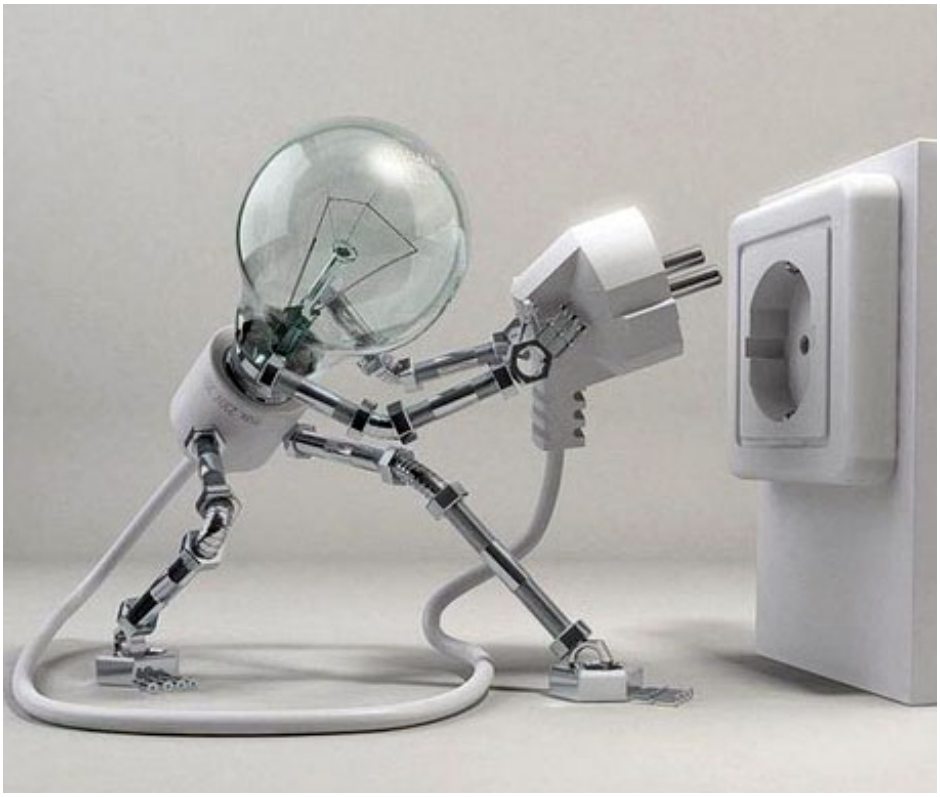
# Сокеты в Python для начинающих

### Предисловие

В далеком для меня 2010 году я писал статью для начинающих про сокеты в Python. Сейчас этот блог канул в небытие, но статья мне показалась довольно полезной. Статью нашел на флешке в либровском документе, так что это не кросспост, не копипаст — в интернете ее нигде нет.



### Что это

Для начала нужно разобраться что такое вообще сокеты и зачем они нам нужны. Как говорит вики, сокет — это программный интерфейс для обеспечения информационного обмена между процессами. Но гораздо важнее не зазубрить определение, а понять суть. Поэтому я тут постараюсь рассказать все как можно подробнее и проще.

Существуют клиентские и серверные сокеты. Вполне легко догадаться что к чему. Серверный сокет прослушивает определенный порт, а клиентский подключается к серверу. После того, как было установлено соединение начинается обмен данными.

Рассмотрим это на простом примере. Представим себе большой зал с множеством небольших окошек, за которыми стоят девушки. Есть и пустые окна, за которыми никого нет. Те самые окна — это порты. Там, где стоит девушка — это открытый порт, за которым стоит какое-то приложение, которое его прослушивает. То есть, если, вы подойдете к окошку с номером 9090, то вас поприветствуют и спросят, чем могут помочь. Так же и с сокетами. Создается приложение, которое прослушивает свой порт. Когда клиент устанавливает соединение с сервером на этом порту именно данное приложение будет ответственно за работу этим клиентом. Вы же не подойдете к одному окошку, а кричать вам будут из соседнего :)

После успешной установки соединения сервер и клиент начинают обмениваться информацией. Например, сервер посылает приветствие и предложение ввести какую-либо команду. Клиент в свою очередь вводит команду, сервер ее анализирует, выполняет необходимые операции и отдает клиенту результат.

### Сервер

Сейчас создайте два файла — один для сервера, а другой для клиента.

В Python для работы с сокетами используется модуль socket:

```
import socket
```

Прежде всего нам необходимо создать сокет:

```
sock = socket.socket()
```

Здесь ничего особенного нет и данная часть является общей и для клиентских и для серверных сокетов. Дальше мы будем писать код для сервера. Это вполне логично — зачем нам писать клиентское приложение, если некуда подключатся :)

Теперь нам нужно определится с хостом и портом для нашего сервера. Насчет хоста — мы оставим строку пустой, чтобы наш сервер был доступен для всех интерфейсов. А порт возьмем любой от нуля до 65535. Следует отметить, что в большинстве операционных систем прослушивание портов с номерами 0 — 1023 требует особых привилегий. Я выбрал порт 9090. Теперь свяжем наш сокет с данными хостом и портом с помощью метода bind, которому передается кортеж, первый элемент (или нулевой, если считать от нуля) которого — хост, а второй — порт:

```
sock.bind(('', 9090))
```

Теперь у нас все готово, чтобы принимать соединения. С помощью метода listen мы запустим для данного сокета режим прослушивания. Метод принимает один аргумент — максимальное количество подключений в очереди. Напряжем нашу бурную фантазию и вспомним про зал с окошками. Так вот этот параметр определяет размер очереди. Если он установлен в единицу, а кто-то, явно лишний, пытается еще подстроится сзади, то его пошлют :) Установим его в единицу:

```
sock.listen(1)
```

Ну вот, наконец-то, мы можем принять подключение с помощью метода accept, который возвращает кортеж с двумя элементами: новый сокет и адрес клиента. Именно этот сокет и будет использоваться для приема и посылке клиенту данных.

```
conn, addr = sock.accept()
```

Вот и все. Теперь мы установили с клиентом связь и можем с ним «общаться». Т.к. мы не можем точно знать, что и в каких объемах клиент нам пошлет, то мы будем получать данные от него небольшими порциями. Чтобы получить данные нужно воспользоваться методом recv, который в качестве аргумента принимает количество байт для чтения. Мы будем читать порциями по 1024 байт (или 1 кб):

```
while True:
    data = conn.recv(1024)
    if not data:
        break
    conn.send(data.upper())
```

Как мы и говорили для общения с клиентом мы используем сокет, который получили в результате выполнения метода accept. Мы в бесконечном цикле принимаем 1024 байт данных с помощью метода recv. Если данных больше нет, то этот метод ничего не возвращает. Таким образом мы можем получать от клиента любое количество данных.

Дальше в нашем примере для наглядности мы что-то сделаем с полученными данными и отправим их обратно клиенту. Например, с помощью метода upper у строк вернем клиенту строку в верхнем регистре.

Теперь можно и закрыть соединение:

```
conn.close()
```

Собственно сервер готов. Он принимает соединение, принимает от клиента данные, возвращает их в виде строки в верхнем регистре и закрывает соединение. Все просто :) В итоге у вас должно было получиться следующее:

```
import socket

sock = socket.socket()
sock.bind(('', 9090))
sock.listen(1)
conn, addr = sock.accept()

print 'connected:', addr

while True:
    data = conn.recv(1024)
    if not data:
        break
    conn.send(data.upper())

conn.close()
```

**Клиент**

Думаю, что теперь будет легче. Да и само клиентское приложение проще — нам нужно создать сокет, подключиться к серверу послать ему данные, принять данные и закрыть соединение. Все это делается так:

```
import socket
```

```
sock = socket.socket()
sock.connect(('localhost', 9090))
sock.send('hello, world!')

data = sock.recv(1024)
sock.close()

print data
```

Думаю, что все понятно, т.к. все уже разбиралось ранее. Единственное новое здесь — это метод connect, с помощью которого мы подключаемся к серверу. Дальше мы читаем 1024 байт данных и закрываем сокет.