.org/redirect-to"
params = {
    "url": "https://ya.ru "
}

response = requests.post(base_url, params=params)
print (response.status_code)
```

Чтобы получать картинки, нужно усовершенствовать наш запрос:

```
import requests

base_url = "https://httpbin.org/image"
response = requests.get(base_url, headers={"accept": "image/webp"})
print(response.status_code)
```

Домашнее задание. Придумайте, как сделать так, чтобы Python смог открыть полученную картинку.

Перейдем теперь ко второй задаче. Попробуем скачать книгу. На сайте [gutenberg.org](https://www.gutenberg.org) можно работать с книгами и текстами, которые уже не обременены авторскими правами.

Попробуем скачать роман Ulysses и сложить его в файл на компьютере:

```
import requests

ulysses_url = "https://www.gutenberg.org/files/4300/4300-h/4300-h.htm "
response = requests.get(ulysses_url)
```

```
if response.status_code == 200:
    print("request OK")
    ulysses_text = response.text
    print(f"text size: {len(ulysses_text)}")
```

Теперь текст можно сохранить:

```
import requests

ulysses_url = "https://www.gutenberg.org/files/4300/4300-h/4300-h.htm "
response = requests.get(ulysses_url)
if response.status_code == 200:
    print("request OK")
    ulysses_text = response.text
    print(f"text size: {len(ulysses_text)}")
    with open("./ulysses.html", "w", encoding="utf-8") as f:
        f.write(ulysses_text)
```

Указание кодировки обязательно, но далеко не обязательно, что у всех сайтов она одинакова. Например, на сайтах для японской или китайской аудитории есть вероятность, что вы столкнетесь с какой-нибудь экзотической кодировкой.

Домашнее задание. Попробуйте зайти на сайт <https://gutenberg.org/>, скачать какую-нибудь книгу и посчитать, сколько в этой книге строк, абзацев, сколько определенных слов. Также можете убрать в книжке все теги — слова между двумя угловыми скобками.

Дополнительные материалы для самостоятельного изучения

1. <https://www.flaticon.com/>
2. <http://httpbin.org/>
3. <https://gutenberg.org/>