Язык Python. Часть 2

Лекция 2

Как данные передаются

- Как работали проводные телефоны
  - От каждого абонента шел провод до телефонной станции
  - И телефонные станции соединены с некоторыми ближайшими проводами
  - И в момент разговора устанавливалось физическое соединение между абонентами

Как данные передаются

- Почему нам важно, как работали проводные телефоны
- И причем здесь Python
  - Потому что их работа соответсвует интуитивным представлениям
  - Но данные в интернете передаются не так
  - А интернет - то место, где Python активно применяется

Как данные передаются

- Данные режутся на кусочки
- Которые называются пакетами
- Есть спецальные устройства - маршрутизаторы
- Каждый пакет посылается в ближайший маршрутизатор

Как данные передаются

- Каждый маршрутизатор
  - Знает своих коллег-соседей
  - По ip-адресу может понять, кому из них лучше передать пакет
  - Отслеживает текущее состояние сети

Как данные передаются

- В такой схеме
  - Пакет можеть потеряться и не дойти
  - Один пакет может дойти несколько раз
  - Ранее отправленный пакет может прийти позже позднее отправленного

И как с этим быть

- Для некоторых приложений это нормально
- Например, для интернет-вещание
- Продублированные пакеты проигнорируем
- Потерявшиеся/опоздавшие ухудшат качество
- Но если их немного - это нестрашно

Но часто это неприемлемо

- Например, для классического сайта
- Представим себе интернет-библиотеку
- А в ней фрагменты текста
  - Пропадают
  - Дублируются
  - Меняются местами

TCP - transmission control protocol

- Можно создать сеанс
- В рамках которого
  - Один участник пишет данные в канал связи как в файл
  - А другой - их читает как из файла
  - Ничего не дублируется, не теряется, не меняется местами
  - И это работает в обе стороны

UDP - unicast datagram protocol

- Просто кидаем пакеты
- Имеем проблемы с доставкой
- Зато дешевле, чем TCP
- Это немножко в стороне от наших интересов

Роли участников

- Один слушает входящие соединения
- Его называют сервером
- Другие подключаются
- Их называют клиентами

Роли участников

- Клиенты посылают запросы
- Сервер на них отвечает
- Ни TCP, ни UDP не обязывают именно к такой схеме
- Но она очень популярна
- Иногда это называют клиент-серверной архитектурой

Поддержка в стандартной библиотеке

- Модуль socket
- Модуль socketserver

Есть тут одна большая проблема

- Допустим, клиенты с сервером решают общую задачу
- Эта задача подразумевает обмен данными
- Которые проходят через маршрутизаторы
- Маршрутизаторы принадлежать не нам
- Данные могут быть прочитаны
- Или изменены

SSL: Secure socket layer

- Будем данные шифровать при посылке
- И дешифровывать при получении
- В сети они будут в зашифрованном виде
- Применяется ко всем протоколам, работающим на базе TCP
- Надо понастраивать, но использование - такое же

Как работает Web

- Браузер - клиент
- Сайт - сервер
- На основе TCP
- Специальный протокол запросов и ответов

Как работает Web

- Адрес ресурса - URL (иногда URI)
- URL - Universal Resource Locator
- URI - Universal Resource Identifier
- Можно увидеть оба варианта
- Есть тонкая разница, но нам она не важна

Структура URL

- https://docs.python.org/3/tutorial/index.html
- https - схема (протокол)
- Иногда бывает http, но очень редко
- HTTP - Hyper-Text Transfer Protocol
- https - http, работающий на зашифрованном канале

Структура URL

- https://docs.python.org/3/tutorial/index.html
- docs.python.org - адрес
- Нужен порт
- Для https подразумевается 443
- Но можно явно указать: https://docs.python.org:443/3/tutorial/index.html
- Если сервер запущен на другом порту, то явно указывать обязательно

Структура URL

- https://docs.python.org/3/tutorial/index.html
- /3/tutorial/index.html - путь
- Индентификатор ресурса внутри сервера
- Как его интерпретировать - зависит от конкретного сервера

Как работает браузер

- Когда надо открыть https://docs.python.org/3/tutorial/index.html
- Смотрит на протокол
- Открывает tcp-сеанс
- С шифрованием
- Чтобы передавать http-запросы и получать ответы

Как работает браузер

- Для открытия сеанса нужны координаты сервиса
- IP-адрес и порт
- IP-адрес определяем по docs.python.org
- Порт - 443 (по умолчанию для https)

Из чего состоит http-запрос

- В первом приближении - из строчек
- Как минимум - начальная часть
- На более высоком уровне - из трех частей
- Первая часть - самая простая
- Тип запроса (метод), путь версия http

Первая строка http-запроса

- Тип запроса в нашем случае - GET
- Путь - /3/tutorial/index.html
- Версия http - HTTP/1.1
- Браузер первой строчкой пишет: GET /3/tutorial/index.html HTTP/1.1

Реакция сервера

- Читает первую строчку
- Определяет тип запроса
- Определяет путь
- Понимает, что надо прочитать определеннный ресурс
- Конкретные действия сервера по чтению ресурса - на его усмотрение

Реакция сервера

- В данном случае - вероятно, что где-то есть каталог
- В котором лежит файл 3/tutorial/index.html
- И сервер прочитает его содержимое

Вернемся к запросу

- Клиент может захотеть сообщить что-то о себе
- Или о своих пожеланиях
- Например, какие-то форматы данных он понимает
- (чтобы сервер знал, что отправлять в ответ)
- Или - должен ли сервер всегда читать ресурс заново или можно его кешировать

Вернемся к запросу

- Для таких данных используется вторая часть запроса
- Она называется - headers (заголовки)
- Состоит из нескольких строчек
- Каждая строчка состоит из имени заголовка
- За которой идет точка с запятой
- И значение заголовка

Вернемся к запросу

- Пример заголовка

```
User-Agent: "Chromium";v="94", "Yan
```

- Их может быть несколько
- Признак конца этой части - пустая строка

Вернемся к запросу

- Третья часть - тело запроса
- В GET-запросах обычно отсутствует
- Но еще бывают POST и PUT
- В них обычно тело есть
- Когда что-то куда-то отправляете - текст пересывается в теле запроса

Ответ сервера

- На любой запроса сервер дает ответ
- Это тоже набор строк
- И тоже из трех частей
- Первая часть - статус

Ответ сервера

- Статус - это одна строка
- Статус начинается с числового трехзначного кода
- Точнее, с его текстового представления
- Потом - пробел и текстовое описание
- Обычный статус: 200 OK

Статус

- Всего статусов - несколько десятков
- О многом говорит первая цифра
- 1 или 2 - в целом успешно
- Чаще всего это 200, но бывают другие варианты

Статус

- 3 - перенаправления по какой-то причине
- Частый вариант - 302
- Ресурс переехал
- Но тех, кто приходит по старой ссылке - перенаправляем
- Частое использование - приходящих на http-версию перенаправлять на https

Статус

- 4 или 5 - ошибки (4 - клиентские, 5 - серверные)
- Частые вариант - 404, 505
- 404 - ресурс переехал
- 500 - внутрення ошибка сервера
- https://en.wikipedia.org/wiki/List_of_HTTP_status_

Другие части ответа

- Вторая часть - заголовки ответа
- Структура - как и у заголовков запроса
- Сами заголовки - другие
- Третья часть - тело ответа
- В ответах на GET-запроса почти всегда есть

Что может быть в теле

- Содержимое html-файла
- Или другого файла на текстовой основе
- Картинка
- Javascript, CSS
- Загрузка одной страницы бразузером обычно порождает много GET-запросов

Возвращаемся в Python

Возвращаемся в Python

- Есть встроенный модуль socket
- Но тут надо копаться в мелких деталях
- Есть встроенная реализация http
- Но это тоже будет громоздко
- А есть сторонняя библиотека requests - стандарт de-facto

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))
```