МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

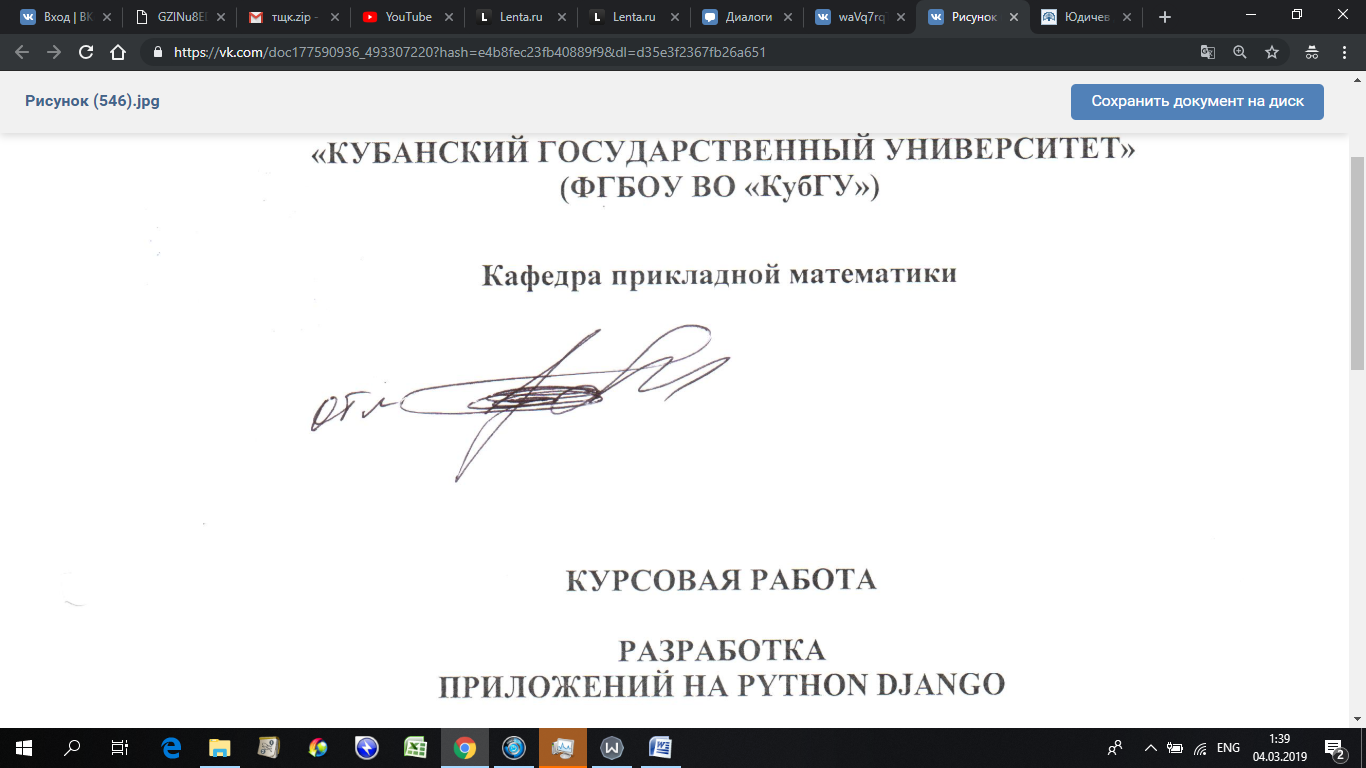
Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Кафедра прикладной математики**



**КУРСОВАЯ РАБОТА**

**РАЗРАБОТКА**

**ПРИЛОЖЕНИЙ НА PYTHON DJANGO**



Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. В. Юдичев

(подпись, дата) (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики курс 3

Направление 09.03.03 Прикладная информатика

Научный руководитель,

канд. экон. наук,

доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. В. Коваленко (подпись, дата) (инициалы, фамилия)

Нормоконтролер,

к. ф.-м. н.\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г. В. Калайдина

(подпись, дата) (инициалы, фамилия)

Краснодар 2018

СОДЕРЖАНИЕ

[Введение 3](#_Toc517885394)

[1 История и архитектура Python и его фреймворка Django 4](#_Toc517885395)

[1.1 История создания языков программирования 4](#_Toc517885396)

[1.2 Современные языки программирования 5](#_Toc517885397)

[1.3 Характеристики Python 6](#_Toc517885398)

[1.4 История создания Python 7](#_Toc517885399)

[1.4.1 Первая версия 8](#_Toc517885400)

[1.4.2 Версия 1.0 и 1.2 8](#_Toc517885401)

[1.4.3 Проект «Программирование для всех» и Python 8](#_Toc517885402)

[1.4.4 Версии 1.6, 1.6.1 и 2.0 9](#_Toc517885403)

[1.4.5 Версии 2.х и 3.х 10](#_Toc517885404)

[1.5 История создания Django 11](#_Toc517885405)

[1.6 Особенности Django 12](#_Toc517885406)

[1.7 Структура Django-приложения 12](#_Toc517885407)

[1.8 Особенности Python 14](#_Toc517885408)

[1.8.1 Исключения 17](#_Toc517885409)

[1.8.2 Переменные 20](#_Toc517885410)

[1.8.3 Группировка операторов 21](#_Toc517885411)

[1.8.4 Цикл for 23](#_Toc517885412)

[1.8.5 Списки 24](#_Toc517885413)

[1.8.6 Кортежи 29](#_Toc517885414)

[1.8.7 Словари 32](#_Toc517885415)

[1.9 База данных PostgreSQL 33](#_Toc517885416)

[2 Разработка клиент-серверного приложения 34](#_Toc517885417)

[2.1 Подготовка к работе 34](#_Toc517885418)

[2.2 Приступаем к работе 39](#_Toc517885419)

[2.3 Настройка urls.py 42](#_Toc517885420)

[2.4 Создание моделей 44](#_Toc517885421)

[2.5 Создание представлений и шаблонов 45](#_Toc517885422)

[2.6 Тестирование 58](#_Toc517885423)

[Заключение 69](#_Toc517885424)

[Список использованных источников 70](#_Toc517885425)

ВВЕДЕНИЕ

На данный момент высокоуровневый язык программирования Python, а в частности его фреймворк Django является очень актуальным при разработке приложений. Большой набор библиотек, инструментов, шаблонов, компактность кода и структурированность – вот причина популярности данного языка программирования.

Целью курсовой работы является создание клиент–серверного приложения на Python Django.

В список поставленных задач входит:

1. Изучить историю создания и архитектуру Python и его фреймворка Django.
2. Создать клиент-серверное приложение, используя фреймворк Django.

Курсовая работа состоит из двух глав, введения, заключения и списка

использованной информации.

В первой главе подробно описывается архитектура и история создания Python и его фреймворка Django.

Во второй главе описывается поэтапное создание приложения «Журнал сделок трейдера».

В заключении приведены основные источники курсовой работы.

# 

# 1 История и архитектура Python и его фреймворка Django

## 1.1 История создания языков программирования

Любую компьютерную программу можно представить в виде некоторого набора последовательных действий (команд), выполнение которых поможет достичь необходимой цели.

Все действия, которые человек выполняет в течение дня, так же содержат некоторую последовательность связанных между собой команд, а некоторые из них мы и вовсе выполняем изо дня в день в течение всей нашей жизни(чистим зубы, например). Получается, что мы в некотором роде компьютеры, выполняющие алгоритмы, написанные на естественном языке(русский или любой другой).

Но для создания программ этот язык не годится, потому что вычислительные машины их попросту не поймут. Они понимают только машинный язык, который состоит из цифр(например, в двоичной системе счисления).

В первое время люди так и программировали с помощью цифр, однако это было очень неудобно по нескольким причинам:

1) для людей естественна словесная форма записи команд, а не цифровая

2) запись даже небольшого алгоритма состояла из огромного количества цифр

С этими неудобствами нужно было бороться. Сначала появились ассемблеры – языки, которые использовали словесно-буквенные обозначения команд вместо числовых. А затем языки высокого уровня, с куда большим функционалом и более простой формой для человека.

Но тут возникает проблема: вычислительным машинам не понятна буквенная форма записи команд. Поэтому под каждый язык программирования создаются трансляторы – программы, которые преобразуют программный код с языка программирования в машинный код.

Механизм этого перевода весьма сложен, при этом выделяют два основных способа трансляции – компиляция программы или  ее интерпретация.

При компиляции весь исходный программный код (тот, который пишет программист) сразу переводится в машинный. Создается так называемый отдельный исполняемый файл, который никак не связан с исходным кодом. Выполнение исполняемого файла обеспечивается операционной системой.

При интерпретации выполнение кода происходит последовательно (можно сказать, строка за строкой). Операционная система взаимодействует с интерпретатором, она не читает исходный код непосредственно.

Выполнение откомпилированной программы происходит быстрее, т.к. она представляет собой готовый машинный код. Однако на современных компьютерах снижение скорости выполнения при интерпретации обычно не заметно.[1]

## 

## 1.2 Современные языки программирования

В наше время существует огромное количество различных языков программирования как похожих друг на друга, так и различающихся между собой.

Все языки имеют какую-то направленность, т.е. способны решать лишь определенные задачи. Поэтому чаще всего они классифицируются по типу решаемых задач.

Я же остановил свой выбор на языке программирования Python.

## 1.3 Характеристики Python

Python – высокоуровневый язык программирования общего назначения, обладающий большой стандартной библиотекой, содержащей множество полезных функций и поддерживающий несколько парадигм программирования.[2]

Разработчики Python придерживаются определенной философии, автором которой является Тим Петерс. Ее содержание легко можно увидеть, запустив интерпретатор и введя в нем команду «import this»:

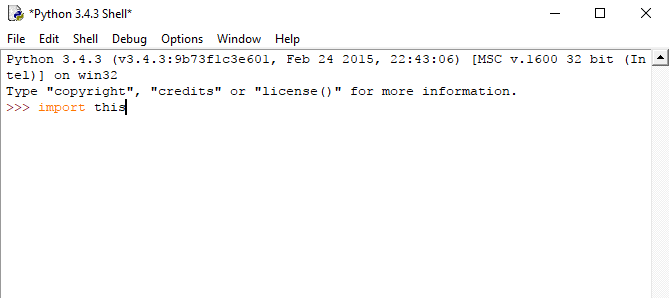


Рисунок 1 – Вызов модуля «this»

Нажав на «Enter», получим следующий результат:

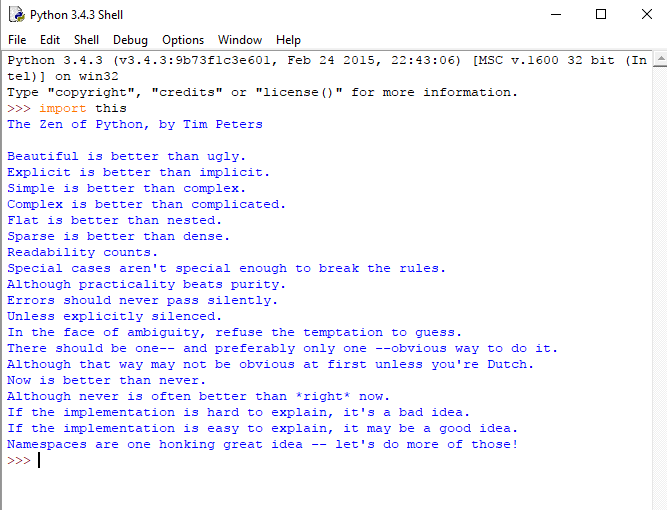


Рисунок 2 – Философия языка Python

Если коротко, то разработчики выступают за красоту, явность и простоту.

Также эта философия содержит много важных правил, например:

«Ошибки никогда не должны замалчиваться»,

«Если реализацию сложно объяснить, то у вас плохая идея»

«Если реализацию легко объяснить, то, возможно, у вас хорошая идея».

## 1.4 История создания Python

Развитие Python происходило в то время, когда активно развивались и набирали популярность многие другие языки программирования (и open–source), такие как Tcl, Perl и (намного позже) Ruby.[3]

Автором языка программирования Python является Гвидо Ван Россум – нидерландский программист. Идея о данной разработке родилась у него еще в начале 1980–х годов, однако к реализации он смог приступить лишь в декабре 1989 года. Работу над проектом он выполнял в Национальном научно–исследовательском институте математики и информатики, расположенном в Амстердаме.

На первый взгляд, может показаться, что название языка возникло от известного семейства опасных змей. В пользу этой теории говорит эмблема на сайте python.org или значки файлов языка, на которых до выхода версии 2.5 был изображен представитель вида пресмыкающихся. Однако на самом деле название было взято в честь известного комедийного британского шоу «Летающий цирк Монти-Пайтона», выходившего в 1970-х годах.

Python задумывался в первую очередь как потомок языка ABC, способный к взаимодействию с операционной системой Amoeba.[4]

### 1.4.1 Первая версия

Первую публикацию своего проекта Гвидо Ван Россум сделал в феврале 1991 года. Это был Python с версией 0.9.0. Уже тогда он обладал достаточно большими возможностями. В частности были доступны функции, обработка исключений, классы с наследованием, основные типы данных: list(список), dict(словарь), str(строка). Также была доступна обработка исключений, сделанная на основе системного языка программирования Модула–3, с добавлением оператора else.[4]

### 

### 1.4.2 Версия 1.0 и 1.2

В январе 1994 года вышла новая версия Python – 1.0, включающая в себя новые возможности – в основном это были средства функционального программирования: свертка списка, лямбда–исчисление, функции высших порядков map и filter.

Следующей была версия 1.2 – она стала последней разработкой Гвидо Ван Россума, выпущенной в период работы в центре математики и информатики.

«С 1995 года Ван Россум продолжил работу над Python-ом в корпорации национальных исследовательских инициатив (CNRI) в городе Рестон, штат Вирджиния, где было выпущено несколько версий языка.[4]

### 

### 1.4.3 Проект «Программирование для всех» и Python

Во время пребывания в CNRI Ван Россум запустил проект "Программирование для всех" (англ. Computer Programming for Everybody, CP4E), предназначенный сделать программирование доступным для большего числа людей, на основе получения базовой "компьютерной грамотности", подобной базовому знанию языка и математики, требуемых большинству работающих. Python играл центральную роль в этой инициативе, благодаря своей нацеленности на ясный синтаксис. Проект CP4E финансировался DARPA(Управление перспективных исследовательских проектов Министерства обороны США), в данное время проект закрыт. И, хотя Python старается быть простым в изучении и не слишком магическим в синтаксисе и семантике, простота его использования непрограммистами не является главной задачей.»[4]

### 

### 1.4.4 Версии 1.6, 1.6.1 и 2.0

В начале 2000х годов основная часть команды разработчиков перешло на BeOpen.com, где сформировалась команда BeOpen Python Lab. В это же время планировалось выпустить версию 2.0.

Однако CNRI потребовала выпустить версию, в которой бы содержались все наработки, созданные командой за время работы в корпорации. Так и вышла версия 1.6 с новым соглашением от CNRI, согласно которому оно регулируется законами штата Вирджиния. Но статья о выборе правовой нормы противоречила GNU GPL(GNU General Public License) –лицензии на свободное программное обеспечение. Фонд свободного программного обеспечения(FSF), CNRI и BeOpen договорились изменить текущую лицензию, совместимую с GPL. ­­­

Поэтому вскоре вышла версия 1.6.1, включающая незначительные доработки и новую лицензию.

Спустя месяц, 16 октября 2000 года была создана версия 2.0, опубликованная на BeOpen.com. Это была и остается по сей день единственной публикацией на BeOpen.

Главным нововведением в данном релизе стало списковое включение, заимствованное из функциональных языков программирования Haskell и SETL, а также система сборки мусора – автоматическое управление памятью, которое освобождает ее, удаляя объекты, которые не будут востребованы приложениями.[4]

### 

### 1.4.5 Версии 2.х и 3.х

Затем были созданы версии 2.1 и 2.2, в которые были добавлены генераторы, и было осуществлено объединение в одной иерархии всех базовых типов и классов, создаваемых программистом, благодаря чему Python стал объектно-ориентированным языком.

Python 3.0 создавался с целью устранения недостатков предыдущих версий. При этом было необходимо соблюдать полную обратную совместимость с версиями 2.х. В частности назрела потребность в удалении дублирующихся конструкций – новых и старых, использующихся для решения одних и тех же задач.

При разработке создатели неизменно следовали своей философии, особенно отдавая предпочтение следующему принципу «должен существовать желательно только один очевидный способ сделать это».

Python 3.0 был выпущен 3 декабря 2008 года, вместе с которым был сделан параллельный релиз 2.6, содержащим функционал 3.0, а так же некоторые возможности, удаленные с версий 3.х.

Затем были разработаны версии 3.1 и 2.7 – последним релизом 2.х.

К тому времени было сделано немало изменений: оператор Print стал функцией, динамическая типизация, переход на Юникод для строк. Все это сделало сложным процесс автоматической трансляции версии 2.х в 3.х. На помощь пришел инструмент под названием «2То3», который вполне успешно выполнял основную работу по переводу.[4]

## 1.5 История создания Django

Django органично росла во время разработки реальных приложений, созданных командой разработки в Лоуренсе, штат Канзас, США. Она родилась в конце 2003 когда программисты газеты «Lawrence Journal-World», Эдриан Холовати (Adrian Holovaty) и Симон Виллисон (Simon Willison), начали использовать язык Python для разработки своих приложений. Команда World Online, ответственная за разработку и поддержку нескольких локальных новостных сайтов, процветала в среде разработчиков, диктуемой срочностью работы журналиста. Для сайтов, включая LJWorld.com, Lawrence.com и KUsports.com, журналисты и менеджеры требовали, чтобы новые возможности и целые приложения были разработаны максимально быстро, часто счёт шёл на дни или на часы. Таким образом, Симон и Эдриан создали среду разработки, экономящую время, исходя из своих потребностей – это был единственный способ, с помощью которого они могли создавать управляемые приложения в жёстких рамках сроков.

Летом 2005, развив эту среду до такого состояния, что она поддерживала большинство сайтов World Online, команда разработчиков, включая Якоба Каплан-Мосса (Jacob Kaplan-Moss), решила выпустить среду в виде программного обеспечения с открытым исходным кодом. Они выпустили его в июле 2005 под именем Django в честь джазового гитариста Джанго Рэйнхарда (Django Reinhardt).

Теперь, по прошествии нескольких лет, Django является грамотным проектом с открытым исходным кодом с десятками тысяч пользователей, над которым работают разработчики по всей планете. Два автора из команды World Online («Великодушные пожизненные диктаторы», Эдриан и Якоб) всё ещё осуществляют общее руководство над развитием среды разработки, но теперь оно в большей степени зависит совместных усилий команды разработчиков. World Online обеспечивает другие важные аспекты, такие как время разработчиков, маркетинговые материалы и хостинг/канал для сайта среды (<http://www.djangoproject.com/>).[7]

1.6 Особенности Django

Основное назначение Django. Так как Django родился в новостной среде, он предоставляет некоторые средства (такие как административный интерфейс), которые хорошо подойдут для контент-ориентированных сайтов, которые предоставляют динамическую информацию из базы данных.

Происхождение Django сформировало культуру его сообщества открытого исходного кода. Так как Django был получен из реального кода, а не был академической разработкой или коммерческим продуктом, он полностью сфокусирован на решение проблем разработки, с которыми сталкивались и продолжают сталкиваться его авторы. В результате, Django постоянно совершенствуется. Основатели среды имеют свой интерес в том, чтобы Django экономил их время, создавал приложения лёгкие в обслуживании и хорошо работал под нагрузкой. При отсутствии других причин, разработчики мотивируются своими собственными эгоистичными желаниями сэкономить своё время и наслаждаться своей работой.[7]

Отличительные особенности Джанго:

1. любой запрос обрабатывается программно и перенаправляется на свой адрес(url);
2. разделение контента и представления с помощью шаблонов;
3. абстрагирование от низкого уровня баз данных.[8]

## 1.7 Структура Django-приложения

Джанго-приложение состоит из четырех основных компонентов.

1. Модель данных: данные являются сердцевиной любого современного Web-приложения. Модель – важнейшая часть приложения, которое постоянно обращается к данным при любом запросе из любой сессии. Любая модель является стандартным питоновским классом. Объектно-ориентированный маппер (ORM) обеспечивает таким классам доступ непосредственно к базам данных. Если бы не было ORM, программисту пришлось бы писать запросы непосредственно на SQL. Модель обеспечивает облегченный механизм доступа к слою данных, инкапсулирует бизнес-логику. Модель не зависит от конкретного приложения. Данными можно манипулировать даже из командной строки, не используя при этом Web-сервер.
2. Представление (view): Представления в Джанго выполняют разнообразные функции, в том числе контролируют запросы пользователя, выдают контекст в зависимости от его роли. View – это обычная функция, которая вызывается в ответ на запрос какого–то адреса (url) и возвращает контекст.
3. Шаблоны: они являются формой представления данных. Шаблоны имеют свой собственный простой метаязык и являются одним из основных средств вывода на экран.
4. URL: это всего лишь механизм внешнего доступа к представлениям (view). Встроенные в url регулярные выражения делают механизм достаточно гибким. При этом одно представление может быть сконфигурировано к нескольким url, предоставляя доступ различным приложениям. Здесь поддерживается философия закладок: url-ы становятся как бы самодостаточными и начинают жить независимо от представления.

В Django поддерживается все то, что и в Python: тут есть доступ ко всем стандартным, и не только, библиотекам Python, плюс встроенный функционал Джанго.[8]

## 1.8 Особенности Python

Python – это интерпретируемый язык программирования.

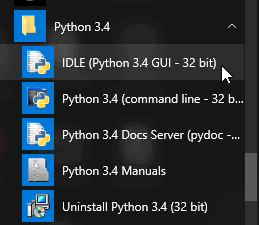


Рисунок 3 – Запуск интерактивного режима

Обладает интегрируемой средой разработки IDLE, в которой есть интерактивный режим:

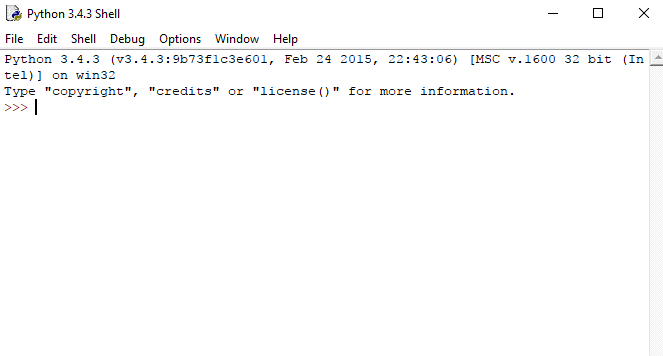


Рисунок 4 – Интерактивный режим

Он очень удобен для тестирования каких–либо конструкций, частей кода, что очень полезно как для начинающих, так и для опытных разработчиков.

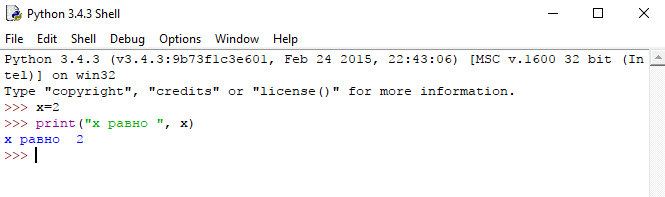


Рисунок 5 – Тестирование некоторых команд

Важной особенностью Python является подсвечивание синтаксиса. Так мы можем определить правильность написания некоторых встроенных команд, функций(если вдруг напишем с ошибкой, то эта часть кода не будет окрашена в определенный цвет).

Но практически во всех случаях требуется написать программу, которую нужно сохранить. Тогда нужно попросту открыть специальное окно, проделав простейшую комбинацию действий: File–New File.

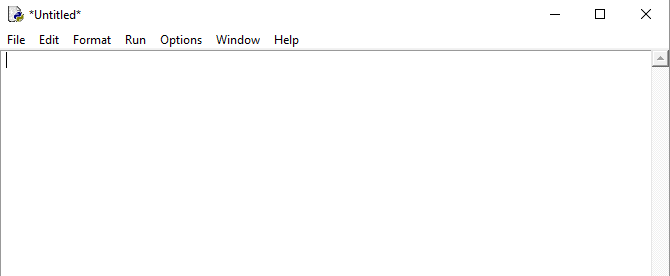


Рисунок 6 – Создание нового проекта

В этом окне и пишется код. Сохранить его можно, нажав на File–Save.

Запустить программу можно следующим путем: Run–Run Module, либо нажав на F5. В этом случае будет предложено сохранить скрипт, если этого не было сделано ранее.

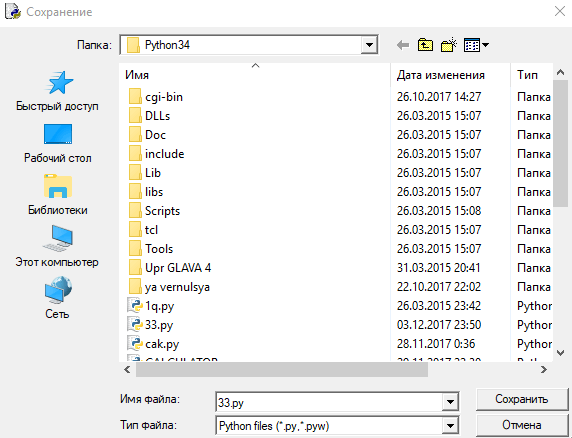
Файлы с кодом обычно имеют расширение “py”:

Рисунок 7 – Сохранение скрипта

На самом деле написать код программы можно в любом текстовом редакторе. Главное соблюдать синтаксис и сохранить в нужном формате разработку.

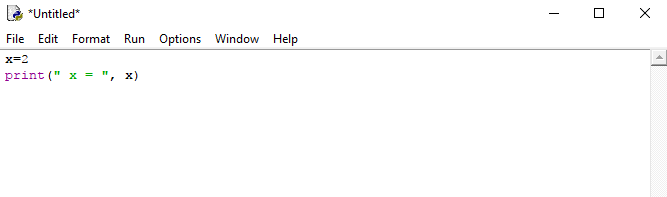


Рисунок 8 – Код программы

При запуске программы ее выполнение происходит в интерпретаторе:

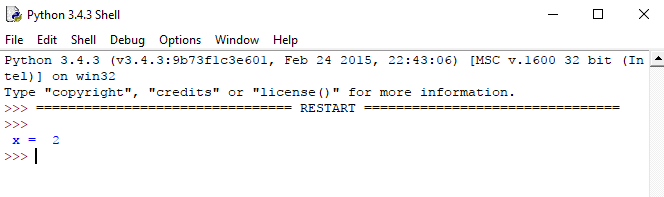


Рисунок 9 – Выполнение написанного кода

Чтобы не запутаться, каждый запуск программы отделяется вставкой «RESTART». Запустим предыдущий код еще раз:

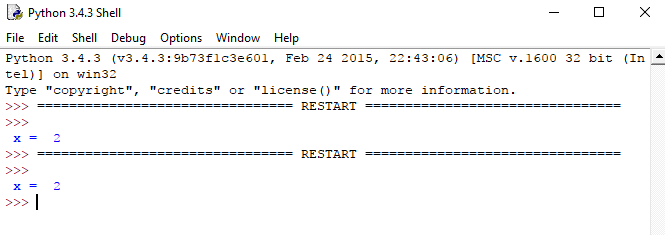


Рисунок 10 – Несколько запусков скрипта отделяются линией RESTART

### 

### 1.8.1 Исключения

Нередко бывает, когда программист совершает ошибки. Они могут быть очень разными: в коде(чаще всего опечатки), в логике, неправильный ввод данных. В этом случае программа прекращает свою работу.

Ошибки в Python правильней будет называть исключениями. Исключения – это отдельный тип данных. С помощью них можно выяснить, какая именно проблема возникла в ходе работы программы(или при запуске).

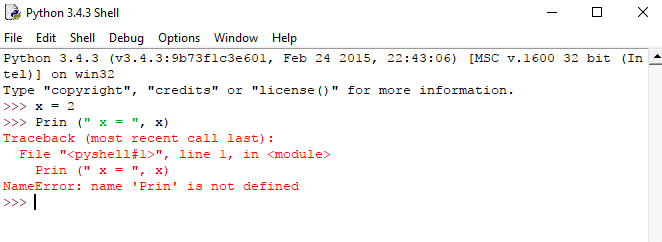


Рисунок 11 – Исключение NameError

В этом случае сработало исключение «NameError» – не найдено переменной/функции с таким именем. Все правильно, ведь мы написали Prin вместо print().

Во второй строке исключения указана строка(line), в которой допущена ошибка.

Еще один пример:

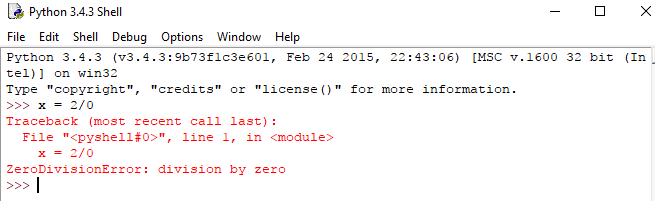


Рисунок 12 – Исключение ZeroDivisionError

Здесь у нас исключение «ZeroDivisionError» – возникает в случае деления на ноль.

В Python важен регистр. Маленькая и большая буква в названии переменных и функций имеют разные значения:

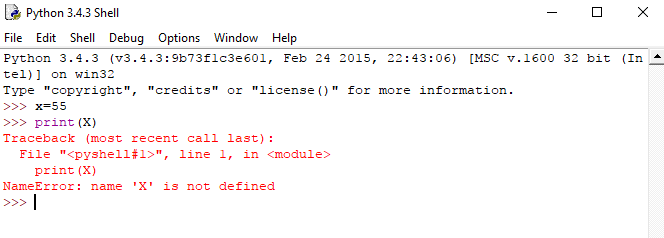


Рисунок 13 – Демонстрация важности регистра

Тут присвоили переменной х значение 55, но пытаемся вывести значение переменной Х.

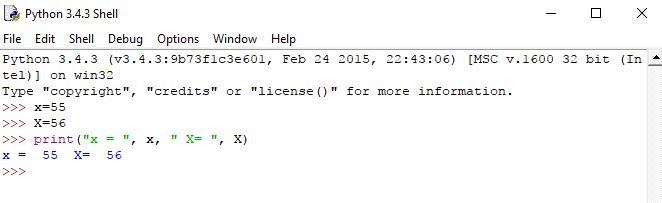


Рисунок 14 – Важность регистра

Видим, что х и Х – совершенно разные переменные.

Пример с функциями:

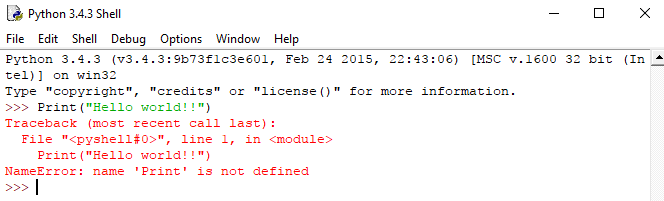


Рисунок 15 – Зависимость названий функций от регистра

В языке существует только print(). Функции Print() нет.

Для того, чтобы в случае возникновения ошибки программа не прекращала свою работу, существует конструкция try-except для обработки исключений. Подробно познакомимся с ней в части 2.

### 

### 1.8.2 Переменные

Python, в отличие от многих языков программирования (Pascal, Basic, Java, Delphi и т.д), не нуждается в описании переменных.

Переменные определяются во время присваивания им какого-либо значения. Тип переменной в этом случае зависит от типа присваиваемого значения.

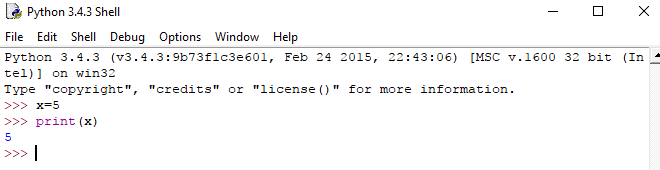


Рисунок 16 – Определение типа переменной во время присваивания

В примере переменной х присвоилось значение 5. Поскольку тип значения числовой – значит, и тип переменной тоже является числовым.

Причем тип переменной не является неизменным. Мы запросто можем присвоить переменной другое значение:

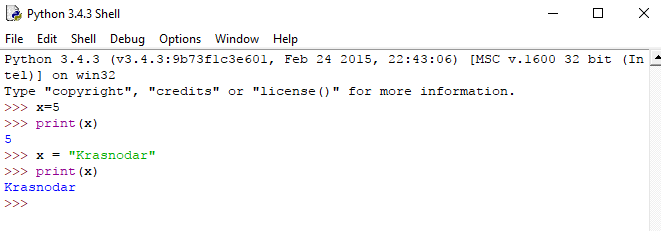


Рисунок 17 – Изменяемость типа переменной

Здесь переменной х было присвоено значение «Krasnodar» строчного типа, а значит, что тип х теперь также строчный

Вообще, в Python, в отличие от других языков, принято говорить, что х ссылается на объект в памяти компьютера. Т.е. во время присваивания значения переменной ей присваивается ссылка на значение. И при вызове функций им передается не фактическое значение переменной, а ссылка. При переприсваивании, как в примере выше, ссылка на объект 5 заменяется ссылкой на объект «Krasnodar».

Интересен случай, когда по обе стороны от «=» стоят переменные, т.е. происходит присваивание одной другой. Тогда просто новая переменная становится лишь другим именем, ссылаясь на ту же область памяти.

### 

### 1.8.3 Группировка операторов

В Python, в отличие от Pascal(где группировка блоков операторов осуществляется за счет конструкции begin-end), Basic(где для таких целей используются команды Next, End if, Wend и тд), Java, C и C++(используются фигурные скобки) группировка осуществляется за счет отступов. Это пробелы или табуляции, ставящиеся в соответствии с заголовком конструкции, которая может при этом быть вложенной в другую конструкцию и, следовательно, тоже быть смещенной вправо относительно «родительской» конструкции.

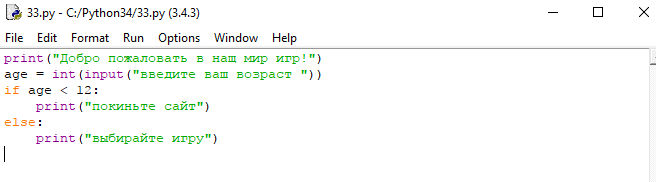


Рисунок 18 – Группировка блоков операторов

Здесь в цикле if блок кода, состоящий из одной строки, сместился вправо.

Пример вложенных конструкций:

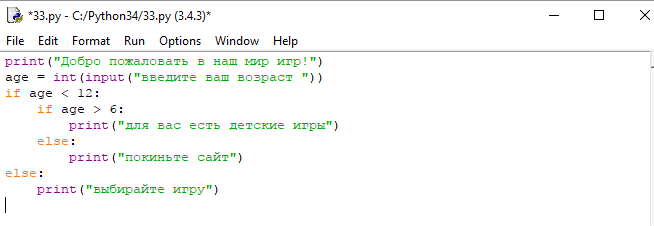


Рисунок 19 – Пример вложенных конструкций

В данном примере цикл if вложен в другой цикл if, поэтому мы можем наблюдать 2 вида табуляции. Один для «главной» конструкции (if age <12), второй, состоящий из двух отступов, для вложенной.

Отличием от других языков является и само обозначение неравенств.

Так, если, например, в Pascal и Basic знак равенства обозначается через равно =, то в Python используется двойное равно ==.

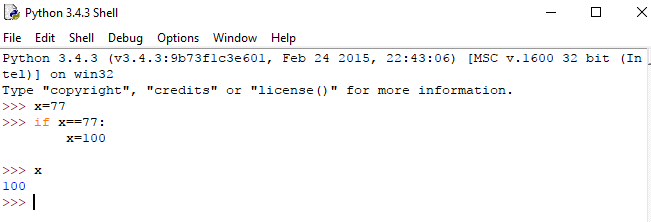


Рисунок 20 – Знаки неравенств в Python

В примере с помощью условного оператора происходит проверка условия. Равно ли х 77? Для проверки использовали два знака равно.

Знак неравенства в Python обозначается !=, а в большинстве других языков используется <>.

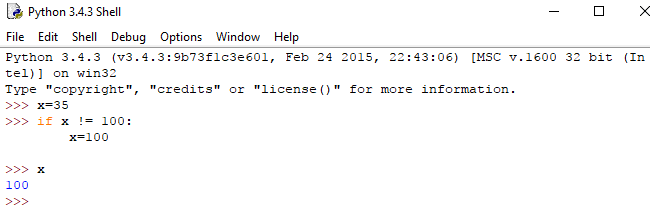


Рисунок 21 – Знак неравенства

Здесь мы проверили условие: х НЕ равен 100?

### 

### 1.8.4 Цикл for

Цикл for в Python также немного отличается от аналогичного в других языках программирования. Так, если в Pascal он имел форму for i = 1 to n do, то в Python немного другой вид: for i in range(начало, конец):

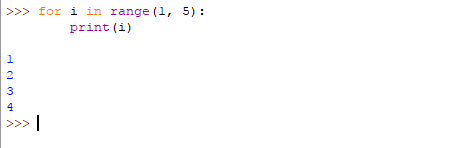


Рисунок 22 – Цикл for

### 

### 1.8.5 Списки

В языках Pascal, C, C++ организация списков была очень неудобной. Для них требовалось хорошо разбираться в работе динамической памяти и указателей. Эта трудоемкость нередко приводила к ошибкам даже у опытных разработчиков. Для упрощения работы со списками были разработаны специальные средства, например, библиотека STL для С++.

Отличительной особенностью Python являются то, что в языке не поддерживается работа с указателями и динамической памятью, что делает его надежным и простым.[5]

Работа со списками очень проста и похожа на работу со строками в Pascal.

Создается он с помощью квадратных скобок.

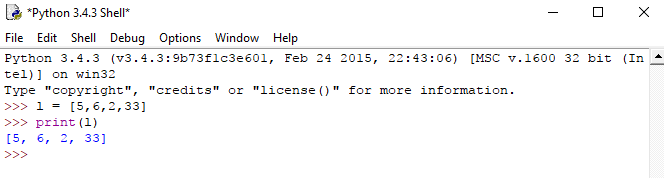


Рисунок 23 – Создание списка

Здесь создали список с 4 элементами. С помощью функции print() можно вывести как весь список, так и отдельные элементы, указав в квадратных скобках номер элемента:



Рисунок 24 – Вывод отдельных элементов списка

Стоит заметить, что индексы элементов списка начинаются с нуля, а значит, максимальный индекс будет равен N-1, где N – количество элементов списка.

Доступ к элементам списка может быть осуществлен и при указании интервала индексов, что делается при помощи двоеточия, что также не характерно для многих языков.

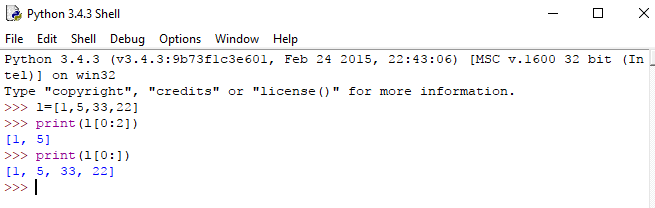


Рисунок 25 – Вывод промежутка списка

В этом случае слева должно быть указано начало промежутка, а справа – конец. Если не будет указано начало, то берется нулевой индекс, а если не указан конец интервала, то максимальный номер индекса.

Интересной особенностью списков в Python является то, что их элементами могут быть значения разных типов данных. В языках, где требуется описание переменных, это невозможно.

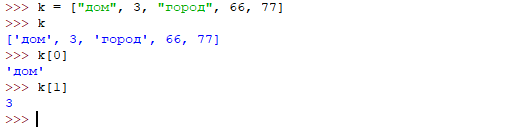


Рисунок 26 – Различные типы данных в одном списке

В примере в списке k содержатся элементы строкового и числового типов.

В списках можно создать матрицу. Для этого достаточно вложить список в список:

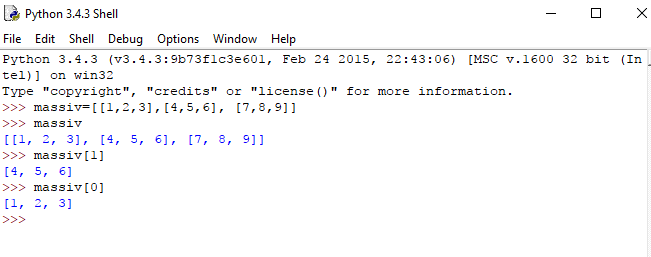


Рисунок 27 – Создание матрицы

В этом случае элементом списка будут элементы другого списка, т.е. «строки» матрицы.

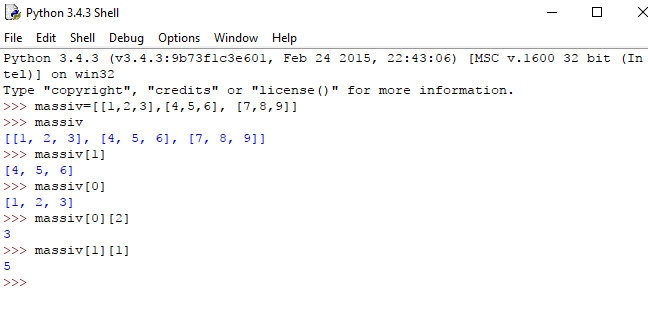


Рисунок 28 – Доступ к элементам матрицы

Здесь мы, не забывая, что индексы в Python начинают считаться с нуля, в первом случае обращаемся к 0 элементу(0 строке матрицы), затем ко 2 элементу( 2 столбцу) этого нулевого элемента.

Еще одной важной особенностью списков, строк, кортежей в Python является доступ к элементам, указывая отрицательные индексы:

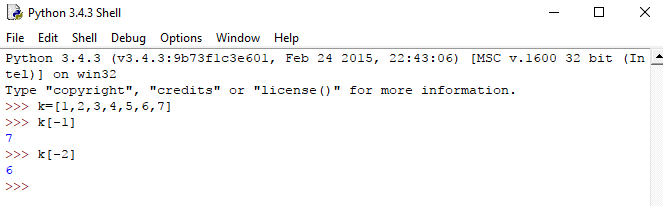


Рисунок 29 – Работа с отрицательными индексами

В этом случае номер -1 равносилен последнему элементу, -2 предпоследнему и т.д.

Эта отличительная черта очень важна, потому что мы можем получить любые элементы конца, не зная при этом их индексы. Этот момент добавляет немалый плюс в упрощение языка.

У списков есть немало методов. Так, append(х) позволяет вставить элемент х в самый конец списка, insert(i,x) вставляет на позицию i элемент х:

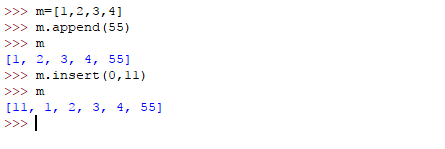


Рисунок 30 – Методы списков

Обратить внимание стоит на метод extend(L) – вставляет список L в конец исходного списка:

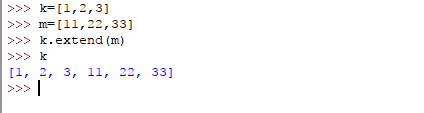


Рисунок 31 – Метод extend()

Метод remove(х) удаляет из списка элемент со значением х:

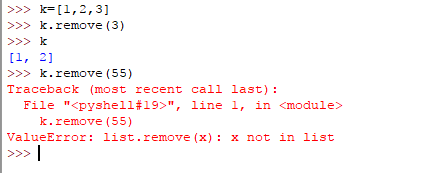


Рисунок 32 – Метод remove()

Если элемента х в списке нет, то выводится исключение ValueError.

Метод clear() полностью очищает список:

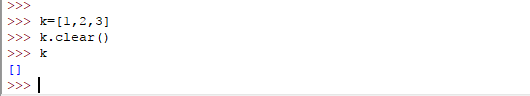


Рисунок 33 – метод clear()

## 

### 1.8.6 Кортежи

Кортежи являются близкими родственниками списков. Одно из очевидных отличий состоит в том, что кортежи, заключаются в круглые скобки.

В отличие от списков, которые позволяют изменять их значения и обладают методами, выполняющими изменения, кортежи относятся к категории неизменяемых объектов, то есть не позволяют изменять свои значения и, отчасти по этой причине, не имеют методов.

Столкнувшись с кортежами впервые, программисты недоумевают, для чего они нужны, если можно использовать списки «только для чтения».

На первый взгляд они правы, но главное назначение кортежей – передавать параметры функциям(при их вызове) и предотвратить возможность изменения их содержимого посторонними функциями.

Это совершенно не означает, что в остальных случаях они бесполезны. Кортежи редко используются самими программистами, но они очень часто используются внутренними механизмами интерпретатора.

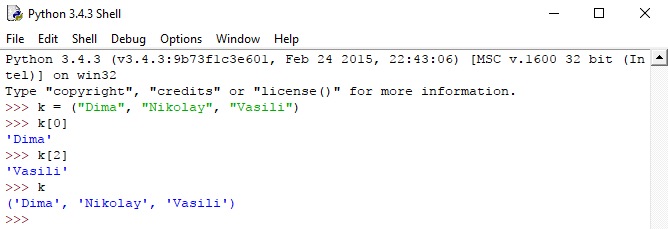


Рисунок 34 – Кортежи

В примере переменной k присваивается ссылка на кортеж с тремя элементами. Доступ к элементам осуществляется так же, как и у списков.

При попытке присвоить второму элементу кортежа новое значение ожидаемо возникает ошибка:

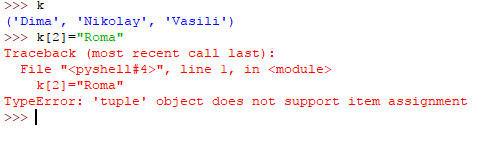


Рисунок 35 – Исключение при присваивании

При определении кортежа, состоящего из одного элемента, обязательно нужно ставить в конце запятую.

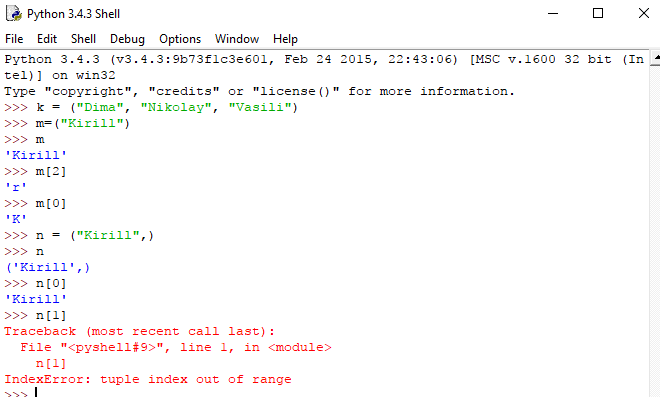


Рисунок 36 – Обозначение кортежа, состоящего из одного элемента

Нужно знать, что не круглые скобки, а запятые задают кортеж, поэтому в примере m – обычная переменная строкового типа, элементами которой являются буквы.

Если же мы поставим запятую, то этот элемент будет не строкой, а элементом кортежа. Поэтому при вызове значения элемента первого индекса возникает ошибка – его попросту нет.

Можно и вовсе отбросить круглые скобки, как это сделано в следующем примере. Однако лучше этого не делать и соблюдать принцип «лучше писать явные определения, чем полагаться на неявное поведение».[6]

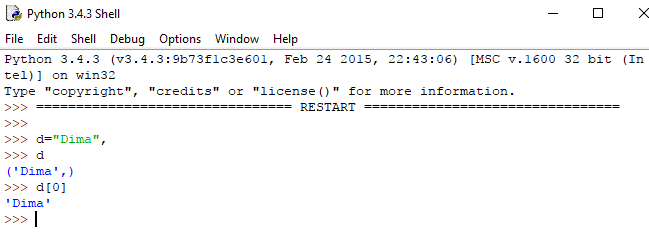


Рисунок 37 – Один из способов обозначения кортежа

### 1.8.7 Словари

Словари являются единственным типом отображений в языке Python. Они являются изменяемыми, неупорядоченными, изменяющимися в размере отображениями ключей на значения и иногда называются хеш-таблицами или ассоциативными массивами. Синтаксис словарей до определенной степени напоминает последовательности, только для доступа к значениям используются не индексы, а ключи, а вместо квадратных (списки) или круглых (кортежи) скобок они определяются с помощью фигурных скобок ({}).

Словари являются одной из важнейших структур данных в языке. Они являются важной составляющей большинства объектов в Python. Независимо от типов объектов и способа их использования высока вероятность, что внутри их заключен словарь, используемый для управления атрибутами объекта. [6]

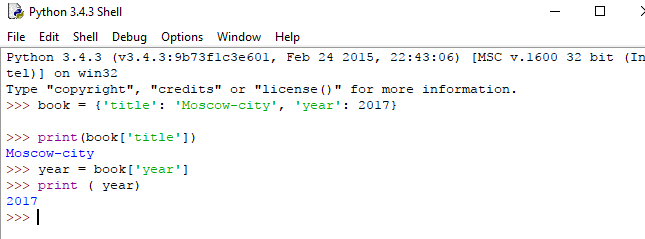


Рисунок 38 – Создание и доступ к элементам словаря

Доступ к элементам словаря осуществляется очень просто. Достаточно к имени словаря дописать квадратные скобки, указав в которых название необходимого элемента.

Умение работы со словарями пригодится нам при создании приложения. В словарях будут содержаться данные, которые нам необходимо обработать.

1.9 База данных PostgreSQL

PostgreSQL не просто реляционная, а объектно-реляционная СУБД. Это даёт ему некоторые преимущества над другими SQL базами данных с открытым исходным кодом, такими как MySQL, MariaDB и Firebird.

Фундаментальная характеристика объектно-реляционной базы данных — это поддержка пользовательских объектов и их поведения, включая типы данных, функции, операции, домены и индексы. Это делает Постгрес невероятно гибким и надежным. Среди прочего, он умеет создавать, хранить и извлекать сложные структуры данных.

# 2 Разработка клиент-серверного приложения

## 2.1 Подготовка к работе

Самое время применить знания о Python на практике!

Будем создавать журнал сделок для трейдера. Для начала откроем командную строку, т.к. все команды в Джанго нам придется осуществлять через нее.

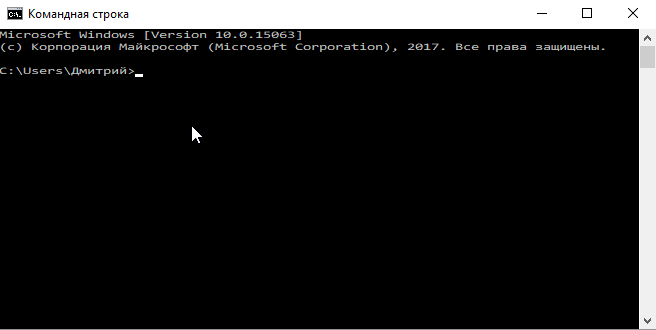


Рисунок 39 – Командная строка

Откроем нашу папку, где будет храниться наш проект. Здесь располагается раннее созданное виртуальное окружение myvenv(my virtual environment), которое позволяет изолировать зависимости Python/Django для каждого отдельного проекта. Это значит, что изменения одного сайта никогда не затронут другие сайты, которые мы разрабатываем.

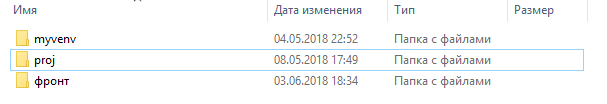


Рисунок 40 – Папки с файлами проекта

Теперь перейдем с помощью командной строки в папку myvenv/Scripts и выполним файл activate – тем самым подключим виртуальное окружение к текущей сессии.

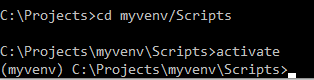


Рисунок 41 – Переход в виртуальное окружение

Как мы видим, перед адресом пути в командной строке появилась название виртуального окружения, заключенного в скобки. Это означает, что подключение прошло успешно.

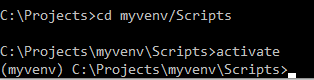


Рисунок 42 – Успешный переход в виртуальное окружение

Физически виртуальное окружение представляет из себя папку, в которой содержатся другие папки с файлами.

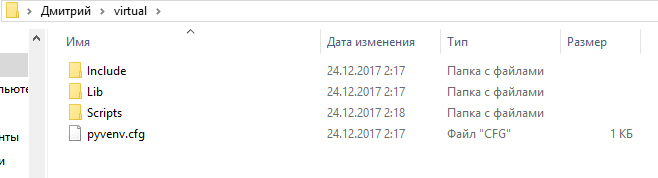


Рисунок 43 – Содержимое виртуального окружения

Далее создаем наш проект следующей командой

3.png

Рисунок 44 – Создание проекта

Проект будет называться journal\_proj. Вот его содержимое:

4.png

Рисунок 45 – Созданный проект

Содержимое папки:

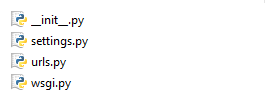


Рисунок 46 – Содержимое jornal\_proj

Файл settings содержит настройки проекта. Их изменения затрагивают все приложения созданные в нем. Если мы его откроем, то увидим следующее:

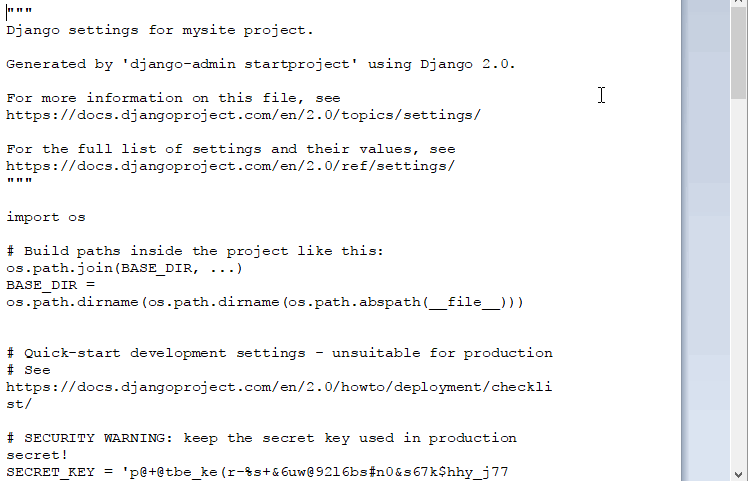


Рисунок 47 – Настройки проекта

Как мы видим, здесь много всяких настроек, но нам понадобятся всего несколько. Установим правильный часовой пояс в строке TIME\_ZONE, поставив ‘Europe/Moscow’, также изменим язык на русский в строке LANGUAGE\_CODE. Это нужно для того, чтобы все уведомления Джанго выводились на русском языке.

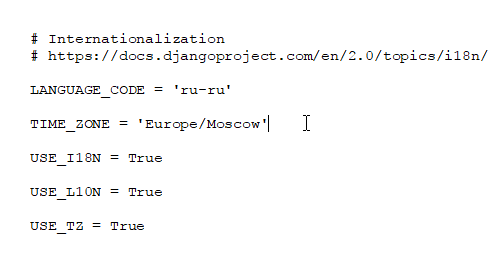


Рисунок 49 – Изменение настроек проекта

В следующем окне настраивается база данных. По умолчанию выбрана стандартная – sqlite3

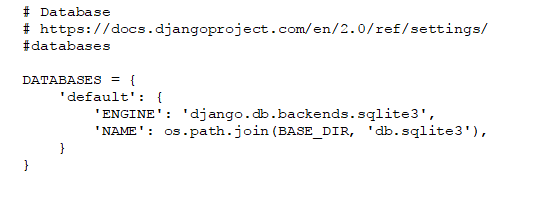


Рисунок 50 – Настройка базы данных

Теперь попробуем запустить сервер и посмотреть, правильно ли работает Джанго. Делается это командой python manage.py runserver

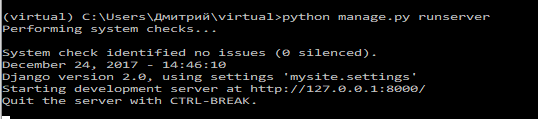


Рисунок 51 – Запуск сервера

В ответе командой строки написано, что ошибок нет, и что на сервер мы можем перейти по адресу <http://127.0.0.1:8000>

Сделаем это, введя в строку браузера адрес

6.4.png

Рисунок 52 – Переход на адрес сервера

Совершив переход, получим следующую страницу:

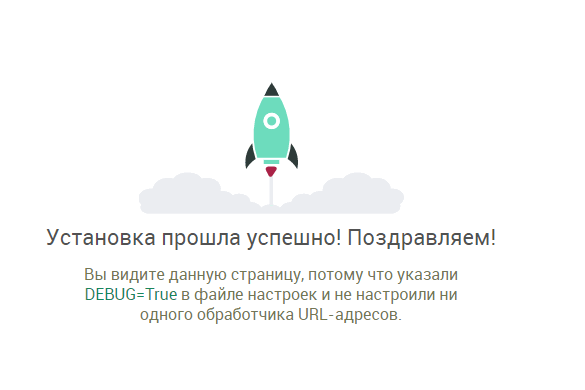


Рисунок 53 – Результат перехода на сервер

Все прошло успешно. Стоит заметить, что эта страница на русском языке, т.к. соответствующий параметр мы указали в настройках.

## 2.2 Приступаем к работе

Создадим теперь приложение под названием «journal»

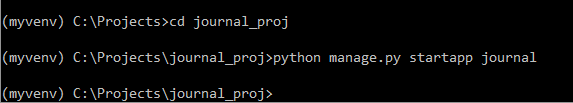


Рисунок 54 – Создание приложения

Рядом с папкой проекта появилась папка с названием приложения

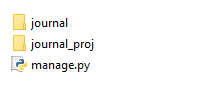


Рисунок 55 – созданное приложение

В ней содержатся следующие файлы:

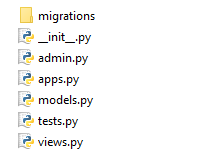


Рисунок 56 – Содержимое приложения

Models.py – нужен для создания моделей данных

Admin.py – для прописывания параметров администрирования

Views.py – представления. Этот файл вызывается в ответ на запрос url-адреса.

Самое время создать базу данных для нашего проекта. Делать это мы будем в PostgreSQL. Для этого откроем PgAdmin 4.

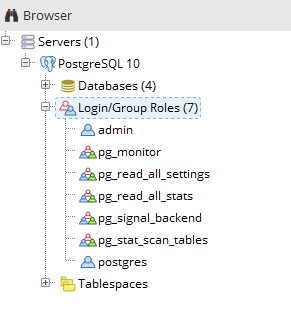


Рисунок 57 – Пользователи базы данных

Для начала создадим суперпользователя, которого назовем admin1 и зададим такой же пароль.

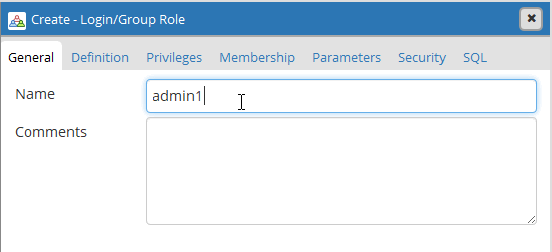


Рисунок 58 – Создание суперпользователя

Зададим ему все права.

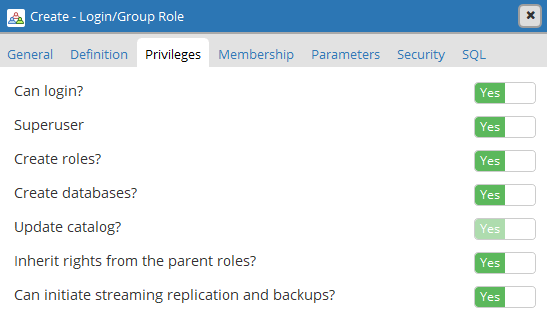


Рисунок 59 – Задание прав

Теперь создадим базу данных, которую назовем jour\_db и укажем в качестве владельца нами созданного суперпользователя admin1.

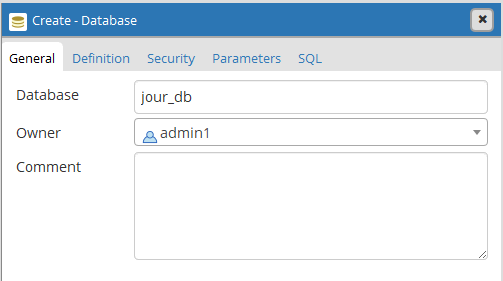


Рисунок 60 – Создание базы данных

Зайдем теперь обратно в настройки проекта и укажем туда нашу базу данных и укажем данные для суперпользователя.

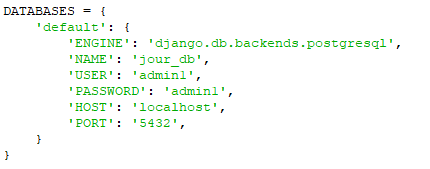


Рисунок 61 – Добавление базы в настройки проекта

## 2.3 Настройка urls.py

Перейдем теперь папку нашего проекта и откроем файл urls.py

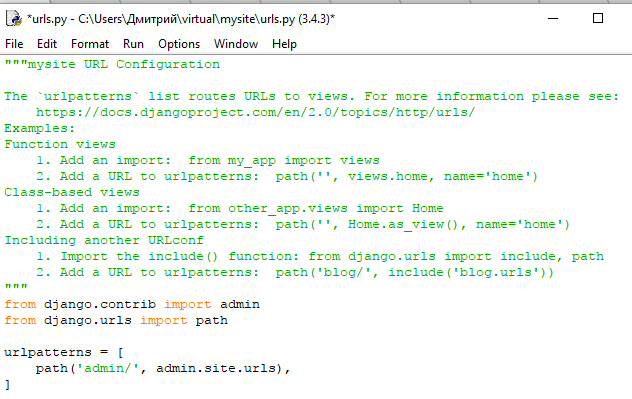


Рисунок 62 – файл urls.py

Как мы помним, urls – это механизм доступа к представлениям. В списке urlpatterns содержатся шаблоны адресов и либо переадресация на другой файл urls либо отображение соответствующих им представлений.

Поскольку запросы идут через urls файл проекта, нам нужно сделать так, чтобы он ссылался на urls файл нашего приложения. Для этого импортируем функцию include, которая позволит это сделать и запишем в список необходимый код.

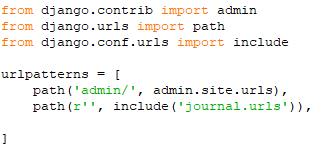


Рисунок 63 – Добавление urls созданного приложения

Django сопоставляет URL–адреса и представления. URL–адрес записывается с помощью регулярных выражений, которые выделяются кавычками. Если в них ничего нет, то подразумевается адрес <http://127.0.0.1:8000>.

Перед регулярными выражениями желательно ставить букву r.  Это намёк для Python, что строка может содержать специальные символы, предназначенные не для самого Python, а для регулярного выражения.

Перейдем в папку нашего приложения и создадим в нем аналогичный файл с адресами. Структура будет такой же.

Начальному адресу <http://127.0.0.1:8000> будет соответствовать функция представления reg – окно с регистрацией пользователя.

Адресу <http://127.0.0.1:8000/auth> - функция authoriz - окно с авторизацией.

Адресу <http://127.0.0.1:8000/main> - функция main - главное меню.

Адресу <http://127.0.0.1:8000/write> - функция writing - окно для заполнения журнала.

Адресу <http://127.0.0.1:8000/jour> - функция read\_jour - окно с содержимым журнала.

Все эти функции будут созданы нами чуть позже.

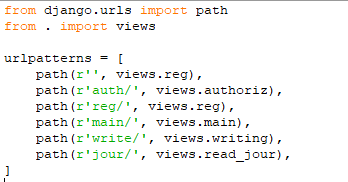


Рисунок 64 – Создание файла с адресами для приложения

## 

## 2.4 Создание моделей

Теперь создадим модели – таблицы в нашей базе данных.

Для этого откроем файл models.py и добавим туда следующий текст:

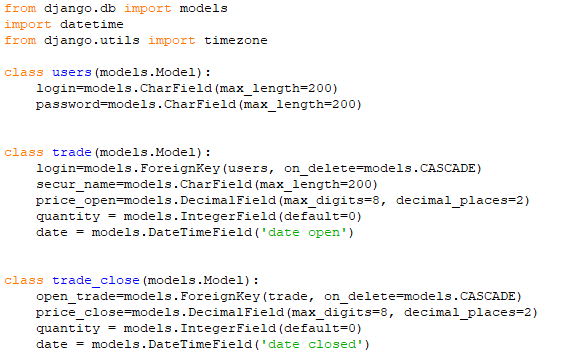


Рисунок 65 – Модели

Таким образом, у нас будет всего 3 таблицы в базе данных.

Первая называется users – список пользователей, которая будет хранить в себе логины и пароли. Тип CharField – текстовые значения. Максимальной длиной укажем 200 символов.

Вторая - trade – список открытых сделок. Содержит в себе номер авторизованного пользователя, который является внешним ключом. При удалении пользователя из системы будут удалены все открытые сделки, связанные с ним (благодаря параметру on\_delete = models.CASCADE). Также эта таблица будет содержать в себе название акции, цену, по которой была открыта сделка, количество акций и дату с временем.

Третья – trade\_close – список закрытых сделок. Содержит в себе номер открытой сделки в качестве внешнего ключа, цену закрытия, количество акций и дату с временем.

Необходимость в создании двух таблиц вместо одной возникла из-за того, что сделки могут закрываться частично, со временем. Либо вообще не закрываться долгий период времени.

Далее с помощью команды Python manage.py makemigrations journal фиксируем список изменений в базе данных, а затем с помощью команды migrate записываем таблицы в базу.

## 

## 2.5 Создание представлений и шаблонов

Перед созданием представления, создадим файлы шаблонов – клиентскую часть. Всего их будет 5.

Сделаем мы это на языке html.

Выглядеть они будут следующим образом

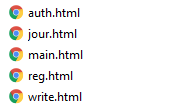


Рисунок 66 – Шаблоны html

У нас будет пять функций представлений – столько же, сколько и шаблонов, которые мы указали в файле urls.py.

В первую очередь импортируем наши модели из файла models.py

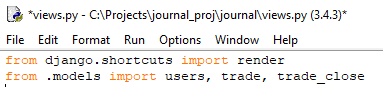


Рисунок 67 – Импорт моделей

Создадим функцию reg, которая будет отвечать за регистрацию новых пользователей. В качестве аргумента она будет принимать POST-запрос, который будет исходить из клиентской части. Затем мы присваиваем переменным значения, которые мы получаем из запроса методом get(). В кавычках при этом указываем название формы ввода (у нас будет 2 формы ввода: одна для ввода логина, которую мы назовем login, а другая для пароля, которую назовем password).

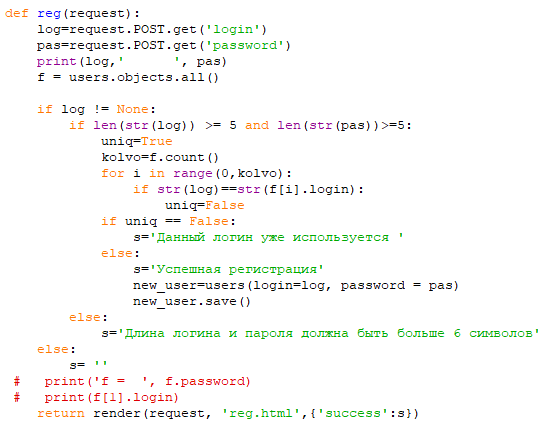


Рисунок 68 – функция представления reg

Далее делаем проверку – если входящий запрос не пустой, то затем проверяем больше ли логин и пароль 5 символов или нет. Если и эта проверка проходит, тогда с помощью цикла for проверяем, имеются ли в таблице users(все записи которой мы в самом начале присваиваем переменной f) логины с таким же названием. Если да, то выведем сообщение о том, что такой логин уже используется, иначе – выведем сообщение об успешной регистрации и сделаем запись в таблицу users.

Сообщение будет передаваться в переменной s, которая в шаблоне будет иметь имя success. И укажем, что эта функция представления связана с шаблоном reg.html, который мы сейчас заполним.

Откроем reg.html, где создадим 2 формы для ввода, которые назовем login и password, а также кнопку для отправки данных на сервер. В самом верху мы указываем метод POST, чуть ниже добавляем пересылаемую переменную success, заключив ее в двойные фигурные скобки.

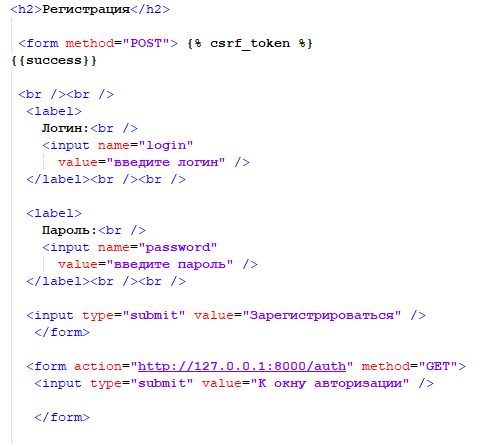


Рисунок 69 – Шаблон для функции reg

Так же добавим кнопку «К окну авторизации», которая переместит нас по адресу, который будет содержать это окно. Цвет фона сделаем светло-салатовым.

В итоге получим такую форму:

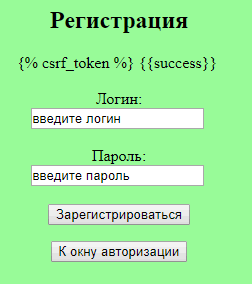


Рисунок 70 – Результат написания шаблона

Поскольку мы еще не запустили сервер, передаваемые переменные не воспринимаются браузером, как теги Джанго.

Теперь создадим функцию представления для авторизации.

В переменные log и pas мы, как и в случае с регистрацией записываем значения полей для ввода логина и пароля. Создаем булевскую переменную auth\_f, в которой будет содержаться состояние авторизации – авторизован пользователь или нет. Далее также мы проверяем, является ли запрос пустым. Если нет, то пытаемся найти в таблице users элемент, который содержит такую комбинацию логина-пароля, если все произошло успешно, то выводим сообщение об успешной авторизации и переменной auth\_f присваиваем значение True. Если вдруг такая комбинация не была найдена в базе, то выведется соответствующее сообщение.

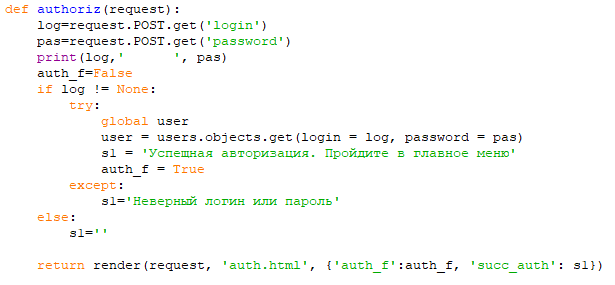


Рисунок 71 – Функция представления для авторизации

Сообщения об успехе/неудаче передаем в клиентскую часть в переменной s1, которая там будет называться succ\_auth, а переменную с результатом передадим с таким же именем.

Так же укажем шаблон, который соответствует этой функции – auth.html, который мы сейчас заполним.

Шаблон будет аналогичным шаблону с регистрацией. Единственное отличие – добавим кнопку «Зарегистрироваться», которая нас направит к окну с регистрацией. Так же добавим кнопку главное меню, которая станет доступной для пользователя только в том случае, если он будет авторизован – переменная состояния auth\_f, которую мы передаем из функции представления, будет равна True.

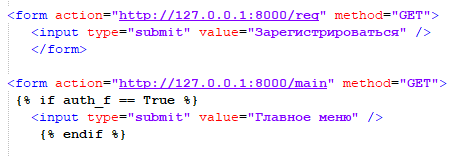


Рисунок 72 – Шаблон для авторизации

Теперь сделаем функцию для главного меню, которую назовем main.

Состоять будет из одной строки, где укажем, что она соответствует шаблону main.html



Рисунок 73 – Функция для главного меню

Займемся шаблоном главного меню. Оно будет состоять из 3 кнопок – Заполнить журнал, Посмотреть журнал, Выйти.

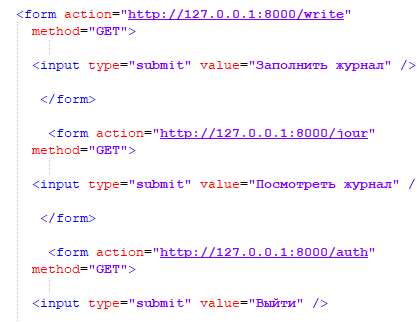


Рисунок 74 – Шаблон главного меню

Кнопка выхода будет ссылаться на окно с авторизацией.

Вернемся к представлениям и создадим функцию для заполнения журнала. Поскольку у нас на одной странице будет форма для заполнения как открытых сделок, так и нет, то разобьем нашу функцию на две части.

В первой части мы будем обслуживать запросы для новых сделок. Здесь все так же, как и в случае с регистрацией – если запрос не пустой, вычленяем из него название акции, цену, по которой была открыта сделка, количество купленных сделок, дату и время (их мы суммируем, т.к. у нас в таблице поле типа datetime – принимает и дату и время сразу, т.е разделение не требуется). Затем записываем данные в таблицу trade.

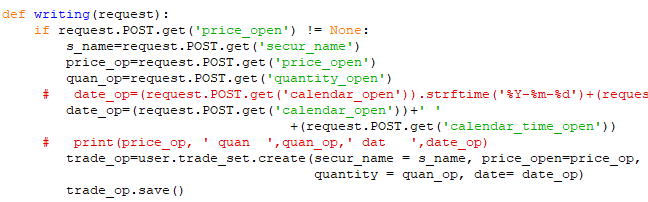


Рисунок 75 – Функция для заполнения журнала

Во второй части мы обрабатываем запросы для формы, где закрывается открытая ранее сделка.

Здесь все аналогично первой части. Только переменной tr\_list присваивается список всех открытых сделок, принадлежащих авторизованному пользователю, который передается в клиентскую часть, где пользователь сможет выбрать какую именно сделку закрыть. Результат вернется в запросе в переменной id\_trade, значение которой мы присвоим переменной id\_tr.

## 

Рисунок 76 – Вторая часть функции для заполнения журнала

Данную функцию мы свяжем с шаблоном write.html.

Заполним его. Для формы открытия сделки нужно 5 форм для ввода информации – название акции, цена, количество, дата и время. Также кнопка «Отправить» - для передачи введенных данных на сервер.

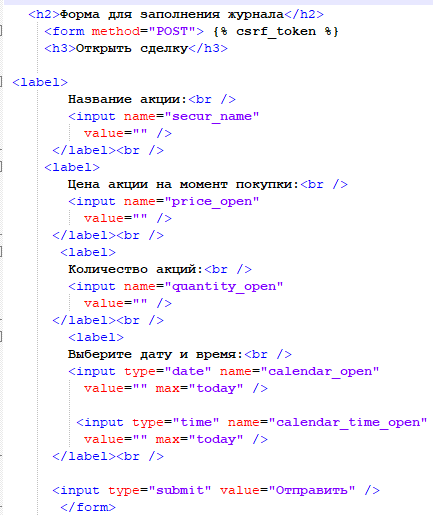


Рисунок 77 – Шаблон заполнения журнала

Далее создадим форму для закрытия сделки. Здесь все аналогично, только эта форма принимает список открытых сделок, из которых с помощью цикла формируется набор номеров открытых сделок для авторизованного пользователя. Делается это с помощью html-тега <select>. Каждый возможный вариант пишется во вложенный тег <option>.

## 

Рисунок 78 – Шаблон заполнения журнала

Посмотрим, что получилось.

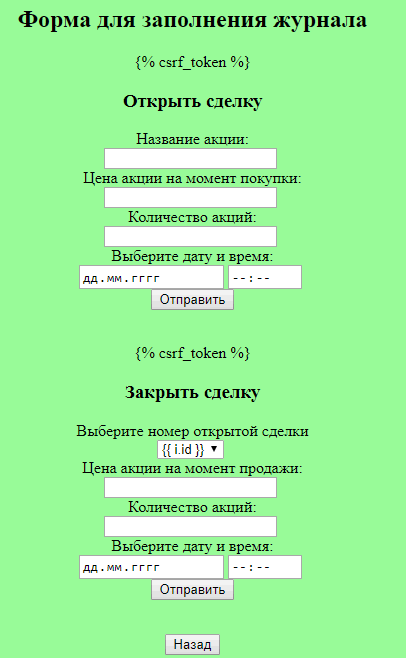


Рисунок 79 – Шаблон заполнения журнала в браузере

Создадим теперь последнюю функцию представления, которая позволит нам прочитать журнал следок. Свяжем ее с шаблоном jour.html, куда будем отправлять переменную tr\_op, которой мы присвоим все сделки, принадлежащие авторизованному пользователю.



Рисунок 80 – Функция для чтения журнала

Заполним теперь шаблон jour.html. Выводить содержимое журнала мы будем в виде таблицы. Таблица задается с помощью тега <table>, после которого задаем параметры таблицы, ее ширину.

Строки таблицы задаются с помощью тега <tr>, а столбцы с помощью <th> (выделяет содержимое жирным) или <td>. Первая строка у нас будет «константой» - в ней будут содержаться названия 8 столбцов нашей таблицы.

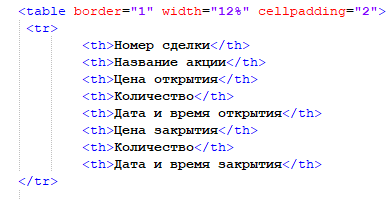


Рисунок 81 – Шаблон чтения журнала

Теперь с помощью цикла осуществим перебор открытых сделок. В каждой строке будет выводиться содержимое каждой открытой сделки, там же будет осуществляться проверка, есть ли у данной открытой сделки какие-нибудь соответствующие ей закрытые сделки. Если да, то выводится первая соответствующая сделка(все остальные будут выведены потом, т.к. для них нужна новая строка). Если нет, то столбцы, которые соответствуют закрытым сделкам, заполняются пустым для данной строки.

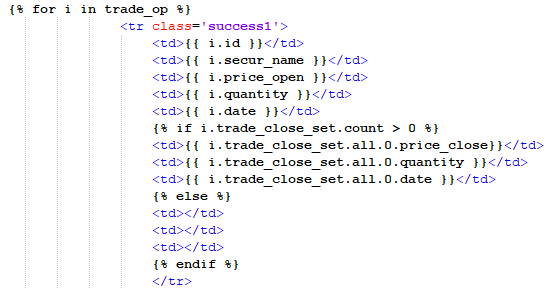


Рисунок 82 – Шаблон чтения журнала

Далее осуществляется проверка. Если у данной открытой сделки есть более одной закрытой, то с помощью цикла осуществляется их перебор. Причем заполнять мы начинаем со второго элемента, т.к. первый заполняется с помощью кода выше. Реализуется это благодаря проверке счетчика цикла forloop.counter. Столбцы для открытых сделок мы заполняем пустыми значениями.

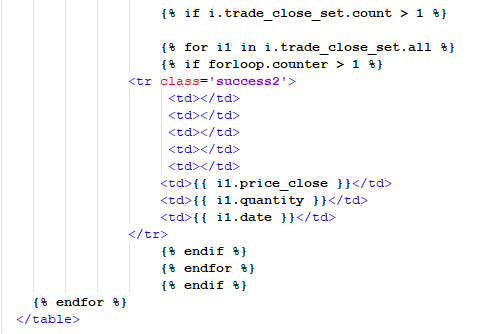


Рисунок 83 – Шаблон чтения журнала

## 2.6 Тестирование

Настало время посмотреть на результаты проделанной работы.

Перейдем к окну с регистрацией. Попробуем ввести логин и пароль меньше 5 символов.

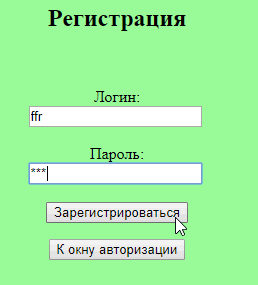


Рисунок 84 – Попытка ввода некорректных данных

Все правильно – выводится оповещение, и пользователь не регистрируется.

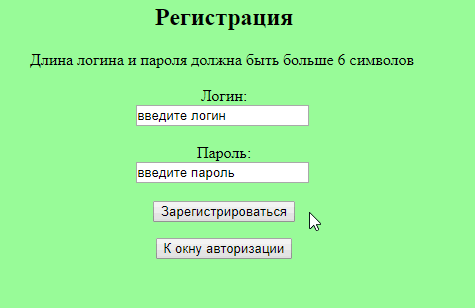


Рисунок 85 – Сообщение об ошибке

Попробуем ввести теперь корректный по длине логин и пароль.

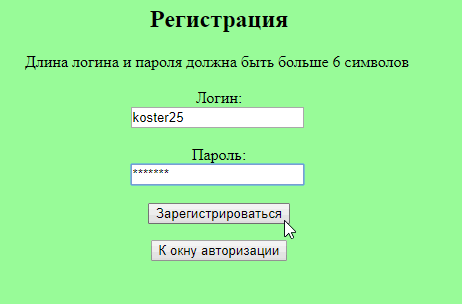


Рисунок 86 – Ввод данных

Как мы видим, регистрация прошла успешно, о чем и говорит нам оповещение.

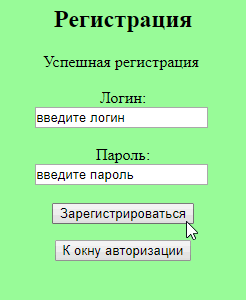


Рисунок 87 – Сообщение об успешной авторизации

Перейдем теперь к окну с авторизацией

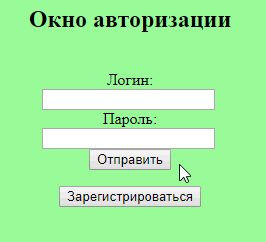


Рисунок 88 – Окно с авторизацией

Попробуем ввести несуществующий логин и пароль.

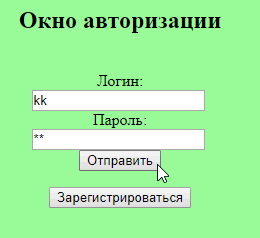


Рисунок 89 – Попытка ввода некорректных данных

Нам будет выведено соответствующее сообщение.

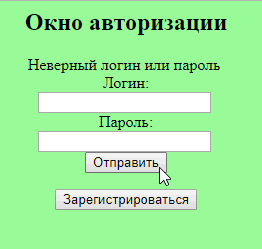


Рисунок 90 – Сообщение об ошибке

Введем теперь корректные данные – ранее зарегистрированного пользователя.

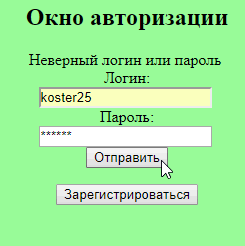


Рисунок 91 – Ввод корректных данных

Как видно авторизация прошла успешно, и внизу стала доступной кнопка для перехода в главное меню журнала.

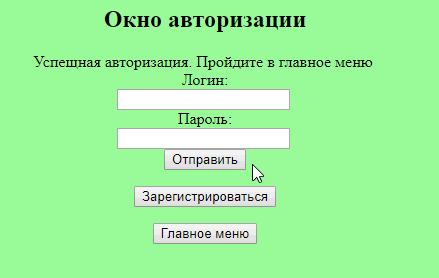


Рисунок 92 – Сообщение об успешной авторизации

Перейдем теперь в главное меню.

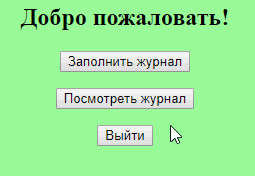


Рисунок 93 – Главное меню

Откроем вкладку для заполнения журнала.

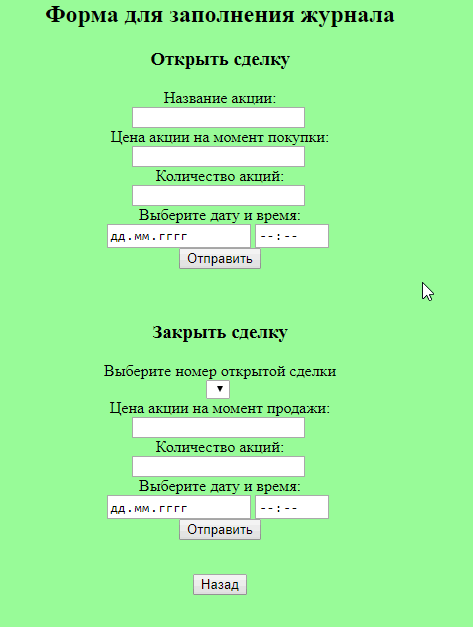


Рисунок 94 – Окно для заполнения журнала

Откроем сделку. Пусть названием акции будет «Сбербанк», количество 100, цена покупки 221.55, дата 22 июня 2018 и время 11:40. Нажмем на кнопку «Отправить»

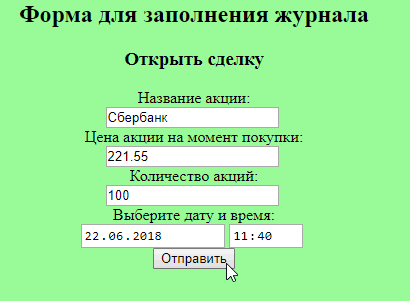


Рисунок 95 – Открытие сделки

Заполним теперь еще одну сделку для Газпрома.

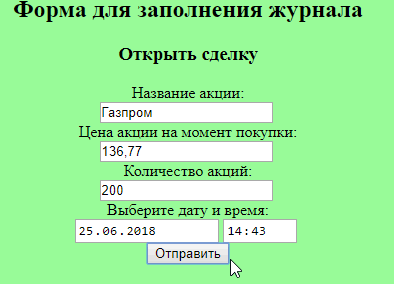


Рисунок 96 – Покупка акций Газпрома

Затем закроем сделку для Сбербанка. Для этого в окне «Закрыть сделку» в качестве номера открытой сделки выбираем 1. Номера сделок можно посмотреть в журнале.



Рисунок 97 – Закрытие сделки

Сделку закроем частично – если мы покупали 100 акций, то продадим 30. И зафиксируем это в журнале, нажав на кнопку отправки данных.

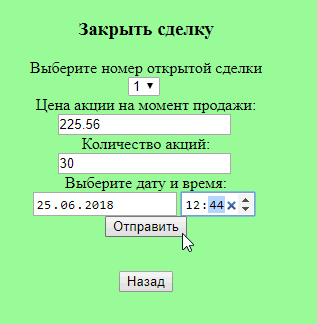


Рисунок 98 – Закрытие сделки для Сбербанка

Допустим, спустя несколько часов мы опять частично продадим акции. Внесем данные в окно.

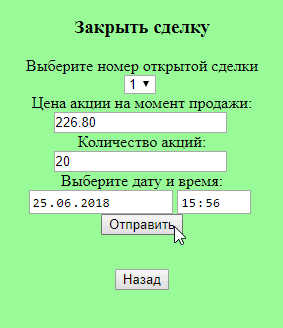


Рисунок 99 – Продажа акций Сбербанка

Посмотрим теперь сделанные записи в журнале. Нажмем на кнопку «Назад», после чего мы попадаем в главное меню. Далее переходим во вкладку просмотра журнала.

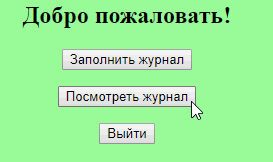


Рисунок 100 – Главное меню

Получаем следующую таблицу

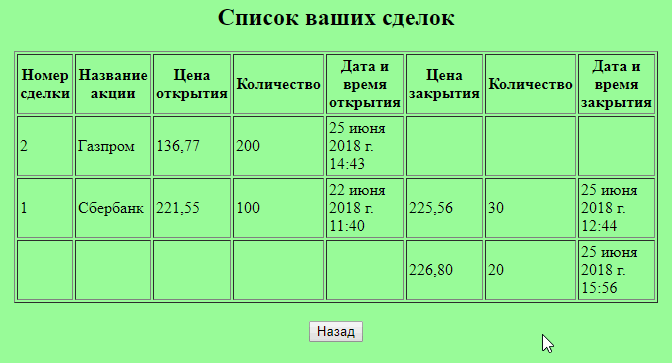


Рисунок 101 – Список сделок

Вернемся назад и откроем сделку с акцией Лукойл. И посмотрим на результат:

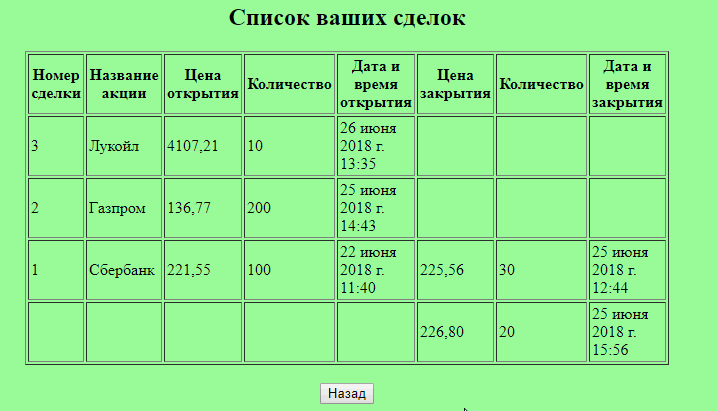


Рисунок 102 – Список сделок после добавления значения

Обратно идем назад и пробуем закрыть третью сделку.

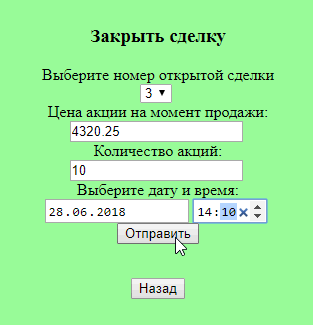


Рисунок 103 – Закрытие сделки для Лукойла

Смотрим на результат.

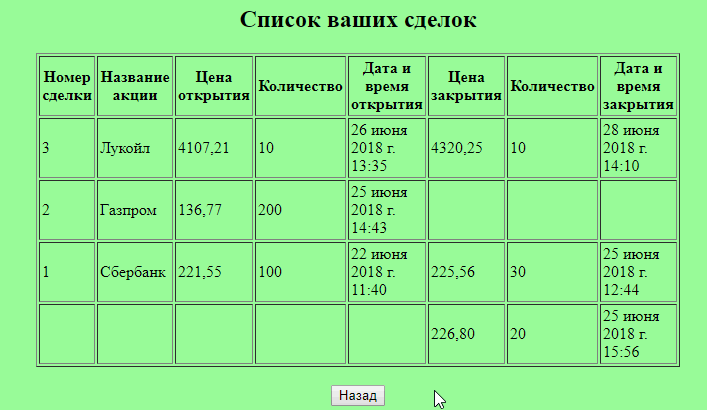


Рисунок 104 – Список сделок

Все работает.

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы была достигнута поставленная цель, а именно, разработано клиент-серверное приложение на Python Django.

Теоретическая часть помогла избежать лишних и абсолютно ненужных ошибок и сыграла немалую роль в понимании основных процессов и структур. В частности сильно пригодилось умение работы со словарями при получении запроса из клиентской части, а так же навыки извлечения элементов из него.

Возможности фраймворка Django языка Python очень велики, что доказывается его активным использованием многими разработчиками.

Главное достоинство Python – простота, в которой мы убедились, достигая нашей цели – создавая клиент-серверное приложение, и о которой говорится в послании разработчиков – философии Python. В большей степени к упрощению написания программ приводит отсутствие необходимости объявлять переменные, тип которых определяется по ходу написания кода автоматически, а также удобная система отступов, что придает коду компактность и структурированность. Наличие большого количества библиотек и инструментов позволяет решать широкий спектр задач различных областей.

Подводя итоги, можно сделать вывод, что Python вполне может подойти для изучения как начинающим программистам, так разработчикам с большим опытом.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Краткая история языков программирования. URL: <http://younglinux.info/python/programminglanguage.php> (дата обращения: 12.06.2018).

2 Wikipedia, Python. URL: <https://ru.wikipedia.org/wiki/Python> (дата обращения: 12.06.2018).

3 A Brief Timeline of Python. URL:[http://python–history.blogspot.ru/2009/01/brief–timeline–of–python.html](http://python-history.blogspot.ru/2009/01/brief-timeline-of-python.html) (дата обращения: 12.06.2018).

4 Wikipedia, История языка программирования Python. URL:<https://ru.wikipedia.org/wiki/История_языка_программирования_Python> (дата обращения: 12.06.2018).

5 Краткий обзор языка Python. URL:<http://www.helloworld.ru/texts/comp/lang/python/python2/index.htm> (дата обращения: 13.06.2018).

6 Форсье Дж., Django. Разработка веб–приложений на Python, 2009. –456 c.

7 История Django. URL: <https://djbook.ru/ch01s03.html> (дата обращения: 13.06.2018).

8 Python Django. URL: [http://ep–z.ru/stroitelstvo/sayt/python/python–django](http://ep-z.ru/stroitelstvo/sayt/python/python-django) (дата обращения: 14.06.2018).

9 Достоинства PostrgeSQL. URL: <https://habr.com/post/282764/> (дата обращения: 14.06.2018).